PERFORMANCE ANALYSIS OF ASYNCHRONOUS CIRCUITS AND SYSTEMS

by

Aiguo Xie

A Dissertation Presented to the
FACULTY OF THE GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(Electrical Engineering)

August 1999

To my parents and sisters.

Acknowledgements

This work would not have been possible without the help of many other people. First among these is my graduate and thesis advisor, Peter A. Beerel. Exactly four years ago, Peter brought me oversea to start this work at USC. He is enthusiastic and is always ready for discussions. Many of the ideas presented in this thesis were inspired during the countless talks with him, sometimes, over the phone deep in the night. He has been patient with me over these years. I received tremendous encouragement from him, especially when I felt down because of the lack of general appreciations of the topic from the field. To a large extent, it was him who saved this work. What I have learned from him is not just a set of technical skills, but has also greatly shaped the way I view our life.

Other members of my program committee, Massoud Pedram, Melvin A. Breuer, Michel Dubois and Peter H. Baxendale (Math Department), have been extremely helpful as well. They carefully read through my thesis proposal and draft. Massoud made critical comments on the state-compression approach for Markovian analysis. Melvin's comments significantly improved the material presented in the introduction chapter. Michel introduced me much of the contemporary work on performance evaluation in the area of computer architecture, which benefited me throughout the program and enlarged the application domain of this work. Lastly, but not the least, Peter (Baxendale) taught me much of the math applied in this thesis. I have been able to catch him for hour-long advisements on technical problems. His quick tutorial on ergodic theory was particularly enlightening. It guided me to prove the almost sure convergence property of TSE sequences, which became the corner stone of several major techniques developed in both parts of the thesis.

Throughout this work, I also received great encouragement and insightful comments from other experts in the field. Among them, I would like to thank especially Alex Yakovlev, Mark Greenstreet, Alex Kondratyev, Ken Stevens, Ken Y. Yun and

Chris J. Myers. Alex (Yakovlev) challenged me with the problem of analyzing his tree-arbiters. He also motivated me to generalize the statistical bounding methods to k-safe rather than 1-safe Petri nets. Mark's comments inspired the idea of applying the state-compression approach to continuous time Markov chains. Alex (Kondratyev), Ken (Stevens), Ken (Yun) and Chris have always been my external support to continue the topic.

There are many other specialists who are slightly outside the field, but have also contributed to this work. Fabio Somezi, E. Allen Emerson, George Papavassilopoulos and Peter J. Haas (IBM) deserve special thanks. A dialog with Fabio led to the idea of using BDDs to decompose a digraph into its strongly connected components (SCCs) (not just the terminal ones). This motivated the recursive SCC-decomposition algorithms summarized in Chapter 5. Discussions with Allen motivated me to looked at a potential application of the SCC-decomposition to bad-cycle-detection problems in model checking. George mentioned to me the classic book on Markov chains by Kai Lai Chung [34], which saved me an unnecessary detour. Recent discussions with Peter (Haas) on estimation of stochastic Petri nets further motivated me to examine possible solutions to asymmetric-choice nets.

I am in debt to my officemates at USC, past and current. They are Ishwardutt Parulkar, Youpyo Hong, Wei-Chun Chou, Peter Yeh, Vida Vakilotojar and Sangyun Kim. Thank you all for the interesting technical chats, "on-line" programming tips and frequent entertainments, all of which have made my stay at USC unforgettable.

There is still a long list of other people who have contributed to this work one way or another. I even cannot recall some of their names, but let me try my best. Mary Zittercob gave great support on documentation from time to time. Janet Martin (my advisor's fiancee) went through several of my paper submissions and presentation rehearsals. She was also the author of many of the figures in the chapters, including the illustration of the Petri net model of the Intel's RAPPID. Finally, the comments from Xiaogang Qiu, Lei Zhuge, Wenlong Dong and Eric Mercer at different stages of this work were constructive as well.

Contents

List of Tables

List of Figures

Abstract

In some applications, well-designed asynchronous systems can dramatically outperform their synchronous counterparts in terms of average speed. One recent example is the Revolving Asynchronous Intel Pentium Processor Instruction Decoder (RAPPID) which achieves 3 times higher average throughput than its comparable synchronous counterpart [103, 115]. Such systems must be highly concurrent and optimized for their average-case performance. Unfortunately, analyzing the performance of asynchronous systems is challenging, and there is an extreme lack of supporting CAD tools. This thesis presents two categories of performance analysis methodologies. The methods in the first category compute the *theoretical* (or exact) values of average performance metrics by modeling systems using Markov chains, and subsequently solving the models using Markovian analysis. The thesis developed sevearl novel methods to attack the *state explosion problem* associated with Markovian analysis of large systems. The methods in the second category compute lower and upper *bounds* on performance metrics of systems that are modeld using stochastic timed Petri nets. The bounds are derived using *finite net executions*, and are evaluated using statistical methods. These methods do not require a state-level analysis, and thus can handle systems of very large size. As an example, a Petri net model of RAPPID with over 900 transitions and 500 places has been successfully analyzed. The resulting bounds are typically much sharper than using any other known method, and are in fact very close to the theoretical values. Moreover, the theory behind the bounding techniques provides a foundation for performance optimization of complex asynchronous circuits and systems. As one of their distinct features, the techniques in both categories are developed to handle arbitrarily dsitributed delays in order to increase the modeling power. All these techniques have been extensively tested and each represents the current state-of-the-art in its

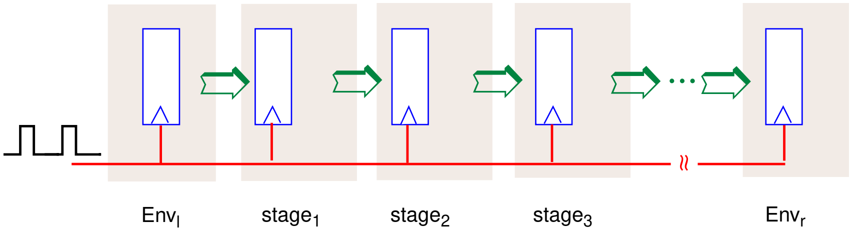
own respect. Due to the generality of the problems addressed, the potential applications of the work presented in this thesis are well beyond asynchronous circuits and systems. For instance, they are directly applicable to a wide class of *discrete event systems* such as embedded systems, database systems and security-oriented networks.

Chapter 1

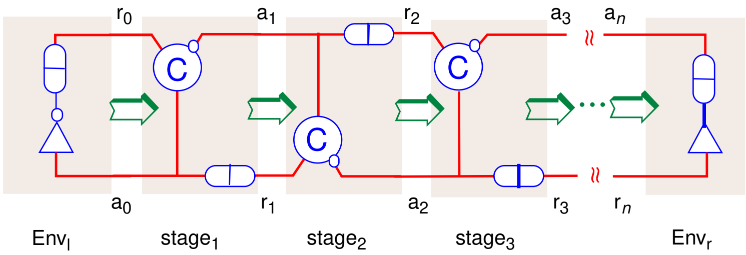
About This Dissertation

1.1 Introduction

Correct operations of digital circuits and systems require proper synchronization of various events. For instance, a comparison of two real numbers cannot be effectively completed until both numbers are known. Physically, such requirements translate to proper *synchronization* of the activities of systems components at various levels of abstraction. Traditionally, digital circuits have been designed using *synchronous*, *asynchronous* schemes and sometimes a combination of the two to achieve such inter-component event synchronization. The activities of *synchronous circuits* are typically sequenced by a *global periodic control signal* commonly referred to as the *clock*. The designers ensure that the operation of every component is completed before certain clock signal arrives. This condition is checked for all possible circumstances under which the components may operate, such as different input combinations, power supplies and chip temperatures. As a result, synchronization among system components is *implicitly* achieved in the sense that the outputs from every component are valid and ready to be consumed before the next clock signal arrives. Consequently, observable events occur only at time instances marked by the arrivals of clock signals. On the other hand, the design of *asynchronous circuits* achieves synchronization *explicitly* by making the outputs of the components available as soon as the they are produced. Typically, this is accomplished through extra *control circuits*. The data-components inform the control circuits the time when their outputs are valid, and the control circuits schedule the operations of the data-components at the time when the new inputs are available. As an example,



(a) A synchronous pipeline,



(b) an asynchronous pipeline.

Figure 1.1: A synchronous pipeline vs an asynchronous counterpart.

Figure 1.1 shows a pipeline circuit designed in two different design styles. In the synchronous pipeline, data move from left to right one stage per clock cycle whereas the data movement in an asynchronous pipeline [118] is controlled locally by the processing status (e.g., `busy` or `idle`) of the neighborhood stages.

Primarily because of their implicit synchronization mechanism, synchronous designs have significantly lower complexity than asynchronous ones in both synthesis and verification. However, asynchronous designs may achieve significantly higher *average* performance than their synchronous counterparts [118, 84, 125, 136, 103, 115] since the data-processing speeds of their components are not determined by the corresponding worst-case delays as is true in synchronous designs.

For instance, in Figure 1.1, the throughput of the synchronous pipeline is determined by the *cycle time* of the clock signal (clock cycle time hereafter) which must be set at least as the worst-case data-processing delay among all the stages[1]. Formally, let us denote the data-processing time of stage i by an interval *random*

[1]Of course, the clock cycle time needs to take into account other factors such as the setup and hold times of latches/registers and possible clock skews. In synchronous pipelines using latches, recent *clock borrowing techniques* may improve the throughput. However, it can be shown that the clock cycle time cannot be smaller than the *amortized* worst-case delays over all the stages.

variable $d_i \in [\delta_i, \Delta_i]$. Then, ignoring other overheads such as setup and hold times of the data latches, the clock cycle time of a synchronous pipeline with n stages is at least $T^L_{sync} = \max_{1 \le i \le n} \Delta_i$. On the other hand, it can be shown that *when ignoring the control circuit overhead*, the *average* cycle time of a corresponding asynchronous pipeline is at most $T^U_{async} = \mathbf{E} \max_{1 \le i \le n} d_i$, assuming the environment is fast enough to deliver or receive the data[2]. It can be checked that $T^U_{async} \le T^L_{sync}$, resulting in a higher average throughput for asynchronous design (e.g., [135]; see also Chapter 8). This throughput difference can be significant especially when the intervals of the stage delays are wide and their means are close to their lower bounds (δ_i's) ([128]), which is often the case in real applications. One of the most challenging problems in high-performance asynchronous design, however, is to keep the control circuit overhead small. A recent example is the asynchronous Intel Pentium processor instruction decoder [103, 115] which achieves more than 3 times higher average throughput than its existing synchronous counterpart.

Unlike synchronous circuits for which system performance can be determined easily from the fixed clock cycle time, it can be challenging to evaluate the performance of asynchronous circuits. Even for the above simple example of an asynchronous micropipeline where stages have arbitrarily distributed data-processing delays, there is no known approach to obtain the exact values of its average throughput. As a result, asynchronous designers are often forced to use *ad hoc* approaches to estimate system performance at different levels of the design hierarchy. In most cases, these *ad hoc* estimates are unreliable. Because of the lack of CAD tools for performance evaluation to guide the design choices, the designers either proceed without sufficient confidence or have to delay the evaluation of their designs until it is too late. Consequently, many asynchronous designs have been unsuccessful, in the sense that the final performance metrics are poorer than their synchronous counterparts. Typically, this is

[2]Intuitively, the data entering stage n must leave the pipeline after a random processing delay of d_n, assuming the environment is sufficiently fast to accept the data. At stage $n-1$, the data first experience a random processing delay of d_{n-1}. Immediately after this delay, the data leave the stage if stage n is `idle`, or gets blocked if stage n is `busy`. However, this blocking time is at most $d_n - d_{n-1}$. Thus, the data experience at most $\max\{d_{n-1}, d_n\}$ time at stage $n-1$. Generally, the data stay for at most $\max_{k \le i \le n} d_i$ time at stage k, and in particular, for at most $\max_{1 \le i \le n} d_i$ at stage 1. The expectation of the time the data staying at stage 1 results in the average cycle time of the pipeline, provided that the environment is sufficiently fast to deliver the input data to stage 1.

because of the overhead incurred by their control circuits, and more noticeably, their unoptimized system architectures. In addition, different applications are suitable for asynchronous implementation to a different degree. For instance, algorithms with an intrinsically low level of synchronization are more favorable to be designed asynchronously than those with an very high level of synchronization. As an example, some communication channel decoding algorithms such as the Fano algorithm [4] using a *sequential* search scheme may benefit more from asynchronous implementation than others such as the Viterbi algorithm [49] which use a *concurrent* search scheme. These experience among others motivates the need of performance evaluation tools to identify application domains that are most suitable for asynchronous implementation, and to optimize an asynchronous system architecture as well as for performance-driven synthesis at the RTL, gate- and transistor-levels.

1.2 Contributions

This thesis presents several efficient techniques for performance analysis of asynchronous circuits and systems. I broadly classify them into two categories.

The techniques in first category were developed to compute the *exact* values of average performance metrics using Markovian analysis. The contributions there are:

- Developing a stochastic modeling formalism to handle delays of arbitrary distributions using time discretization. The models are specified as a set of probabilistic communicating finite state machines, and are capable of specifying essentially all types of system behaviors, including both AND- and OR-causality, concurrency, choice and arbitration. They are equivalent to discrete-time finite state Markov chains [126].

- Deriving a thorectical framework from which the above Markov chain models can be analyzed to obtain average *time separations of events* (TSEs), which cover many important system performance metrics such as average throughput, latency and response time [126].

- Speeding up by orders of magnitude the analysis of large finite state Markov chains by attacking the state-explosion problem. This is achieved by developing

(1) an efficient symbolic techniques using binary decision diagrams (BDDs) for the structural analysis (state classification) [130, 131], and (2) a novel technique called *state compression* to accelerate the subsequent stationary analysis [127, 132].

- Demonstrating the effectiveness and the efficiency of the above techniques on a number of chip designs.

Other contributions there include a symbolic technique to decompose a large graph into strongly connected components (SCCs). Since SCC decomposition is a fundamental graph problem, the technique is expected to have wide applications. One such application is in *bad cycle detection*, a generic problem for *model checking* in the area of formal verification.

Although the techniques in the first category facilitate Markovian analysis of much larger systems than using traditional techniques, the size of systems they can handle is still limited in many cases. Nevertheless, these techniques are suitable for performance-driven synthesis of small- to medium-size control circuits.

The techniques in the second category take a radically different approach. Instead of computing the theoretical values of the performance metrics, they give performance *estimates* by deriving sharp lower and upper bounds using statistical methods [135, 134]. Besides, the models are analyzed in the *continuous time* domain, which removes the limitation of the first category as a result of the time-discretization. The contributions there are:

- Definition of TSE statistics (i.e., average, variance, and distribution of TSEs) in stochastic timed Petri nets with unique- and free-choice and with arbitrary distributed delays based on an infinitely long random simulation.

- Characterization of a large class of transition pairs for which their TSE statistics exist.

- Development of closed-form expressions for upper and lower bounds on these TSE statistics.

- Explanation of how the upper and lower bounds can often be made arbitrarily close to each other using more complicated expressions at the expense of additional run-time.

- Demonstration of how these closed-form expressions of the bounds can be efficiently evaluated with arbitrary high accuracy and confidence using simple statistical methods. Consequently, the mean of the TSE bounds serves a good estimate of the TSE statistics with a well defined error interval.

- Development of **USC-PET**, a fully automated Performance Evaluation Tool implementing the above this approach. Demonstration of the efficiency of USC-PET in a variety of examples, including a performance analysis of a full-scale stochastic timed Petri model of RAPPID (with over 900 transitions) in less than 1.5 hours of CPU time.

1.3 Organization

The remainder of this thesis is organized as follows. Chapter 2 describes the system modeling formalism. Part I presents the analysis techniques of the first category, which covers Chapters 3, 4,5 and 6. The techniques of the second category are presented in Part II which consists of Chapters 7, 8, 9. Chapter 10 summarizes this work, outlines some of the future work, discusses several open issues in analyzing systems with arbitration.

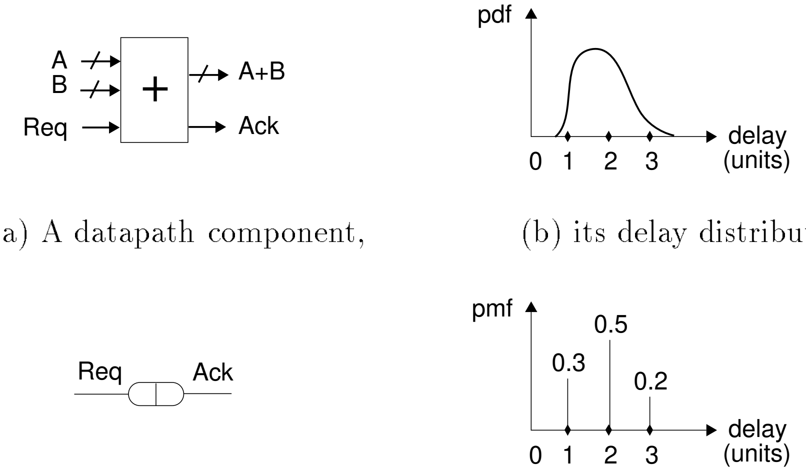
Chapter 2

System Models

This chapter introduces two types of system modeling formalism. One of the common goals of both types of the formalism is to allow specifications of system component delays with *arbitrary* probability distributions. Historically, in order to simplify the subsequent analysis, almost all known performance models specify system component delays either as random variables with special distributions (e.g., exponential distributions) or as constants. Models with these restricted delay specifications are widely used, for example, in *queueing networks* [70] and more generally in *discrete event dynamic systems* (DEDS) [60, 26]. Unfortunately, most components in asynchronous circuits and systems have interval delays that cannot be suitably modeled by random variables with above restricted type of distributions. Experiments show that such delay models can cause significant accuracy loss in performance analysis (e.g., [128]).

In the first modeling formalism, systems are specified as a set of *communicating finite state machines* (FSM) (e.g., [121]) and component delays are *discretized*. The model transits from one state to another only at time instants that are multiple of some *time unit* (or time step). The distinct feature of this framework is that it is always possible to achieve delay models with arbitrarily high accuracy by decreasing the size of the time step. Models from this framework are analyzed as *discrete-time Markov chains* (DTMC) (e.g., [53]) (see Part I).

In the second modeling formalism, systems are specified as *stochastic timed Petri nets* (e.g., [140], [99], [63]). Unlike existing research in stochastic timed Petri nets which assumes exponential delays or constant delays, this formalism allows specification of virtually any delay distributions provided that they all have *finite means*.



(a) A datapath component, (b) its delay distribution,

(c) abstraction of the component, (d) time discretization.

Figure 2.1: Probabilistic modeling and time discretization.

Part II of the thesis provides efficient techniques to analyze models specified with the second formalism.

The remaining of this chapter describes the two types of the modeling formalism in detail.

2.1 Stochastic discrete-time models

There have been developed many types of formalism to specify asynchronous systems. Traditionally, these specifications target synthesis and verification applications 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., [44, 64, 22]. However, in order to obtain system characteristics such as average throughput and average power consumption, the system needs to be adequately modeled in a probabilistic sense. In particular, the probability of choice branches to be taken and the distribution of interval delays must be explicitly specified.

We focus on explaining how we derive our performance models from an architectural description of the system specification. 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 datapath and control components. Our goal is to obtain performance models for these system from simple ones of both the datapath and control components.

The operation of an asynchronous datapath 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 value of the datapath inputs. However, the size of such a model would likely grow exponentially with respect to the datapath's bitwidth, making its analysis computationally intractable. In practice, one can often achieve useful performance models by abstracting the delay of the datapath components to a random variable with an estimated probability distribution that takes into account input statistics (e.g., common v.s. rare input combinations) [126, 32]. Consequently, the actual data signals need not be modeled, thereby dramatically simplifying the complexity of the system model. As an example, Figure 2.1 (b) plots the distribution of the delay of an adder in Figure 2.1 (a). In some cases, of course, this form of 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 datapath 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, datapath units 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 (see e.g., [32]).

The issues regarding modeling asynchronous controllers are made more obtuse because, unfortunately, there is no universal specification language for asynchronous controllers. In fact, there are many different languages with varying expressiveness, including asynchronous finite state machines [121], burst mode machines [41] and its derivations (e.g., [93], [137]), various classes of Petri-nets [95] Signal Transition

Graphs [33], and Timed Event/Level (TEL) structures [9], and sets of connected processes (e.g.,[46, 83]) that are variants of communicating sequential processes (CSP) [61].

As mentioned earlier, most of these specification languages do not specify the delays through the controllers and all of them need not specify delay distributions. For performance modeling, however, delay and sometimes their distributions are necessary. For example, one performance model for a burst-mode controller specifies the latencies of each state transition as a constant [126]. An alternate model useful for analyzing the stochastic effects of variations in chip operating temperatures and/or the manufacturing process, may model the related delays probabilistically. Even under constant temperature and nominal process assumptions, some controllers do exhibit non-fixed delays and should be modeled probabilistically. For instance, delays of an arbiter in metastability can be modeled as an exponential distributed random variable with proper parameters [28, 123]. In addition, the probability distribution of input (environmental) choices can be modeled in the same manner that we model datapath output signals that dictate the controller behavior (such as the result of a comparator).

In most of these cases, the untimed next-state function of the system consisting of communicating datapath and controller elements can be modeled as boolean functions of inputs, outputs, and perhaps some auxiliary variables (e.g., variables related to places of a Petri-net). Consequently, the untimed state space of the system can be generated [42, 100, 87], which is useful for some formal verification purposes. Verifying systems with timing assumptions also requires the next-state function to model delays [9, 122]. For performance analysis, the next-state function must also include a notion of the distribution of delays. Informally, the performance of an asynchronous system can be derived from as a set of interdependent random variables that essentially defines a stochastic process whose state space is spanned by all these random variables. This process evolves according to the distributions of these random variables as well as their interdependencies.

The larger the system to be modeled and the higher the desired accuracy of the model leads to more random variables required. The key issue that we now address is how to efficiently analyze such models.

2.1.1 Approximating arbitrarily distributed delays

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. Consequently, their exact behavior is hard to capture using previously known stochastic processes that can be efficiently analyzed. For this reason, many researchers proposed methods to approximate arbitrary distributions with special distributions that facilitate more efficient analysis. This subsection briefly reviews these approaches and the subsequent subsection describes the approach we adopted in Part I for Markovian analysis of asynchronous systems.

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 [52]. In other cases, the resulting system can be modeled as a stochastic or timed Petri net [80, 90, 63]. Both of these types of memoryless delay models can be analyzed as a finite state Markov chain where the states in the Markov chain have appropriately defined semantics [80, 90, 63]. 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 marking of places in the Petri net.

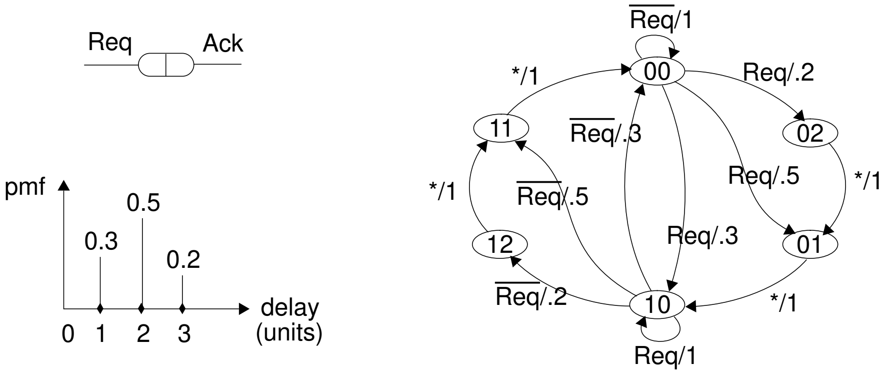
A second approach approximates the arbitrary distribution using a fixed delay. For example, the fixed delay can be chosen to be the mean of the arbitrary distribution. These type of delay models are incorporated into Generalized Timed Petri Nets (GTPNs) which can also be converted into a Markov chain by augmenting a notion of time into the semantics of a state [63]. For example, the state can be associated with a fixed time after which the Markov chain transits to some next state.

A third approach approximates the arbitrary distribution using time discretization (see e.g., [89, 90]). The result is that events can now occur only at time instants that are multiple of some unit time step. In other words, the delay interval (of reals) is discretized into an interval of integers (with a unit time step). The probability distribution function (pdf) is accordingly converted into a probability mass function (pmf) by interval integrations. Similar to the other two approaches, time discretization also loses some accuracy [89]. However, it is always possible to trade discretization resolution (and consequently the computational cost) with accuracy of the final performance estimates. As in the other two approximations, the time discretized system model can be converted to a finite state Markov chain (e.g., [90]).

2.1.2 Time-discretized models

In part I of this proposal, 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 modeled using an exponentially or geometrically distributed delay. A notable exception is the delay an arbiter experiences when resolving metastability [28]. Another exception may be the data inter-arrival time from the environments which sometimes may be reasonably approximated with exponential distribution. Beyond these cases, most system components (e.g., datapath components, controllers) exhibit bounded delays with arbitrary distributions which cannot be reasonably approximated as exponentially distributed random variables. Moreover, when exponential distributions do need be modeled, they can be approximated with arbitrary accuracy in discrete time using a discrete geometric distributions. Second, we believe that it is often not adequate to model bounded delays with their mean values, as was proposed in [72]. 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 [128]. For instance, the average throughput of a micropipeline strongly depends on the second moment or the variance of the stage delays [128].

As an example of time discretization, we may discretize the adder delay whose distribution is shown in Figure 2.1 (b) and model it as a discrete random variable whose pmf is shown in Figure 2.1 (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 physical system signals, we call them *timers*. The value of a `timer` is probabilistically assigned when a corresponding physical system signal is enabled. It then decreases by 1 each time the system model makes a state transition. When it reaches 0, the corresponding system signal fires (i.e., makes a desired transition).



(a) (b)

Figure 2.2: Local STG of an abstracted adder: (a) its probabilistic delay model, and (b) its state space where `state := (Ack, timer)` and `transition label := input-condition/probability`.

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. In other words, each state transition takes precisely one time unit. As a concrete state space example, let us consider again the the abstract adder model in Figure 2.1 whose discretized delay distribution is redrawn in Figure 2.2 (a). Figure 2.2 (b) shows its (local) 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., `Req` makes a positive transition) which enables a positive transition on signal `Ack`. According to the delay distribution of the adder, the `timer` is then randomly assigned a value at next time step. Specifically, at time instant $k+1$, variable `timer` receives a value of 2 with probability 0.2, a value of 1 with probability 0.5 and a value of 1 with probability 0.3. We note that if `timer` is assigned a value of k where $(0 \leq k \leq 2)$, 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 and so forth.

Experiments show that this time-discretized model can yield high-quality system performance estimates (compared with extensive simulation) with fairly coarse time discretization [126]. 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 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 [75] and less accurate GTPN models of asynchronous systems [72].

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 describe above, we create the Markov chain (see [126] for details). Of course, translations from other specification formalisms and other levels of design abstraction are also possible. For example, systems specified using VHDL at 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 notation 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.

2.1.3 Discrete-time boolean models

The timed system is modeled as a finite state automaton $M = \langle I, Q, T, Q_0, O \rangle$ where Q is the state space spanned by the set of state variables V, Q_0 is the set of initial states, $T \subseteq Q \times Q$ represents the state transition relation, I and O are the system inputs and outputs, respectively. We assume the environment is considered as part of the timed systems so that the system is closed and both I and O are empty. That is, $M = \langle \emptyset, Q, T, Q_0, \emptyset \rangle$.

A state variable $v \in V$ either represents a physical signal or helps model the passage of time (i.e., timer). The range of a state variable v is denoted by $R(v)$ which is boolean if v is a signal variable or an integer interval $[L, U]$ if v is a time variable where L and U are the lower and upper bounds of the corresponding timer. We may write state space as $Q = \bigotimes_{v \in V} R(v)$ where a state $s \in Q$ is the valuation of all state variables. We denote $s(v)$ the value of state variable v in state s.

The transition relation $T \subseteq Q \times Q$ consists of state transitions $(s, s') \in Q \times Q$ each modeling the possible evolution of the systems from state s to state s' after *one* time step. We say s is the current state and s' the next state in the state transition. The transition relation is derived from the set of partial transition relations $\{T_v \subseteq Q \times Q, v \in V\}$, where T_v constraints the evolution of state variable v in terms of current system state and the evolution of all other state variables. T is obtained from all T_v's by *synchronous* composition semantics in which state variables evaluate simultaneously. Consequently, the transition relation T is the intersection of all the partial transition relations, i.e.,

$$T = \cap_{v \in V} T_v \tag{2.1}$$

More specifically, system M may be viewed as a set of communicating nondeterministic finite state machines, $\{M_v, v \in V\}$. Each finite state machine $M_v = \langle I_v, Q_v, T_v, Q_{0_v}, O_v \rangle$ contains only one state variable v, i.e., $Q_v = R(v)$, and communicates with all other machines through input I_v and output O_v. In particular, we allow I_v to be a subset of both current and next state values of all other state variables, i.e., $I_v \subseteq \bigotimes_{\substack{u \in V \\ u \neq v}} [R(u) \otimes R(u)]$ whereas the output consists of the current and next state values of the state variable v, i.e., $O_v \subseteq R(v) \otimes R(v)$. Then, the transition relation of M_v is $T_v \subseteq I_v \otimes [R(v) \otimes R(v)]$. Finally, $Q_{0_v} \subseteq Q_v$ is the set of initial states which partially describes the initial state set Q of M, i.e., $Q_0 = \bigotimes_{v \in V} Q_{0_v}$.

For further description, let us introduce following nations. $Q(B)$ is the space spanned by the state variables in a subset B of V, i.e., $Q(B) = \bigotimes_{v \in B} R(v)$. Particularly, we have $Q(\emptyset) = \emptyset$, $Q(V) = Q$ and $Q = Q(B) \otimes Q(\overline{B})$ where $\overline{B} = V \backslash B$, the complement set of B. In the case where B contains only one state variable v, we sometimes write $\overline{B} = V \backslash v$ for convenience.

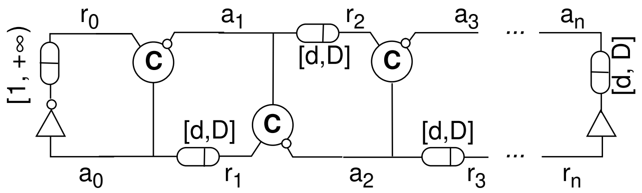


Figure 2.3: An n-stage FIFO.

For the purpose of this work, each partial transition relation T_v is specified by a case statement which we formalize as an *ordered* set G_v of conditional assignments. Each conditional assignment $(g : o)$ in G_v consists of a condition g and an associated assignment o. The condition g is a boolean predicate of current (system) state and next state values of all state variables excluding v, a domain defined by $Q \times Q(V \backslash v)$. At least one condition in the case statement must be true for every element of $Q \times Q$ (and thus for every transition (s, s')). The first true condition and its associated assignment are said to be active. An assignment is a set of functions from $Q \times Q(V \backslash v)$ to $R(v)$. Thus, for every element in $Q \times Q(V \backslash v)$ the active assignment defines a set of values $R(v)_a \subseteq R(v)$. Then, T_v defines a set of transitions (s, s') from current state s such that $s'(v) \in R(v)_a$.

Example Figure 2.3 shows a simple FIFO with $n \geq 0$ stages. Each stage consists of a C-element and a delay line. All delays are quantized into time steps. The C-elements have unit delay (i.e., one time step) and the delay lines have integer delay in the interval $[d, D]$. The left environment has an integer delay in the interval $[1, +\infty]$, while the right environment has a fixed integer delay D_r. The delay line of stage i is modeled by a timer w_i with delay range $[0, D]$. To model the range of delays, w_i is non-deterministically set to any integer value in the interval $[d, D]$ each time the input to the corresponding delay line changes, and is decremented by one each subsequent state transition. The output of the delay line is asserted when w_i reaches 0.

The partial transition relations that model the timed behavior of the FIFO are specified using case statements. Each stage is modeled as follows:

$$
\begin{array}{llll}
T_{a_i} & := & \text{case} & \\
& & r_i \neq a_{i+1} \quad : \quad \{r_i\}\,; & \\
& & 1 \qquad\qquad\quad : \quad \{a_i\}\,; & \text{for } i = 0, 2, \ldots, n-1. \\
T_{r_i} & := & \text{case} & \\
& & w_i{}' \;\&\; w_i \neq 0 \quad : \quad \{a_i\}\,; & \\
& & 1 \qquad\qquad\quad : \quad \{r_i\}\,; & \text{for } i = 1, 2, \ldots, n. \\
T_{w_i} & := & \text{case} & \\
& & a_{i-1}{}' \neq a_{i-1} \quad : \quad \{d, \ldots, D\}\,; & \\
& & w_i > 0 \qquad\quad : \quad \{(w_i - 1)\}\,; & \\
& & 1 \qquad\qquad\quad : \quad \{w_i\}\,; & \text{for } i = 1, 2, \ldots, n.
\end{array}
$$

The left and right environments are modeled similarly. □

2.1.4 Probabilistic state transition relations

A *proabilistic transition relation* P maps each state transition $(s, s') \in Q \times Q$ onto a probability such that the sum of the probabilities of the state transitions from any state is 1, i.e.,

$$\sum_{s' \in Q} P(s, s') = 1 \quad \text{for any } s \in Q \tag{2.2}$$

In other words, P corresponds to a *stochastic* state transition matrix with rows indexed by current states and columns indexed by next states such that the sum of probabilities in any row equals 1. Sometimes, we say P is stochastic if equation 2.2 holds.

To obtain this model we augment the conditional assignments of each variable v. We annotated the elements of each assignment of next state value of v (i.e., v') with probabilities that add to 1. For each state transition (s, s'), this defines a function $prob : R(v) \rightarrow [0, 1]$ which maps each value in $R(v)_a$ onto the annotated probability for that value given in the active assignment. Values for v' not in $R(v)_a$ (i.e., the set $R(v) \backslash R(v)_a$) are mapped to 0 probability by default. We then define a partial probabilistic transition relation $P_v : Q \times Q \rightarrow [0, 1]$ for each variable v such that

$P_v(s,s') = prob(s'(v))$ for every state transition (s,s') where function *prob* is defined by the active assignment of variable v under the corresponding state transition.

The property that the annotated probabilities for state variable v over the values in each of its assignment add to 1 may be formally written as follows.

Property 1

$$\sum_{s_1 \in R(v)} P_v(s, s_1 \otimes s_2) = 1 \tag{2.3}$$

for any $s \in Q$ *and any* $s_2 \in Q(V \backslash v)$.

The probabilistic transition relation $P : Q \times Q \rightarrow [0,1]$ is determined as follows:

$$P(s,s') = \prod_{v \in V} P_v(s,s') \tag{2.4}$$

for all $(s,s') \in Q \times Q$.

Equation 2.4 suggests that the probability of each state transition is defined by a set of independent choices of assignments whose probabilities multiply. This, however, is not guaranteed without further restriction on the model. To see this, consider the following example:

Example Suppose the system consists of 2 state variables a and b with partial probabilistic state transitions defined as:

$$\begin{array}{lllll} P_a & := & \text{case} & & \\ & & b' & : & \{1\}; \\ & & 1 & : & \{0\}; \end{array} \qquad \begin{array}{lllll} P_b & := & \text{case} & & \\ & & a' & : & \{1\}; \\ & & 1 & : & \{0\}; \end{array}$$

Both P_a and P_b satisfy the property given by Equation 2.3. Combined, they define two transitions from state 00, i.e., $(00,00)$ and $(00,11)$. Evaluating Equation 2.4, each state transition has probability 1 and their sum equals 2, a violation of Equation 2.2. Thus, the probabilistic state transition relation $P = P_a P_b$ is not stochastic. $\square$

The violation in the above example occurs because the above provability assignment does not capture the non-determinism hidden in this model. Intuitively, this is because, for both P_a and P_b under a given current state 00, multiple conditional assignments can become active so that the choice made is between two guards, not between value assignments in one guard. More precisely, one sees the transition of variable a depends on transition of b and vice versa. This forms a *cyclic dependence* which introduces extra non-determinism that cannot be capture by simply requiring the transition behavior of individual state variable to satisfy the property given by Equation 2.3.

More generally, for every given current state, there is a dependence graph among the transition behaviors of all state variables. Each state variable corresponds to a vertex of the graph and there is an edge from vertex x to vertex y if the transition behavior of the state variable corresponding to vertex x depends on that of the state variables corresponding to vertex y. The remaining of this subsection shows that if the dependence graph corresponding to every possible current state is cycle free, then the probabilistic transition relation P is stochastic.

Let us first formalize the dependence among state variables and the cycle-free requirement on the dependence graphs of the model.

Definition 1 *Let u and v be two state variables in V. Variable u depends on variable v in the current state $s \in Q$, denoted by, $u \xrightarrow{s} v$, if P_u is a function of v' in state s. That is,*

$$u \xrightarrow{s} v \iff \exists s', s'' \in Q \ s.t. \ s'(v) \neq s''(v) \wedge P_u(s, s') \neq P_u(s, s'').$$

We denote by $D(s, v)$ the set of state variables that depends on v assuming the current state is s, i.e.,

$$D(s, v) \triangleq \left\{ u \in V \mid u \xrightarrow{s} v \right\}.$$

Lemma 1 *Suppose $u, v \in V$ and $u \in D(s, v)$, then $D(s, u) \subseteq D(s, v)$.*

Proof Dependency is transitive. ■

Lemma 1 captures the transitivity of the variable dependence relation. That is, if $w \xrightarrow{s} u$ and $u \xrightarrow{s} v$, then $w \xrightarrow{s} v$.

The *non-cyclic-dependence* property of the model can be formally stated as follows.

Property 2 *For every state variable $v \in V$ and every state $s \in Q$, $v \notin D(s, v)$.*

In the previous examples, one may check that $b \in D(00, a)$ and $a \in D(00, b)$. Thus, $a \in D(00, b) \subseteq D(00, a)$ by Lemma 1. Therefore, the model in this example has cyclic dependence among variables. In fact, since $b \in D(00, a) \subseteq D(00, b)$ for a similar reason, we have $D(00, a) = D(00, b) = \{a, b\}$.

In the sequel, we assume that there is no cyclic dependence among state variables so that Property 2 holds.

Definition 2 *A subset B of V is* closed *under current state $s \in Q$ if for any $u \in B$ and any $v \in \overline{B}$, $u \notin D(s, v)$. Or equivalently, $B \cap (\bigcup_{v \in \overline{B}} D(s, v)) = \emptyset$.*

Intuitively, every state variable in a closed subset of V depends no more than the state variables in that subset. Two special closed sets are $\emptyset$ and V.

Lemma 2 *Property 2 is equivalent to the fact that for any non-empty closed set C of V in the current state s, there exists a variable $v \in C$ such that $C - \{v\}$ is a closed in state s.*

Proof Let $|C|$ denote the size of C. Suppose that in the current state $s \in S$, for every variable v in closed set $C \subseteq V$, $C - \{v\}$ is not closed. In other words, for every $v \in C$, there exists a variable $u \in C - \{v\}$ such that $D(s, u) \not\subseteq C - \{v\}$. But C is closed so that $D(s, u) \subseteq C$. This means $v \in D(s, u)$, i.e., $u \xrightarrow{s} v$. Therefore, there is a sequence of variables $v_{1_i} \in C : i = 1, 2, \cdots, m \leq |C|$ such that

$$\begin{aligned} &v_{1_i} \in D(s, v_{1_{i-1}}), \text{ for } i = 2, 3, \cdots, m \\ &v_{1_1} \in D(s, v_{1_m}). \end{aligned}$$

By Lemma 1, we then have $D(s, v_{1_m}) \subseteq D(s, v_{1_1})$. Therefore, $v_{1_1} \in D(s, v_{1_1})$, which contradicts the cycle-free dependence assumption of Property 2. ■

The following theorem gives the key observation on the transition behaviors of the state variables in a closed subset.

Theorem 1 *Suppose the model satisfies Properties 1 and 2. For any non-empty closed subset C of V under current state $s \in Q$,*

$$\sum_{s_1 \in Q(C)} \prod_{v \in C} P_v(s, s_1 \otimes s_2) = 1 \textit{ for any } \; s_2 \in Q(\overline{C}).$$

Proof (By induction on the cardinality of C)
Base case: $|C| = 1$. Let v_i by the only variable of C. Since C is closed, $v_j \notin D(s, v_i)$

for any $v_j \in \overline{C} = V \backslash v$. Thus, $\sum_{s_1 \in Q(C)} \prod_{v \in C} P_v(s, s_1 \otimes s_2) = \sum_{s_1 \in R(v_i)} P_{v_i}(s, s_1 \otimes s_2) = 1$. The second equality of the above equation is due to Property 1.

Suppose the theorem holds for the cases up to $|C| = m \geq 1$. Then, for the case $|C| = m + 1$:

By Lemma 2 under assumption of Property 2, there exists a variable $v_i \in C$ such that $C - \{v_i\}$ is a closed subset of V. For simplicity, let this subset be denoted by $C' = C - \{v_i\}$. Then,

$$
\begin{aligned}
& \sum_{s_1 \in Q(C)} \prod_{v_j \in C} P_{v_j}(s, s_1 \otimes s_2) \\
= & \sum_{s_1 \in Q(C)} \prod_{v_j \in C'} P_{v_j}(s, s_1 \otimes s_2) \cdot P_{v_i}(s, s_1 \otimes s_2) \\
= & \sum_{s_1' \in Q(C')} \sum_{s_1'' \in Q(\{v_i\})} \prod_{v_j \in C'} P_{v_j}(s, s_1' \otimes s_1'' \otimes s_2) \cdot P_{v_i}(s, s_1' \otimes s_1'' \otimes s_2) \\
= & \sum_{s_1' \in Q(C')} \prod_{v_j \in C'} P_{v_j}(s, s_1' \otimes \{s_1'' \in Q(\{v_i\})\} \otimes s_2) \sum_{s_1'' \in R(v_i)} P_{v_i}(s, s_1' \otimes s_1'' \otimes s_2) \\
= & \sum_{s_1' \in Q(C')} \prod_{v_j \in C'} P_{v_j}(s, s_1' \otimes \{s_1'' \in Q(\{v_i\})\} \otimes s_2) \text{ (by Property 1)} \\
= & \ 1 \text{ (since } |C'| = m \text{ and by the assumption on the case } |C| = m)
\end{aligned}
$$

■

Theorem 1 states that if the model satisfies Properties 1 and 2, the probabilistic transition relation composed by multiplying the partial probabilistic transition relations of the state variables in any non-empty closed subset C of V is stochastic over the state space spanned by the ranges of state variables in C. Since V is a closed set itself, the following corollary regarding the stochasticity of the model is immediate.

Corollary 1 *Suppose the model satisfies Properties 1 and 2, then its probabilistic transition relation P defined by Equation 2.4 is stochastic. That is, $\sum_{s' \in Q} P(s, s') = 1$ for any $s \in Q$.*

Proof Notice the fact that V itself is a closed state variable set and $Q = Q(V)$. Then, the corollary is a direct consequence of Theorem 1 by replacing C with V:

$$\sum_{s' \in Q} P(s, s') = \sum_{s' \in Q(V)} \prod_{v \in V} P_v(s, s') = 1.$$

■

Example Consider the FIFO shown in Figure 2.3 in which we assign the probability distribution for the delays of the delay lines of each stage as follows: probability that the delay line takes 1 unit of delay is 0.2, 2 units of delay 0.6 and 3 units of delay 0.2. The corresponding partial transition relation for the timer variable w_i is annotated as follows:

$$\begin{array}{lll} T_{w_i} & := & \text{case} \\ & & \begin{array}{lll} a_{i-1}' \neq a_{i-1} & : & \{1'0.2, 2'0.6, 3'0.2\}\,; \\ w_i > 0 & : & \{(w_i - 1)\}\,; \\ 1 & : & \{w_i\}\,; \qquad\qquad \text{for } i = 1, 2, \ldots, n. \end{array} \end{array}$$

The conditional assignments that have only one element in their corresponding assignment are deterministic and thus the element is, by default, assigned probability 1. Similarly, we specify the partial probabilistic transition relation of the timer variable in the delay line of the left environment as:

$$\begin{array}{lll} T_{w_0} & := & \text{case} \\ & & \begin{array}{lll} a_0' \neq a_0 & : & \{1\}\,; \\ w_0 > 0 & : & \{0'p, 1'q\}\,; \\ 1 & : & \{w_i\}\,; \qquad \text{where } p + q = 1. \end{array} \end{array}$$

This models the delay of the left environment whose delay distribution is geometric with a parameter p. □

2.2 Stochastic timed Petri nets

Our second modeling formalism is based on Petri nets (e.g., [95]). Petri nets are widely accepted models for discrete event systems. Historically, Petri nets are used for system control, synthesis and verification purpose. When associated with time, they have also been used for performance analysis [63, 89, 92, 23]. Asynchronous circuits and systems have long been modeled and analyzed as Petri nets [98, 72]. Compared with the modeling formalism described in the previous section, incorporating Petri net models increases the modeling capability of our tool, which better links the tool to industrial application. This section reviews general concepts of (untimed) Petri nets. We then introduce stochastic delays and choice probabilities specific to timed Petri nets. In particular, the stochastic delay assumptions here are significanlty more general than those in any existing time Petr net models. Further details on Petri nets can be found in [92]. We leave the discussion of some well-known properties of Petri nets in Part II where we will also prove some important properties of many random processes derived from these models.

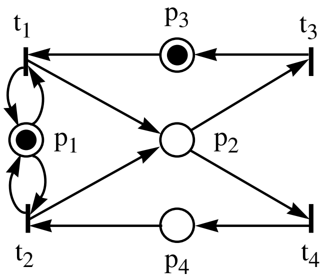
2.2.1 Petri net models

A Petri net is a triple $N = (P, T, F)$ where P is the set of places, T the set of transitions, and $F \subseteq (P \times T) \cup (T \times P)$ the flow relation. The preset of $x \in P \cup T$ denotes the set $\bullet x = \{y \in P \cup T \mid (y, x) \in F\}$, and its poset $x\bullet = \{y \in P \cup T \mid (x, y) \in F\}$.

A Petri net N is called a *state machine* if every transition has at most one input and output place, i.e., $\bullet t \leq 1 \geq t\bullet, \forall t \in T$. It is a *marked graph* if every place has at most one input and output transition, i.e., $|\bullet p| \leq 1 \geq |p\bullet|, \forall p \in P$. A place p is a choice place if $|p\bullet| > 1$. A state machine can model *choice* but not *concurrency* whereas a marked graph can model *concurrency* but not *choice*.

A Petri net N is called *free-choice* (FC) if $\forall p_1, p_2 \in P$, $p_1\bullet \cap p_2\bullet \neq \emptyset \Rightarrow |p_1\bullet| = |p_2\bullet| = 1$, i.e., neither p_1 nor p_2 is a choice place. Equivalently, in a FC net, if two transitions share an input place, they may not have any other input places. Both state machines and marked graphs are FC nets. A Petri net is *extended free-choice* (EFC) if $\forall p_1, p_2 \in P$, $p_1\bullet \cap p_2\bullet \neq \emptyset \Rightarrow p_1\bullet = p_2\bullet$. An EFC net can be translated into a FC net [11]. A choice place p is called *unique choice* if no two output transitions of p are ever enabled simultaneously. A choice place that is neither (extended) free-choice or unique choice is called an *asymmetric choice*. Asymmetric choice is necessary to model arbitration.

Figure 2.4 shows an example net that contains both free-choice and unique choice. Place p_1 is a unique choice whereas p_2 is a free-choice. Notice that tokens in p_1 never need make a decision as who to fire since p_3 and p_4 can never be simultaneously marked. In other words, the decision as whether t_1 or t_2 will fire is completely controlled by the p_2 choice place.



$X(p_1) = 5,$
$X(p_2) \sim \texttt{exp}(2.5),$
$X(p_3) \sim \texttt{uniform}(2, 4),$
$X(p_4) \sim \texttt{uniform}(5, 10).$

$\mu(p_2, t_3) = 0.4,$
$\mu(p_2, t_4) = 0.6 = 1 - \mu(p_2, t_3).$

(a) The Petri net, (b) its stochastic assumption.

Figure 2.4: A stochastic Petri net example.

A *marking* is a mapping $M : P \rightarrow \mathbf{N} = \{0, 1, 2, \cdots\}$. The number of tokens in place p under marking M is denoted by $M(p)$. If $M(p) > 0$, one says p is in the *support* set of M. A transition t is *enabled* at marking M if all its input places have at least one token, i.e., $M(p) \geq 1, \forall p \in \bullet t$. When t is enabled, it may fire. The firing of t removes one token from each place in its preset and deposits one token to each place in its poset, leading to a new marking M', denoted by $M[t\rangle M'$. Let M be a marking of the net. A sequence of transitions $\sigma = t_0 t_1 \cdots t_m$ is a firing sequence from marking M iff $M_k[t_k\rangle M_{k+1}$ for $k = 1, \cdots, m$ with $M_0 = M$. In that case, one also writes $M[\sigma\rangle M'$.

A *marked* Petri net Σ is a tuple (N, M_0), where N is a Petri net and M_0 is the initial marking of N. Sometimes, we use N_Σ to stress N is the underlying net of Σ. The set of reachable markings of Σ is denoted by set $R(M_0)$. Σ is *live* iff every transition of N will be eventually enabled from every $M \in R(M_0)$. It is *safe* if no place ever marked with more than one token, i.e., $M(p) \leq 1$ for every $M \in R(M_0)$. It is *k-bounded* if no place is ever marked with more than k tokens.

2.2.2 Stochastic assumptions

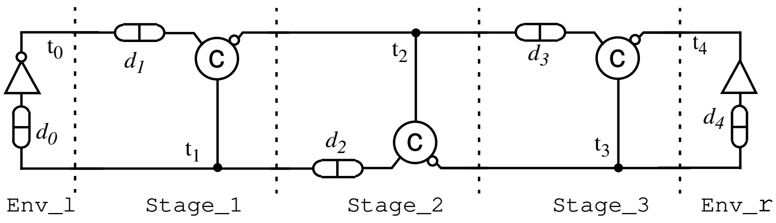
In this work, we associate delays with places. That is, a token flowing into a place p must experience a random delay associated with p before it is *available* to be consumed by an output transition of p. The delays experienced by different tokens are independent. We also assume that delays associated with different places are independent. The firing of a transition is assumed to be instantaneous. Let $X(p)$ be the random variable denoting the delay associated with place p. For each such random delay variable $X(p)$, let $F_{X(p)} : R \to [0, 1]$ denote its distribution function, i.e., $F_{X(p)}(x) = \text{Prob}(X(p) \leq x)$. Unlike some other research in stochastic timed Petri nets, we do not put any restriction on the distribution functions except that they all have finite moments up to a sufficiently high order.

For each free-choice place $p \in P$, we assume there is a probability mass function (p.m.f.) μ that resolves the choice. Specifically, for each output transition t of p, we have $\mu(p, t)$ denoting the probability that t consumes the token in p each time it is marked. Of course, $\sum_{t \in p\bullet} \mu(p, t)$ must equal 1. In case p is a unique choice, we of course have $\mu(p, t) = 1$ where t is the unique output transition of p that is currently enabled. For an asymmetric choice place p, the situation is more involved. Chapter 10 discusses more on its stochastic choice assumption.

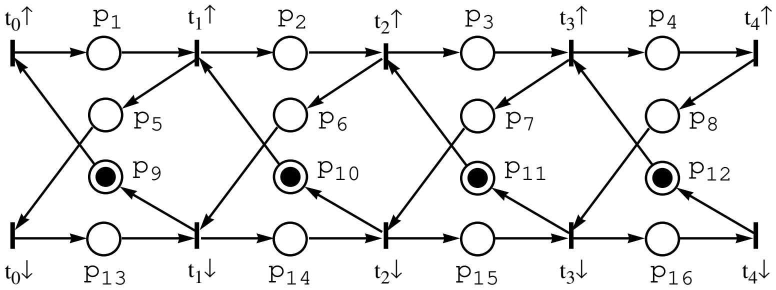
Example Figure 2.4.(b) lists a possible stochastic assumption on delay distributions and choice p.m.f. on the Petri net in Figure 2.4.(a). In particular, place p_1 takes a constant delay of 1, the delay in p_2 is exponentially distributed with a parameter 2.5, the delay in p_3 is uniformly distributed between 2 and 4, and so forth. In addition, with probability 0.4, t_3 fires each time the free-choice place p_2 receives a token, and with the remaining probability, t_4 fires. The other choice place p_1 is not assigned a p.m.f. since it is a unique choice place. □

We close this section with the Petri net model of the micropipeline similar to the one discussed in the previous section. Since such a micropipeline does not have choice, it can be modeled by a marked graph. In Chapter 9, we will see a Petri net model of a micropipeline that can exhibit choice.

Example Figure 2.5(b) shows a marked graph that models the behavior of a three-stage micropipeline depicted in Figure 2.5(a). The pipeline has three stages between two environments (a sender, i.e., Env_l, and a receiver, i.e., Env_r). Initially, the outputs of C-elements are all low. When `Env_l` has data ready, it sends a *request* to `stage_1` . by flipping the logic value of signal t_0. When `stage_1` finishes processing the data, it latches the result. Meanwhile, it sends a request to `stage_2` and simultaneously, sends an acknowledge back to `Env_l` by flipping the logic value of t_1. Following that, `stage_2` starts processing and when it is done, it latches the result, and meanwhile it sends a request to the `stage_3` and an *acknowledge* back to `stage_1`. When `stage_2` is busy processing data, `stage_1` may possibly be processing new data if it receives another request from `Env_l`, demonstrating potential



(a)



(b)

Figure 2.5: An asynchronous micropipeline: (a) the control circuit, and (b) its marked graph model.

concurrency of the system. However, `stage_1` cannot latch its new result until it receives an acknowledge from `stage_2` to avoid ruining the data being processed at `stage_2`. This latter constraint shows the potential *data blocking*. The above behavior is captured by the marked graph in Figure 8.1(b). With the initial marking as shown, we may check that the marked graph is live and safe. □

Part I

Markovian Analysis

Chapter 3

Markov Chain Theory of Asynchronous Systems

3.1 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. More formally, X is a Markov chain if for all $n \geq 1$ and all $j, i, x_{n-1}, \cdots, x_1, x_0 \in S$, $Pr(X_{n+1} = j \mid X_n = i, (X_{n-1} = x_{n-1}, \cdots, X_1 = x_1, X_0 = x_0)) = Pr(X_{n+1} = j \mid X_n = i)$, where $Pr(A \mid B)$ denotes the conditional probability that A occurs given that B occurs. 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 point n. It is sometimes desirable to make use of a stronger property, called the *strong Markov property* [53] which deals with the situation where the corresponding conditioning is taken at a random time T rather than at a fixed time. 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 $m \geq 0$, $i, j \in S$, and all sequence (x_k) of states.

If the probabilities governing X are independent of time, X is *time-homogeneous*. So, if X is a Markov chain and $Pr(X_{n+1} = j \mid X_n = i)$ is independent of time n,

[1] A random variable T taking values in $\{0, 1, 2, \cdots\}$ is called a stopping time of a Markov chain 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.

then X is time-homogeneous. In this case, we define a matrix P whose element at the i-th row and j-th column, $P_{ij} = Pr(X_{n+1} = j \mid X_n = i) = Pr(X_1 = j \mid X_0 = i)$. Clearly, we have $\sum_{j \in S} P_{ij} = 1$ 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 [126], assume X denotes a time-homogeneous Markov chain.

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

A state is called *recurrent* if the chain starting from that state will return to it with probability 1. Otherwise, it is called *transient*. Hence, once visited, a recurrent state is expected to be re-visited infinitely often whereas a transient state is expected to be re-visited only a finite number of times. Two states *communicate* (or are *strongly connected*) if there exists a cycle containing both states. Similarly, a subset of states communicate if any two states of the subset communicate. A *recurrent class* is a maximal communicating subset that contains only recurrent states. If S contains transient states or it has multiple recurrent classes, X is called *reducible*. Otherwise, X is *irreducible*. 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 recurrent class have the same period. Consequently, X is called periodic (aperiodic) if all its states are periodic (aperiodic).

Let $M = (S, P)$ be an irreducible *finite* state Markov chain in discrete time where S is the state space and P is the 1-step 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, $\lim_{n\to\infty} P\{M_n = i\}$ 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., $\lim_{n\to\infty} \frac{1}{n} \sum_{k=0}^{n} P\{M_k = i\}$, converges to a constant. Let this constant be denoted by π_i as well. Regardless of its periodicity, M has the following nice property:

$$\pi = \pi P \tag{3.1}$$

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

In this chapter, we assume the Markov chains to be analyzed are irreducible for the following reasons. The state classification technique developed in [130, 131] can be used to efficiently identify transient states and all recurrent classes of a Markov chain. If the original Markov chain has only one recurrent class (which is often the case [56, 130]), then all transient states can be removed, reducing the problem to finding the stationary probability distribution of an irreducible Markov chain. If the original Markov chain has multiple recurrent classes, one may first restrict the stationary analysis to each recurrent class. If necessary, one may further compute the limiting probability for each of the recurrent classes to be hit from a given initial distribution by performing a transient analysis. As described in Section 6.5, our implementation supports both these analyses.

3.2 Stationary analysis: the Power method

In this section, we review a traditional iterative method, the Power method (see e.g., [117]), to obtain the stationary distribution of a large irreducible Markov chain. We will use this method to solve a reduced Markov chain in Section 6.3.

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

Let us first recall some terms in basic matrix algebra since we will use them several times in this work. 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 $\xi A = \lambda \xi$. Consequently, ξ is a (left-hand) eigenvector of A corresponding to eigenvalue λ. In general, a square matrix has N eigenvalues (not necessarily distinct) denoted by $\lambda_1, \lambda_2, \cdots, \lambda_N$ if N

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

Suppose 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. Let this vector be 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 $\mathcal{C}, \alpha > 0$ are 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.

3.2.1 Computing π using the Power method: 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: $1 = \lambda_1 > |\lambda_2| \geq \cdots |\lambda_{|S|}|$ [68]. From (3.1), we see the stationary distribution of M is an eigenvector of P, in this case, corresponding to the simple dominant eigenvalue 1. Thus, π can be iteratively computed using the Power method as follows:

$$\pi^{(n+1)} = \pi^{(n)}P, \ n = 0, 1, \cdots \tag{3.2}$$

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.

3.2.2 Computing π using the Power method: 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 (3.1). In fact, the sequence $\{\pi^{(n)} : n \geq 0\}$ obtained from the iteration described by (3.2) oscillates.

However, the sequence $\{\pi^{(nd+r)} : n \geq 0\}$ converges, where $r = 0, 1, \cdots, d-1$ [73]. Denoting $\pi(r) = lim_{n\to\infty}\pi^{(nd+r)}$, we have $\pi(r) = \pi(r)P^d$ by Chapman-Kolmogorov equations [68, 73], where P^d is the d-th power of P. Hence, $\pi(r)$, $(r = 0, 1, \cdots d)$ are all eigenvectors of P^d corresponding to its eigenvalue 1. Moreover, eigenvalue 1 is d-multiple and dominant in this case, 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 $\pi = \lim_{n\leftarrow\infty} \frac{1}{n}\sum_{k=0}^{n}\pi^{(k)} = \frac{1}{d}\sum_{r=0}^{d-1}\pi(r)$ [73]. 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.,

$$\overline{\pi}^{(n)} = \frac{1}{d}\sum_{r=0}^{d-1}\pi^{(nd+r)} \tag{3.3}$$

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 value $r = 0, 1, \cdots, d-1$.

Equation (3.3) describes the iterative procedure of the Power method that we use when chain M is periodic where each iteration involves d vector-matrix multiplications. Note that by setting $d = 1$, the procedure reduces to the simple Power method used in the aperiodic case. Note that the period of M is found using a symbolic version of a simple algorithm described in [73] which is very efficient in practice.

3.3 Average time separations of events

In this section, we relate performance metrics to a notion of *average time separations of events*. We show that the latter can be derived from another notion called *sojourn times* [105] of a Markov chain.

3.3.1 Sojourn times

Consider a irreducible DTMC $M = \{M_n : n \geq 0\}$ with finite state space Q. Denote by A a proper subset of Q and $\overline{A} = Q - A$, the complementary subspace of A. Further, assume that both A and $\overline{A}$ have at least one state from the recurrent class of M. Clearly, both A and $\overline{A}$ will be visited infinitely often by M. Partition $(A, \overline{A})$ of the state space Q induces a decomposition of the transition probability matrix P into four sub-matrices and the stationary distribution π into two sub-vectors:

$$\begin{pmatrix} P_{AA} & P_{A\overline{A}} \\ P_{\overline{A}A} & P_{\overline{A}\overline{A}} \end{pmatrix} \quad \text{and} \quad \pi = (\pi_A, \pi_{\overline{A}}).$$

The time spent by M in subset A during its $n^{th} (n > 0)$ visit to A is a random variable and called the n^{th} *sojourn time* of M in A. More formally, a sojourn of M in subset A is a sequence $M_l, \cdots, M_{l+m}$ where $M_l, \cdots, M_{l+m-1} \in A, M_{l+m} \in \overline{A}$ and if $l > 0$ then $M_l \in \overline{A}$. This sojourn starts at time l, ends at time $l + m$, and lasts for m time steps. Let $N_{A,k}$ denote the k^{th} sojourn time of M in A. Rubino *et al.* [105]

show that the running average of $\{N_{A,k} : k \geq 1\}$ converges *in mean* to a constant called the sojourn time of M in A. Let it be denoted by J_A, then

$$J_A = \lim_{k\to\infty} \frac{1}{n} \sum_{l=1}^{k} \mathbf{E} N_{A,l} = \frac{\pi_A \mathbf{1}_A^T}{\pi_A (I - P_{AA}) \mathbf{1}_A^T}. \tag{3.4}$$

The quantity $\pi_A \mathbf{1}_A^T$ is the stationary probability that M is in set A, and $\pi_A(I - P_{AA})\mathbf{1}_A^T$ is essentially the probability of the chain leaving A. In this sense, the above equation states that the sojourn time of M in A is the reciprocal of the probability of the chain leaving A given that it is in A.

We now show that the running average of the sequence $\{N_{A,k} : k \geq 0\}$ also converges *almost surely*.

Lemma 3 *Let M be an irreducible finite state Markov chain. Let A be an non-empty subset of the state space of M. Then, the running average of the sequence $\{N_{A,k} : k \geq 1\}$ defined above converges almost surely to J_A given in Equation 3.4. That is,*

$$\frac{1}{n} \sum_{k=1}^{n} N_{A,k} \longrightarrow J_A, \ \textit{almost surely.} \tag{3.5}$$

The importance of this lemma lies in the fact that the running average of the sojourn time sequence converges to a constant for *almost all possible runs* of the Markov chain. As will be clear later that this implies the consistency of the system performance from one random run to another.

To prove Lemma 3, we need the following ergodic theorem of Markov chains (see [34], Theorem 15.2).

Theorem 2 *([34]) If $\{X_n : n \geq 0\}$ is an irreducible Markov chain with finite state space S and f is a function defined from S to $(-\infty, +\infty)$, then*

$$P(\lim_{n\to\infty} \frac{1}{n} \sum_{s=0}^{n} f(X_s) = \sum_{j\in S} \pi_j f(j)) = 1 \tag{3.6}$$

provided that the series converges absolutely.

Proof of Lemma 3 Let V_k be the random variable from which the k-th sojourn of X in A starts. Then, the sequence $\{V_{k\geq 1}\}$ is an irreducible Markov chain. To see it is Markovian, fix an entrance state, say, state i, for the k-th sojourn. The Markov property of X itself ensures that the entrance state of $(k+1)$-th sojourn is independent of the entrance points of sojourns prior to the k-th. The irreducibility of the sequence is due to that of X.

Let A' be the state space of this auxiliary Markov chain. Of course, A' is a nonempty subset of A. Further, let $N^i_{A,m}$ be the random variable denoting the length of the sojourn starting at state i for the m-th time. It is ready to check that $N^i_{A,m}$ are independent identically distributed (iid) random variables. Let N^i_A be the generic random variable denoting the sojourn times in A starting at state i. We first show that $\frac{1}{n}\sum_{k=1}^n N_{A,k} \xrightarrow{a.s.} \sum_{i\in A'} \pi'_i \mathbf{E} N^i_A$ where π' is the stationary distribution of the Markov chain $\{V_{k\geq 1}\}$.

With the aid of the Kronecker function $\delta_i(v) = \begin{cases} 1 & \text{if } v = i \\ 0 & \text{otherwise} \end{cases}$, for any possible sample path ω of X, we have

$$\begin{aligned}
& \lim_{n\to\infty} \frac{1}{n} \sum_{k=1}^{n} N_{A,k}(\omega) \\
= & \lim_{n\to\infty} \frac{1}{n} \sum_{k=1}^{n} \sum_{i\in A'} N_{A,k}(\omega)\delta_i(V_k(\omega)) \\
= & \lim_{n\to\infty} \sum_{i\in A'} \frac{1}{n} \sum_{k=1}^{n} N_{A,k}(\omega)\delta_i(V_k(\omega)) \\
= & \lim_{n\to\infty} \sum_{i\in A'} \frac{1}{n} \sum_{m=1}^{l_i(n,\omega)} N^i_{A,m}(\omega) \\
= & \lim_{n\to\infty} \sum_{i\in A'} \frac{l_i(n,\omega)}{n} \frac{1}{l_i(n,\omega)} \sum_{m=1}^{l_i(n,\omega)} N^i_{A,m}(\omega),
\end{aligned}$$

where $l_i(j,\omega) = \sum_{k=1}^{j} \delta_i(V_k(\omega))$ denoting the number of sojourns in A that starts from state i out of the first j sojourns.

Since A' is recurrent, $\lim_{n\to\infty} l_i(n,\omega) = +\infty$ almost surely. Therefore,

$$\frac{1}{l_i(n,\omega)} \sum_{m=1}^{l_i(n,\omega)} N^i_{A,m(w)} \longrightarrow \mathbf{E} N^i_A, \text{ almost surely.}$$

according to the *strong law of large numbers* due to the fact that $N^i_{A,j}(\omega)$ $(j = 1, 2, \ldots)$ are i.i.d random variables with geometric distribution, implying finite first and second moments.

Applying Theorem 2 on process $\{V_k : k \geq 1\}$ and taking $f = \delta_i$, we have

$$\frac{l_i(n,\omega)}{n} \longrightarrow \pi'_i, \text{ almost surely.}$$

Thus, we have proved $\frac{1}{n}\sum_{k=1}^{n} N_{A,k}(\omega)$ converges to $\sum_{i\in A'} \pi'_i \mathbf{E} N^i_A$ almost surely.

Next, we show that the sequence $\{\frac{1}{n}\sum_k^n \mathbf{E}N_{A,k}\}$ converges to $\sum_{i\in A'} \pi'_i \mathbf{E}N^i_A$. Indeed, we have

$$\begin{aligned}
& \lim_{n\to\infty} \frac{1}{n}\sum_{k=1}^{n} \mathbf{E}N_{A,k} \\
= \; & \lim_{n\to\infty} \frac{1}{n}\sum_{k=1}^{n}\sum_{i\in A'} P(V_k = i)\mathbf{E}N^i_{A,l(k)} \\
= \; & \lim_{n\to\infty} \sum_{i\in A'} \mathbf{E}N^i_A(\frac{1}{n}\sum_{k=1}^{n} P(V_k = i)) \\
= \; & \sum_{i\in A'} \pi'_i \mathbf{E}N^i_A.
\end{aligned}$$

The last equality is due to the fact that the partial sum $\frac{1}{n}\sum_{k=1}^{n} P(V_k = i)$ converges to π'_i since A' is irreducible. The proof is then completed by following the result of Rubino in Equation (3.4). ■

3.3.2 Average cycle time and throughput

We define an *event* as a transition of a state variable. For simplicity, this section will discuss only events of boolean state variables. Extensions to enumerate state variables are straightforward.

Denote $v(n)$ the value of variable v at time n. We say variable v makes a transition at time n if $v(n) \neq v(n-1)$. Let $Q^+_v = \{s \mid s(v) = 1, s \in Q\}$ and $Q^-_v = \{s \mid s(v) = 1, s \in Q\}$ where $s(v)$ is the value of variable v in state s. Clearly, Q^+_v and Q^-_v partition Q. Let us denote by $v^+(v^-)$ an event that corresponds to a state transition $t \in Q^-_v \times Q^+_v (t \in Q^+_v \times Q^-_v)$.

The average cycle time of event v^+, denoted by C_{v^+}, is the asymptotic time difference between two consecutive occurrences of v^+. Let the tuple (v^+, l) be the time of l^{th} occurrence of v^+ and $\Delta_{v^+,l} = (v^+, l+1) - (v^+, l)$ be the time difference between the $(l+1)^{th}$ and l^{th} occurrences of v^+. Note that both (v^+, l) and $\Delta_{v^+,l}$ are random variables. Then, the cycle time C_{v^+} is defined to be:

$$C_{v^+} = \lim_{k\to\infty} \frac{1}{k} \sum_{l=1}^{k} \Delta_{v^+,l}. \tag{3.7}$$

Since the time difference of l^{th} occurrence of $v^+(v^-)$ and its succeeding occurrence of $v^-(v^+)$ is the l^{th} sojourn time of $Q_v^+(Q_v^-)$, we can determine C_{v^+} using sojourn times.

Theorem 3 *The running average of cycle time sequence due to event v^+ converges almost surely and in mean to a constant for which the following equation holds.*

$$C_{v^+} = \lim_{k\to\infty} \frac{1}{k} \sum_{l=1}^{k} N_{Q_v^+,l} + \lim_{k\to\infty} \frac{1}{k} \sum_{l=1}^{k} N_{Q_v^-,l} = J_{Q_v^+} + J_{Q_v^-}, \tag{3.8}$$

where J denotes the sojourn time of the corresponding subset of states as given in Equation 3.4.

Proof By the fact that both Q_v^+ and Q_v^- are both non-empty subsets of Q plus the results of Lemma 3 and Rubino's Equation (3.4). ■

The average cycle time of v^- is defined and can be determined similarly. In fact, we have $C_{v^-} = C_{v^+}$.

The throughput of the system is then defined to be the reciprocal of the average cycle time of some corresponding *indicating* event (such as the rising edge of a completion signal from a functional unit). Consequently, if the indicating event is v^+, then the corresponding throughput is computed as $\frac{1}{C_{v^+}}$. This naturally extends to more general indicating events.

3.3.3 Average time separation of events (TSE)

To determine other performance metrics such as latency, we need to study the average time difference between multiple events.

We define the time separations between events α and β as a sequence of time intervals, $\mathcal{I}(n) \triangleq \{I_k = [a_k, b_k] : k = 1, 2, \cdots, N_{\alpha\to\beta}(n)\}$ up to time n, where the starting and ending times of each interval are recursively defined as follows:

$$\begin{aligned}
a_0 &= \min\{i : \alpha \in E_i\}, \\
b_k &= \min\{i : \beta \in E_i, i \geq a_k\}, \qquad \text{for all } k \geq 0 \\
a_k &= \min\{i : \alpha \in E_i, i > b_{k-1}\}, \quad \text{for all } k \geq 1 \text{ and} \\
N_{\alpha\to\beta}(n) &= i : b_i \leq n, a_{i+1} > n,
\end{aligned}$$

where E_i is the set of events that occur at time i.

Note that the length of time interval I_k, denoted by $\ell(I_k)$ is a random variable. In addition, I_k has zero length if $a_k = b_k$. We define the average time separation between event α and β, denoted by $\tau_{\alpha\to\beta}$, as the asymptotic behavior of $\ell(I_k)$:

$$\tau_{\alpha\to\beta} \triangleq \lim_{n\to\infty} \frac{1}{N_{\alpha\to\beta}(n)} \sum_{k=1}^{N_{\alpha\to\beta}(n)} \ell(I_k) = \lim_{n\to\infty} \frac{1}{|\mathcal{I}(n)|} \sum_{I\in\mathcal{I}(n)} \ell(I) \tag{3.9}$$

where $|\mathcal{I}(n)|$ denotes the total number of intervals in sequence $\mathcal{I}(n)$.

Theorem 4 *Let α and β are two events of an asynchronous systems that is modeled as an irreducible finite state Markov chain. Then, their average time separation $\tau_{\alpha\to\beta}$ defined in Equation 3.9 converges to a constant almost surely and in mean.*

Proof The theorem is proved by similar arguments made in the proof of Theorem 3. Details are omitted. ■

Again, $\tau_{\alpha\to\beta}$ can be computed using sojourn times. The idea is to partition the time intervals into two sets: intervals that have non-zero length, $\mathcal{I}_1(n) = \{I_k \mid \ell(I_k) > 0,\ I_k \in \mathcal{I}(n)\}$, and those that have zero-length, $\mathcal{I}_2(n) = \{I_k \mid \ell(I_k) = 0,\ I_k \in \mathcal{I}(n)\}$. This can be accomplished by introducing three extra state variables into the model, say x, y and z. The behavior of x is defined such that sojourns of subspace $x = 1$ are in one-to-one correspondence with time intervals in $\mathcal{I}_1(n)$. Similarly, the behavior of z is defined such that the sojourns of subspace $z = 1$ are in one-to-one correspondence with time intervals in $\mathcal{I}_2(n)$. Finally, the behavior of y is defined such that the total number of time intervals is the number of sojourns of subspace $y = 1$. Figure 3.1 gives the definitions of these three variables. It is

```
Px  :=  case
        u ≠ u' & u' = 1 & v = v'  :  {1};   /* if α but not β occurs */
        v ≠ v' & v' = 1           :  {0};      /* if β occurs */
        1                         :  {x};   /* otherwise */
Py  :=  case
        u ≠ u' & u' = 1 & v ≠ v'
        & v' = 1                  :  {1};   /* if α and β both occur */
        1                         :  {0};   /* otherwise */
Tz  :=  case
        x = 0 & x' = 1 | z = 0
        & z' = 1                  :  {1};   /* if x or z rises */
        1                         :  {0};   /* otherwise */
```

Figure 3.1: Definition of the auxiliary variables.

important to note that the new model with these three auxiliary variables added is still Markovian because the variables can not introduce cycles in the dependency graph.

To illustrate how the auxiliary state variables track the time separation intervals, assume that both α and β are positive transitions, e.g., $\alpha = u^+$ and $\beta = v^+$. Figure 3.2(a) illustrates the situation where the intervals have non-zero length while Figure 3.2(b) shows a zero-length interval where α and β happen simultaneously. Sojourn times of $x = 1$ equal the corresponding $non - zero$ time intervals between u^+ and v^+. Sojourns of $y = 1$ or $z = 1$ both have unit length.

It is now easy to relate the average time separation between α and β to the sojourn time of $x = 1$ by elementary probability analysis. From Equation 3.9, we have

$$\tau_{\alpha \to \beta} = \lim_{n \to \infty} \frac{\mathcal{I}_1(n)}{\mathcal{I}(n)} \times [\frac{1}{\mathcal{I}_1(n)} \sum_{I \in \mathcal{I}_1(n)} \ell(I)] \tag{3.10}$$

$$P\left\{\ell(I) > 0 \mid I \in \lim_{n \to \infty} \mathcal{I}(n)\right\} \times J_{x=1} \tag{3.11}$$

where $J_{x=1}$ is the average sojourn time of subspace $x = 1$.

The term $P\left\{\ell(I) > 0 \mid I \in \lim_{n\to\infty} \mathcal{I}(n)\right\}$ in Equation 3.11 is the limiting probability for an arbitrary time interval in $\mathcal{I}(n)$ to have non-zero length. It can be determined by the following equation:

$$P\left\{\ell(I) > 0 \mid I \in \lim_{n \to \infty} \mathcal{I}(n)\right\} = \frac{\pi_{y=1}\mathbf{1}_{y=1}^T - \pi_{z=1}\mathbf{1}_{z=1}^T}{\pi_{y=1}\mathbf{1}_{y=1}^T} \tag{3.12}$$

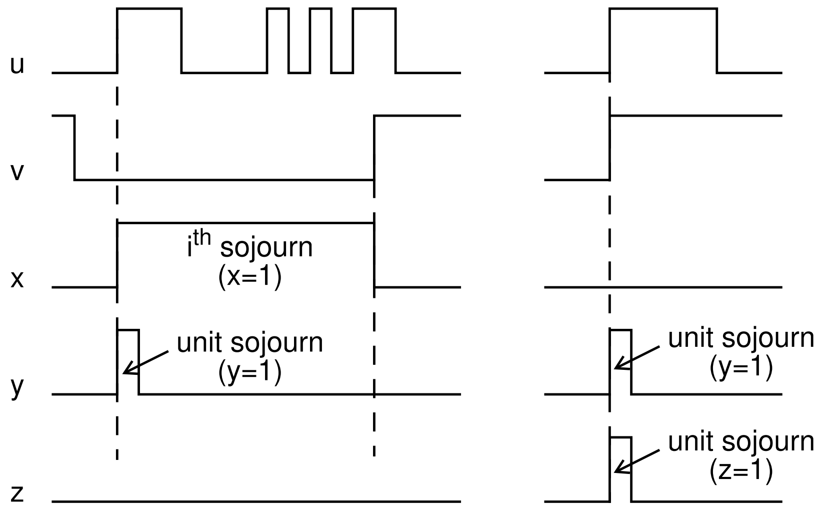


Figure 3.2: The separations between u^+ and v^+ and the sojourns of the auxiliary variables when (a) u^+ and v^+ do not occur at the same time, and (b) when u^+ and v^+ occur at the same time.

where $\pi_{y=1}$ and $\pi_{z=1}$ are the vectors representing the stationary probabilities of the states in subspace $y = 1$ and $z = 1$, respectively, and $\mathbf{1}_{y=1}$ and $1_{z=1}$ are all-one vectors of proper length.

3.4 Symbolic Markovian analysis

The principal difficulty with Markovian analysis is the computational challenge in dealing with the large state spaces of real systems. In particular, the most time consuming part of the analysis is the computation of the stationary distribution of the Markov chain. Once the stationary distribution is computed, the time spent in calculating the average TSEs is negligible. This section demonstrates how we address this problem by combining iterative Power method with symbolic representations using *binary decision diagrams* (BDDs) [16] and *algebraic decision diagrams* (ADDs) [7] to compute the stationary distribution. In chapter 6, we present a more powerful approach to stationary analysis of Markov chains modeling asynchronous systems which uses a novel technique called *state compression* [127, 132].

3.4.1 Symbolic reachability analysis

The first step in our Markovian analysis is to find all reachable states, which requires a reachability analysis. After that, we need to determine if the resulting Markov chain is irreducible. More generally, a state classification is required to determine all recurrence classes and all transient state of the Markov chain. For now, we assume that Markov chain contains only one recurrent classes so that all transient states cannot affect that stationary behavior of the systems and thus can be ignored. In Chapter 4, a powerful symbolic state classification technique is presented. The importance of finding all the reachable states is that we can then restrict the transition matrix P to the reachable state space which usually significantly reduces the size of the ADD that represents P, and hence improve the computational efficiency.

We propose to use a well-known symbolic approach for reachability analysis [86, 20, 38]. to mitigate the the state explosion problem. The approach is iterative and works as follows. Let Q_0 be the set of initial states and Q_k be all the states reachable from Q_0 in at most k steps from Q_0. The set Q_k can be computed iteratively as follows:

$$Q_{k+1} = Q_k \cup \{s' \mid \exists s, s \in Q_k - Q_{k-1}, (s, s') \in T\},$$

where $Q_{-1} = \emptyset$ and T is the (boolean) transition relation the system introduced in Section 2.1.3). The iteration continues until the *frontier set* $Q_k - Q_{k-1}$ is empty, reaching a fixed point.

This fixed point is computed using BDDs. Let $\mathcal{T}_v(V, V')$ be the BDD corresponding to the partial transition relation T_v, where V and V' denote the BDD variables for the current and the next state spaces, respectively. Then, the global (or monolithic) transition relation of a timed system $\mathcal{T}(V, V')$ is a *conjunction* of BDDs of all the partial transition relations, i.e., $\mathcal{T} = \bigwedge_{v \in V} \mathcal{T}_v$. Let $\mathcal{Q}_k(V)$ be the BDD corresponding to Q_k. Then, the set of reachable states is the fixed point of the following computation:

$$\mathcal{Q}_{k+1}(V') = \mathcal{Q}_k(V') \vee [\exists_{v \in V} \mathcal{Q}_k(V) \wedge \neg \mathcal{Q}_{k-1}(V) \wedge \mathcal{T}(V, V')].$$

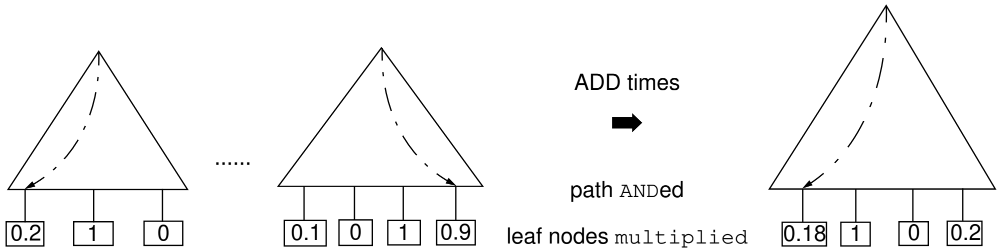


Figure 3.3: Construction of the ADD for the probabilistic transition relation.

3.4.2 Symbolic Power method

Let $\mathcal{P}_v(V, V')$ represent the ADD for the partial probabilistic transition relation P_v. Then, the probabilistic transition relation can be represented by an ADD $\mathcal{P}(V, V')$ which is constructed using ADD `times` as follows:

$$\mathcal{P}(V, V') = \cdot_{v \in V} \mathcal{P}_v(V, V')$$

Figure 3.3 illustrates the construction of $\mathcal{P}$ from $\mathcal{P}_v$'s.

The symbolic calculation of the Power method is the same as the explicit one except that vector-matrix multiplication is replaced with ADD `times`. To accelerate the convergence, however, we take P^{iter}, a higher order of P to be the multiplicative. To choose a proper power number $iter$, we perform iterative squaring on $P^{2*iter} \leftarrow P^{iter} \cdot P^{iter}$ and $iter \leftarrow 2*iter$ starting with $iter = 1$. The squaring process terminates when either the $\texttt{size}(P^{iter})$ exceeds a user definable constant or $\texttt{size}(P^{2*iter}) > 2\texttt{size}(P^{iter})$. This optimization yields faster convergence at the expense of more computation in each iteration step, resulting in a short run-time. Moreover, many ADD/BDD size minimization techniques [7, 110] can be used to reduce the size of $\mathcal{P}(V, V')$ with respect to all the unreachable states.

Chapter 4

State Classification

4.1 Introduction

The technique to compute the average time separations of events (TSE) developed in the previous chapter assumes the Markov chain model has a unique recurrence class. If the Markov chain has multiple recurrence classes, it is necessary to identify all the recurrence classes and treat each of them as a separate Markov chain. In this case, a *transient analysis* may also be required to compute the probability that each of the recurrence class is eventually hit from the initial state (or states). For the stationary analysis, removing all the transient states also improves computational efficiency. Besides, by treating each recurrence class as a single state, transient analysis can be speeded up significantly if a large portion of the state space is recurrent, which is typically the case. In fact, even when the number of the recurrence classes is expected to be one, it is often necessary to verify this is the case because modeling errors or design bugs may introduce extra recurrence classes. Unfortunately, the size of the (reachable) state space of Markov chains modeling from real systems are usually very large, for instance routinely exceeding 10^6, making identification of recurrence classes computational challenging.

This chapter describes a symbolic technique [130, 131] based binary decision diagrams (BDDs) [16] to efficiently classify the states of a large Markov chain. It is worthwhile to note Markov chains are an important class of stochastic processes that can effectively model randomly evolving systems. They have vast applications in engineering including the analysis of system reliability [77, 139], control [114, 106, 15], power consumption [79, 8] and performance [63, 72, 126, 66, 119]. State classification is a generic problem in all these applications.

The structure of the state space of a Markov chain can be represented by a directed graph where vertices correspond to the states and edges correspond to the transitions among the states. One method to classify the state space is to *explicitly* identify all the (maximal) *strongly connected components* (SCCs) of the graph. Components (those called terminal SCCs) that do not have edges leading to other components correspond to the recurrent state classes and all other components correspond to the set of transient states. Since Markov chains derived from real systems often have huge state space [63, 56, 126], this method can be intolerably slow because it analyzes states *sequentially*. Another method for structural analysis is described in [56][1] which uses *implicit* (symbolic) techniques based on *binary decision diagrams* (BDDs) [16] so that all the SCCs of the corresponding graph are *concurrently* computed and the terminal SCCs are subsequently identified. This method adopts the recursive paradigm in [85] to compute all the states that can be reached from individual states, or from a graph point of view, the *transitive closure* of the corresponding graph. Unfortunately, this step tends to be *memory-intensive* and *time-consuming*.

As mentioned earlier, the technique described in this chapter also relies on BDDs but exploits the fundamental property of transience and recurrence of a Markov chain. The technique *sequentially* explores the state space by repeatedly applying symbolic reachability analysis. Newly reached transient states and recurrence class are detected after each reachability analysis. Since most systems contain only a few recurrence classes and a large portion of their state spaces are recurrent, the number of times reachability analysis is performed is often very small. Finally, because reachability analysis is typically much faster and less memory-intensive than the transitive closure computation, our method dramatically outperforms the one discussed in [56] in almost all the examples we have tested. In particular, we observed computational time reductions of up to several orders of magnitude, and the new method solves all the examples where the previous method fails due to insufficient memory.

[1] We note, however, that the main thrust of [56] is symbolic algorithms for stationary analysis with less emphasis on structural analysis.

It should be noted that a similar algorithm has been independently proposed by Qadeer, Brayton, Singhal and Pixley in [97, 112]. However, only the case of a single TSCC is considered in their work. On the contrary, we provide a state-space pruning procedure that enables us to consider multiple TSCCs. It should also be noted that the algorithm of [97, 112] has been conceived and used in a completely different application domain. The problem they have considered is that of finding a *design replacement* of sequential circuits without designated initial states. Roughly speaking, a new, possibly smaller design (e.g., with fewer latches), is an *n-cycle delay replacement* [97, 112] of an original design if the environment can not distinguish them once they have operated for n cycles after power-up. One aggressive approach to obtaining such a new design is to preserve the desirable steady-state behavior of the old design for a single terminal SCC and constrain the other states in the new design to fall into the terminal SCC within n-cycles.

The remaining of the chapter is organized as follows. Section 4.2 reviews more details of Markov chains necessary for further discussion, and the previous work on state classification of Markov chains. Section 4.3 describes the principles, proofs, and the algorithm of our method. Experimental results are given in Section 4.4.

4.2 Preliminaries and previous Work

4.2.1 More about Markov chains

Let $X = (P, S)$ be a (finite state discrete-time) Markov chain where P is the transition matrix and S the state space. To consider the relationship between states of S, one needs the notion of *communication*. A state $i \in S$ communicates with state $j \in S$, denoted by $i \rightarrow j$, if there is a positive probability that state j will eventually be visited once state i is visited. Or equivalently, $p_{ij}(n) = Pr(X_n = j \mid X_0 = i) > 0$ for some $n > 0$. Otherwise, state i does not communicate with state j, denoted by $i \not\rightarrow j$. States i and j *intercommunicate* if $i \rightarrow j$ and $j \rightarrow i$, written $i \leftrightarrow j$. Using the notation of communication, one can redefine recurrence and transience property of a state. For example, state $i \in S$ is recurrent if $\forall j \in S$ i.e., $i \rightarrow j$, $j \rightarrow i$ as well. Otherwise, state i is transient, i.e, $\exists j \in S$ i.e., $i \rightarrow j$ but $j \not\rightarrow i$.

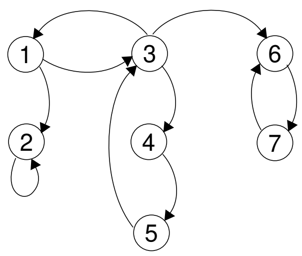


Figure 4.1: A small Markov chain example.

It can be checked that a recurrent state is visited infinitely often once it is visited, while the expected number of the times X visits a transient state is bounded from above.

Many of the nice properties of Markov chain X can be derived from the famous *Chapman-Kolmogorov* theorem [68] which expresses the probability of X to go from one state to another in $n \geq 0$ time steps in terms of the 1-step transition matrix P. One such property concerning the communication relation is as follows.

Theorem 5 *If $i \to j$ and $j \to k$, then $i \to k$.*

By the definition of a recurrent state and Theorem 5, it is easily verified that if states i and j intercommunicate, either both of them are recurrent or both of them are transient.

Theorem 5 implies that $\leftrightarrow$ is transitive, symmetric and reflexive. Thus, $\leftrightarrow$ is an equivalence relation so that the state space can be partitioned into equivalence classes according to $\leftrightarrow$. That is, $S = T \cup R_1 \cup R_2 \cdots \cup R_K$ where T is the set of transient states, R_1 through R_K are K recurrence classes each containing recurrent states that do not communicate with states in other classes.

Example Figure 4.1 shows a Markov chain with 7 states. State 1 is transient because $1 \to 2$ but $2 \not\to 1$. States 3, 4 and 5 are also transient for similar reasons. The chain has two recurrence classes, namely, $\{2\}$ and $\{6, 7\}$. □

We have seen that the transience and recurrence property of states can be derived from the their communication relation. In particular, the structure of the state space can be represented by a directed graph $G = (V, E)$, where the vertices in V are in one-to-one correspondence with the states in S. There is an edge e_{uv} from vertex u to v iff $p_{ij} > 0$ where the vertices u and v correspond to the states i and j. Then,

$i \to j, i, j \in S$ means there is a path from vertex u to v in G. Similarly, $i \leftrightarrow j$ means vertices u and v are strongly connected. We call graph G the boolean transition graph of X.

4.2.2 Previous work

Recently, Hachtel et al. proposed a symbolic method [56] which finds the recurrent classes by *concurrently* identifying all terminal SCCs of the transition graph G associated with the Markov chain.

The method symbolically computes *all* the SCCs in G by creating the transitive closure TC of G which describes the set of states that each state of S communicates with. One may think of TC as a boolean matrix of size $S \times S$, i.e, $TC_{ij} = \begin{cases} 1 & \textit{if} \quad i \to j, \\ 0 & \textit{otherwise.} \end{cases}$ Therefore, checking if $i \to j$ is equivalent to checking if the element $TC_{ij} = 1$. Suppose it is determined that $i \to j$ by the fact that $T_{ij} = 1$. To see if $j \to i$ so that i, j are strongly connected, one merely needs to check if $T_{ji} = 1$. Thus, the second step in the method is to *term-wise* multiply TC with its transpose, TC^t, yielding a symmetric matrix (let it be denoted by $TC * TC^t$). Clearly, i and j are in the same SCC *iff* $(TC * TC^t)_{ij} = (TC^t * TC)_{ji} = 1$. All above operations are performed symbolically using BDDs.

Next, each SCC is represented by a particular state (called the representative state) of the SCC using the so called *datum priority function*[57] of the corresponding BDD. A new directed graph consisting of these representative states is then constructed so that it has an edge from one (representative) state i to another (representative) state j *iff* there is an edge from some state in the SCC represented by i to some state in the SCC represented by j. This new graph is *acyclic* and its leaf vertices represent the terminal SCCs.

The bottleneck of the method is the computation of the transitive closure TC of G. To this end, a recursive algorithm presented in [85] which also uses BDD-based symbolic operations is adopted. Nevertheless, it can still take intensive amount of CPU time and memory to compute TC when G is large.

4.3 State classification by iterative application of reachability analysis

The method presented in this section does not compute the transitive closure of the graph G. Rather, it relies on the definition of a transient/recurrent state. Once we determine a state to be transient/recurrent, the transience and recurrence properties of other states that communicate with this state are deduced and they are removed from further consideration.

4.3.1 The principles

We start by defining the *forward* and *backward* sets of a state.

Definition 3 *The forward set of state $i \in S$, denoted by $\mathcal{F}(i)$, is the set of states that i communicates with. That is, $\mathcal{F}(i) = \{j \in S \mid i \to j\}$. Similarly, the backward set of state i, denoted by $\mathcal{B}(i)$, is the set of states that communicate with i. That is, $\mathcal{B}(i) = \{j \in S \mid j \to i\}$.*

The following lemma shows a simple property of forward and backward sets.

Lemma 4 *Let $i, j \in S$. If $j \in \mathcal{F}(i)$, then $\mathcal{F}(j) \subseteq \mathcal{F}(i)$. Similarly, if $j \in \mathcal{B}(i)$, then $\mathcal{B}(j) \subseteq \mathcal{B}(i)$.*

Proof The proof is a direct consequence of the transitivity property of relation $\to$ given in Theorem 5. Suppose $j \in \mathcal{F}(i)$, i.e., $i \to j$, then, $\forall k \in \mathcal{F}(j)$, i.e., $j \to k$, we have $i \to k$, meaning $k \in \mathcal{F}(i)$ so that $\mathcal{F}(j) \subseteq \mathcal{F}(i)$. Similarly, if $j \in \mathcal{B}(i)$, i.e., $j \to i$, then, $\forall k \in \mathcal{B}(j)$, i.e., $k \to j$, we have $k \to i$, which implies $\mathcal{B}(j) \subseteq \mathcal{B}(i)$. ■

Theorem 6 *A state $i \in S$ is recurrent if and only if $\mathcal{F}(i) \subseteq \mathcal{B}(i)$. In other words, i is transient if and only if $\mathcal{F}(i) \not\subseteq \mathcal{B}(i)$.*

Proof Suppose $i \in S$ is recurrent, then $\forall j \in S$ such that $i \to j$, i.e., $j \in \mathcal{F}(i)$, we have $j \to i$ by the definition of a recurrent state. That is to say, $j \in \mathcal{B}(i)$. Therefore, $\mathcal{F}(i) \subseteq \mathcal{B}(i)$. Conversely, if $\mathcal{F}(i) \subseteq \mathcal{B}(i)$, which means $\forall j \in S$ such that $i \to j$, we have $j \in \mathcal{F}(i) \subseteq \mathcal{B}(i)$, and hence $j \to i$. Then i is recurrent by definition. The fact that a state is either recurrent or transient concludes the proof. ■

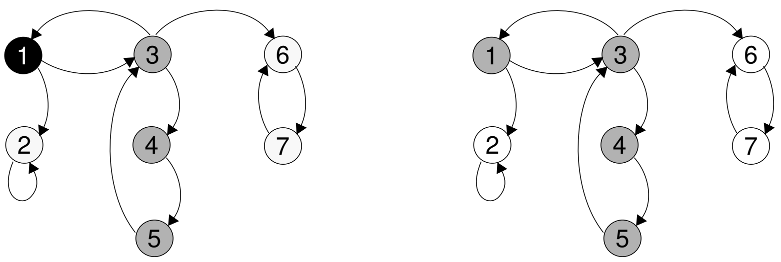


Figure 4.2: Determining if a state is recurrent: (a) computing $\mathcal{F}(1)$ and $\mathcal{B}(1)$, (b) states in $\{1\} \cup \mathcal{B}(1)$ all transient.

Theorem 7 *If state $i \in S$ is transient, then states in $\mathcal{B}(i)$ are all transient. If state i is recurrent, on the other hand, states in $\mathcal{F}(i)$ are all recurrent. In the latter case, set $\mathcal{F}(i)$ is a recurrence class, and set $\mathcal{B}(i) - \mathcal{F}(i)$ (if not empty) contains only transient states.*

Proof Suppose state i is transient. By Theorem 6, $\mathcal{F}(i) \not\subseteq \mathcal{B}(i)$, i.e., $\exists k \in \mathcal{F}(i)$ such that $k \notin \mathcal{B}(i)$. Now, suppose state $j \in \mathcal{B}(i)$, then $j \to k$ so that $k \in \mathcal{F}(j)$. This is because $i \in \mathcal{F}(j)$ so that $\mathcal{F}(i) \subseteq \mathcal{F}(j)$ by Lemma 4. On the other hand, $\mathcal{B}(j) \subseteq \mathcal{B}(i)$ since $j \in \mathcal{B}(i)$. Therefore, we have state $k \in \mathcal{F}(j)$ but $k \notin \mathcal{B}(j)$ since $k \notin \mathcal{B}(i)$, which implies $\mathcal{F}(j) \not\subseteq \mathcal{B}(j)$ so that j is transient by Theorem 6.

If state i is recurrent, i.e., $\mathcal{F}(i) \subseteq \mathcal{B}(i)$ by Theorem 6, then, $\forall j \in \mathcal{F}(i) \Longrightarrow i \leftrightarrow j$ by definition so that we have $\mathcal{F}(j) \subseteq \mathcal{F}(i)$ and $\mathcal{B}(i) \subseteq \mathcal{B}(j)$ by Lemma 4. Thus, $\mathcal{F}(j) \subseteq \mathcal{F}(i) \subseteq \mathcal{B}(i) \subseteq \mathcal{B}(j)$, which implies j is recurrent by Theorem 6. Finally, if i is recurrent and $\mathcal{B}(i) - \mathcal{F}(i)$ is not empty, let state $k \in \mathcal{B}(i) - \mathcal{F}(i)$, we merely need to show that $\mathcal{F}(k) \not\subseteq \mathcal{B}(k)$ so that k is transient. In fact, $k \in \mathcal{B}(i) \Longleftrightarrow i \in \mathcal{F}(k)$, and $k \notin \mathcal{F}(i) \Longleftrightarrow i \notin \mathcal{B}(k)$, which implies $\mathcal{F}(k) \not\subseteq \mathcal{B}(k)$. ■

Example Figure 4.2(a) shows the small example Markov chain in Section 4.2. To determine if state 1 is recurrent, we first compute $\mathcal{F}(1) = \{1, 2, 3, 4, 5, 6, 7\}$ and $\mathcal{B}(1) = \{1, 3, 4, 5\}$. Since $\mathcal{F}(1) \not\supseteq \mathcal{B}(1)$, not only is state 1 transient, all states in $\mathcal{B}(1)$ are transient as well. Thus, only states 2, 6 and 7 need to be further classified. □

Theorem 6 determines if a state i is recurrent by checking if its forward set is contained in its backward set. If it is, then a recurrence class has been found which equals the forward set. Moreover, according to Theorem 7 if the backward set properly contains the forward set, those states in the backward set not belonging to the forward set are all found to be transient. In the case the forward set is not contained in the backward set, we have found a set of transient states equal to $\{i\} \cup \mathcal{B}(i)$.

The next subsection gives our symbolic algorithm for state classification based on the above theory.

4.3.2 The algorithm

We associate the transition graph $G = (S, E)$ with a boolean transition relation Q defined from $S \times S$ into $\{0, 1\}$ such that $Q(s, s') = 1$ iff $e_{s,s'} \in E$. We say s is the current state while s' is the next state. To represent the relation Q with BDDs, let X and Y be two sets of boolean variables encoding current states and next states, respectively. Details of BDDs and their operations can be found in [16].

4.3.2.1 Computing forward and backward sets

Computation of the forward set can be performed exactly as the symbolic reachability analysis for finite state machines using *fixed-point calculations* [20, 38].

The idea of fixed point calculation may be explained by the concept of the frontier set, FS. Initially, the reachable state set, RS, is set to $\emptyset$, and the frontier set is set to be the initial states. That is, $RS^{(0)} = \emptyset$, and $FS^{(0)} = I$, where I is the set of initial states. Next, the set of states, Z, that can be reached from the current frontier set $FS^{(0)}$ in one time step (or one state transition) is computed. Those states that are newly reached during this iteration are taken as the frontier for the next iteration. The process repeats until the frontier set is empty.

The first procedure given in Figure 4.3(a) computes the forward set of state s where $\exists$ and $\wedge$ denote BDD operations of existential quantification and AND, respectively. Applying the same idea, the backward set of s can be computed similarly.

4.3.2.2 Classifying states

forward_set(s, Q) {
  $RS^{(0)} \leftarrow \emptyset, FS^{(0)} \leftarrow \{s\}$;
  $k \leftarrow 0$;
  while$(FS^{(k)} \neq \emptyset)$ {
    $FS^{(k+1)}(x) = \exists_y FS^{(k)}(y)Q(y,x) \wedge \overline{RS^{(k)}(x)}$;
    $RS^{(k+1)}(x) = RS^{(k)}(x) \vee FS^{(k+1)}(x)$;
    $k \leftarrow k+1$; }
  return$(\mathcal{F}(s) \leftarrow RS^{(k)})$; }

(a)

backward_set(s, Q) {
  $RS^{(0)} \leftarrow \emptyset, FS^{(0)} \leftarrow \{s\}$;
  $k \leftarrow 0$;
  while$(FS^{(k)} \neq \emptyset)$ {
    $FS^{(k+1)}(x) = \exists_y FS^{(k)}(y)Q(x,y) \wedge \overline{RS^{(k)}(x)}$;
    $RS^{(k+1)}(x) = RS^{(k)}(x) \vee FS^{(k+1)}(x)$;
    $k \leftarrow k+1$; }
  return$(\mathcal{B}(s) \leftarrow RS^{(k)})$; }

(b)

Figure 4.3: Computing the forward set and backward set.

State_classification(S, Q) {
  $S' \leftarrow S$;
  while$(S' \neq \emptyset)$ {
    $s \leftarrow$ random_take(S');
    $\mathcal{F}(s) \leftarrow$ forward_set(s, Q);
    $\mathcal{B}(s) \leftarrow$ backward_set(s, Q);
    if$(\mathcal{F}(s) \wedge \overline{\mathcal{B}(s)} = \emptyset)$ {
      report $\mathcal{F}(s)$ a recurrence class;
      report $\mathcal{B}(s) \wedge \overline{\mathcal{F}(s)}$ all transient;
      $S' \leftarrow S' \wedge \overline{\mathcal{B}(s)}$; }
    else {
      report $s \vee \mathcal{B}(s)$ all transient;
      $S' \leftarrow S' \wedge \overline{s \vee \mathcal{B}(s)}$; } } }

Figure 4.4: The state classification algorithm.

Figure 4.4 shows the state classification algorithm. The algorithm accepts as an input the states S to be classified and *iteratively* partitions S into all its recurrent classes and transient states. Initially, S is copied into the set S' which represents the remaining states to be classified. During each iteration, a state from S' is taken as a *trial* state whose forward set and backward set are computed using the procedures given in Figure 4.3. Next, the trial state is determined to be either recurrent or transient by checking the containment of its forward and backward sets according to Theorem 6. States that communicate with this trial state are determined to be recurrent or transient according to Theorem 7 and are removed from S'.[2] The loop terminates when there is no state left to be classified (i.e., S' is empty).

The set S is typically the set of states reachable from any initial state specified in the model, because the classification of unreachable states (i.e., states not reachable from the given set of initial states) is usually not of interest [56, 126]. Alternatively, however, one can partition all 2^n states of the state space, where n is the number of state variables in the model, by setting S to be the BDD 1. Note that in this case that our algorithm completes as usual when S' becomes empty.

In the experiments, we find S using an initial standard reachability analysis from all initial states. In contrast, however, the transitive-closure-based approach does not in principle require this step because the set of reachable states can be trivially found once the transitive closure of the model is derived. In practice, however, the transitive closure computation is typically much more computationally expensive than standard reachability analysis. Consequently, we can speed up the transitive closure approach by first computing the reachable states using standard reachability analysis and then minimizing the other BDDs involved using the unreachable states as don't cares [56].

The function $random\_take(S')$ works as follows. It randomly takes a state, say s_1 from S' as a start point, and searches for a state s_2 not yet considered to be a trial state but reachable from s_1 within a user definable number of steps. The intuition is that, state s_2 is more likely to be recurrent because $s_2 \in \mathcal{F}(s_1)$. Since this search

[2] Note that the case in which the trial state is recurrent is only useful when finding all TSCCs (because at this point one TSCC has already been found) and this step was not considered by Qadeer et al. [97, 112].

can be done iteratively and after each iteration only one state is selected to be the new frontier set, the time spent in this function is negligible.

4.3.2.3 The complexity

In general, the new algorithm has much better complexity than the method in [56]. This is because the reachability analysis is performed only for part of the state space, i.e, for those trial states returned by function *random_take*() whereas the transitive closure of graph G required by the latter is equivalent to the knowledge of the reachable sets for all the states in S.

The worst case complexity of our algorithm can occur when each iteration determines the transience or recurrence property of only the corresponding trial state because its backward set contains only itself. Then, exactly $|S|$ iterations would be required. This could happen for instance if (1) every state in S forms a single-state recurrence class or (2) all states except one are transient and each transient state reaches only the sole recurrent state. Since most real systems contains relatively few recurrence classes and a significant portion of the state space is recurrent, very few (typically only one) iterations are performed so that the worst case complexity is rarely observed. Another case where our algorithm may also be slow is when all states (nearly) form a loop. This case does not require a large number of iterations, but the time spent in the reachability analysis during each iteration could be significant.

Since the complexity of the algorithm depends on the structure of the Markov chain, it is also possible to quantify it in a probabilistic sense. To do this, however, appears difficult without some assumptions. Let T be the set of transient states, and R_1 through R_K be the $K(K > 0)$ recurrence classes. We make the following assumptions to simplify the analysis.

`Assumption 1.` *The K recurrence classes have roughly the same size, say $|R|$. Thus, the size of the state space $|S|$ is approximately $|T| + K|R|$.*

`Assumption 2.` *The structure of the transient portion of state space is such that on average a randomly taken transient state is reachable from half of all transient states.*

`Assumption 3.` *On average, about $\frac{1}{K}$ of the transient states reach a given recurrent class.*

Let us first characterize the expected total number of state visits by the algorithm. (A state visit here refers to an instance when a state is visited.) Let this quantity be denoted by a function $G(|T|, |R|, K)$.

For the first iteration, with probability $p = \frac{|T|}{|T|+K|R|}$, the randomly chosen trial state is transient, and with probability $1-p$, recurrent. If the trial state is transient, then the number of state visits is upper bounded by $2|T|+K|R|$. On the other hand, if the trial state is recurrent, then the number of state visits is about $2|R| + \frac{|T|}{K}$ by assumption 3. Therefore, the expected number of state visits during first iteration is $p \cdot (2|T| + K|R|) + (1-p) \cdot (2|R| + \frac{|T|}{K}) \leq 2(|T| + |R|)$.

Moreover, if the trial state is transient, then after first iteration, the problem reduces to classifying a new state space with the same recurrence classes (since no recurrence class has been removed) but with half the transient states by assumption 2. In other words, the remaining problem is characterized by $G(\frac{|T|}{2}, |R|, K)$. If the trial state is recurrent, on the other hand, the problem reduces to classifying a new state space with $\frac{1}{K}$ fraction fewer transient states by assumption 3 and one fewer recurrent class, which is characterized by $G(\frac{K-1}{K}|T|, |R|, K-1)$. This reasoning leads to the recurrence relation for function G,

$$\begin{aligned} G(|T|, |R|, K) \leq\ & 2(|T| + |R|) + \\ & p \cdot G(\frac{|T|}{2}, |R|, K) + (1-p) \cdot G(\frac{K-1}{K}|T|, |R|, K-1). \end{aligned}$$

The boundary condition for G is $G(0, |R|, K) = 2K|R|$, where all states are recurrent. Using the *substitution method* [37] to do induction, it is easily verified that for all integers $|T|(|T| \geq 0)$, $|R|(|R| \geq 1)$ and $K(K \geq 1)$, we have $G(|T|, |R|, K) \leq c \cdot \max\{4|T|, K|R|\}$ where c is any constant no smaller than 8. Thus, $G(|T|, |R|, K) = O(|S|)$. This means that with the given assumptions, the expected total number of state visits is linear in the size of the state space.

Now, if we further assume that the symbolic technique keeps roughly the same searching efficiency (i.e., number of states visited per second) as the reachability analysis iterates and the state space gets pruned. Then, the time complexity of our algorithm on average has the same order of complexity as that of symbolic reachability analysis that determines the state space of the original Markov chain.

4.4 Experimental Results

We implemented both our method and the one presented in [56] based on the CUDD [7] and SMV [86] packages on a SPARC20 with 256 Megabytes of memory. The efficiencies of the two methods are compared for many circuit models as well as a queueing network model. All of them are specified in SMV format.

4.4.1 Synchronous and asynchronous circuits

Our synchronous examples are the synchronous bus arbiter (*syncarb*) [86], the mutual exclusion element (*mutex*), the counter circuit (*counter*), and a 3-stage synchronized pipeline (*periodic*) which are all from the SMV package. Our asynchronous examples include the distributed mutual exclusion (*dme*) circuit [43, 86], the pausible clock interface (*pci*) [138], the asynchronous differential equation solver [138] with its estimated version (*diffeq_est*) and its back-annotated version (*diffeq_bka*) both of which were analyzed symbolically for their average performance in [126] and finally the asynchronous FIFO circuit *fifo* also discussed in [126].

According to the results in Table 4.1, all the circuits contain a single recurrence class. The synchronous ones have no transient states. Among the asynchronous ones, only the *diffeq* and the *fifo* contain a significant fraction of transient states.

Our method finishes all examples while the previous method fails on half of them due to insufficient memory. Comparing the run time, except on the counter circuit, our method dramatically outperforms the previous method. With more experiments, we notice that for the *fifo* example, the CPU time required by our method increases polynomially while the state space grows exponentially. This is because the BDD size of its transition relation Q grows polynomially with each added stage. Similar performance is achieved in the *syncarb*, *pci* and *dme* examples. The exception of the counters is because their state spaces are complete graphs for which the hashed recursive transitive closure computation is particularly efficient [85].

4.4.2 Closed queueing networks

Finally, we consider a behavioral model of a *closed queueing network* [70]. Each queue is modeled as a single server with finite storage capacity. For simplicity, we assume each server has no input/output buffers. We model the delay of the server with a bounded time interval with a discrete probability distribution. In practice, this delay model is more realistic than the typical models with exponential delays. Figure 4.5(a) shows the overall system structure and Figure 4.5(b) specifies the behavior of a single server as a finite state machine.

Notice that because of its closed nature of the system, once (randomly) initialized, the number of jobs in the system remains constant. Consequently, this example has multiple recurrence classes.

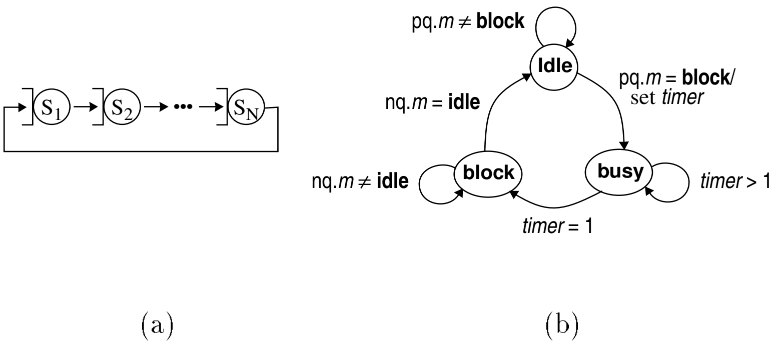


Figure 4.5: A closed queueing network: (a) the system structure, and (b) the behavior of one server.

Initially, a server may or may not have a job to process, i.e., it is randomly initialized as either `idle` or `busy`. The server leaves `idle` mode if there is an incoming job (pq.m = `block`) and becomes `busy`. Meanwhile, a timer (modeled by a `timer` variable) is randomly set to a value from the specified service time interval. Variable `timer` decrements each time step, denoting the residual service time. One time step after the timer reaches 1, the server moves to `block` mode and stays there until the next queue (nq) is `idle`. Then, it returns to `idle` in the subsequent time step.

Suppose each server has a service time anywhere between 1 and 7 steps. The reachable state space grows roughly by a factor of 6.5 with each added queue. As shown in Table 4.1 (cqn's), the number of recurrence classes is $N+1$ where N is the number of queues. In fact, under the given service time assumption, one can show that each recurrence class uniquely determines the long-run system behavior for a fixed number of jobs.

The results show that the run time of our method grows polynomially whereas that of the previous method increases exponentially, implying an exponential speed up. The previous method fails to handle systems with more than 6 queues due to insufficient memory while our method can handle systems with many more queues.

Examples	State space				CPU secs	
	Total states	Reachable states	Transient states	Recur. classes	Previous method	Our method
dme3cells	1.80×10^{16}	6,579	0	1	417	7.1
dme6cells	3.25×10^{32}	8.217×10^{6}	0	1	$8,140^{\dagger}$	123.7
syncarb5	32,768	5,120	0	1	0.63	0.1
syncarb5	1.07×10^{9}	1.049×10^{7}	0	1	2.97	0.28
mutex8	524, 288	2, 048	0	1	0.48	0.18
periodic	6.87×10^{10}	3, 952	0	1	$1,577^{\dagger}$	1.3
counter8	256	256	0	1	0.23	0.18
counter16	65,536	65,536	0	1	0.43	100.9
pci-2-1	4.29×10^{9}	14, 556	0	1	$3,514^{\dagger}$	14.7
pci-4-1	2.88×10^{17}	1.76×10^{7}	0	1	$34,620^{\dagger}$	116.5
pci-8-1	1.30×10^{33}	5.24×10^{13}	0	1	$11h^{\ddagger}$	426.3
diffeq-est	4.29×10^{9}	7, 632	2,762	1	$3,180^{\dagger}$	4.0
diffeq-bka	2.75×10^{11}	11, 036	6,176	1	$1,059^{\dagger}$	6.5
fifo2	65, 536	98	2	1	8.0	0.23
fifo4	1.68×10^{7}	1,150	18	1	72.2	0.58
fifo8	1.10×10^{12}	171,518	730	1	$1,535^{\dagger}$	2.9
fifo16	4.72×10^{21}	4.258×10^{9}	2.48×10^{6}	1	$11h^{\ddagger}$	24.2
fifo32	5.44×10^{39}	7.736×10^{17}	2.88×10^{13}	1	$11h^{\ddagger}$	277.0
cqn2	1,024	21	3	3	0.45	0.1
cqn4	1.05×10^{6}	1,693	15	5	24.7	1.7
cqn6	1.07×10^{9}	80,874	63	7	1,097	14.3
cqn8	1.10×10^{12}	3.083×10^{6}	255	9	8, 138†	60.9
cqn10	1.26×10^{15}	1.328×10^{8}	1,043	11	3, 776†	230.8
cqn12	1.15×10^{18}	5.614×10^{9}	5,535	13	1, 808†	573.0

Table 4.1: Experimental results: † out of memory, ‡ incomplete. In each example, the field of "total states" represents the size of the entire state space (including unreachable portion) spanned by all the state variables. The number of recurrent states is not shown which is the difference between the number of reachable states and that of the transient states.

Chapter 5

State Space Decomposition

5.1 Introduction

The previous chapter describes a BDD-based symbolic algorithm to enumerate all terminal strongly connected components (TSCCs) of the state space. Recall that identifying all TSCC of the reachable state space is an indispensable step in Markovian analysis. In this chapter, I would like to slightly deviate from the scope of Markovian analysis to extend the above symbolic algorithm to identify all (maximal) strongly connected components of a state space.

More generally, the problem addressed in this chapter is to decompose a directed graph (digraphs) into its SCCs, which is a fundamental graph problem [2] and has many important applications in CAD. Generally speaking, SCC decomposition often divides a digraph problem into subproblems, one for each SCC. The solution to the original problem can be constructed by combining the solutions to the subproblems, sometimes with the aid of the component graph (i.e., the structure of connections among SCCs). Using an explicit data structure such as an *adjacency-list* or an *adjacency-matrix* [2], the decomposition of a digraph $G(V, E)$ (with V being the set of its nodes and E the set of its edges) can be solved in linear time (i.e., $O(V + E)$) using an *explicit* depth-first search [120, 2]. However, in many real applications, the size of the digraph can be very large (e.g. with more than 10^{20} nodes), which prohibits the use of explicit methods.

Although targeted for the problem of identifying all TSCCs, the transitive-closure-based method [56] can be easily adapted to compute all SCCs. As mentioned in the previous chapter, the key disadvantage of this method is that it needs the transitive closure of the adjacency-matrix of the digraph which can be expensive to obtain. Nevertheless, it is the only known implicit method for this purpose. The algorithm proposed in this chapter takes advantage of the fact that digraphs resulting from real applications often contain a limited number of SCCs. Similar to the algorithm proposed in the previous chapter where all TSCCs are computed sequentially, it iteratively identifies all SCCs using only reachability analysis. Experiments show that the new algorithm is much faster and solves much larger problems than the transitive-closure-based method adapted from [56], especially when the graph has a small number of SCCs.

The remainder of this chapter is organized as follows. Section 5.2 reviews implicit representation of digraphs using BDDs and the transitive-closure-based method (TC-based method below) for computing all SCCs. Section 5.3 details our reachability-analysis-based method (RA-based method below). The performance of the two methods are compared in Section 5.4. Section 5.5 outlines a potential application of our algorithm to formal verification. The major results in this chapter have been previously summarized in [133].

5.2 Preliminaries and the TC-based method

Let $G(V, E)$ denote a digraph where V is the set of its nodes, and for any two nodes $u, v \in V$, $(u, v) \in E$ iff there is an edge from u to v. A vector $(v_1, v_2, \cdots, v_{n+1})$ of nodes is a *path* of length $n+1$ (from v_1 to v_{n+1}) iff $(v_i, v_{i+1}) \in E$, for all $i = 1, \cdots, n$. Node v is *reachable* from u (denoted by $u \rightsquigarrow v$) if there is a path from u to v. A (maximal) *strongly connected component* (SCC) is a (maximal) subset of nodes where every pair of nodes are reachable from each other. Below, when we say SCCs, we refer to the maximal ones. We denote by $\mathcal{A}(G)$ the set of SCCs in G. For convenience, we call a node a non-SCC node if it does not belong to any SCC, an SCC node otherwise.

Let the nodes be labeled with distinct numbers from $\{1, 2, \cdots, |V|\}$. The digraph can then be represented by an adjacency matrix $M_{|V|\times|V|}$ [2] whose element $M(u, v)$ is 1 if $(u, v) \in E$, and 0 otherwise. The digraph may also be represented by a BDD $N(X, Y)$ by encoding the rows and columns of its adjacency matrix M with two sets of BBD variables X and Y. The BDD $N(X(u), Y(v))$ evaluates to 1 if $M(u, v) = 1$

where $X(u)$ and $Y(v)$ are the vectors encoding the labels of u and v, respectively. Details of BDDs and their common operations can be found in [16]. For convenience, we sometimes use node u to refer to the BDD encoding of its labeling.

The TC-based symbolic method first computes the transitive closure of N, denoted by N^*. By definition, $N^*(u, v) = 1$ iff $u \rightsquigarrow v$. The transpose of N^* denoted by N^{*t} has the property that $N^{*t}(u, v) = 1$ iff $v \rightsquigarrow u$. Thus, u and v belong to a same SCC iff $N^*(u, v) \wedge N^{*t}(u, v) = 1$, where $\wedge$ denotes the BDD AND [16]. Consequently, the union of all SCC states can be computed as $\exists_Y N^*(X, Y) \wedge N^{*t}(X, Y)$, where $\exists$ denotes the BDD *existential quantification* [16]. Finally, the SCC containing a particular node v can be computed as $\exists_Y X(v) \wedge N^*(X, Y) \wedge N^{*t}(X, Y)$.

5.3 The RA-based method

Computing the transitive closure of the adjacency-matrix of digraph as required by the TC-based method is equivalent to computing the reachability set of every node (the set of nodes reachable from the given node). Our method which is based on reachability analysis (the RA-based method below) needs to compute the reachability sets of potentially very few nodes. To better describe our method, we need to introduce a few notations related to the reachability sets and several properties of such sets.

5.3.1 More on forward and backward sets

Recall that the forward set $F(v)$ of a node $v \in V$ is the set of nodes reachable from v. That is, $F(v) = \{u \in V \mid v \rightsquigarrow u\}$. Similarly, its backward set $B(v)$ is the set of nodes that can reach v. That is, $B(v) = \{u \in V \mid u \rightsquigarrow v\}$. One of the properties of $F(v)$ and $B(v)$ is given in Lemma 5, which states that the intersection of the two sets (if not empty) is an SCC. Additionally, v is a non-SCC node if the intersection is empty.

Lemma 5 *[131, 97, 112] If v is a node of $G(V, E)$, then $F(v) \cap B(v) \in \mathcal{A}(G)$.*

A subset $U \subseteq V$ is *SCC-closed* if no SCC intersects with both U and its complement $V \backslash U$. That is, any SCC must be either completely contained in such a set or they do not overlap.

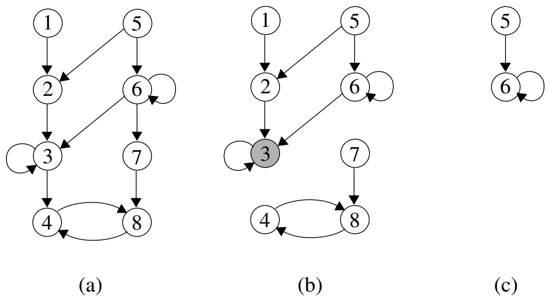


Figure 5.1: A digraph and its decomposition.

Lemma 6 *Both backward and forward sets of any node are SCC-closed.*

Proof Since a diagraph and its (edge-) reversed graph have the same set of SCCs, we just need to show that any backward set is SCC-closed. Consider a node u, and assume its backward set $B(u)$ is not SCC-closed. Thus, there is be a non-empty SCC, say A such that $A \cap B(u) \neq \emptyset$ and $A \backslash B(u) \neq \emptyset$. Let x and y be any nodes of $A \cap B(u)$ and $A \backslash B(u)$, respectively. Since x and y are nodes of A, y must be a node of $B(x)$. Moreover, since x is a node of $B(u)$, $B(x) \subseteq B(u)$ according to Lemma 4. Thus, y is a node of $B(u)$, which means $A \backslash B(u) \subseteq B(u)$. The latter can be true only if $A \subseteq B(u)$ or $A = \emptyset$, which contradicts the assumption. ■

Theorem 8 *The difference of the backward sets of any two nodes is SCC-closed.*

Proof Let u and v be any two nodes. Without less of generality, let us prove the difference $D = B(v) \backslash B(u)$ is SCC-closed. If $D = B(v)$ or $D = \emptyset$, the result is trivial. Now suppose $D \neq B(v)$ and $D \neq \emptyset$, but it is not SCC-closed. Then, there must be an SCC, say A such that $A \cap D \neq \emptyset$ and $A \backslash D \neq \emptyset$. Since $B(v)$ is SCC-closed by Lemma 6, we have $A \subseteq B(v)$, and thus $A \backslash D \subseteq B(v)$. Consequently, $\emptyset \neq A \sim D \subseteq B(v) \backslash D \subseteq B(u)$. Therefore, $A \cap B(u) \neq \emptyset$. Besides, $\emptyset \neq A \cap D = A \cap (B(v) \backslash B(u)) \subseteq A \cap (V \sim B(u)) = A \backslash B(u)$, where V is the set of nodes of the diagraph. Hence, $B(u)$ is not SCC-closed. This contradicts Lemma 6. ■

Remark 1 *By similar arguments, one can show that the difference of any two of $B(u)$, $B(v)$, $F(u)$ and $F(v)$ is SCC-closed, and more generally, that the difference of any two SCC-closed sets is SCC-closed.*

```
SCC_DECOMP(N, V)
    V' ← V;
    while(V' ≠ 0) {
        v ← random_take(V');
        B(v) ← backward_set(v, V', N);
        SCC_DECOMP_RECUR(v, B(v), N);
        V' ← V' ∧ ¬(v ∨ B(v));
    }
```

Figure 5.2: The top-level algorithm.

Example In the digraph shown in Figure 5.1(a), $B(3) = \{1, 2, 3, 5, 6\}$, $F(3) = \{3, 4, 8\}$, and $B(4) = \{1,2,3,4, 5,6,7, 8\}$. From Lemma 5, $B(3) \cap F(3) = \{3\}$ is an SCC. From Lemma 6, $B(3)$ is SCC-closed. Moreover, the difference $B(4) \backslash B(3) = \{4, 7, 8\}$ is also SCC-closed from Theorem 8. □

5.3.2 The algorithm

Lemma 6 and Theorem 8 suggest that a digraph may be first partitioned into backward sets (or differences of backward sets) of some of its nodes, and then the computing of SCCs of the graph can be restricted to each of the partitions. This leads to a divide-and-conquer strategy. For instance, the digraph in Figure 5.1(a) can be partitioned into $B(3)$ and $B(4) \backslash B(3)$, which reduces the problem to finding SCCs within the two subsets independently, as illustrated in Figure 5.1(b).

Because a digraph G and its transpose G^t (the graph obtained by reversing all the edges of G) has the same set of SCCs [2], the above divide-and-conquer strategy may also be carried out in term of forward sets. In fact, the partitioning is also possible to be carried out by interleaving the computation of backward sets and forward sets. However, our experience shows that none of these three different partitioning schemes leads to a significant advantage over the others. For this reason, we focus on explaining the algorithm based on the backward-set partitioning scheme.

At the top level, the algorithm first picks a node from the set of remaining nodes (the entire set V at the beginning), computes its backward set restricted to the remaining nodes, and then decomposes the backward set into SCCs and a set of non-SCC nodes. This process repeats until the remaining set of nodes is empty.

```
FMD_PRED(W, U, N)
    pred←0;
    front←W;
    bound←U;
    while(front ≠ 0) {
        x←∃_Y front(Y) ∧ N(X,Y) ∧ bound(X);
        y←∃_Y bound(Y) ∧ N(X,Y) ∧ bound(X);
        front←x ∧ ȳ;
        pred←pred ∨ front;
        bound←bound ∧ ¬front;
    }
    return pred;
```

Figure 5.3: Computing the predecessors with finite maximum distance.

The procedure is formally described in Figure 5.2. In the algorithm, function `random_take`(V') randomly picks a node from the set V'. Function `backward_set`(v, V', N) returns the backward set of v *restricted to* set V', which can be implemented efficiently using the fixed-point computation [20]. The remainder of this subsection explains the core procedure `SCC_DECOMP_RECUR`(v, $B(v)$, N) which recursively decomposes $B(v)$ into SCCs and a set of non-SCC nodes.

To describe the procedure `SCC_DECOMP_RECUR`, we introduce a concept of predecessors with finite maximum distance, also referred to as *FMD predecessors*. A node u is an FMD predecessor of node v if any path from u to v has finite length. That is to say, any path from u to v must not pass any SCC. A necessary condition for this is that u must be a non-SCC node. The set of FMD predecessors of a set W of nodes is defined to be the set of all FMD predecessors of nodes in W, denoted by $FMD\_PRED(W)$. As an example, in Figure 5.1(b), $FMD\_PRED(\{3\}) = \{1,2\}$. Node 5 is not a FMD predecessors of node 3 because it may pass the SCC $\{6\}$ to reach node 3. Figure 5.3 gives an implicit algorithm that iteratively computes the set $FMD\_PRED(W)$ restricted to a set U.

The recursive procedure `SCC_DECOMP_RECUR`(v, $B(v)$, N) works as follows. It first computes the forward set $F(v)$ of v restricted to $B(v)$. If $F(v)$ is not empty, then it is an SCC due to Lemma 5. Otherwise, node v is a non-SCC node. In either case, $F(v) \vee v$ can be removed from $B(v)$ from further consideration. Next, the FMD predecessors of $F(v) \vee v$ is computed, and are subsequently removed from further

```
SCC_DECOMP_RECUR(v, B(v), N)
    F(v)←forward_set(v, B(v), N);
    if(F(v) ≠ 0)
        report F(v) an SCC;
    else
        report v non-SCC;
    x←F(v) ∨ v;
    R←B(v) ∧ x̄;
    y←FMD_PRED(x, R, N);
    report y non-SCC;
    R←R ∧ ȳ;
    IP←backward_set(y ∨ x, R, N);
    while(R ≠ 0) {
        v←random_take(IP);
        B(v)←backward_set(v, R, N));
        SCC_DECOMP_RECUR(v, B(v));
        R←R ∧ ¬(R ∨ B(v));
        IP←IP ∧ ¬(IP ∨ B(v));
    }
```

Figure 5.4: Recursive decomposition of a backward set.

consideration. Then, from the remaining set of nodes, the procedure computes the set of immediate predecessors (IP) of these already removed nodes. To decompose the remaining set of nodes, a node u is randomly picked from the IP set and its backward set $B(u)$ is computed. The procedure then calls itself to decompose $B(u)$. After it returns, u and $B(u)$ are removed from the remaining set of nodes and the IP set gets updated. If the remaining set of nodes is not empty, another node w is picked up from the updated IP set, and its backward set $B(w)$ is recursively decomposed. This process repeats until the remaining set of nodes is empty. A formal description of the procedure is given in Figure 5.4.

Example In figure 5.1(b), a call to `SCC_DECOMP_RECUR`(3, B(3), N) first computes $F(3, B(3), N) = \{3\}$, and reports $F(3) \cap B(3) = \{3\}$ to be an SCC. Next, it computes $FMD\_PRED(\{3\}) = \{1, 2\}$ which is reported to be a set of non-SCC nodes, and the IP set is computed to be $\{5, 6\}$. At this point, the procedure has reduced the problem to decomposing the digraph in Figure 5.1(c). Suppose node 6 is picked from the IP set, the procedure calls `SCC_DECOMP_RECUR`(6,$B(6)$,N) which, in this case, leaves no more nodes to be decomposed when it returns. □

Example	State space			CPU seconds	
	$\lvert V\rvert$	$\lvert V'\rvert$	SCCs	TC	RA
abp	484	444	11	5.41	0.40
arbit	5,568	5,568	1	797	0.52
bakery	2,886	2,604	31	197	3.24
coherence	94,748	94,688	2	`mo`	41.0
cdnew	186,876	139,520	3,469	`to`	202
eisenberg	1,611	1,609	1	`to`	0.49
emodel	34,133	376	12	to	96.0
scheduler	2.4e+6	2.4e+6	1	`mo`	1.19
minMax30	2.1e+26	2.1e+26	1	`mo`	6.68
tcp	3.9e+22	-	-	`mo`	`to`

Table 5.1: The experimental results where $|V|$ is the number of reachable states, $|V'|$ denotes the number of states belonging to SCCs, `mo` denotes memory out and `to` denotes time out after 1 hour of CPU time.

5.3.3 The complexity

As shown in Figure 5.4, procedure `SCC_DECOMP_RECUR` performs a constant number of reachability analyses before it calls itself. Each reachability analysis must have completed in $O(\mathcal{D})$ BDD operations where $\mathcal{D}$ is the diameter of the graph. Since a call to `SCC_DECOMP_RECUR` removes at least one node from further consideration, `SCC_DECOMP_RECUR`(v,$B(v)$,N) cannot call itself for more than $|B(v)|$ times. Thus, a trivial upper bound on the complexity of our algorithm is $O(|V| \times \mathcal{D})$ in BDD operations.

From our experiments, we observe that a call to `SCC_DECOMP_RECUR` typically removes an SCC before the procedure calls itself. In practice, the algorithm terminates in $O(|\mathcal{A}(G)| \times \mathcal{D})$ time (in BDD operations). Note further that the reachability analysis performed by our algorithm is always restricted to the subset of the nodes that is currently under consideration. In other words, the analysis needs a potentially much smaller number of BDDs operations than the diameter of the graph. In this respect, our complexity analysis above is pessimistic.

5.4 Experiments

We have implemented both TC-based and RA-based methods with `vis-1.3` [48]. In particular, the transitive closure of adjacency-matrix of the graph is computed by iteratively squaring the matrix. This section compares their performance using the examples distributed with the `vis-1.3` package each of which specifies a finite state machine. The methods are used to compute the SCCs in the reachable state space of the finite state machines. The package is run on a Sun Ultra 10 with 640 Mbytes of memory. Table 5.1 gives the experimental results of a representative set of the examples.

Since the TC_method simultaneously computes all SCCs, we report its run time as soon as it computes the transition closure of the adjacency-matrix and ANDs the transitive closure with its transpose. That is, the time to list all the SCCs is not included for in the reported run times of the TC_method.

As shown in the Table, the TC-method solves only a few small examples whereas our RA-based method solves most of them. Even for those solved by the TC-based method, the RA-based method is faster by orders of magnitude. The RA-based method runs out of time in the last example. The reason is that the example contains excessive SCCs and the RA-based method cannot finish reporting all of the them in time, which highlights the limitation of our method.

5.5 Concluding remarks

Computing strongly connected components (SCCs) of a directed graph is a generic graph problem. The proposed BDD-based implicit algorithm iteratively applies reachability analysis and computes SCCs sequentially. It has been used to compute the SCCs in the reachable state space of a set of finite state machines. Compared with an existing symbolic method which requires the transitive closure of the adjacency-matrix of the graph, our method is shown to be much faster, requires much less memory, and consequently solves much larger problems. Nevertheless, when there are excessive SCCs, our approach may not complete in a reasonable amount of time. Consequently, it might be interesting to explore a hybrid approach between our RA-based method and the TC-based method, which may be able to find multiple SCCs of relatively small size simultaneously but without computing the transitive closure of the entire graph.

As a potential application, the proposed method may be adapted to the *bad-cycle-detection* problem in formal verification [47, 58]. This will be discussed more in Chapter 10.

Chapter 6

Accelerating Markovian Analysis: State Compression

6.1 Introduction

The state-explosion problem brings computational challenge to several steps of analysis of Markov chain models of asynchronous systems. Chapter 4 has solved to a large extend the state classification step. However, in most cases, it is the subsequent computation of stationary distribution of the Markov chain that forms the bottleneck of the entire analysis. This chapter is devoted to attacking this computational bottleneck.

The computation of the stationary distribution of a Markov chain is equivalent to solving a linear system with as many unknowns as the number of states in the chain. The generally preferred technique is the well known *Power method* (see e.g., [117]) which iterates from some initial distribution (or initial guess) until it converges according to some user-specified accuracy requirement. Since the Markov chains resulting from real designs can possess huge state spaces, e.g, with hundreds of thousands of states, the Power method can take intolerably long before getting a good approximation of the stationary distribution. A promising way to speed up this phase of the analysis is to use *symbolic techniques* [126, 56]. Although it is then possible to handle some systems with hundreds of signals, it can still take a long time to complete.

There are many proposed techniques to speed up iterative approaches such as Power method for large linear systems. Examples are *pre-conditioning* [96] and *orthogonal multi-vector iteration* [116]. It is unclear whether these sophisticated techniques are likely to give a significant improvements, however, because they impose significant overhead. In addition, there is an intensive ongoing research trend in probability theory on the *lumpability* of Markov chains [1, 104]. 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 Markov property. A necessary and sufficient condition for this to be possible is all states within a subset has 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 system to satisfy this condition and get a much smaller Markov chain [60]. Therefore, this approach cannot be considered as a general method to reduce the computational complexity.

The method presented in this chapter 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 [28, 123]). Using time-discretization to model these systems as Markov chains leads to state spaces that have *limited feedback*. More precisely, let the (reachable) state space S be partitioned into two subsets, B and $\overline{B}$, such that any two consecutive occurrences of any state in $\overline{B}$ is always interleaved by at least one occurrence of some state in B. In graph terminology, B is a *feedback vertex set* (FVS) of states. It will be shown 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, the chapter uses 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 the our method and is called *state compression*. Next, the stationary distribution of the new Markov chain is computed by the Power method or even by direct (non-iterative) methods if the new Markov chain is sufficiently small. Finally, the stationary distribution of the original Markov chain is expanded from that of the new Markov chain just computed. This third step is called *expansion*. All these steps can be implemented using symbolic techniques. Figure 6.1 depicts an overview of our method.

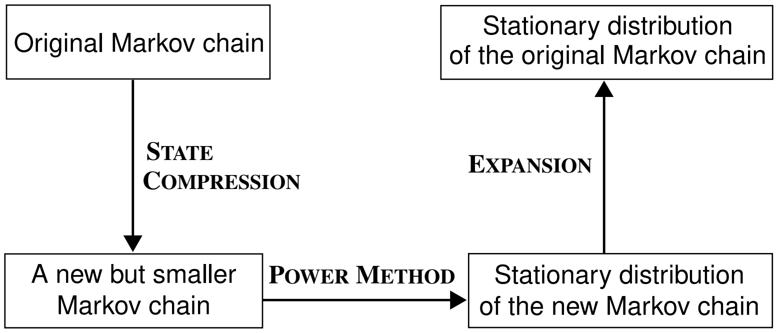


Figure 6.1: Block diagram of our method.

It is desirable to find a minimum FVS in order to make the state space of the new Markov chain small. However, this is a well-known *NP*-complete problem [50]. Many heuristic algorithms to find a *minimal FVS* [113, 29, 94] exist, but they are all too expensive for large graphs. Two simple heuristics are given to symbolically search for a good FVS which is not necessarily the minimum.

The significant contributions of this approach are two folds. 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, this method can analyze larger systems than is possible using the method in [126].

The chapter also briefly discusses the potential application of this work to other application areas that use Markovian analysis. For example, Markovian analysis is heavily used in the performance analysis and scheduling of networks and distributed systems (e.g., [70, 63, 30]). As observed in our application, the state-of-the-art is still limited by the exploding computational complexity. The chapter will note the different characteristics of these Markov chains and identify the conditions under which the approach presented in this chapter may be helpful to their analysis.

Section 6.2 identifies characteristics of asynchronous systems that lead to models with a small FVS and thus would most benefit from our proposed approach. Section 6.3 gives a detailed description of the proposed method as well as the theorems that have been proved to check the correctness of the method. Section 6.4 describes several heuristic algorithms to find a good feedback set to form the state space of the new Markov chain. Implementation issues are noted in Section 6.5. experimental results are given in Section 6.6. The chapter is concluded in Section 6.7.

6.2 Feedback vertex sets in the derived Markov chains

For a detailed discussion on the derivation of Markov chains modeling asynchronous circuits and systems, see Chapter 2. This subsection analyzes 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. These characteristics will be contrasted with those of the 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* [53] (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 a unit time step.

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. However, system components mostly have bounded firing times. Even for those components that have unbounded firing times, like arbiters, the self-loop only occurs in the small fraction of the states in which the unbounded event is enabled. Moreover, to have a self-loop, the unbounded event cannot be concurrently enabled with a transition that has bounded delay, further reducing the number of self-loop states. Another notable self-loop state represents a system that is internally idle (no internal event is enabled) but the environment is experiencing a geometrically distributed delay. Since both of these situations typically represent a small fraction of the state space, the fraction of self-loop states is usually small.

In contrast, Markov chains modeling synchronous circuits usually have a large fraction of self-loop states (e.g., see [78, 56]). The reason is that Markovian analysis of synchronous circuits targets power rather than performance analysis. Thus, their target is the long-term probability of state transitions and a state in their models 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 states without a self-loop.

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. Experiments show that Markov chains of asynchronous systems with mostly components having bounded delays have a small FVS. There are at least two reasons as described below.

The first reason is that the state space of the resulting Markov chain usually has few states with self-loops as just discussed. This may be considered as a necessarily condition for a small FVS to exist since every state that has a self-loop must be in any FVS, The second reason is that the state space tends to be spare in the sense it contains many directed acyclic graphs (DAG) like structures. Note that if the root state of a DAG is in a FVS, all other states of that DAG need not be in the FVS. There are two sub-reasons that lead to the existence of these DAG-like structures in our Markov chain model.

- The first reason is again related to time discretization. For each system event that has a bounded delay, a dedicated integer-valued timer is used to denote the corresponding passage time (cf. Section 2.1.2). Suppose such an event e with its delay upper bounded by n time steps 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, consider the STG of the abstract adder model in Figure 2.2. Since `Req` may make a positive transition when the model is in state (00), event `Ack`↑ can be enabled in state (00). Moreover, `Ack`↑ can take up to 3 time steps to occur once it is enabled. Thus, there must be a enabled DAG of event `Ack`↑ rooted at state (00) which has a depth of 3. Indeed, states (00), (02) and (01) correspond to this a DAG which has a depth of 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 event that can be enabled in a state within the enabled DAG of event e, this depth is not decreased either. In contrast, the enabled DAG of these events combined with the enabled DAG of event e may form a larger DAG that contain more states.

  Consequently, there tends to be many DAG-like structures in the state space which may lead to a small FVS since a FVS need not contain any state of an enabled DAG except possibly its root.

- Secondly, 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* [136] and *pausible clock interface* *circuits* [138]. 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 exist 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 load applied to the micropipeline gets reduced. All these examples will be discussed more in Section 8.6.

The existence of small FVS's of asynchronous systems having many bounded delays serves 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. As argued above, models with many discretized bounded delays or fixed delays will have a small fraction of self-loop states. In addition, the size of these delays impacts the size of the DAG-like structures within the state space. 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.

6.3 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 an 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 is to obtain the stationary distribution of the reduced Markov chain using the traditional power method. The third step, *called expansion* mathematically relates the stationary distribution of the reduced Markov chain to that of the original Markov chain.

More specifically, let $M = (S, P)$ be an irreducible Markov chain with large state space S. For our state compression step, we restrict M to an arbitrary subset of states S' ($S' \subset S$), and define a new Markov chain $M' = (S', P')$. It will be clear that if S' is a FVS of states, P' can be defined without matrix inversion, thereby simplifying the algorithmic implementation, and motivating our approach.

The stationary distribution of M' can be computed using the traditional Power method. Note that because M' may be much smaller than M, the implementation of this step may be much less computationally expensive than applying the power method to M. Finally, the expansion step mathematically relates the stationary distributions of the two Markov chains.

The following three subsections describe the mathematics behind each of these three steps in order. Symbolic algorithms for their implementation are described in Sections 6.4 and 6.5.

6.3.1 State compression

The first part of this section describes the construction of a new Markov chain M' with a (hopefully) smaller state space S' whose stationary distribution has an important but simple relationship to part of the stationary distribution of M. The remainder of this section shows that by selecting S' to be a FVS significantly reduces the overhead for the construction and subsequent analysis of M'.

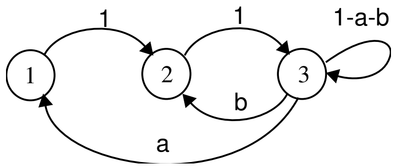
6.3.1.1 Constructing a new Markov chain

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

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

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

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



(a) STG

(b) Transition matrix P

Figure 6.2: A 3-state Markov chain.

Example (Decomposition of P and π) The transition matrix of a Markov chain is usually depicted as a *labeled state transition graph (STG)* with vertices denoting the states and labels on the edges denoting the 1-step transition probabilities. Figure 6.2 shows the *STG* and the transition matrix of a Markov chain with 3 states.

Assume S' contains the first two states of S, i.e., $S' = \{1, 2\}$, and of course, $\overline{S'} = \{3\}$. Then, P and π are accordingly decomposed as follows.

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

□

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

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

To construct the compressed Markov chain with the above-mentioned simple scaling property, one possible way is to construct a new random process X that keeps track of all time instances when the original Markov chain M is in a state of S' and skip all other time instances when M is not in any state of S'. Note that the states of S' keep the same relative occurrence frequency in process X as they have in Markov chain M. This idea may be best described by a small example.

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

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

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

$$\begin{array}{rccccccccc} X_n: & 1 & 2 & & & 2 & & 1 & 2 & \cdots \\ \text{time } n: & 0 & 1 & & & 2 & & 3 & 4 & \cdots \end{array}$$

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

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

$$X = \{X_n = M_{\tau(n)} : n \geq 0\} \tag{6.1}$$

where function τ is defined recursively:

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

Theorem 9 *If M is a time-homogeneous irreducible Markov chain, then process X defined by (6.1) 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 $\tau(n)$ in the sequence is a stopping time of M. Thus, we have

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

Hence, X is Markovian.

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

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

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

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

Corollary 2 *$\pi_{S'} = \mu\pi'$, 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 $n \leq 0$. The stationary probability of state i, i.e, π', equals the proportion of time M' has spent in state i in the limit. That is, $\pi' = \lim_{n\to\infty} \frac{1}{n} N_i'(n)$. Similarly for M, we have $\pi = \lim_{n\to\infty} \frac{1}{n} N_i(n)$. Thus, for any given states $i, j \in S'$, we have

$$\frac{\pi_i'}{\pi_j'} = \lim_{n\to\infty} \frac{N_i'(n)}{N_j'(n)} = \lim_{n\to\infty} \frac{N_i(\tau(n))}{N_j(\tau(n))} = \lim_{m\to\infty} \frac{N_i(m)}{N_j(m)} = \frac{\pi_i}{\pi_j}. \tag{6.2}$$

Moreover, by Theorem 9, $i, j \in S'$ are recurrent, meaning $\pi_i', \pi_j' > 0$. Then, the claim of the corollary is equivalent to (6.2). ■

From Corollary 2, we see that that expansion for $\pi_{S'}$ from π' can be done by a simple scaling as expected. The determination of the constant μ will be discussed later in this section.

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

Theorem 10 *Markov chain M' has a transition matrix*

$$P' = P_{S'} + P_{S'\overline{S'}}(I - P_{\overline{S'}})^{-1} P_{\overline{S'}S'}. \tag{6.3}$$

The proof of Theorem 10 relies on following lemma.

Lemma 7 *Let $M = (S, P)$ be a time-homogeneous irreducible Markov chain. Let $\overline{S'}$ is 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 not leave subset $\overline{S'}$ and after $k(k \geq 0)$ time steps it is in state j is:*

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

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

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

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

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. ■

Proof of Theorem 10 From Lemma 7, $P^k_{\overline{S'}}\to 0$ 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

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

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

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

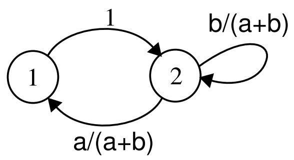
$\sum_{u,v \in \overline{S'}} Pr(M_1 = j \mid M_0 = v) \cdot$
$\sum_{k=0}^{\infty} Pr(M_k = v, \text{ if } k > 1, (M_{k-1}, \cdots, M_1) \in \overline{S'} \mid M_0 = u) Pr(M_1 = u \in S' \mid M_0 = i)$
(exchange summations, and apply time-homogeneous property of M)
$= (P_{S'})_{ij} + (P_{S'\overline{S'}}(\sum_{k=0}^{\infty} P_{\overline{S'}}^{k}) P_{\overline{S'}S'})_{ij}$ (Lemma 7)

■

Example (State comporession) Continuing with the previous example, Equation (6.1) defines a new Markov chain $M' = (S', P')$ where $S' = \{1, 2\}$ and its transition matrix P' is determined by (6.3). That is,

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

Figure 6.3 depicts the labeled STG for M'. Moreover, any direct method (or the Power method) can be used 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 $\mu = \frac{2a+b}{1+2a+b}$. □



(a) STG (b) transition matrix P'

Figure 6.3: The compressed Markov chain M'.

6.3.1.2 Finding a proper S'

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

However, when determining the transition matrix for M' using Theorem 10, 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, in the proof of Theorem 10, it is shown $\lim_{k\to\infty} P^k_{\overline{S'}} = 0$. This implies that matrix $I - P_{\overline{S'}}$ is invertible and can be determined as:

$$(I - P_{\overline{S'}})^{-1} = I + P_{\overline{S'}} + P^2_{\overline{S'}} + \cdots \tag{6.4}$$

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

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

$$(I - P_{\overline{S'}})^{-1} = I + P_{\overline{S'}} + P^2_{\overline{S'}} + \cdots + P^{K-1}_{\overline{S'}} \tag{6.5}$$

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

In fact, if S' is chosen in such away that there is no cycle in $\overline{S'}$, then, $(P_{\overline{S'}})^k = \mathbf{0}$ for all $k \geq K$ where K is the length of the longest path in $\overline{S'}$. Consequently, the inversion of $I - P_{\overline{S'}}$ can be determined by (6.5) without an error. This is a key observation of this chapter. Moreover, matrix $P_{\overline{S'}}$ is typically spare so that the multiplications and summations involved in (6.5) 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 in graph terminology where a FVS is a subset of vertices of the graph and by removing all the edges connected with the vertices in this subset, the graph becomes acyclic.

Example (Choosing S' as a 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. There are several reasons, however, which suggest that we should not search for the minimum FVS to be S'. First, finding the minimum FVS is *NP*-complete [50]. Secondly, as we will further explain, there is a trade-off associated with the total analysis time between the size of S' and $\overline{S'}$. Therefore, we propose simple heuristics to efficiently find a good FVS for S' in Section 4.

6.3.2 Computing π'

Once the compressed Markov chain M' is constructed, its stationary distribution π' can be computed iteratively using the Power method described in Sections 3.2.1 and 3.2.2.

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

As we know, the convergence rate relates to the distribution of the eigenvalues of the transition matrix (or that of the d-the power of the transition matrix if the Markov chain is periodic). Generally, it is the magnitude of the subdominant eigenvalue, λ_{d+1} of matrix P^d, that determines the convergence rate if the chain has a periodic $d \geq 1$. 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 several real examples suggest this rarely happens. One possible explanation to this may be stated as follows. Any given Markov chain $M = (S, P)$ may be considered as a random sample from the family of all possible Markov chains with the same state space as that of M. Consequently, one may assume the eigenvalues of P are uniformly distributed in the interval $[0, 1]$ except the largest eigenvalue 1. It may be further assumed that the eigenvalues of P', the transition matrix after state compression, are also uniformly distributed in the interval $[0, 1]$. Then, it may be deduced 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.

6.3.3 Expanding π' to π

The last step in the analysis is to expand the stationary distribution π' of M' to π of M. Corollary 2 relates π' to $\pi_{S'}$. The following theorem further relates π' to $\pi_{\overline{S'}}$.

Theorem 11 *If π and π' are the stationary distributions of M and M', respectively, then*

$$\pi_{\overline{S'}} = \mu \pi' P_{S'\overline{S'}} P^*_{\overline{S'}} \tag{6.6}$$

where, μ is the normalization factor which is the same as in Corollary 2 and is equal to:

$$\mu = \frac{1}{1 + \pi' P_{S'\overline{S'}} P^*_{\overline{S'}} \mathbf{1}^T} \tag{6.7}$$

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

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

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

But $(I - P_{\overline{S'}})$ is invertible and it equals $P^*_{\overline{S'}}$, 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 2.

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

Theorem 11 combined with Corollary 2 enables us to expand the stationary distribution of M' to that of M. Recall that $P^*_{\overline{S'}}$ in (6.6) is the reflective transitive closure of $P_{\overline{S'}}$ (defined by 6.4) 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 (6.3), we have $M' = (1)$ and thus $\pi' = (1)$.

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

$$\begin{cases} \pi_{S'} = (\pi_3) = \mu(1) \quad \text{(by Corollary 2)} \\ \begin{aligned} \pi_{\overline{S'}} &= (\pi_1, \pi_2) = \mu(1) \begin{pmatrix} a & b \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \\ &= \mu \begin{pmatrix} a & a+b \end{pmatrix} \text{(by Theorem 11).} \end{aligned} \end{cases}$$

Following (6.7), we compute $\mu = (1 + 2a + b)^{-1}$. □

In fact, the value for μ is typically not important. What is important is the *relative values* computed for all the states in S by the expansion step. These relative values represent the *relative frequencies* of the corresponding states to be visited by the original Markov chain M in the long run which is generally sufficient for further analysis of performance metrics (as well as power estimation [56]).

6.4 Heuristics for state compression

This section describes several 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 6.5

Let us assume an irreducible Markov chain M has a state space S and a transition matrix P. The following definitions are needed to describe the heuristics.

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

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

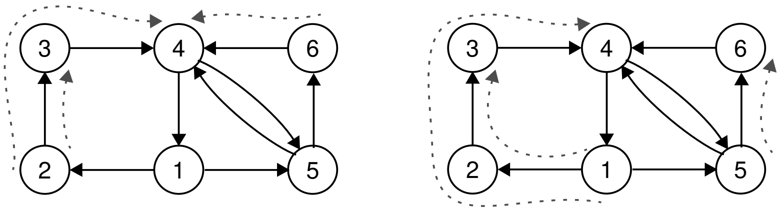
Note that a non-decision state can reach other states only through its sole next state. Similarly, a non-merging state can be reached only through its sole previous state.

Definition 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 Figure 6.4. By definition, every single state is both a forward-string and a reverse-string. One can list several forward-strings of length more than 1. Examples are $(2, 3)$, $(2, 3, 4)$ and $(6, 4)$. The latter two are also maximal. Example reverse-strings are $(1, 2, 3)$, $(1, 2, 3, 4)$, and $(5, 6)$. The latter two are also maximal. □

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



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

Figure 6.4: Forward- and reverse-string examples.

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.

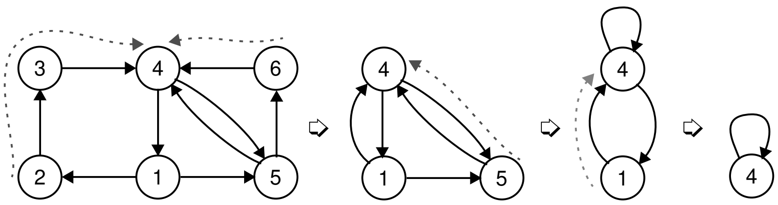


Figure 6.5: Forward-string-based iterative state compression.

6.4.1 Forward-string-based compression

Let us first give two simple lemmas.

Lemma 8 *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 Suppose σ is a maximal forward-string of M. Then, M has at least one decision state which is the tail of σ.

Next, the length of σ must be bounded by $|S|$ because otherwise there is a state 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 where every state should be expected to be visited infinitely often. This completes the proof. ■

Lemma 9 *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, $S_2 = (S_1 - \{s\}) \cup \{tail\ of\ \sigma\}$.*

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

Lemmas 8 and 9 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 following theorem.

Theorem 12 *If S has decision states, there is a FVS containing only decision states. Otherwise, S forms a loop, and hence any single state is a FVS.*

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 8, all maximal forward-strings in S (including those containing s) have a bounded length with a decision tail state. The proof concludes immediately with Lemma 9. ■

Our first heuristic for state compression is thus to search for all the decision states of S. If there is no decision state, any single state of S forms a FVS which must be the smallest. Otherwise, we select all the decision states as a FVS. The heuristics can be applied iteratively. That is, each time the heuristic is applied, a new Markov chain is created. Since a decision state in the original Markov chain may become a non-decision state in the new Markov chain, the heuristic may be applied again to get an even smaller Markov chain. This process continues until the heuristic fails to compress the Markov chain any further. Figure 6.5 illustrates this idea.

The corresponding algorithm is shown in Figure 6.6. The iterative procedure **String_Compress** with the *direction* parameter set to key word *FORWARD* performs the iterative compression. It returns immediately if all states currently left are

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**
        Let S' := the set of decision states in S;
    else
        Let 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 (6.3) and (6.5);
    return **String_Compress**(S', P', D);
end

Figure 6.6: State compression based on forward-string concept.

decision states. If not, it checks whether they are all non-decision states (forming a loop) in which case it takes an arbitrary state as a FVS and returns a Markov chain containing only this state. Otherwise, it takes all the decision states as a FVS, creates a new Markov chain with the FVS as its state space and contructs its transition matrix. Then the procedure calls itself to try compression on the new Markov chain. As we will further note in Section 6.5, the probability transition matrix only needs to be computed for the Markov chain built on the final FVS. Boolean versions of their transition matrices will suffice for all intermediate Markov chains. However, in the above algorithm, the probability transition matrix is computed for intermediate chain to ease the exposition.

6.4.2 Further compression based on reverse-string

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, the forward-string-based heuristics compresses forward-strings down to their tail states. This same idea can be applied on the reverse-strings. That is, we may compress all reverse-strings up to their head states. These two compression ideas can be combined.

To avoid the 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*. Figure 6.7 shows the algorithm that adopts both forward-string-based and reverse-string-based compressions.

```
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
```

Figure 6.7: State compression based on both forward-string and reverse-string concepts.

The compressed Markov chain returned by Algorithm 2 will be no larger than that returned by Algorithm 1. One fact worth noticing, however, is during each of its iteration, states in $\overline{S'}$, i.e., those not in the compressed Markov chain M', are all non-decision states with respect to the current Markov chain. This means that the sub-matrix, $P_{\overline{S'}}$, contains only 0's and 1's and hence is a Boolean matrix. This will speed up the computation of $P_{\overline{S'}}$ when implemented using symbolic techniques (as will be explained in the following section). Thus, in some cases Algorithm 1 can result in less overall CPU time than Algorithm 2 even though it involves analyzing a larger Markov chain.

6.5 Implementation

The method discussed in the previous sections has been embedded in a modified version of the symbolic performance analysis tool described in [126]. The method serves as an alternative for the computation of stationary distribution of Markov chain models in contrast to the standard Power method previously adopted by the tool. In particular, we make note some implementation details for each of the three steps of the 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 [130, 131]. Specifically, all transient states (if any) are symbolically identified and excluded from further analysis. If there are multiple recurrent classes, the tool also performs a *transient analysis* 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 recurrent classes.

Second, we note some technical details related to improving the speed of the state compression step. Rather than computing the probabilistic transition matrix for a new Markov chain during each iteration of state compression, we compute its Boolean transition matrix which is sufficient for further compression in the sense that it is guaranteed to find the same 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 (6.3) and (6.5). In addition, when computing P', we do not compute the reflective probabilistic transitive closure $P'^*_{\overline{S'}}$ independently. Instead, we first left multiply $P_{\overline{S'}}$ by the quantity $P_{S'\overline{S'}}$ and then iteratively compute for $P'^*_{\overline{S'}}$. This is because the

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

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

where $\overline{\pi}^{(n)}$ is the average of subsequence of $\{\pi^{(i)} : i \geq 0\}$ of length d as defined by (3.3).

There are other convergence criteria recommended in the literature. For instance, one might take the relative supremum norm [117] 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 applications one has [62]. 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 [101]. The difference is a constant factor. This is true for all other usually referenced norms as well. In our applications which target the average metrics of performance or power (i.e., emphasizing the stationary probabilities of subsets of the state space rather than those of individual states), we observe that the criterion based on the norm 1 measure is generally sufficient.

Finally, the expansion step symbolically implements the two equations given in Corollary 2 and Theorem 11. This step is usually computationally cheap since it involves only two vector and matrix multiplication as suggested by (6.6). Recall that the probabilistic transitive closure $P^*_{\overline{S'}}$ needed in (6.6) has been computed during state compression. The constant μ in (6.6), as pointed out earlier, need not to be computed for performance analysis purpose.

6.6 Experimental results

The state-compression algorithm has been evaluated on a number of Markov chains models of asynchronous systems derived using the abstract and time-discretization techniques described in Section 6.2.

6.6.1 The FIFO

The first experiments is a FIFO with 6 stages and trivial environments as shown in Figure 6.8. 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 $(1'0.1, 2'0.1, 3'0.8)$, i.e., with probability 0.1, 0.1 and 0.8, it has a delay of 1 , 2 and 3 units, respectively. The environment at the right side acknowledges requests with a fixed delay of 2 units upon requested. The environment at left side (LSE hereafter) requests with a fixed delay of $\delta \geq 1$ units upon acknowledged.

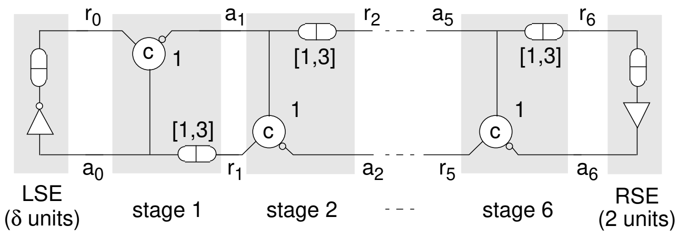


Figure 6.8: A 6-stage FIFO with simple environments.

Our experiment varies the delay of the LSE from 1 unit to 10 units. If we think of an empty stage (free of data) as a bubble, the average number of non-empty FIFO stages at any time instance decreases with an increase in the LSE delay. This suggests that the FIFO behaves more "sequentially" as the LSE gets slower, implying a sparser state space with a smaller feedback set. In the extreme case (not shown), when the LSE is very slow, only one 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 Figure 6.9 we see a trend of increasing state compression ratio as the delay of the LSE increases.

Note, however, that the overall state compression ratio in this example 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.

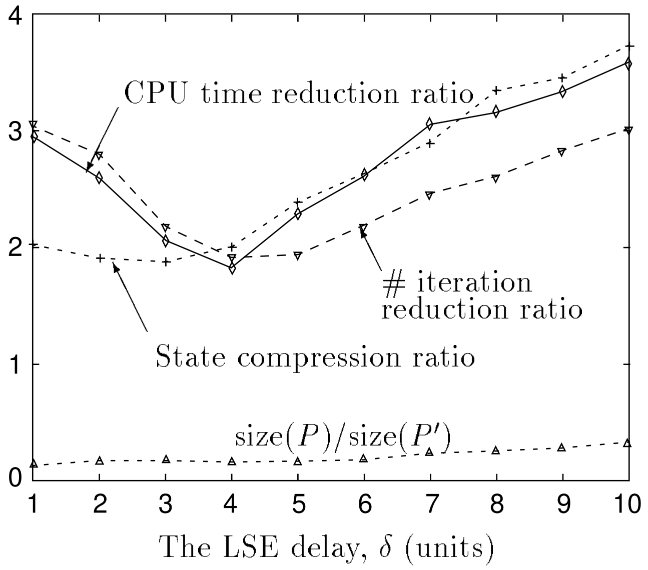


Figure 6.9: Speedup in FIFO analysis.

Figure 6.9 also plots the speedup in stationary analysis using the state-compression-based method over the standard Power method. The CPU time reduction ratio closely tracks with the state compression ratio when the LSE has a relatively large delay (over 4 units). When the LSE is relatively fast (with a delay less than 4 units), the CPU time reduction ratio tracks more closely with the number of iterations needed for convergence. Another interesting fact is the ratio between the sizes (in number of ADD [7] nodes) of the transition matrices P and P'. Clearly, the ADD for P' is significantly more complex than the ADD for P. However, we see that the larger size of P' does not prevent the state-compression-based from achieving a significant reduction in overall CPU time. On the whole, we observe a speedup of close to or over a factor of 2. The model with the LSE set to 4 units requires the longest CPU time to solve (slightly over 15 minutes on a Sparc 20 with 256M of memory). In all cases, the time for state compression and expansion are negligible.

6.6.2 The Differential Equation Solver

The second example which has been analyzed for performance metrics in [126] is the behavioral model of the differential equation solver [136]. As in [126], we consider its estimated and back-annotated versions. The latter has more accurate specifications that account for some wire delays.

Table 6.1 gives the sizes of their reachable state space and the set of recurrent states. The back-annotated version achieves over three times the state compression ratio of the estimated version which is much higher than that of the FIFOs. This is largely because the modules in the DIFFEQs are *loosely coupled* and rather sequential. Due to the state compression, the number of power iterations to convergence is dramatically reduced (cf. Table 6.2). The curves in Figure 6.10 illustrate convergence of the distance of the probability vectors from two consecutive iterations. They clearly suggest that the Markov chains in both estimated and back-annotated versions possess a much higher convergence rate after state compression.

Examples	Number states			compression ratio
	reachable	recurrent	after compression	
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 6.1: State compression in the DIFFEQ and PCI analyses.

Table 6.3 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 requires a larger portion of the CPU time for state compression mainly because their structures allow for more state compression than for the FIFO models. It may be conjectured that the larger state compression ratio can be attributed to the fact that DIFFEQ solver's circuit structure has relatively fewer and larger cycles than the FIFO.

Examples	Number of Power iterations			Size of transition matrix in number of ADD nodes		
	w/o comp.	w/ comp.	reduction ratio	w/o comp.	w/ comp.	ration
DiffeqEst.	247	39	7.5	3,384	3,258	1.04
DiffeqBka.	210	14	15	4,456	3,323	1.34
PCI2to1	$> 4,287$	35	> 122	8,915	10,040	0.89

Table 6.2: Reduction on iteration numbers in the DIFFEQ and PCI analyses.

Example	CPU time w/o comp. (seconds)	CPU time w/ comp. (seconds)				Speedup
	Power iter.	Comp.	Power iter.	Expan.	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/incomplete	84.4	50.0	2.6	137	> 315

Table 6.3: Speedup in the DIFFEQ and PCI analyses using state compression.

6.6.3 A pausible clocking interface

Figure 6.11 shows two asynchronous modules (the senders) communicating with a synchronous module (the receiver) through a variation of a pausible clocking interface (PCI) [138] 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 MEs and an AND gate). It continues to run freely after receiving requests from the senders if there is a clear difference between the arrival time of the requests and that of the clock edge. However, if this time difference is significantly small so that synchronization failure may occur, either the clock is *paused* by stretching its falling edge or the corresponding AFSM delays its request out (a transition of signal R_{xp}). 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.

The following timing assumptions are made 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 acknowledgment is $0.1^{n-1} * 0.9$. Similarly, the delay at the receiver side is also geometrically distributed with a parameter of 0.9. The mutual exclusion element *mutex* has a unit delay and is assumed to be fair with simultaneously arriving requests. Table 6.1 lists the sizes of the reachable state space, the recurrent state set and the state space after compression. The model achieves a state compression ratio close to 6. A slightly closer study of the PCI behavior suggests that any cycle in its state space represents at least one complete clock cycle as generated by the ring oscillator, a possible explanation for the PCI to have a high state compression ratio.

Comparing the overall CPU times required to compute the stationary distribution, the standard Power method fails to reach the convergence after 12 hours of CPU time whereas the method proposed in this chapter using state-compression converges within 3 minutes yielding a speedup of over two orders of magnitude. In particular, in the state-compression-based method, time spent in expansion is again negligible, but the time spent in state compression takes over half of the total CPU time as in the back-annotated version of the DIFFEQ model.

6.6.4 Synchronized processes

The example is a set of concurrently running processes synchronized at some special points. Figure 6.12 shows a simple configuration of $n + 1$ ($n > 1$) processes from $\mathtt{p_1}$ to $\mathtt{p_{n+1}}$ each depicted by an inverter. All processes excepted $\mathtt{p_{n+1}}$ 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 $\texttt{C}_2$), or the average waiting (idle) time for each of the processes. One might then perform a Markovian analysis in order to obtain these measures [126].

The state space of this simple configuration, unfortunately, increases roughly by a factor of 4 with each added process. Taking $n = 24$ processes, we face a state space of over 10^{15} states, excluding the possibility to use any explicit approaches on currently available machines. However, due to the symmetry involved in the configuration, we can successfully employ the symbolic technique [126] to obtain the stationary probability of each of the state using standard Power method with memory usage less than 30 Mbytes. Unfortunately, it takes over 14 hours to complete on a Sparc20 assuming the Power method starts with the initial distribution where each state is equiprobable.

By the technique proposed in this chapter, the state space can be compressed down to a single state for every possible configuration in Figure 6.12 (i.e., for every possible value of m). In fact, that single state is always one of the two global synchronization states where all processes are idle and element $\texttt{C}_2$ 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 $\texttt{C}_1$). 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 $\texttt{C}_2$), which implies the system has a small feedback set in its state space.

The Power method on the compressed chain converges immediately. It turns out that the compression step takes most of the run times. For n = 24, it takes slightly over 20 minutes at most when $m = 1$, and less than 10 minutes when $m = 24$. The time for expansion is negligible. Thus, one achieves a reduction of over a factor of 40 in CPU time. Figure 6.13(a) shows that the overall run time is a small polynomial function of the system size while the number of states grows exponentially.

Figure 6.13 plots the average cycle time of the entire procedure computed as a function of m. Further, one can show in this example that the sum of the average waiting time and the busy time of each process equals the average cycle of the system. Since the average busy times are the same for all processes (i.e., 2.5 units), their average waiting times are also the same which is 2.5 units less than the average cycle time for a fixed m.

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

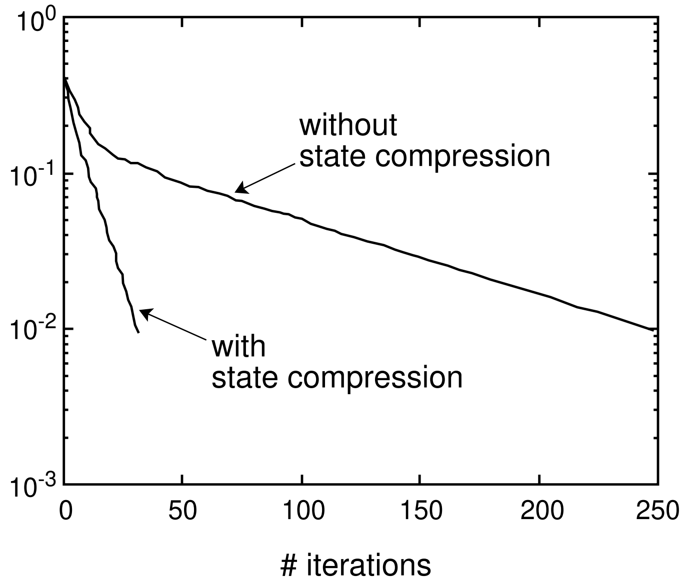
6.7 Concluding remarks

Markov chain models currently formalized from specifications of asynchronous systems require intensive CPU time to find their stationary distributions. The present chapter proposes 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. It is further proposed the 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. Experiments show that many asynchronous systems possess a FVS that occupy a small portion of their reachable state spaces. As a consequence, the proposed 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.

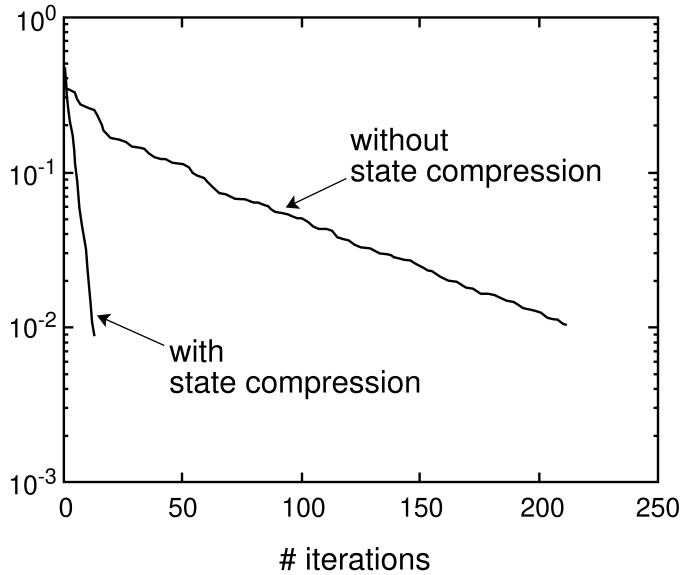
The application of the proposed technique may extend beyond the analysis of asynchronous systems. It has been argued that analyzing the time behavior of systems with mostly bounded delays will require a time discretization-based modeling approach that will typically lead to a Markov chain with a small 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 by our approach. This suggests that the approach presented in this chapter may have applicability in analysis of other types of systems modeled by discrete stochastic Petri nets and generalized timed Petri nets which has 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. In addition, it may be possible to leverage off of recent work in partial-order-based formal verification [74] to find a good FVS from component specifications.

It might also be interesting future work to try to construct a small Markov chain that is sufficient for targeted analysis without performing complete reachability analysis. Alternatively, it might be possible to directly obtain the reduced Markov chain defined over a FVS without ever examining the entire state space. 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. In Part II of this thesis, we present statistical techniques that avoid these expensive state-level analyses, and thus solve much larger systems.



(a) The estimated version.



(b) The backannotated version.

Figure 6.10: Convergence of $\|x^{(n)} - x^{(n-1)}\|$ in the DIFFEQ model.

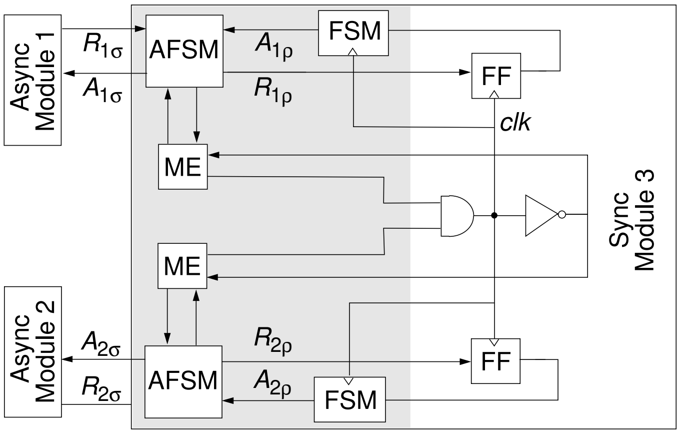


Figure 6.11: A pausible clocking interface.

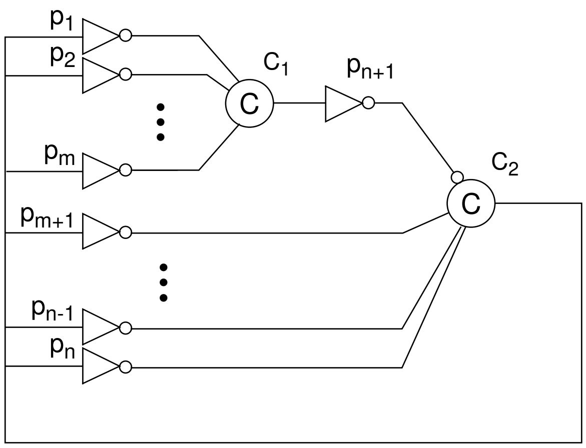
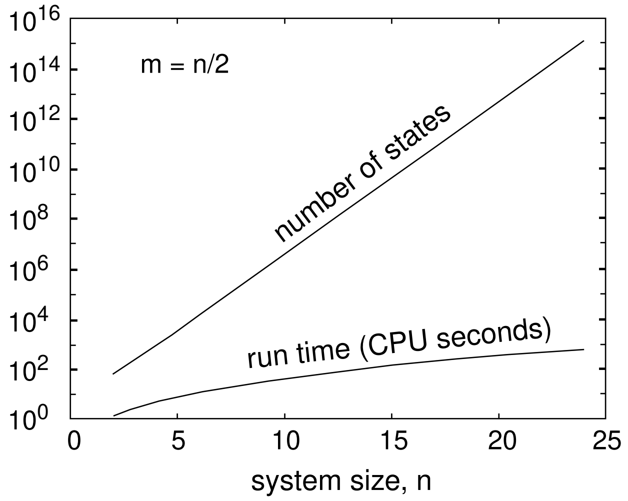
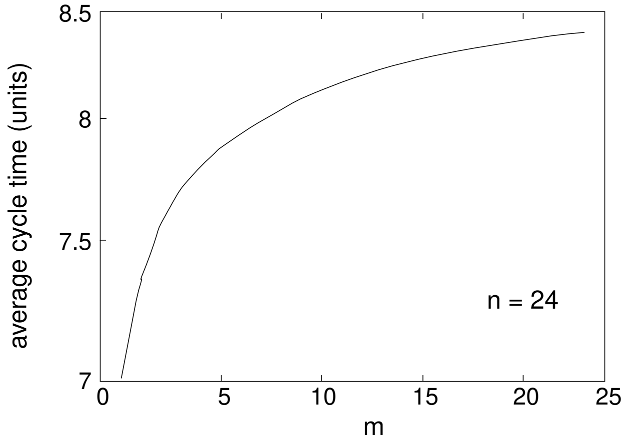


Figure 6.12: Synchronized processes.



(a) state space size vs overall run time



(b) system performance

Figure 6.13: Experimental results of synchronized processes.

Part II

Statistical Analysis

Chapter 7

Overview

7.1 The change of goal: bounding performance metrics

Part I of this thesis was devoted to computing the *exact values* of performance metrics, more precisely, the average time separations of events (TSEs). Our effort there was mainly on attacking the *state-explosion* problem in Markovian analysis of large systems. It should be noted that the state-explosion is inherent in virtually all performance analysis problems when theoretical performance measures are pursued, which demands a state-level analysis. It should be also emphasized that the state-explosion problem is not unique to performance analysis. For example, it is also the major hurdle in synthesis and verification. So far, the size of systems that our techniques can manage is still limited in many cases. For instance, even with the state-compression approach, the symbolic performance evaluation tool developed in Part I cannot complete the analysis of a micropipeline circuit with more than 8 stages within one hour on a Sun Sparc20 workstation.

Taking the other side of the coin, we devote the present part to performance evaluation using *statistical analysis* instead of *analytical methods*. In particular, we change our goal from computing the exact values of performance metrics to obtaining their statistical *estimates*. More specifically, we will develop efficient techniques to obtain lower and upper bounds on performance metrics which themselves have well-defined probabilistic error intervals. The mean of the bounds then serves an unbiased estimate of the corresponding performance metric, as illustrated in Fig. 7.1. These

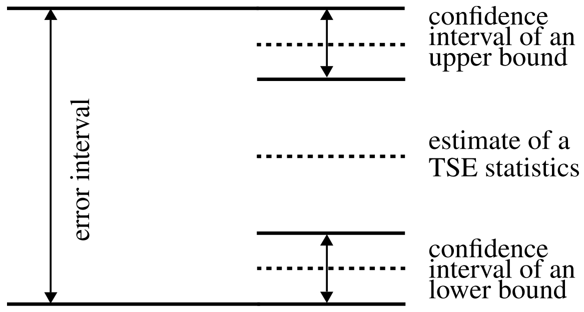


Figure 7.1: Relationship among the estimates.

techniques can handle dramatically larger systems. As an example, the average throughput of a micropipeline circuit of 32 stages can be bounded within an interval of almost 0 width within several minutes. In addition, we will demonstrate that the bounds can be often made arbitrarily sharp, and thus improve the quality of the estimates, with polynomially increased run time.

Also different from Part I where we used time-discretization to handle arbitrarily distributed component delays, we now move on to *continuous time* models. In particular, the widely accepted *stochastic timed Petri net* models are adopted. However, unlike other research in stochastic timed Petri nets where delays are assumed to be either fixed or exponentially distributed, we allow the delays to be arbitrarily distributed provided that they all have finite means. In a word, the shift of our system model from communicating stochastic finite state machines in Part I to stochastic timed Petri nets does not weaken the modeling capability of our tool.

Another difference is that the techniques of this part are capable to estimate not only the averages of TSEs but also other useful statistics such as their variance as well as their distributions.

The theoretical foundation to derive bounds on TSE statistics is based on a notion of a *segment*. Roughly speaking, a segment is an untimed event graph corresponding to an untimed execution of the STPN that starts and ends with the same marking. For the STPNs with free-choice, a random time execution of the net can be partitioned into a sequence of independent and identically distributed (iid) segments. Within each segment, a lower and an upper bound on the TSE instance is derived using a simple longest path analysis and by ignoring the history prior to the segment considered. The latter is achieved by assuming the tokens may flow into the source places of the segment at any time between $(-\infty, \infty)$. As a result, one obtains a sequence of iid lower and upper bounds which can be used to obtain an unbiased estimate of the bounds using simple statistical methods. This approach has been fully automated. The resulting tool is named USC-PET which is depicted in Fig. 7.2. Note that USC-PET essentially solves the problem of a cyclic system by transforming the system to a finite[1] set of acyclic systems.

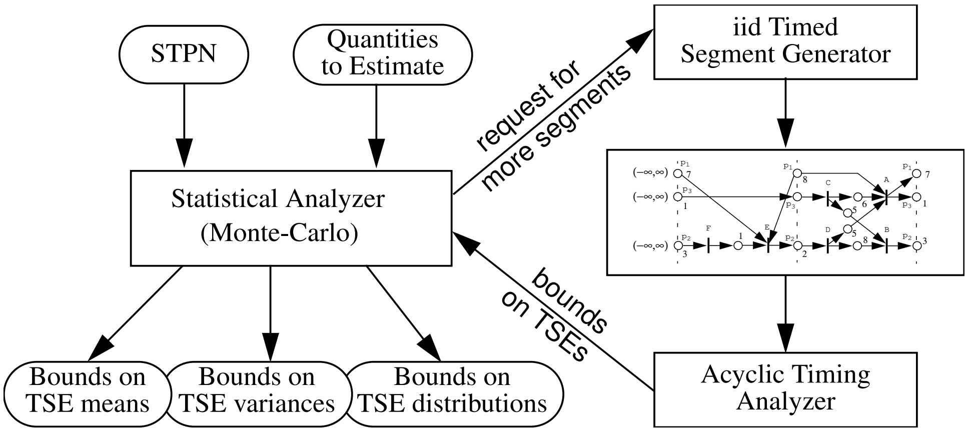


Figure 7.2: Overview of USC-PET.

7.2 A comparison with existing work

To make a better comparison of the techniques described in this part with existing work on performance analysis of stochastic timed Petri nets and other related models, we classify all these work into four categories depending on their solutions and the target performance quantities. Fig. 7.3 illustrates this classification. Within each category, works vary in the expressiveness and the size of system models that can be analyzed.

The first category consists of analytical methods that compute exact values of performance metrics. In [98], Ramamoorthy et al used Karp's theorem [67] to determine the average throughput of a timed marked graph with deterministic firing delays in polynomial time. Burns [22] modeled asynchronous circuits using an event-rule system similar to marked graphs with fixed delays and used linear programming

[1]Finite with probability 1, to be precise.

	Exact	Bounds
Analytical	√	√
Simulation/ Random Sampling	√	×

Figure 7.3: A classification of performance analysis techniques for timed Petri nets.

to find the average cycle time of events. Lee [76] extended Burns' work to handle systems with OR-causality. Although computationally efficient, these techniques are severely limited in applicability because they cannot handle systems with choice and must model component delays using their mean values. There have been proposed many other algorithms to deal with such systems. Dasdan et al. recently gave a detailed comparison among these algorithms [40].

To handle choice, Kudva et al [72] adopted Generalized Timed Petri Nets (GTPNs) [63] to model asynchronous circuits in which component delays are again modeled using their mean values. As usual, the model is converted into a continues time Markov chain (CTMC) and solved for its stationary probability distribution. In other Petri-net based approaches, the delays are assumed to be either fixed (possibly zero) or exponentially distributed and the analytical tool is either a discrete or continuous time Markov chain (see e.g., [82]). In addition to the restricted delay modeling, the size of the models that can be analyzed is limited by the state explosion problem.

Our work presented in Part I also fall into the first category, but can handle arbitrarily distributed delays. To mitigate the state-explosion problem, we proposed a technique called state-compression to speed up the symbolic computation of the stationary distribution of their DTMC models. Other efforts to mitigate the state explosion problem include Buchholz's work on (exact and weak) lumpability [17, 18, 19]. Nevertheless, even with these advances, the state explosion problem has limited Markov chain based approaches to small models.

To circumvent the state explosion problem, Queuing models have been explored. For example, Greenstreet [52] studied the performance of closed asynchronous ring structures. Unlike Petri nets, however, queuing models are less expressive in specifying synchronization and consequently are applicable only to a relatively small special class of asynchronous circuits. Moreover, there is no simple solution to such models with general delay distributions.

Another approach to circumvent the computational challenges of the above techniques is to analytically compute bounds on performance metrics rather than their exact values. For example, Campos et al's work [24] computes throughput bounds in a limited class of timed Petri nets. These bounds are defined by considering all possible delay distributions with fixed means. Unfortunately, they are often too wide to be practically useful. In addition, the existing work in timing analysis of asynchronous circuits often deals with system models with delays expressed by intervals and bounds the time separation between system events (TSEs). Although typically targeted to system verification, such bounds may serve as upper and lower bounds of average performance metrics that are independent of the actual distribution of the delays. Examples are Ebergen and Berks' work [45] on extreme (tight bounds) and amortized response times of asynchronous pipelines with tree structures. Hulgaard and Burns [64] give extreme TSEs for timed Petri nets with free-choice and/or unique-choice. Their algorithm often works well but still has exponential computational complexity in the worst case. For more general Petri nets, however, all known techniques must perform timed state space analysis in order to compute extreme TSEs (e.g., [3, 9]), which is often computationally expensive.

Opposite to the above analytical techniques, several research groups have studied simulation-based approaches to finding performance estimates that can be applied to models with general delay distributions. For example, Shelder et al [109] developed performance estimates for regenerative timed Petri nets using independent replicants and Glynn and Iglehart [51] proposed using standardized time series method. For non-regenerative nets with arbitrary delay distributions, Haas [54] recently proposed a technique for performance estimation using standardized timed series. Unfortunately, we are not aware of any currently available tool which implements Haas' technique and consequently its computational efficiency is unclear.

Lastly, the techniques presented in the present part form a new category of performance analysis methods, where random simulation (more precisely, a combination of random sampling and longest path analysis) is used to efficiently derive performance bounds.

Chapter 8 develops techniques for stochastic marked graphs, a class of Petri nets that does not have choice but can model concurrency. Micropipeline circuits can be modeled by such a Petri net. Chapter 9 extends the result to Petri net that exhibits a limited class of choice called free-choice and/or unique choice.

Chapter 8

The Choice-Free Systems

8.1 Introduction

This chapter is devoted to deriving performance bounds of choice-free stochastic systems. Such systems can often be modeled using stochastic timed *marked graphs* (e.g., [95]), also known as timed *event-graphs* (e.g., [35, 59]) or *decision-free Petri nets* (e.g., [98]). Although marked graphs are a restrict class of Petri net [92], they constitute an adequate model for many real-world systems, including a large class of asynchronous circuits and embedded systems. To ease the exposition of the proposed technique, we focus on bounding the *average* timed separation of events (TSEs). The derivation of other statistics of TSEs will be presented in the next chapter.

The problem of computing the average TSEs in basic classes of Petri nets has been studied extensively over the last two decades (e.g., [67, 98, 59, 35, 5, 23, 24, 39, 72, 44, 126]) although the majority of the work was on average system throughput. The pioneer work by Karp [67] solved this problem for mean cycle time for marked graphs with deterministic delays. In the stochastic cases where delays are not fixed, the problem has been addressed using Markovian analysis (e.g., [63, 72, 126]) and event-driven simulation. As mentioned before, the Markovian analysis techniques are often limited to small systems because of the *state explosion problem* ([17, 56, 127]). For large complex systems, event-driven simulation is often used to obtain some estimates of the performance metrics. Because of the inherent cyclic nature (hence the memory) of general concurrent systems, simple simulation-based approaches suffer from lack of confidence in analyzing the steady-state system behavior (e.g., [31]). Haas recently developed a sophisticated simulation-based approaches e.g.,

[54] which gives the confidence interval of the resulting estimates. However, the approach may require significantly more computational effort [55] and no tool is currently available that implements this approach.

Because of these difficulties with finding the exact value of the average cycle time, many researchers have resorted to efficiently finding lower and upper bounds of the average cycle time. For special marked graphs (i.e., with tree-like structures), Ebergen and Berks give nearly exact bounds on *amortized* response time [44]. Campos et al [23] give average throughput bounds using linear programming and later extended their results to free-choice Petri nets [24, 111]. A distinct feature of their bounds is that they are independent of higher moments (except the first moment) of the delays. In practice, these bounds can be efficiently computed at structure level instead of state-level although exponential run time may also be hit in the worst-case. Unfortunately, in many real applications, their bounds are too loose to be useful.

In comparison, the bounds obtained by the technique proposed in this chapter tend to be much sharper, and typically are very close to the true value of the average TSE. It is well-known that all possible executions of a Petri net can be captured by an infinite unfolding of the net [87]. The challenge we are faced with is to characterize the average delay determined by this infinite unfolding in finite time. Fortunately, for marked graphs, the infinite unfolded graph has a repeated structure. Consequently, we show that we can analyze a finite unfolding of the graph to obtain lower and upper bounds on the TSE of the targeted events. In particular, the chapter presents a method to identify the degree of unfolding sufficient to find the bounds and the proof that such a finite unfolding always exists. More specifically, to obtain the bounds from this finite unfolding we identify a set of reference events from which the delays to the targeted pair of events are analyzed. By ignoring the time separation between reference events we obtain bounds that are independent of the long-term history of the net. It can then be proved that the derived bounds are valid for the infinite unfolding. The principle penalty for ignoring the time separation between reference events is that sometimes it leads to wide bounds. However, this penalty can be reduced (often arbitrarily) by analyzing a larger unfolding. In all the experiments we have done, the technique yields sharp lower and upper bounds, and exhibits a time complexity that is polynomial in system size.

The idea of bounding stochastic measures of sequential systems using statistical methods is not new. In particular, Kozhaya and Najm's used statistical methods to bound the power estimation of synchronous sequential circuits (modeled as finite state machines) [71]. Their method is similar to ours in that the infinite execution of the sequential circuit is analyzed using a finite execution. In their case they ignore the initial state of the sequential circuit whereas in our case we ignore the time separation between reference events. Another difference between the two works is that our analysis yields closed form equations which sometimes can be computed using analytical techniques.

The remaining of this chapter is organized as follows. Section 8.2 formally defines the average TSEs in stochastic timed marked graphs introduced in Chapter 2 and proves the existence of average TSEs. A detailed description of the bounding technique is given in Section 8.3. Section 8.4 briefly discusses several approaches to evaluate the derived formulae of the bounds. The next section is devoted to an easy extension of the technique described in Section 8.3 to further sharpen the bounds. Experimental results are described in Section 8.6.

8.2 TSEs in stochastic timed marked graphs

8.2.1 Some known results of marked graphs

Let $N = (P, T, F)$ denote a Petri net where P is the set of places, T the set of transitions, and $F : (P \times T \cup T \times P) \rightarrow \{0, 1\}$ the flow function (or the incident matrix). Chapter 2 gives details of the definition and firing rules of a Petri net. Recall that the Petri net is a *marked graph* if $|\bullet p| = |p\bullet| = 1, \forall p \in P$, i.e., every *place* has a unique input and output transition. In the remaining of this chapter, assume N is a marked graph.

Let $\Sigma = (N, M_0)$ be the *marked* net of N where M_0 is a the initial marking of Σ. Let $R(M_0)$ denote all reachable marking of Σ from M_0. Recall also that Σ is *live* iff every transition can be enabled from every $M \in R(M_0)$. It is *safe* if no place ever contains more than one token, i.e., $M(p) \leq 1$ for every $M \in R(M_0)$.

It is a well-known result in the literature [10] that a live and safe (LS) marked Petri net has no source or sink places and no source or sink transitions. This implies that a LS Petri net can be partitioned into a set of strongly connected components each evolving independently of others. In the sequel, we assume the the marked graph is strongly connected. In particular, we restrict ourselves to live and safe marked graphs (LSMG's).

Lemma 10 [10] *For a live and safe (marked) Petri net, there is no source or sink places and no source or sink transitions. That is,* $\forall x \in P \cup T$, $\bullet x \neq \emptyset$, $x\bullet \neq \emptyset$.

The necessary and sufficient condition [36] for a marked graph to be live and safe requires a simple definition of a *circuit*. A sequence $(v_1, v_2, \cdots, v_{m+1})$ is a (directed) *path* if $v_i \in P \cup T$ and $(v_i, v_{i+1}) \in F$ for every $1 \leq i \leq m$. A *circuit* is a path with its head and tail being identical. Below, when we write *circuits*, we refer to elementary ones, i.e., those not containing any other circuits.

Lemma 11 [36] *Let* (G, M_0) *be a marked graph. It is live iff every circuit is initially marked, i.e.,* $\mu_{M_0}(\xi) \geq 1$, $\forall \xi \in C(G)$. *It is live and safe iff every place belongs to a circuit* $\xi \in C(G)$ *for which* $\mu_{M_0}(\xi) = 1$.

Lemma 12 [5] *For a connected marked graph, a firing sequence results in the initial marking if and only if it fires all transitions a same number of times.*

Below, we will use the micropipeline circuit previously described in Chapter 2 (Figure 2.5) as a running example. For convenience, we redraw it in Figure 8.1. In Chapter 2, we have already checked that this marked graph is live and safe with the given initial marking.

8.2.2 Delay models

Let us quickly review the stochastic delay assumptions of our stochastic timed marked graph models introduced in 2 where we associate delays with places[1]. For each place p, $X(p)$ is the random variable denoting the delay associated p. The delays experience by different tokens in the same place are assumed to be independent.

[1]Sometimes, delays are associated with transitions or both in the literature. For marked graphs, one can always model these cases in our place-delay semantics by adding extra places and transitions.

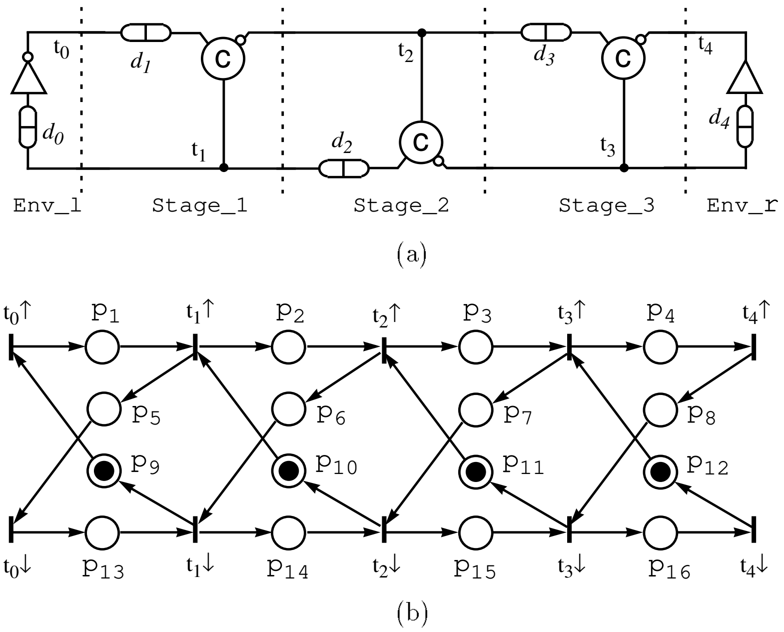


Figure 8.1: An asynchronous micropipeline: (a) the control circuit, and (b) its marked graph model.

For any two distinct places p and p, $X(p)$ and $X(q)$ are also assumed to independent. Function $F_{X(p)} : R \to [0,1]$ denotes the distribution function of $X(p)$. That is, $F_{X(p)}(x) = Prob(X(p) \leq x)$. The actual firing of a transition is assumed to be instantaneous.

In the micropipeline model (Figure 8.1), let us assume the C-elements (inside stages), the inverter and the buffer (inside the environments) all have unit delay. The (random) delay variables ($d_0 \sim d_3$) represent the data processing delays in corresponding stages and environments. It is not difficult to determine the delay on each places in marked graph model (Figure 8.1(b)). For instance, $X(p_1) = d_1 + 1$, $X(p_2) = d_2 + 1$, etc.

In a marked graph, since each place has a unique input and output transition, the basic timing constrains among the firing of transitions can be easily expressed through the delays on places as follows. Suppose $\tau(t^{(k)})$ denotes the time when transition $t \in T$ fires for its k-th time. In particular, we define $\tau(t^{(0)}) = 0$ if $t^\bullet \cap I \neq \emptyset$. Then, for every $k > 0$, we have,

$$\tau(t^{(k)}) = \max_{s \in T, s\bullet \cap \bullet t \neq \emptyset} \tau(s^{(k-\epsilon)}) + X(s\bullet \cap \bullet t). \tag{8.1}$$

where ϵ is the initial occurrence index offset. That is, $\epsilon = 1$ if $s\bullet \cap \bullet t \in M_0$, and 0 otherwise.

8.2.3 Average cycle time and time separation of events

The k-th occurrence time of a transition $t \in T$, namely, $\tau(t^k)$ is a random variable. The k-th time separation between transitions s and t with an occurrence offset ε is also a random variable which we denote by $\overline{\gamma}^{(k)}(s, t, \varepsilon)$. That is,

$$\gamma^{(k)}(s, t, \varepsilon) = \tau(t^{(k+\varepsilon)}) - \tau(s^{(k)}), k \in \mathbf{N}, \varepsilon \geq -k.$$

The above time separation sequence, i.e., $\{\gamma^{(k)}(s, t, \varepsilon) : k \geq 0\}$ forms a stochastic process. There are cases, for instance when all delays are deterministic, in which this sequence can be periodic in k. However, what we are interested is the *average* of this sequence over all k, namely, the average time separation between s and t as defined below.

Definition 4 *Let (G, M_0) be a stochastic timed MG. The average time separation between transitions s and t with occurrence index-offset ε is* $\lim_{n\to\infty} \frac{1}{n} \sum_{k=1}^{n} \gamma^{(k)}(s, t, \varepsilon)$.

Note that the average *cycle time* of a transition u is a special case of the average TSE defined above with $s = t = u$ and $\varepsilon = 1$. It is well-known that all the transitions in a live stochastic marked graph has the (finite) same average cycle time [5, 6].

Theorem 13 *[5, 6] (Weak ergodicity) For a LS stochastic timed MG (G, M_0) as defined above, there exists a constant $0 \leq c < \infty$ such that (8.2) holds for every $t \in T$.*

$$\lim_{k\to\infty} \frac{1}{k}\tau(t^{(k)}) = \lim_{k\to\infty} \frac{1}{k}\mathbf{E}\tau(t^{(k)}) = c \quad \text{a.s. and in mean,} \tag{8.2}$$

We shall show that the limit in Definition 4 in fact converges for every transition pair (s, t) and every finite ε. Theorem 14 states this result which generalizes the *weak ergodicity property* of the cycle time (of one transition) [5, 6] to that of the time separations of two arbitrary transitions.

Theorem 14 *Let s, t be two arbitrary transitions of a LS stochastic timed MG (G, M_0) as defined in Section 8.2.2. Their average time separation (with occurrence offset ε) converges to a constant $\overline{\gamma}(s,t,\varepsilon)$ almost surely and in mean, i.e.,*

$$\lim_{n\to\infty} \frac{1}{n} \sum_{k=1}^{n} \gamma^{(k)}(s,t,\varepsilon) = \overline{\gamma}(s,t,\varepsilon) \quad \textit{a.s., and in mean.}$$

Moreover, $(-h+\varepsilon)\cdot c \leq \overline{\gamma}(s,t,\varepsilon) \leq (h+\varepsilon)\cdot c$ where h is the minimum number of places in all the circuits containing s, t, and c is the average cycle time of G.

Our proof of Theorem 14 relies on a special case of Kingman's subadditive ergodic theorem [69].

Theorem 15 (Kingman's Ergodic Theorem) *Let $\{Y_{l,n}, l < n \in \mathbf{Z}\}$ be a doubly-indexed stationary random process such that*

$$\begin{aligned} \textit{boundedness:}\quad \mathbf{E}Y_{0,n} &\geq -Cn, n > 0, \textit{ for some finite constant } C > 0, \\ \textit{subadditivity:}\quad Y_{l,n} &\leq Y_{l,m} + Y_{m,n}, \forall l < m < n. \end{aligned}$$

There exists a constant ϕ such that the sequence $\{\frac{Y_{0,n}}{n}, n > 0\}$ converges in mean and almost surely, i.e., $\lim_{n\to\infty} \frac{\mathbf{E}Y_{0,n}}{n} = \phi$, and $\lim_{n\to\infty} \frac{Y_{0,n}}{n} = \phi$, w.p.1.

Proof of Theorem 14 The theorem is first proved for $\varepsilon = 0$. Extension to any fixed ε is argued similarly.

Let s, t be as given in the assumption. From the *coupling argument* [5], there is a positive integer $K < \infty$ almost surely such that for any $t, s \in T$, the sequence $\gamma^{(k)}(s,t,0) = \{\tau(t^{(k)}) - \tau(s^{(k)}) : k > K\}$ is stationary.

Next, for all $0 < l < m$, let us define $Z_{l,m} = \sum_{k=l+1}^{m} \gamma^{(k)}(s,t,0)$. Clearly, Z is additive, i.e., for all $0 < l < m < n$, $Z_{l,n} = Z_{l,m} + Z_{m,n}$. We shall further show that for large n, $\mathbf{E}Z_{K,n}$ is bounded from below by $-(c\cdot h)\cdot n$ where c is the average cycle time of the graph.

For any $k > h$, we have

$$\tau(t^{(k)}) \geq \tau(s^{(k-h)}) \tag{8.3}$$

because t cannot have fired h times more than s at any instant under the safeness assumption. It can then be checked that,

$$Z_{K,n} = \sum_{k=K+1}^{n} \tau(t^{(k)}) - \tau(s^{(k)}) \geq - \sum_{k=n-h+1}^{n} \tau(s^{(k)}). \tag{8.4}$$

By taking expectation on both sides of (8.4), the boundedness of $\mathbf{E}Z_{K,n}$ follows Theorem 13 when n is large.

As such, $Z_{K,n}$ verifies the conditions of Kingman's ergodic theorem, from which we deduce that $\lim_n \frac{1}{n}\sum_{k=1}^{n}\gamma^{(k)}(s,t,0) = \lim_n \frac{1}{n}Z_{0,n}$ equals to a constant almost surely. Moreover, this constant is lower bounded by $-c \cdot h$. It is further upper bounded by $c \cdot h$ because (8.3) is also valid if s and t are interchanged.

For the case where $\varepsilon \neq 0$, the theorem can be checked with similar arguments. In particular, the sequence $\{\gamma^{(n)}(s,t,\varepsilon)\}$ is almost surely stationary for large n because $\{\gamma^{(n)}(r,r,1)\}$ is almost surely stationary for all $r \in T$ and large n. The only difference is that the average time separation is now bounded by $c \cdot (h + \varepsilon)$ from above and $c \cdot (-h + \varepsilon)$ from below. ∎

Consequently, under fairly weak assumptions on delay models, we see that the average of the time separation sequence of arbitrary transitions converges almost surely to some constant. Bounding this constant for arbitrary event pairs is the focus of this chapter. For convenience, we refer to (s,t,ε) as an *event separation triple*, meaning we are interested in the average TSE $\overline{\gamma}(s,t,\varepsilon)$.

8.3 Bounding the average TSEs

One obvious challenge in bounding the average TSEs is that they are defined over infinite execution of the timed marked graph. In this section, we show that the average TSEs can be bounded by identifying and analyzing a special finite segment (meaning that we do not have to explicitly analyze the infinite execution). We show such a finite segment can always be found for any event separation triple (s,t,ε).

8.3.1 The duality of the bounds

In many timing analysis problems, finding a lower bound on a delay variable can often be transformed into finding an upper bound on a related delay variable. This is also true in our case.

Note that for any $j, \varepsilon \in \mathbf{N}$ s.t. $j > -\varepsilon$, from the definition of γ, we have $-\gamma^{(j)}(s,t,\varepsilon) = -t^{(j+\varepsilon)} + s^{(j)} \triangleq \gamma^{(j+\varepsilon)}(t,s,-\varepsilon)$. Therefore, if one finds an upper bound, say, U on $\overline{\gamma}(t,s,-\varepsilon)$, then $-U$ is a lower bound on $\overline{\gamma}(s,t,\varepsilon)$. Because of this duality, we shall be concerned only with finding the upper bounds from now on.

8.3.2 Unfolding

The *untimed* behavior of a marked graph (G, M_0) (either in terms of transition firing sequence or marking sequence) can be directly reasoned through net unfolding [87]. In this subsection, we discuss some simple properties of the unfolded graph of (G, M_0) which serve as key bases of later sections.

The unfolding of (G, M_0) results an acyclic marked graph $H = (P_H, T_H, F_H)$. All source places of H are marked initially. The sets P_H and T_H represent the instances of places in P and transitions in T, respectively. The flow relation F_H captures the causality of the firing of transitions as the net evolves. As an example, Figure 8.3(a) shows a finite unfolding of the marked graph in Figure 8.1 where every transition is instanced exactly once. It may be useful to note that from the unfolded graph, one may enumerate all possible transition sequences leading from one reachable marking to another.

Figure 8.2 gives a simple procedure that unfolds (G, M_0) once. It instances every transition of G exactly once, and returns an unfolded graph $G^{(0)} = (T^{(0)}, P^{(0)}, F^{(0)})$. The (inverse) labeling function $\ell : T^{(0)} \cup P^{(0)} \rightarrow T \cup P$ maps the instanced transitions and places to their corresponding ones in G.

Lemma 13 *Procedure 1 terminates for every LSMG (G, M_0). Moreover, when it returns, $M = M_0$.*

Proof Suppose there is a transition $t \in T$ that cannot be instanced into $G^{(0)}$. Since G is choice free, this mean there is a place in $\bullet t$ that is never instanced into $G^{(0)}$, which implies transition $\bullet p$ is never instanced. In addition, ${}^{\bullet}p$ must not be t because otherwise p and t form a circuit, forcing $p \in I$ by Lemma 11. We may repeat the same argument on transition $\bullet p$ and deduce that no transition of T can ever be instanced (enabled), contradicting the assumption that G is live. Finally, since the procedure instances every transition exactly once, the final marking M returns to the initial marking M_0 by Lemma 12. ■

Procedure 1 **Unfold**$((G, M_0))$ /* $G = (P, T, F)$ */

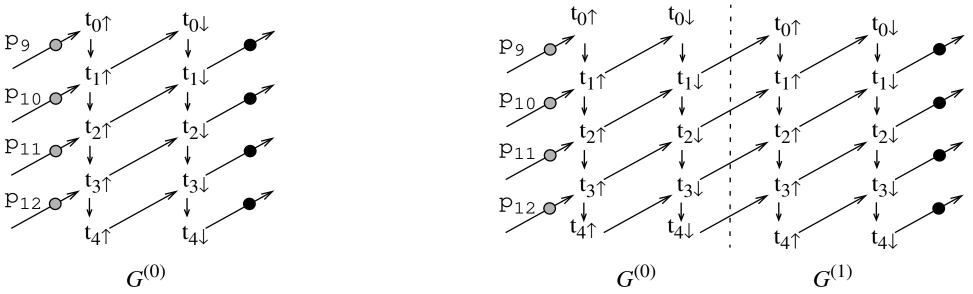
```
    P(0)←∅, T(0)←∅, F(0)←∅, M←M0;            /* initialize */
    foreach (p ∈ P s.t. M0(p) > 0)
        P(0)←P(0) ∪ {p(0)}, ℓ(p(0))←p;       /* make a copy of p and label it */
    while T ≠ ∅ {
        pick a t ∈ T s.t. t is enabled at M;
        T(0)←T(0) ∪ {t(0)}, ℓ(t(0))←t;       /* make a copy of t, and label it */
        foreach (p ∈ •t)
            F(0)←(p(0), t(0)), M(p)←0;       /* add an edge from p(0) to t(0),
                                                erase token in p */
        foreach (p ∈ t•) {
            P(0)←P(0) ∪ {p(0)}, ℓ(p(0))←p;   /* make a copy of p, and label it */
            F(0)←(t(0), p(0)), M(p)←1; }     /* add an edge from t(0) to p(0),
                                                put a token in p */
        T ← T \ {t}; }                       /* t has been instanced */
    return G(0) = (T(0), P(0), F(0)) and ℓ;
```

Figure 8.2: The unfolding procedure.

Lemma 13 verifies that when all the transitions are instanced exactly once, the sink places of the unfolded graph coincide with the support set of the initial marking (recall that a place p is in the support set of M if $M(p) > 0$). This is intuitive because for a marked graph if a firing sequence which fires all transitions an equal number of times, it leads back to the same marking [91]. We call the above unfolding the 0th unfolding of G.

Example The 0th unfolding of the marked graph in Figure 8.1(b) is shown in Figure 8.3(a). It starts with the initial marking (shaded) and ends with the same marking (dark) after instancing all the transitions once. In the figure, we have omitted the instanced places, assuming there is a place instanced on each edge. In addition, we have dropped the occurrence indices of transitions in order to simplify the exposition. (They should be clear from the indices of the unfolding subgraph, e.g., $G^{(0)}$.) □

Starting from the marking obtained from the 0th unfolding, we may repeat the unfolding process and obtain the 1st unfolded graph $G^{(1)}$ (e.g., Figure 8.3(b)) and so



(a) Unfolding once, (b) unfolding twice.

Figure 8.3: Unfolding the marked graph in Figure 8.1(b).

forth. For all $j, k \in \mathbf{Z}_+$ such that $j \leq k$, let us define an unfolding segment $H(j,k)$ to be the unfolded subgraph generated from j-th to k-th unfolding, i.e.,

$$H(j,k) = \cup_{i=j}^{k} G^{(i)} \tag{8.5}$$

For convenience, we define a *shift* operator Δ on a *subgraph* of an unfolded segment. It changes the indices of all the elements of the subgraph by a constant. For example, $\Delta_l H(j,k)$ results in a same graph as $H(j,k)$ except the indices of all the nodes and edges are added by an amount l. In particular, the fact that $G^{(j)}$ (i.e., $H(j,j)$) and $G^{(j+1)}$ (i.e., $H(j+1,j+1)$) are isomorphic (due to Lemma 13) is equivalent to saying $H(j,j) = \Delta_{-1} H(j+1,j+1)$. More generally, we have the following simple lemma which states that the unfolded graph of a LSMG has repeated unfolding segments.

Lemma 14 *For all $j, k, l \in \mathbf{N}$, the unfolding segments $H(j+l, k+l)$ and $H(j,k)$ are identical under the shift operator, i.e., $H(j+l, k+l) = \Delta_l H(j,k)$.*

Proof Following Lemma 13, for any $i, j \in \mathbf{Z}_+$, $H(i,i) = G^{(i)}$ is checked to be identical to $G^{(j)} = H(j,j)$ under Δ with a proper shift amount. ■

8.3.3 Reference sets

In this subsection, we derive an upper bound on average TSE $\overline{\gamma}^{(k)}(s,t,\varepsilon)$ for any fixed $k > 0$. In the next subsection, we will show that the derived upper bound is indeed an upper bound on $\overline{\gamma}(s,t,\varepsilon)$ by showing it is an upper bound on $\overline{\gamma}^{(k)}(s,t,\varepsilon)$ for all sufficiently large k. To derive this upper bound, we need the concept of *reference sets* of the separation triple (s,t,ε) which will be formally introduced below. Roughly speaking, a reference set cuts every path from the source places of the unfolded graph $H(0,\infty)$ to $t^{(k+\varepsilon)}$, and every event in the reference set has a path leading to $s^{(k)}$. Our key idea is that under any delay assignment, we can derive an upper bound on $\overline{\gamma}^{(k)}(s,t,\varepsilon)$ by considering only the delays on paths from reference events to the targeted events $s^{(k)}$ and $s^{(k+\varepsilon)}$, but not any further history nor the occurrence time of the reference events.

For all $i,j \in \mathbf{N}$ s.t. $i \leq j$, let $\mathcal{D}(i,j)$ be the space generated by all possible delay assignments of places inside the unfolding segment $H(i,j)$. That is, $\mathcal{D}(i,j) = \otimes_{p\in H(i,j)} X(\ell(p))$. (Recall that function $\ell(.)$ maps an instanced place to its corresponding place in the origin marked graph.) Then, $\mathcal{D} = \lim_{j\to\infty} \mathcal{D}(0,j)$ is the space generated by delay assignments of places of all possible timed executions[2]. In order to stress a particular delay assignment D under which random quantities are referred, we subscript them with D. For instance, $X_D(\ell(p))$ means the delay value of place $p \in H(0,\infty)$ under delay assignment D, and ${\gamma_D}^{(j)}(s,t,\varepsilon)$ denotes the value of random variable $\gamma^{(j)}(s,t,\varepsilon)$ under D.

Let $\rho \in H(0,\infty)$ be a path in the unfolded graph. We denote by a random variable $\delta(\rho)$ the sum of random delay variables associated with all the places along path ρ, namely, $\delta(\rho) = \sum_{p\in\rho} X(\ell(p))$. Let u,v be any two nodes in $H(0,\infty)$. Denote by $\mathcal{P}(u,v)$ the set of all paths from u to v, namely, $\mathcal{P}(u,v) = \{\rho \in H(0,\infty) \mid x \overset{\rho}{\rightsquigarrow} y\}$.

From timing constraint 8.1, we verify that if there is a path from events u to v, v must occur after u by at least the maximum sum of delays on all the paths from u to v. Formally, $\forall u,v \in H(0,\infty)$, s.t., $\mathcal{P}(u,v) \neq \emptyset$, we have,

$$\tau(u) + \max_{\rho\in\mathcal{P}(u,v)} \delta(\rho) \leq \tau(v). \tag{8.6}$$

Under a given delay assignment D, we say that u is *critical* for v if (8.6) holds in equality. In that case, there is a path $\rho^* \in \mathcal{P}(u,v)$ so that $\delta_D(\rho^*) = \tau_D(v) - \tau_D(u)$ and we call ρ^* a critical path from u to v. It is understood that any subpath of a critical

[2]Roughly speaking, a timed execution of a timed marked graph is a tuple (N,f) where N is an unfolding of the graph and f indicates the occurrence time of each event in N as well as the delay assigned to the places in N.

path is also critical under the same delay assignment. Let S be the source nodes of $H(0, \infty)$, then at least one node in S is critical for every transition $t \in H(0, \infty)$. The criticality of timing along the paths can be described using the concept of *cut sets* defined below.

Definition 5 (*Cut set*) *A cut set for a transition $t \in H(0, \infty)$ is a set of transitions $C(t)$ in $H(0, \infty)$ such that any path leading from a source place of $H(0, \infty)$ touches at least one transition in $C(t)$. Moreover, every transition in $C(t)$ has a path leading to t. Formally, $\forall p \in M_0, \forall \rho \in \mathcal{P}(p^{(0)}, t)$, $\rho \cap C(t) \neq \emptyset$ and $\forall u \in C(t)$, $u \rightsquigarrow t$. The cut set is minimal if it does not contain any other cut set of t.*

Example Considering the unfolded graph in Figure 8.3(b), $\{t_{1\downarrow}{}^{(0)}, t_{2\downarrow}{}^{(0)}\}$ is a cut set for $t_{1\uparrow}{}^{(1)}$, but not for $t_{1\downarrow}{}^{(1)}$ because it does not cut the path from p_{12} to $t_{1\downarrow}{}^{(1)}$ through transition $t_{3\uparrow}{}^{(0)}, t_{4\uparrow}{}^{(0)}, t_{3\downarrow}{}^{(0)}$, and $t_{2\uparrow}{}^{(1)}$, nor is it a cut set for transition $t_{0\uparrow}{}^{(0)}$ because $t_{2\downarrow}{}^{(0)} \not\rightsquigarrow t_{0\uparrow}{}^{(1)}$. Note that the set is minimal for $t_{1\uparrow}{}^{(1)}$. □

Lemma 15 *Let $C(t)$ be a cut set for transition $t \in H(0, \infty)$. Then, for any delay assignment $D \in \mathcal{D}$,*

$$\tau_D(t) = \max_{u \in C(t)} [\tau_D(u) + \max_{\rho \in \mathcal{P}(u,t)} \delta_D(\rho)] \tag{8.7}$$

Proof For any $u \in C(x)$, we have $\tau_D(t) \geq \tau_D(u) + \max_{\rho \in \mathcal{P}(u,t)} \delta_D(\rho)$. Therefore,

$$\tau_D(t) \geq \max_{u \in C(t)} \tau_D(u) + \max_{\rho \in \mathcal{P}(u,t)} \delta_D(\rho). \tag{8.8}$$

Further, under given D, there is a critical path, say $\rho^* \in \mathcal{P}(v, t)$, from some source node $v \in S$ to t. Let $w \in \rho^* \cap C(t) \neq \emptyset$. By sub-criticality argument, $\tau_D(w) + \delta_D(\rho_1^*) = \tau_D(t)$ where $w \overset{\rho_1^*}{\rightsquigarrow} t$ and $\rho_1^* \subseteq \rho^*$. Thus, (8.8) holds only with equality. ■

The above lemma shows that the occurrence time of a transition in the unfolded graph can be completely determined by the occurrence time of transitions in any of its cut sets plus the delays assigned on the places on paths leading from cut set transitions to it. In other words, any other knowledge regarding the timing of unfolded graph further in the history is then irrelevant.

We now define the *reference sets* of a pair event separation pair which is the key concept in the remaining sections.

Definition 6 *Let* $(s^{(j)}, t^{(j+\varepsilon)})$ *be any valid transition pair. A subset of transitions* $\mathcal{R}$ *in* $H(0,\infty)$ *is a reference set for the pair if*

1. R *is a cut set of* $t^{(j+\varepsilon)}$,
2. $\forall u \in \mathcal{R}, u \rightsquigarrow s^{(j)}$.

A minimal reference set is one that does not contain any other reference set.

Intuitively, $\mathcal{R}$ is a set of transitions that cut every path from a source place of the unfolded graph to $t^{(j+\varepsilon)}$. Moreover, every transition in $\mathcal{R}$ has paths leads to $s^{(j)}$. Note that if R is a minimal reference set, it is necessary to be a minimal cut set of $t^{(j+\varepsilon)}$.

Example Consider the unfolded graph in Figure 8.3(b). We can check $\{{t_{1\downarrow}}^{(0)}, {t_{2\downarrow}}^{(1)}, {t_{3\downarrow}}^{(0)}\}$ to be a (minimal) reference set for pair $({t_{1\uparrow}}^{(1)}, {t_{1\downarrow}}^{(1)})$. It is not a reference set for pair $({t_{0\uparrow}}^{(1)}, {t_{0\downarrow}}^{(1)})$ because it is not a cut set for either ${t_{0\uparrow}}^{(1)}$ or ${t_{0\downarrow}}^{(0)}$. In fact, for this limited unfolding, there is no subset of transitions in $G^{(0)}$ that is a reference set for $({t_{0\uparrow}}^{(1)}, {t_{0\downarrow}}^{(1)})$. □

For any two transitions $u, v \in H(0,\infty)$, we define a random variable $\delta^*(u,v)$ which measures the maximum delay on any path from u to v. That is,

$$\delta^*(u,v) = \max_{\rho \in \mathcal{P}(u,v)} \delta(\rho). \tag{8.9}$$

Example Let us consider transition ${t_{0\uparrow}}^{(0)}$ and ${t_{0\uparrow}}^{(1)}$. There are two paths connecting the two events, namely,

$$\begin{aligned}\mathcal{P}({t_{0\uparrow}}^{(0)}, {t_{0\uparrow}}^{(1)}) \;=\; & \{({t_{0\uparrow}}^{(0)}, {p_1}^{(0)}, {t_{1\uparrow}}^{(0)}, {p_5}^{(0)}, {t_{0\downarrow}}^{(0)}, {p_{13}}^{(0)}, {t_{1\downarrow}}^{(0)}, {p_9}^{(0)}, {t_{0\uparrow}}^{(1)}), \\ & ({t_{0\uparrow}}^{(0)}, {p_1}^{(0)}, {t_{1\uparrow}}^{(0)}, {p_2}^{(0)}, {t_{2\downarrow}}^{(0)}, {p_6}^{(0)}, {t_{1\downarrow}}^{(0)}, {p_9}^{(0)}, {t_{0\uparrow}}^{(1)})\}\end{aligned}$$

Thus, $\delta^*({t_{0\uparrow}}^{(0)}, {t_{0\uparrow}}^{(1)}) = \max\{X(p_1) + X(p_5) + X(p_{13}) + X(p_9), X(p_1) + X(p_2) + X(p_6) + X(p_9)\}$. We will repeatedly use such random variables defined by δ^*. □

The following lemma gives an upper bound of the average TSE of (s,t,ε) at its k-th occurrence.

Lemma 16 *Let* $\mathcal{R}$ *be a reference set of transition pair* $(s^{(j)}, t^{(j+\varepsilon)})$. *Then,*

$$\mathbf{E}\gamma^{(j)}(s,t,\varepsilon) \le \mathbf{E}\max_{u\in\mathcal{R}} [\delta^*(u, t^{(j+\varepsilon)}) - \delta^*(u, s^{(j)})] \tag{8.10}$$

Proof First, let $\mathcal{R}'$ denote a cut set of $s^{(j)}$ s.t. $\mathcal{R} \subseteq \mathcal{R}'$. From Lemma 15, we have

$$\tau(t^{(j+\varepsilon)}) = \max_{u \in \mathcal{R}} [\tau(u) + \delta^*(u, t^{(j+\varepsilon)})], \text{ and} \tag{8.11}$$

$$\tau(s^{(j)}) = \max_{u \in \mathcal{R}'} [\tau(u) + \delta^*(u, s^{(j)})] \geq \max_{u \in \mathcal{R}} [\tau(u) + \delta^*(u, s^{(j)})] \tag{8.12}$$

where we have used the notion of δ^*. Deducting (8.12) from (8.11), we get

$$\gamma^{(j)}(s, t, \varepsilon) \leq \max_{u \in \mathcal{R}} [\tau(u) + \delta^*(u, t^{(j+\varepsilon)})] - \max_{u \in \mathcal{R}} [\tau(u) + \delta^*(u, s^{(j)})] \tag{8.13}$$

According to Lemma 15, at least one transition in $\mathcal{R}$ is critical for $t^{(j+\varepsilon)}$ under any delay assignment. Particularly, if $v \in \mathcal{R}$ is critical for $t^{(j+\varepsilon)}$, we have $\tau(v) + \delta^*(v, t^{(j+\varepsilon)}) = \tau(t^{(j+\varepsilon)})$ and hence,

$$\gamma^{(j)}(s, t, \varepsilon) \leq \delta^*(v, t^{(j+\varepsilon)}) - \delta^*(v, s^{(k)}).$$

Considering the possibility for each of the events in $\mathcal{R}$ to be critical for $t^{(j+\varepsilon)}$, we conclude,

$$\gamma^{(j)}(s, t, \varepsilon) \leq \max_{v \in \mathcal{R}} [\delta^*(v, t^{(j+\varepsilon)}) - \delta^*(v, s^{(k)})]$$

and the desired result of the lemma follows. ■

What is important to note is that the bound given by Lemma 16 is independent of the occurrence time of the events in the reference set. It only relies on the structure of the paths leading from events in the reference set to the targeted event pair and the delay distributions on the places of these paths. This implies that if there is a reference set for every sufficiently large j such that the structure between the reference set and the corresponding targeted events is identical (under the shift operator Δ) for every j, we may effectively upper bound the average TSE of (s, t, ε) by analyzing a finite segment of the infinite unfolding of the marked graph.

The proof of the lemma heavily uses the fact that under any delay assignment, there is at least one transition in the reference set that is *critical* for the event $t^{(j+\varepsilon)}$ and the corresponding timing equality given by Lemma 15, i.e., $\tau(t^{(j+\varepsilon)}) = \tau(u) + \delta^*(u, t^{(j+\varepsilon)})$ if u is critical for $t^{(j+\varepsilon)}$. However, since $\tau(s^{(j)})$ is at least $\tau(u) + \delta^*(u, s^{(j)})$ due to the fact that u has a path to $s^{(j)}$, we know $\gamma^{(j)}(s, t, \varepsilon)$ is at most $\delta^*(u, t^{(j+\varepsilon)}) - \delta^*(u, s^{(j)})$, which eliminates the occurrence time of u. Thus, under any circumstance,

$\gamma^{(j)}(s,t,\varepsilon)$ is bounded from above by $\max_{u\in\mathcal{R}}[\delta^*(u,t^{(j+\varepsilon)}) - \delta^*(u,s^{(j)})]$. The desired result follows immediately if we take the expectation of the above bound. For convenience, we write

$$U^{(j)}(\mathcal{R},s,t,\varepsilon) = \mathbf{E}\max_{u\in\mathcal{R}}[\delta^*(u,t^{(j+\varepsilon)}) - \delta^*(u,s^{(j)})] \tag{8.14}$$

in order to emphasize the fact that the resulting bound is derived from the reference set $\mathcal{R}$.

8.3.4 The bounds

In the previous subsection, we derived an upper bound for the expected j-th time separation of the targeted event pair, i.e., $\overline{\gamma}^{(j)}(s,t,\varepsilon)$, based on one of its reference sets (if there is one). In this subsection, we derive an upper bound on the average time separation of the event pair over all j, i.e., $\overline{\gamma}(s,t,\varepsilon)$. Towards this end, we show the existence of a reference set $R(s^{(j)},t^{(j+\varepsilon)})$ for pair $(s^{(j)},t^{(j+\varepsilon)})$ for every j no less than some constant $\pi(s,t,\varepsilon)$. This ensures we can always find an upper bound for $\overline{\gamma}^{(j)}(s,t,\varepsilon)$ for sufficiently large j using the result in the previous subsection. Finally, we show that such upper bounds for all sufficiently large j are the same by showing the shiftability of the corresponding reference sets under the shift operator Δ. Consequently, we can obtain an upper bound by analyzing a finite segment for only a fixed j. The remainder of this section formalizes this result.

8.3.4.1 Existence of R sets and their shiftability

That constant $\pi(s,t,\varepsilon)$ we mentioned above is related to the token distribution under the initial marking M_0. We need to define two quantities related to the token distribution before we state the result of this subsection.

Let u,v be two arbitrary transitions of a LSMG G which has an initial marking M_0. We define $\phi(u,v)$ to be the minimum number of tokens under the initial marking in any path ρ of G leading u to v. More precisely,

$$\phi(u,v) = \min_{\rho\in\mathcal{P}} \mu_{M_0}(\rho)$$

where $\mu_M(\rho)$ denotes the number of tokens on the path ρ under the marking M. For instance, $\phi(t_{3\uparrow}, t_{1\uparrow}) = 1$ and $\phi(t_{3\downarrow}, t_{1\uparrow}) = 2$ in our example marked graph (Figure 8.1(b)).

In other words, $\phi(u, v)$ measures the minimum number of tokens one must encounter in order to traverse from transition u to v under M_0. Among all transitions of G, there must be a transition u^* that maximizes this measure. That is,

$$\phi(u^*, v) = \max_{u \in T} \phi(u, v).$$

We use $\Phi(v)$ to denote this number. That is, $\Phi(v) = \phi(u^*, v)$. The importance of $\Phi(v)$ lies in the following fact. It can be shown that for any transition u of G, there must be path from $u^{(j)}$ to $v^{(j+\Phi(v))}$ for every $j \geq 0$. For instance, in Figure 8.1(b), we may check that $\Phi(t_{1\uparrow}) = 2$. As a result, any transition in the unfolded graph $G^{(0)}$ must have a path leading to $\Phi({t_{1\uparrow}}^{(2)})$.

With the foregoing definitions, we state the following theorem which shows that for sufficiently large k, there always exists a reference set for event separation triple (s, t, ε).

Theorem 16 *Let s, t be arbitrary transitions of G. For any $j \geq \pi(s, t, \varepsilon) = \max(\Phi(s), \Phi(t) - \varepsilon, 1 - \varepsilon)$, there is a subset of transitions A in $G^{(0)}$ which is a reference set for pair $(s^{(j)}, t^{(j+\varepsilon)})$. Moreover, $\Delta_k A$ is a reference set for the pair $(s^{(j+k)}, t^{(j+\varepsilon+k)})$ for every $k > 0$.*

Before we prove Theorem 16, we must first show following two simple lemmas.

Lemma 17 *Let (G, M_0) be a LSMG and H its unfolding segment. Let $j, k \in \mathbf{N}$ such that $k > j$. There is a path $\rho \in H(j, k)$ such that $t^{(j)} \overset{\rho}{\rightsquigarrow} t^{(k)}$ for every $t \in T$.*

Proof It is sufficient to show that the lemma holds for every k s.t. $k = j + 1$ and $j > 0$. However, since G is live and safe, Lemma 11 ensures t belong to some circuit $\xi \in C(G)$ such that $\mu_I(\xi) = 1$. Let p be the only marked place in ξ under initial marking. According to Lemma 13, we must have $p^{(j)} \rightsquigarrow t^{(j)} \rightsquigarrow p^{(j+1)} \rightsquigarrow t^{(j+1)}$ after unfolding G for $(j + 1)$-th time. ∎

Lemma 18 *For any transitions s, t of a LS marked graph G, there is a path ρ in $H(j, k)$ such that $s^{(j)} \overset{\rho}{\rightsquigarrow} t^{(k)}$ iff $k \geq j + \phi(s, t)$.*

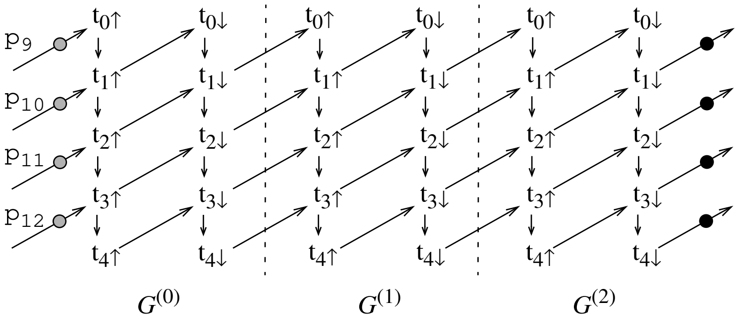


Figure 8.4: The unfolding segment $H(0,3)$ of the marked graph in Figure 8.1(b).

Proof Let $\rho \in G$ such that $s \overset{\rho}{\rightsquigarrow} t$ and $\mu_{M_0}(\rho) = \phi(s,t)$. Let $p_1, p_2, \ldots, p_{\phi(s,t)}$ be the marked places of ρ under initial marking such that $p_m \rightsquigarrow p_{m+1}$ for $1 \leq m < \phi(s,t)$. With similar argument in the proof of Lemma 17, we verify that there is a path in $H(j, j+\phi(s,t))$ along which we have $s^{(j)} \rightsquigarrow {p_1}^{(j+1)} \rightsquigarrow \cdots \rightsquigarrow {p_m}^{(j+m)} \rightsquigarrow \cdots \rightsquigarrow {p_{\phi(s,t)}}^{(j+\phi(s,t))} \rightsquigarrow t^{(j+\phi(s,t))}$. However, since $k > j + \phi(s,t)$, we have $t^{(j+\phi(s,t))} \rightsquigarrow t^{(k)}$ according to Lemma 17. Verifying the reverse part of the lemma is trivial. ■

Proof of Theorem 16 First, since $j + \varepsilon \geq 1$ as assumed by the theorem, any path leading a source node of $H(0,\infty)$ to transition $t^{(j+\varepsilon)}$ must pass through at least one transition in $G^{(0)}$. Moreover, since $j + \varepsilon \geq \Phi(t)$, every transition in $G^{(0)}$ must have path leading to $t^{(j+\varepsilon)}$. Therefore, there is at least one subset of transitions in $G^{(0)}$ which is a cut set for $t^{(j+\varepsilon)}$. Let this cut set be denoted by A. Next, for any transition v in A, $v \rightsquigarrow s^{(j)}$ according to Lemma 18 because $j = \Phi(s) \geq \phi(v,s)$. Therefore, A is reference set for $(s^{(j)}, t^{(j+\varepsilon)})$.

For the remaining of the theorem, suppose first $\Delta_k A$ is not a cut set for $t^{(j+\varepsilon+k)}$. Then, there exists a path ρ and a source node v of $H(0,\infty)$ such that $v \overset{\rho}{\rightsquigarrow} t^{(j+\varepsilon+k)}$. Further, any path must pass through the set of source nodes of $H(k,\infty)$ as implied by Lemma 12. Let ρ visit a source node $u^{(k)}$ of $H(k,\infty)$. Thus, we find a path $\rho' \subseteq \rho$ such that $\rho' \cap A = \emptyset$ but $u^{(k)} \overset{\rho'}{\rightsquigarrow} t^{(j+\varepsilon+k)}$. According to Lemma 14, we have $\Delta_{-k}\rho' \in H(0,\infty)$ such that $\Delta_{-k}\rho' \cap A = \emptyset$ but $u^{(0)} = \Delta_{-k}u^{(k)} \overset{\Delta_{-k}\rho}{\rightsquigarrow} \Delta_{-k}t^{(j+\varepsilon+k)} = t^{(j+\varepsilon)}$. This contradicts the fact that A is a cut set of for $t^{(j+\varepsilon)}$. Finally, Lemma 18 ensures every transition in $\Delta_k A$ leads to $s^{(j+k)}$. ■

Intuitively, since $j + \varepsilon \geq \Phi(t)$ and $j + \varepsilon \geq 1$ as assumed by the theorem, there must be a subset of transitions, say A, of $G^{(0)}$ which is a cut set for $t^{(j+\varepsilon)}$. Moreover, since $j \geq \Phi(s)$, every transition in set A has a path leading to $s^{(j)}$. This verifies

A is a reference set for the pair $(s^{(j)}, t^{(j+\varepsilon)})$. More importantly, it is shown that this reference set is structurally shiftable. We emphasize that the quantity $\pi(s,t,\varepsilon)$ defined in this theorem identifies a sufficient amount of net unfolding that guarantees the existence of a shiftable reference set for the separation triple. Thus, in some cases, a reference set can be found with less unfolding.

Example In the marked graph of our micropipeline example (Figure 8.1(b)), let us consider transition pair $({t_{1\uparrow}}^{(j)}, {t_{1\downarrow}}^{(j)})$ where we have set $\varepsilon = 0$. We verify that $\Phi(t_{1\uparrow}) = \Phi(t_{1\downarrow}) = 2$. Therefore, for every $j \geq \max\{\Phi(t_{1\uparrow}) = 2, \Phi(t_{1\downarrow}) - \varepsilon = 2 - 0, 1 - 0\} = 2$, the pair has a reference set in $G^{(0)}$. For instance, if we unfold the marked graph one more time from Figure 8.3 (b) as shown in Figure 8.4. We see $\{{t_{1\downarrow}}^{(0)}, {t_{2\downarrow}}^{(0)}, {t_{3\downarrow}}^{(0)}\}$ is a reference set for $({t_{1\uparrow}}^{(2)}, {t_{1\downarrow}}^{(2)})$. Besides, $\{{t_{1\downarrow}}^{(m)}, {t_{2\downarrow}}^{(m)}, {t_{3\downarrow}}^{(m)}\}$ must be a reference set for $({t_{1\uparrow}}^{(2+m)}, {t_{1\downarrow}}^{(2+m)})$ at every $m > 0$. □

8.3.4.2 Bounds on $\overline{\gamma}$

The following lemma shows the invariance property of the probability distribution of the upper bound of the TSE given in (16) under the operator Δ.

Lemma 19 *Let $R \subseteq G^{(0)}$ be a reference set for pair $(s^{(j)}, t^{(j+\varepsilon)})$ where $j \geq \pi(s,t,\varepsilon)$. For any $m > 0$, we have*

$$U^{(j+m)}(\Delta_m R, s, t, \varepsilon) = U^{(j)}(R, s, t, \varepsilon). \tag{8.15}$$

Proof With the given assumption, we have $\Delta_m R$ as a valid reference set for $(s^{(j+m)}, t^{(j+m+\varepsilon)})$ according to Theorem 16.

Consider an arbitrary transition $v^{(m)} \in \Delta_m R$. For any path $\rho \in \mathcal{P}(v^{(j+m)}, t^{(j+\varepsilon+m)})$, there is a path $\rho' \in \mathcal{P}(v^{(j)}, t^{(j+\varepsilon)})$ which is identical to ρ under operator Δ according to Lemma 14. That is, $\rho' = \Delta_{-m}\rho$ and, of course, $v^{(j)} \in R$. In addition, the delays experience by tokens flowing into a given place is assumed to independent identically distributed. Therefore, $\delta(\rho) = \sum_{p\in\rho} X(\ell(p)) \overset{L}{\sim} \sum_{p\in\rho'} X(\ell(p)) = \delta(\rho')$. Conversely, if $u^{(0)}$ is any transition in R and ρ' is a path in $\mathcal{P}(u^{(0)}, t^{(j+\varepsilon)})$, there is a path ρ in $\mathcal{P}(u^{(m)}, t^{(j+\varepsilon+m)})$ such that the sums of place delays along ρ' and ρ share the same probability law. By similar arguments, we claim the same invariance property property transitions $s^{(j)}$ and $s^{(j+m)}$. Thus, for any $v^{(m)} \in \Delta_m R$, and hence, $v^{(0)} \in R$, we have $\delta^*(v^{(m)}, t^{(j+\varepsilon+m)}) \overset{L}{\sim} \delta^*(v^{(0)}, t^{(j+\varepsilon)})$. The lemma is thus immediate by combining this observation with definition of U in (8.14). ■

This lemma ensures that the upper bound on the average TSE $\gamma^{(k)}(s,t,\varepsilon)$ given in (8.14) is valid for every sufficiently large k. In other words, this bound is shiftable. The result follows from the shiftability of the reference set when it is sufficiently far from the targeted events plus the fact that the unfolded graph has repeated segments (both structurally and i.i.d delay distributions on repeated places).

We are now ready to state our main theorem.

Theorem 17 *Let $R \subseteq G^{(0)}$ be a reference set for the pair $(s^{(\pi(s,t,\varepsilon))}, t^{(\pi(s,t,\varepsilon)+\varepsilon)})$. Then,*

$$\overline{\gamma}(s,t,\varepsilon) \leq \mathbf{E} \max_{u \in R} [\delta^*(u, t^{(\pi(s,t,\varepsilon)+\varepsilon)}) - \delta^*(u, s^{(\pi(s,t,\varepsilon))})]. \tag{8.16}$$

Proof By Theorem 14, we know that the sequence of time separations $\{\gamma^{(k)}(s,t,\varepsilon) : k > 0\}$ converges almost surely and in mean to $\overline{\gamma}(s,t,\varepsilon)$. That is, $\overline{\gamma}(s,t,\varepsilon) = \lim_{n\to\infty} \frac{1}{n} \sum_{k=1}^{n} \mathbf{E}\gamma^{(k)}(s,t,\varepsilon)$. Since $\pi(s,t,\varepsilon)$ is finite, we have

$$\overline{\gamma}(s,t,\varepsilon) = \lim_{n\to\infty} \frac{1}{n} \sum_{k=\pi(s,t,\varepsilon)}^{n} \mathbf{E}\gamma^{(k)}(s,t,\varepsilon).$$

By Lemma 16, it is upper bounded as

$$\overline{\gamma}(s,t,\varepsilon) \leq \lim_{n\to\infty} \frac{1}{n} \sum_{k=\pi(s,t,\varepsilon)}^{n} \mathbf{E}[\delta^*(u, t^{(k+\varepsilon)}) - \delta^*(u, s^{(k)})].$$

Finally, Lemma 19 ensures that for all $k \geq \pi(s,t,\varepsilon)$,

$$\mathbf{E} \max_{u \in calR} [\delta^*(u, t^{(k+\varepsilon)}) - \delta^*(u, s^{(k)})] = \mathbf{E} \max_{u \in \mathcal{R}} [\delta^*(u, t^{(\pi(s,t,\varepsilon)+\varepsilon)}) - \delta^*(u, s^{(\pi(s,t,\varepsilon))})],$$

which concludes the proof. ■

The result follows Lemma 19 and weak ergodicity theorem in Section 8.2.3 which states that the average TSE equals the average of the expected TSE at all the occurrences of the separation triple.

The bounds given in (8.16) involves taking the expectation of maximum of a set of random variables. Note that the expectation sign does not pass through the maximum sign in general. That is maximization is not linear under expectation,

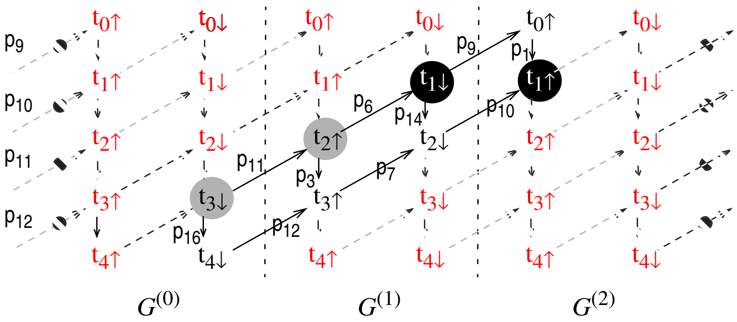


Figure 8.5: Bounding average TSE $\overline{\gamma}(t_{1\uparrow}, t1_{\downarrow}, 0)$ in a micropipeline.

which would otherwise make the evaluation of our bounds much easier. In particular, there is no easy analytical solution to the expectation of such complicated random variables. Nevertheless, this difficulty can be overcome by the techniques discussed in the next section.

8.4 Evaluating the bounds

We can image several different approaches to evaluate the upper bound derived in the previous section (given by (8.16)). For some special graph structures and delay distributions, we may come up with exact (analytical) result for the bounds. More generally, our bounds can be efficiently evaluated using standard statistical techniques such as Monte-Carlo sampling to get arbitrarily high confidence level and small error interval. Below, we briefly discuss the two approaches.

8.4.1 Analytical solution

First, let us demonstrate a possible analytical solution to the bounds using our micropipeline example.

Consider bounding $\overline{\gamma}(t_{1\downarrow}, t_{1\uparrow}, 1)$[3]. For convenience, we redraw the unfolded graph of the corresponding marked graph in Figure 8.4.1. Similar to the examples in previous sections, we see that $\{{t_{1\downarrow}}^{(0)}, {t_{2\downarrow}}^{(0)}, {t_{3\downarrow}}^{(0)}\}$ is a reference set for $({t_{1\downarrow}}^{(1)}, {t_{1\uparrow}}^{(2)})$. There are in fact many reference sets for the event pair. For instance, $R = \{{t_{3\downarrow}}^{(0)}, {t_{2\uparrow}}^{(1)}, {t_{1\downarrow}}^{(1)}\}$

[3]It can be shown that the average throughput of the micropipeline is simply the reciprocal of $\overline{\gamma}(t_{1\downarrow}, t_{1\uparrow}, 1)$.

is also a reference set for $({t_{1\downarrow}}^{(1)}, {t_{1\uparrow}}^{(2)})$ (cf., the solid part of Figure 8.4.1) . In addition, we check that $\Delta_k R = \{{t_{3\downarrow}}^{(k)}, {t_{2\uparrow}}^{(k+1)}, {t_{1\downarrow}}^{(k+1)}\}$ is a reference set for $({t_{1\downarrow}}^{(k+1)}, {t_{1\uparrow}}^{(k+2)})$ at every $k \geq 0$, which means the reference set R is shiftable. Therefore, an upper bound on $\overline{\gamma}(t_{1\downarrow}, t_{1\uparrow}, 1)$ is $\mathbf{E}\max_{v\in R}[\delta^*(v, {t_{1\uparrow}}^{(2)}) - \delta^*(v, {t_{1\downarrow}}^{(1)})]$.

Suppose each control circuit element (i.e., the C-element, inverter and the buffer) takes the same constant delay c. In the figure, we have $X(p_{11}) = X(p_{12}) = X(p_6) = X(p_7) = X(p_{10}) = c$. Besides, $X(p_{16}) = c + d_4, X(p_3) = c + d_3, X(p_{14}) = c + d_2, X(p_1) = c + d_1$ and $X(p_9) = c + d_0$, where d_0 through d_4 are the (random) data processing delays of the environments and the stages as shown in Figure 8.1. By inspection on the relevant paths in the figure, we see that $\max_{v\in R}[\delta^*(v, {t_{1\uparrow}}^{(2)}) - \delta^*(v, {t_{1\downarrow}}^{(1)})] = c + \max\{d_0 + d_1, d_2, d_3, d_4\}$ and hence an upper bound on $\overline{\gamma}(t_{1\downarrow}, t_{1\uparrow}, 1)$ is $c + \mathbf{E}\max\{d_0 + d_1, d_2, d_3, d_4\}$.

Let us assume the data process delays at each stage and the left-side environment are independent and uniformly distribution in an interval (d, D) where $(d \leq D)$. That is, $d_i \sim \text{uniform}(d, D)$. Suppose further the right-side environment always has data ready (i.e., $d_0 = 0$) so that the pipeline is fed as fast as possible. With these assumptions, we know from the *order statistics* (e.g., [25]) that $\frac{1}{D-d}[\max\{d_0 + d_1, d_2, d_3, d_4\} - d]$ has a *beta*(4,1) distribution. According to the special property of the *beta* family [25], we obtain the following result: $\mathbf{E}\frac{1}{D-d}[\max\{d_0 + d_1, d_2, d_3, d_4\} - d] = \frac{4}{5}$. Consequently, $\overline{\gamma}(t_{1\downarrow}, t_{1\uparrow}, 1)$ is upper bounded $c + \frac{1}{5}d + \frac{4}{5}D$.

This result can be easily generalized to an n-stage micropipeline ($n \geq 0$) with similar delay assumptions where we argue that $\overline{\gamma}(t_{1\downarrow}, t_{1\uparrow}, 1) \leq c + \frac{1}{n+2}d + \frac{n+1}{n+2}D$. Note that as n increases, the upper bound gets close to $c + D$ from below. From a theoretical point of view, it might also be interesting to notice that as $n \to \infty$, the above pipeline keeps a *positive* average throughput which is at least $\frac{1}{c+D}$.

With a similar reasoning, we arrive at a lower bound on $\overline{\gamma}(t_{1\downarrow}, t_{1\uparrow}, 1)$ equal to $c + \max\{\mathbf{E}(d_0 + d_1), \mathbf{E}d_2, \cdots, \mathbf{E}d_{n+1}\}$ for an n-stage micropipeline where d_1 through d_n are the (random) data processing delays at the stages, d_0 and d_{n+1}, the data processing delay at the right- and left-side environments. If we make similar delay assumptions as before (i.e., assuming identical and uniformly distributed data processing delays), we have $\overline{\gamma}(t_{1\downarrow}, t_{1\uparrow}, 1) \geq c + \frac{1}{2}(d + D)$.

8.4.2 Statistical simulation

The above analytical approach to evaluating the derived bounds can usually be applied only to restricted types of delay distributions. Moreover, for large systems, the procedure to apply such an analytical approach tends to be tedious and hard to automate. On the other hand, standard statistical techniques can be used to estimate our bounds with a sufficiently high quality in the probabilistic sense. In our experiment, we used the well-known Monte-Carlo simulation approach (e.g., [88]) which we briefly outline below. More advanced statistical techniques such as *stratified and importance sampling* [102] may be applied as well.

Recall that at the beginning of Section 8.3.3, we discussed the space $D(i,j)$ generated by all possible assignments of delays on places in the unfolding segment $H(i,j)$, $0 \leq i \leq j$. Our upper bound (8.16) $U(R,s,t,\varepsilon)$ on $\overline{\gamma}(s,t,\varepsilon)$ is the expectation of a random variable which in turn is a function of the random delay assignment drawn from $D(0,k)$ where $k = \max\{\pi(s,t,\varepsilon), \pi(s,t,\varepsilon)+\varepsilon\}$ (cf. Theorem 17). That is, $U(R,s,t,\varepsilon) = \mathbf{E}W$ where random variable $W = \max_{u\in R}[\delta^*(u, t^{(\pi(s,t,\varepsilon)+\varepsilon)}) - \delta^*(u, s^{(\pi(s,t,\varepsilon))})]$. Let F_W denote the distribution function of W.

The intuition behind Monte-Carlo simulation approach is the Central Limit Theorem [53] which deals with the partial sum of *i.i.d* random variables. In our case, if $W_1, W_2, \cdots$ are independent random variables all having distribution function F_W, then for large n, the random variable $\frac{1}{n}S_n$ approaches to $\mathbf{E}W$ where $S_n = W_1 + W_2 + \cdots + W_n$. More precisely, $\frac{1}{n}S_n$ approaches a normal distribution with mean $\mathbf{E}W$ and variance $\frac{1}{n}\sigma_W^2$ where σ_W is the variance of W. Since $\frac{1}{n}\sigma_W^2$ decrease to zero as n increases, any *realization* of the random variable $\frac{1}{n}S_n$ is a good (unbiased) estimate of $\mathbf{E}W$. In fact, the following result is well-known (e.g.,[88]). For any given relative error interval β and a confidence level $1-\alpha$, we have $P\{|\frac{1}{n}S_n - \mathbf{E}W/\mathbf{E}W| < \beta\} > \alpha$ as long as

$$n > (\frac{z_{\alpha/2}\sigma}{\beta\mathbf{E}W})^2, \tag{8.17}$$

where $z_{\alpha/2}$ is defined such that the tail probability to its right under normal distribution is $\alpha/2$. The process of this realization of $\frac{1}{n}S_n$ is commonly know as Monte-Carlo simulation approach. The process involves a random realization of n i.i.d random

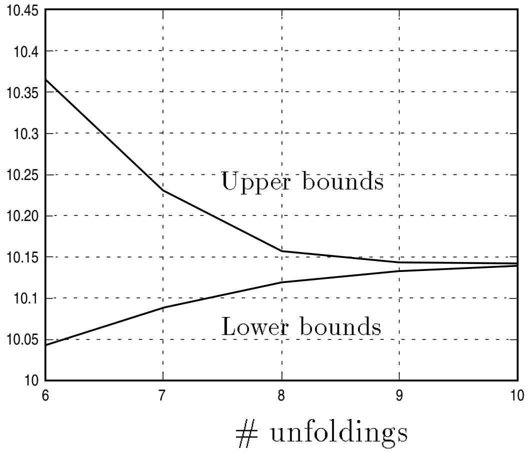


Figure 8.6: Convergence of lower and upper bounds on $\overline{\gamma}(t_{1\downarrow}, t_{1\uparrow}, 1)$ in an 8-stage micropipeline as the unfolding amount increases. (Monte-Carlo sampling w/ a relative error interval of 1% and a confidence level of 99%)

variables (W_1 through W_n) also called as a *random sample* of W with size n. Equation (8.17) is referred to as stopping criterion.

In practice, the true variance of W, i.e., σ_W^2 is not known and is commonly replaced by the variance of the sample, i.e., $S^2 = \frac{1}{n-1}\sum(W_i - \frac{1}{n}S_n)^2$. Similarly, $\mathbf{E}W$ in (8.17) is replaced by the mean of the sample, i.e., $\frac{1}{n}S_n$ itself. As a consequence, $z_\alpha/2$ in (8.17) has to be replaced by $t_\alpha/2$ which is defined such that the tail probability to its right under the student t-distribution [88] is $\alpha/2$.

The relative error interval and the confidence level can be chosen to trade the quality of the estimate with the run time. In practice, a relative error interval of 1% and confidence level of 99% are sufficient.

8.5 Improving the bounds

With the unfolding amount suggested by Theorem 17, the resulting bounds are typically much sharper than using other known method (e.g., [23, 44]). Equipped with statistical evaluation technique, we can make the bounds even sharper by simply increasing the amount of unfolding. In fact, the lower and upper bounds can converge very quickly as the unfolding amount slightly exceeds the number suggested by Theorem 17. Figure 8.5 shows the convergence behavior of the lower and upper bounds in one of our micropipeline experiments (to be discussed in the next section) for which the suggested unfolding amount by Theorem 17 is 6.

Note also that the time complexity typically increases linearly in the amount of unfolding. This is mainly because the required sample size is roughly independent of the problem size, as will be shown in the next section. This latter fact is also observed in many other applications using statistical evaluation, e.g., [21].

8.6 Experiments

The proposed bounding technique has been coded in C on a Sun UltraSparc10 with 640Mb of main memory. It was tested on asynchronous micropipelines [118] and self-timed ring circuits [124]. For a comparison of the resulting bounds, we also implemented the state-of-the-art technique proposed by Campos et al [23, 24] using Lindo [107] for the required linear programming. This section discusses the experimental results. Since Campos et al's technique cannot be easily bound quantities other than average cycle time, only average cycle time bounds are reported.

Our first set of experiments is on bounding the average cycle time of a series of asynchronous micropipelines whose circuit structure and timed marked graph model have appeared in the earlier sections (cf., Figure 8.1). The experiments are made by varying the number of stages in the pipeline as well as the distributions of component delays. Apparently, one can thus design many such experiments. In particular, we vary the number of stages from 2 to 32 and choose `uniform` and `beta` families [25] for the distribution of data processing delay of pipeline stages. Among the many possible reference sets, we chose the one that corresponds to a column of transitions in the first unfolding (i.e., $G^{(0)}$) excluding the top and bottom transitions.

The left-half of Table 8.1 lists the bounds on the average cycle time of the micropipelines where the data processing delay in each stage is uniform distributed between 1 and 11 (units). The other half of the Table lists the results of the same micropipelines but the stages experiencing a `beta`-shaped [25] processing delay also ranging from 1 and 11. The scaled `beta` distribution has a parameter pair (1,4). The two environments are set to their highest speed, i.e., they do not experience data processing delays. All the control components (C-elements, buffer and inverters) are

# stages	Uniform distributions				Beta distributions			
	Campos et al's		Ours		Campos et al's		Ours	
	Lower	Upper	Lower	Upper	Lower	Upper	Lower	Upper
2	14	32	17.28	17.29	8	20	9.798	9.800
4	14	60	19.29	19.30	8	36	11.11	11.14
8	14	116	20.19	20.24	8	68	12.01	12.01
16	14	228	20.69	20.78	8	132	12.44	12.48
32	14	452	21.05	21.09	8	260	12.68	12.72

Table 8.1: Computed bounds on the average cycle time of transitions in micropipelines. Note that all transitions in a timed marked graph have same average cycle time $\overline{\gamma}$. The relative error interval is set to 0.5% and the confidence level is set to 99.5% during Monte Carlo sampling.

# stages	# unfoldings	Uniform distributions		Beta distributions	
		Sample size	Run time (secs)	Sample size	Run time (secs)
2	4	19,359	0.58	30,277	1.05
4	6	9,217	1.01	21,610	2.63
8	10	5,626	2.96	17,425	10.35
16	18	4,490	15.50	15,877	61.70
32	36	4,260	182.1	14,885	499.9

Table 8.2: Run time statistics of the bounding technique in the micropipeline experiments.

assumed to have a unit delay. In the Monte Carlo sampling, we set a relative error interval of .5% and confidence level of 99.5%.

In all 10 experiments, our technique achieves much shaper bounds than Campos et al's. We note that the technique suggested by Ebergen and Berks [44] (which is specialized in tree-like pipeline structures) yields an upper bound of 24 for all the ten experiments.[4] While significantly better than Campos et al's upper bound, they are not as sharp as ours.[5]

[4]Strictly speaking they upper bound the *amortized worst case* response time.

[5]Note that Ebergen and Berks' technique does not compute lower bounds on the amortized worst case.

# stages	Uniform distributions				Beta distributions			
	Campos et al's		Ours		Campos et al's		Ours	
	Lower	Upper	Lower	Upper	Lower	Upper	Lower	Upper
3	26.5	28.5	26.50	26.50	22	24	22.00	22.11
4	30	38	30.00	30.04	24	32	23.96	24.00
8	60	76	59.98	60.00	48	64	47.95	48.00
16	120	152	119.93	120.08	96	128	95.89	96.14
32	240	304	239.91	240.07	192	256	191.88	192.10

Table 8.3: Computed bounds on the average cycle time of transitions in self-timed rings. The relative error interval is set to 0.5% and the confidence level is set to 99.5% during Monte Carlo sampling.

Table 8.2 lists the number of unfoldings performed, the sample size in Monte Carlo sampling and the run time for each experiment. The sample size remains roughly the same as the number of stages increases. The overall run time grows almost quadratically in the number of stages. We also note that due to the strong cyclic structure of the micropipelines, state-of-art Markovian approach (e.g., [127]) suffers from the state explosion problem and cannot handle the micropipeline models of more than 8 stages within a reasonable amount of time.

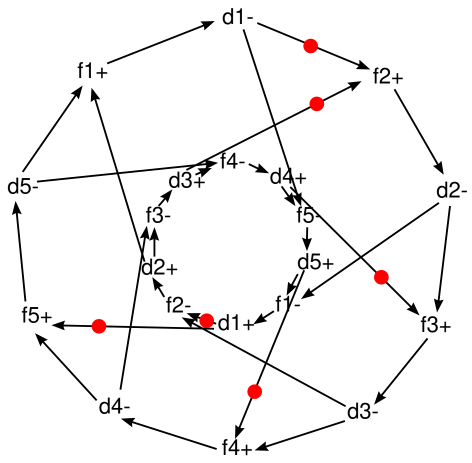


Figure 8.7: The marked graph that models a 5-stage self-timed ring circuit.

Our second set of experiments are a series of self-timed ring structures due to Williams [124]. Figure 8.7 shows the timed marked graph model of a 5-stage ring with the **PS0** configuration where each stage contains a precharging functional block.

(Places are omitted in the figure for a brevity.) With the initial marking shown in Figure 8.7, there is one data-token flowing around the ring. Due to space limitations, we refer the readers to [124] for a detailed description of the model.

Similar to the micropipeline experiments above, we vary the number of stages from 3 to 32. (At least 3 stages are required for a working ring structure [124].) For the stage *evaluation* delays (including the *completion* detection), we consider `uniform` and `beta` distributions similar to the data processing delays in the micropipeline experiments, but ranging from 5 to 10. The *precharge* delay of each stage is assumed to be the same. We choose the poset of the initial marking to be the reference set. (In fact, the poset of the initial marking is always guaranteed to be a reference set for a sufficiently unfolded net.) Table 8.3 lists the bounds on the average cycle time of the ring (i.e., the average time for a data-token to circle through the ring once). Again, the bounds obtained by our technique are much sharper than those using Campos et al's.

Chapter 9

The Free-Choice Systems

9.1 Introduction

In the previous chapter, we presented a technique to estimate the average TSEs in stochastic timed marked graphs with arbitrary delays. The technique gives upper and lower bounds on average TSEs and then estimates the average TSE by taking the mean of the derived upper and lower bounds. Consequently, unlike estimates derived from simple random simulation, the derived estimate has a well-defined error. Moreover, technique can trade-off the computational effort with the width of the derived bounds, thereby improving the quality of the estimate at the cost of additional run-time. The technique exploits the fact that any infinite timed execution of a marked graph has a unique repeated event structure. This event structure, combined with the system timing models, is used to derive closed-form expressions of average TSE bounds by effectively ignoring the history of the net execution previous to this structure. When considering one or more such repeated structure prior to the one considered, the expression yields sharper bounds.

The focus of this chapter is to extend the above technique to free-choice systems, more precisely, the systems that can be modeled by stochastic timed Petri nets with (extended) free-choice and/or unique-choice. The challenge here is that an infinite execution of such nets no long has a unique repeated structure. In fact, it has infinitely many distinct repeated structures so that the previous fixed-structure analysis is no longer applicable. To deal with this difficulty, we use statistical sampling approach to dynamically generate a sufficiently large set of repeated structures and derive bounds on performance metrics based on these sampled structures. Finally, we combine these derived bounds to obtain bounds on the targeted statics of the performance metrics.

The basic material in this chapter has been summarized in [135, 134]. The organization of the remainder of the chapter is as follows. Section 9.2 reviews stochastic timed Petri nets with choice, defines the average TSEs defined on timed executions of such models, and proves the existence of average TSEs for a large class of event pairs. Section 9.3 provides an overview of our approach, which is detailed in Sections 9.4, 9.5, 9.6, 9.7 and 9.8. Section 9.9 describes several case studies.

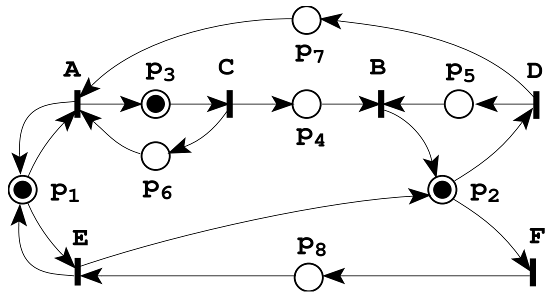
9.2 TSEs in stochastic timed Petri nets

As in Chapter 2, let $N = (P, T, F)$ denote a Petri net and $\Sigma = (N, M_0)$ the marked net of N with the initial marking being M_0. Σ is assumed to be live and safe, which implies its underlying net N is strongly connected (cf. Section 8.2). We further assume that N has only (extended) free-choice and/or unique choice places. Nets with more general choice places, i.e., asymmetric choices, is significantly more difficult to analyze and is discussed briefly in future work. Nevertheless, the models considered here still cover a large class of asynchronous systems. For details of (extended) free-choice and unique-choice nets, see Chapter 2.

The stochastic aspects of the models come in the delay assumptions on all the places and choice probability of the choice places. Both of them are described in Chapter 2. The delay assumptions have been reviewed in Chapter 8. Recall that for each free-choice place $p \in P$, there is a probability mass function (p.m.f.) μ that resolves the choice. That is, for every output transition t of p, we have $\mu(p, t)$ denoting the probability that t consumes the token in p each time it is marked.

Fig. 9.1(a) shows an example of such a net, where p_1 is unique-choice and p_2 is free-choice. Note that tokens in p_1 never needs to choose which of its two output transitions to fire since p_6, p_7 and p_8 can never be simultaneously marked.

Fig. 9.1.(b) lists a possible stochastic assumption on delay distributions and choice p.m.f. on the net. For example, the delay on place p_1 is uniformly distributed between 5 and 10, the delay on p_2 is exponentially distributed with a parameter 1.5, the delay on p_3 is 1, the delay on p_4 is `beta`-shaped with a parameter of $(1, 3)$ which



$$
\begin{aligned}
&X(p_1) \sim \texttt{uniform}(5, 10),\\
&X(p_2) \sim \texttt{exp}(1.5),\\
&X(p_3) = 1,\\
&X(p_4) \sim 1 + 5 \times \texttt{beta}(1, 3).\\
&\cdots\\
&\mu(p_2, D) = 0.6,\\
&\mu(p_2, F) = 0.4 = 1 - \mu(p_2, D).
\end{aligned}
$$

(a) The Petri net, (b) its stochastic assumption.

Figure 9.1: A stochastic Petri net example.

is located at 1 and scaled by a factor of 5, and so forth. In addition, with probability 0.6, D fires each time p_2 receives a token, and with the remaining probability, F fires. The other choice place p_1 is not assigned a p.m.f. since it is unique-choice.

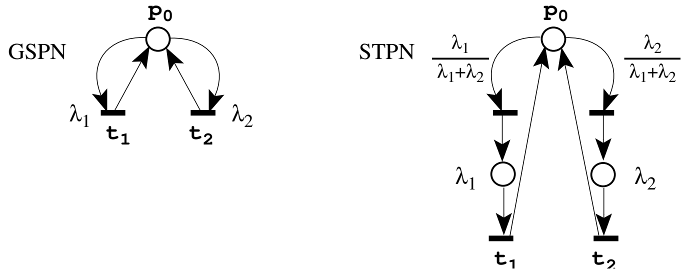


Figure 9.2: Translating GSPNs to STPNs.

One of the most general class of timed Petri nets is the GTTD_SPN where delays may be arbitrarily distributed [82]. A major difference between GTTD_SPNs and our STPN models is in the choice resolution semantics. In GTTD_SPNs, every choice, whether free-choice or asymmetric choice, is resolved through a transition race, i.e., the earliest firable transition in conflict consumes the tokens in the choice place. We call such a process as *dynamic* choice resolution. In contrast, our STPN models considered in this Chapter only support *static* choice resolution. Although GTTD_SPNs are very general, existing analytical solutions apply only to a subclass of them. GSPNs [81] fall into this subclass where the exponential GTTD_SPNs where delays are either exponentially distributed or zero. It can be shown that a GSPN without arbitration choice can be translated into a performance equivalent STPN of ours as shown in Fig. 9.2.

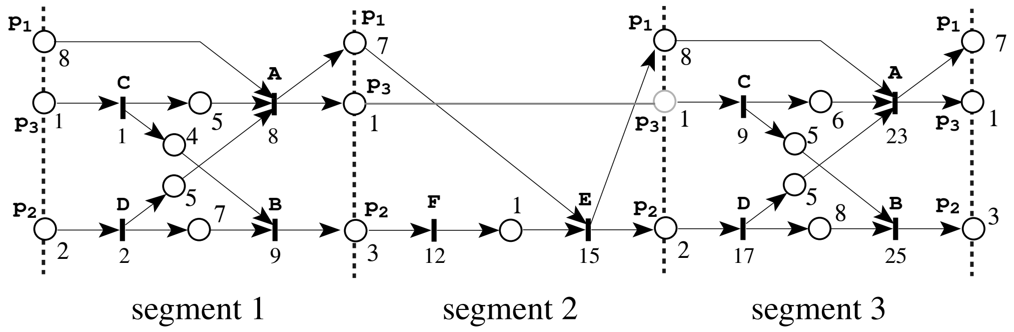


Figure 9.3: A timed execution of the stochastic Petri net in Fig. 9.1.

9.2.1 Timed executions

We call a possible run of an STPN a *timed execution* where choices are resolved and places are assigned random delays. In particular, we call a firing of a transition an *event*. A timed execution can be described as a sequence of events and their occurrence times. Alternatively, it can be depicted by a timed event graph which also describes the causality among events.

For example, Fig. 9.3 shows the event graph of a timed execution of the STPN in Fig. 9.1. In this execution, D fires after p_2 is marked for the 1st and 3rd times whereas F fires after p_2 is marked for the 2nd time. The numbers along the (instanced) places denote the delay values assigned to them. For convenience, we write $t^{(k)}$ and $p^{(k)}$ to denote the k-th event corresponding to transition t and the k-th instance of a place p, respectively. In the figure, these instance indices are dropped for brevity.

Formally, a timed execution π of Σ is a triple (N_π, d_π, ℓ) where $N_\pi = (P_\pi, T_\pi, F_\pi)$ is an acyclic event graph (or marked graph), d is a function $P_\pi \rightarrow \mathbf{R}$ that denotes the (constant) delay value of each place in P_π and a labeling function $\ell : P_\pi \cup T_\pi \rightarrow P \cup T$ that maps each instanced places and transitions to their corresponding ones in Σ. We use a function τ to denote the occurrence times of events. For the timed execution shown in Fig. 9.3, the corresponding functions d and τ are illustrated along the instanced places and transitions. For a given timed execution, the occurrence time of event $t^{(k)}$ is determined as follows:

$$\tau(t^{(k)}) = \max_{\substack{(s^{(j)}, p) \in F_\pi, \\ p \in \bullet t^{(k)}}} \tau(s^{(j)}) + d(p) \tag{9.1}$$

where the term $\tau(s^{(j)}) + d(p)$ reduces to $d(p)$ if $\bullet p = \emptyset$ in π, i.e., if p is a source place of π.

Note that for Petri nets with only free-choice and unique-choice, the set of event graph structures resulting from all possible timed executions coincides with that from all possible untimed processes [65, 108]. This fact will be exploited later to decouple the choice decisions from the timing behavior.

9.2.2 TSEs and their statistics

Given a timed execution π, the Time Separation of an Event pair (TSE) is the time distance between their occurrences. More precisely, the TSE of event pair $(s^{(k)}, t^{(k+\varepsilon)})$, denoted by, $\gamma^{(k)}(s, t, \varepsilon)$, is:

$$\gamma^{(k)}(s, t, \varepsilon) = \tau(t^{(k+\varepsilon)}) - \tau(s^{(k)}) \tag{9.2}$$

where τ is the function associated with π, and (s, t, ε) is called the corresponding *separation triple*. When considering π as a random time execution, we view the TSE $\gamma^{(k)}(s, t, \varepsilon)$ as a random variable. Consequently, the TSE sequence $\{\gamma^{(k)}(s, t, \varepsilon) : k = 1, 2, \cdots\}$ is a random process.

Definition 7 *The average TSE due to separation triple (s, t, ε) in a STPN, denoted by $\overline{\gamma}(s, t, \varepsilon)$, is the average of the corresponding TSEs of an infinite timed execution of the STPN. That is,*

$$\overline{\gamma}(s, t, \varepsilon) = \lim_{n \to \infty} \frac{1}{n} \sum_{k=1}^{n} \gamma^{(k)}(s, t, \varepsilon). \tag{9.3}$$

As we will show in Section 9.9, many system performance metrics such as average throughput and latency can be directly expressed as the average TSEs of some indicating event pairs, which is much easier than expressing them from the stationary distribution of system states.

Definition 8 *The variance of the TSE due to separation triple* (s, t, ε) *is:*

$$\sigma^2_{\gamma(s,t,\varepsilon)} = \lim_{n\to\infty} \frac{1}{n} \sum_{k=1}^{n} |\gamma^{(k)}(s, t, \varepsilon) - \overline{\gamma}(s, t, \varepsilon)|^2 \tag{9.4}$$

Equivalently, $\sigma^2_{\gamma(s,t,\varepsilon)} = \lim_{n\to\infty} \frac{1}{n} \sum_{k=1}^{n} {\gamma^{(k)}}^2(s, t, \varepsilon) - \overline{\gamma}^2(s, t, \varepsilon)$.

Definition 9 *The frequency function of the TSE due to separation triple* (s, t, ε) *is a mapping* $F_{\gamma(s,t,\varepsilon)} : \mathbf{R} \to [0, 1]$ *such that*

$$F_{\gamma(s,t,\varepsilon)}(x) = \lim_{n\to\infty} \frac{1}{n} \sum_{k=1}^{n} \mathbf{1}_{\gamma^{(k)}(s,t,\varepsilon)\leq x} \tag{9.5}$$

where the indicating function $\mathbf{1}_A$ *evaluates to 1 if the subscript expression* A *is true, and 0 otherwise.*

It is the above defined statistics that we want to characterize in this chapter. Toward this end, it is natural to first question their existence. There are cases where the average TSE does not exist. For instance, let us consider transitions D and F of the Petri net in Fig. 9.1. The difference between the numbers of occurrences of D and F will be infinite almost surely as time progresses. Consequently, their average TSE diverges almost surely for any finite occurrence offset ε provided that the net has at least one place with a positive mean delay. However, for a wide class of TSE pairs, their average TSEs exist [135]. Condition 1 below formally characterizes such a class with the aid of the notion of *steady markings*. A steady marking is one that can be reached from all reachable markings. Since the reachability graph of a LB Petri net with only free-choice and unique-choice is irreducible [13], its unique terminal strongly connected component contains all the steady markings of the net.

Condition 1 *If* M *is a steady marking and* σ *is a firing sequence such that* $M[\sigma\rangle M$, *then* $\vec{\sigma}(s) = \vec{\sigma}(t)$.

The condition requires that the net fires both transitions the same amount of times in order to traverse any cycle of steady markings. In practice, we expect this condition to be guaranteed by the user. However, it can also be checked using structural analysis for free-choice nets and a reachability analysis [131] to determine the terminal components for nets with both free-choice and unique-choice.

The following theorem [135] verifies that when a separation triple satisfies Condition 1, its TSE sequence generated by a random timed execution is *weakly ergodic* and thus the average TSE exists.

Theorem 18 *Let Σ be a LB Petri net that has only free-choices and unique-choices and satisfies stochastic assumptions given in Section 2.2.2. For any transition pair (s,t) for which Condition 1 holds, its corresponding average TSE with a fixed occurrent offset ε is a finite constant $\overline{\gamma}(s,t,\varepsilon)$ almost surely and in mean. That is,*

$$\mathrm{Prob}(\lim_{n\to\infty} \frac{1}{n}\sum_{k=1}^{n} \gamma^{(k)}(s,t,\varepsilon) = \overline{\gamma}(s,t,\varepsilon)) = 1, \tag{9.6}$$

$$\lim_{n\to\infty} \frac{1}{n}\sum_{k=1}^{n} \mathbf{E}\gamma^{(k)}(s,t,\varepsilon) = \overline{\gamma}(s,t,\varepsilon). \tag{9.7}$$

The proof of Theorem 18 sketched below uses (1) the weak ergodicity of cycle time sequence of transitions [6], and again (2) the Kingman's ergodic theorem [69] (see also Chapter 8).

In [6], Baccelli et al proved the weak ergodicity of average cycle time sequence in a wide class of discrete event systems (DES) which enter the so-called *monotone-separable* framework. Roughly speaking, the monotonicity here means delaying one event will not cause any forthcoming events events to occur earlier. The separability here is trivial, which requires that the system returns to a background (or empty) state whenever the system becomes idle. Other requirements for a system to fall into the framework include *causality* in a usual sense that the initiation of an activity should not occur earlier than its completion. Stochastic timed Petri nets with free-choice and unique choice enter this framework. Those with more general choice such as arbitration choice (timing-dependent choice) may not satisfy the above monotonicity requirement, thus may fall outside the framework. Therefore, for the stochastic Petri nets considered in this chapter, the average cycle time sequence of any transition converges to a finite number almost surely and in mean. Note that the average cycle times of different transitions are not necessarily the same.

Proof of Theorem 18 (*Sketch*) We first prove for the $\varepsilon = 0$ case. Checking the thorem for the case where $\varepsilon \neq 0$ is then trivial.

Let s, t be two transitions of Σ that verify Condition 1. From the *coupling argument* [53] plus the stationarity of a random timed execution of Σ, we deduces that there exists a positive integer $K < \infty$ such that their TSE sequence $\gamma^{(k)}(s, t, 0) = \{\tau(t^{(k)}) - \tau(s^{(k)})\}$ becomes stationary almost surely. In addition, for all $0 < l < m$, we let $Z_{l,m} = \sum_{k=l+1}^{m} \gamma^{(k)}(s, t, 0)$. Clearly, Z is additive since $Z_{l,n} = Z_{l,m} + Z_{m,n}$. for all $0 < l < m < n$.

Next, we show that for large n, $\mathbf{E}Z_{L,n}$ is bounded from below by $-c\dot{n}$ for some finite constant c and some finite integer L. From Condition 1, we know that t never can have fired infinitely many times than s. That is, there is a positive integer h such that for any firing sequence σ of Σ, $|\#_\sigma(s) - \#_\sigma(t)| \leq H$. Consequently, $\tau(t^{(k)}) \geq \tau(s^{(k+h)})$ for all $k > H$. It can then be checked that for all $n > L = \max\{K, H\}$

$$Z_{L,n} = \sum_{k=L+1}^{n} \tau(t^{(k)} - \tau(s^{(k)}) \geq - \sum_{k=n-H+1}^{n} \tau(s^{(k)}). \tag{9.8}$$

Taking expectation on both sides of (9.8), the desired bounded of $\mathbf{E}Z_{L,n}$ follows the weak ergodicity of cycle time of a transition as described earlier.

As such, $Z_{L,n}$ verifies the conditions of Kingman's ergodic theorem (cf. Chapter 8), and we deduce that the sequence $\lim_{n\to\infty} \frac{1}{n} \sum_{k=1}^{n} \gamma^{(k)}(s, t, 0) = \lim_{n\to\infty} \frac{1}{n} Z_{0,n}$ equals to a constant almost surely and in mean. For the case where $\varepsilon \neq 0$, the theorem can be checked by similar arguments.

■

At first glance, Condition 1 may seem rather restrictive. However, if there is a cycle of steady markings along which s an t fire different amount of times, then the difference of occurrence times of s and t is likely to grow unboundedly as time proceeds. According to the definition of average TSE, their average TSE $\overline{\gamma}(s, t, \varepsilon)$ is then likely to be infinite (positive or negative) for every fixed ε. As an example, let us consider transitions t_3 and t_4 of the Petri net in Figure 9.1(a). They do not satisfy Condition 1 since from the initial marking M_0 as shown, the firing sequence $t_1 t_3$ leads the net back to M_0 which can be checked as a steady marking. As a result, one can show that their average TSE does not exist since for every random firing sequence σ, $\lim_{|\sigma|\to\infty} |\vec{\sigma}(t_3) - \vec{\sigma}(t_4)| \to \infty$ almost surely, unless the choice p.m.f of place p_2 evaluates strictly the same at t_3 and t_4, i.e., $\mu(p_2, t_3) = \mu(p_2, t_4) = \frac{1}{2}$.

In some cases, it may be of interest to compute average time distance between some special occurrences of transition pair (s, t) which does not satisfy Condition 1. We state the following two cases where the *average time distance* still exists and can be computed by the approach described in this chapter.

Case 1 *The interested average time distance is between an occurrence of s to the ε-th occurrence of t following the occurrence of s.*

Such an average time distance may make sense if s occurs less frequently than t, although it does not satisfy the definition of average TSE set earlier. In fact, if only the matched occurrences of s and t are considered (by skipping the unmatched occurrences of t), Condition 1 is again met. For instance, the pair (t_3, t_4) in the above example may be considered within this case.

Case 2 *The pair (s, t) is such that for every T-invariant $\vec{v}$, if $\vec{v}(s) > 0$, $\vec{v}(t) > 0$, and vice versa. Moreover, $\vec{v}(s)/\vec{v}(t)$ is a constant.*

In this case, one may split the occurrences of each of the two transitions into the occurrences of some new transitions, and consider the average TSE of a pair of new transitions corresponding to s and t. The new transition pair satisfies Condition 1.

The existence of TSE variance for the STPNs considered in this chapter requires more consideration. In practice, the delays on places are finite or their moments are finite up to a sufficiently high order. In such cases, we believe when a TSE sequence is weakly ergodic (e.g., as characterized by Theorem 18), the existence of its variance (and in fact, any of its finite-order moments) is guaranteed.

Similarly, when the TSE sequence $\{\gamma^{(k)}(s, t, \varepsilon), k = 1, 2, \cdots\}$ is weakly ergodic, we believe its frequency function, F_γ, defined as the limit in (9.5) exists for every fixed x almost surely and in mean. In that case, we will call F_γ the *distribution* of the TSE.

Below, it is assumed that the interested transition pair satisfies Condition 1 which serves as the basis of the other two cases.

Sections 9.3, 9.4 and 9.5 are devoted to the bounding technique for average TSEs. Sections 9.6 and 9.7 extend the technique to derive bounds on TSE variance and distributions.

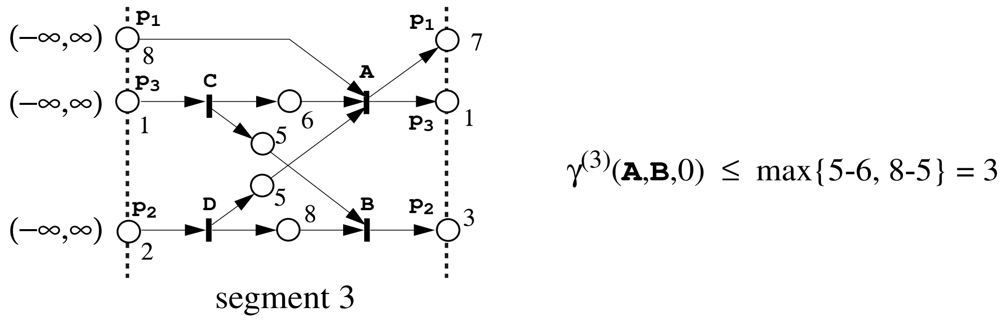


Figure 9.4: Achieving bounds on TSE by using limited history by assuming the token available time range to be $(-\infty, \infty)$for every source place.

9.3 Overview of the approach

To better describe the approach, let us first describe how to derive bounds on a TSE instance in a given timed execution using limited history. For this purpose, a timed execution is partitioned into segments. Roughly speaking, a segment is a portion of a timed execution that starts and ends at a same marking. We argue that bounds on a TSE instance can be derived using the knowledge of the segment which contains the TSE instance.

Let us consider analyzing the time separation from $A^{(2)}$ to $B^{(2)}$ in segment 3 of the timed execution shown in Figure 9.3. According to the timing in the figure, $A^{(2)}$ occurs at time 23 and $B^{(2)}$ occurs at time 25. Thus, their time separation is 5. In our approach, we will ignore the exact token available times in the source places p_1, p_2 and p_3 of segment 3, and deduce TSE bounds using only the timing within segment 3. In particular, by assuming the token available times in the source places to be anywhere within $(-\infty, \infty)$, a simple longest path analysis yields an upper bound of 3 on the TSE instance. This is illustrated in Figure 9.4. The duality of the problem gives a lower bound.

To obtain bounds on the average TSEs, our approach analyzes a finite number of timed execution segments using the path analysis alluded to above. More precisely, by randomly generating timed execution segments and analyzing each of them separately, and averaging the results, we indeed get valid bounds on the average TSE with arbitrarily high precision. In order to determine the number of segments needed to guarantee the required precision, we use a simple statistical method i.e., Monte-Carlo approach. Finally, we observe that the bounds can often be dramatically improved by using more history, i.e., using more segments prior to the one considered.

The proof of the correctness of this approach requires the development of two key facts. First, the structure of the k-th segment in a random timed execution is independent of index k (the location of the segment in the timed execution). From this, we can reason about segments that are infinitely far in the future. Secondly, multiple TSEs corresponding to a given transition pair may start in one segment and they are not independent. To overcome this dependence, we introduce a notion of a grouped TSE which is the sum of TSEs that start in a same segment, and describe an approach to compute bounds on the average grouped TSE. We then show that when divided by the average number of TSE pairs in a group, these bounds become the bounds on the average TSE.

Section 9.4 formally defines segments and grouped TSEs. Section 9.5 describes the path analysis method used to obtain the actual bounds in more detail.

9.4 Partitioning infinite timed executions

9.4.1 Definition of a segment

Let $\pi = (N_\pi, d, \ell)$ be a timed execution of a stochastic Petri net $\Sigma = (N, M_0)$. It is well-known that every (untimed) reachable marking of Σ induces a cut ξ (a set of instanced places) that partitions the event graph N_π into two parts [12]. The portion of the event graph in between two different cuts due to the same reachable marking M is called a *segment*. More precisely, if σ is a firing sequence that starts from a cut ξ and ends at another cut ξ' such that $\ell(\xi) = \ell(\xi) = M$, then the portion of N_π between ξ and ξ' is a segment and denote it by $S(\xi, \sigma)$. We say $S(\xi, \sigma)$ is *minimal* if it does not contain any other segment starting from ξ. For the timed execution shown in Figure 9.3, there are three minimal segments corresponding to the reachable marking where places p_1, p_2 and p_3 are marked. This example also shows that for Petri nets with choices, there can be many (minimal) segments whereas for those without choices (i.e, marked graphs), the minimal segment is unique. In the sequel, when we write segments, we refer to minimal ones.

One simple property of a random time execution of a Petri net considered in this chapter is that the structures of its segments are independent of each other. This is simply because the structure of a segment is determined by the choices made on the places inside the segment and choices made in different segments are independent. As a result, the sequence of segments generated by a random time execution has a property that the structure of a segment is not determined by the location of the segment in the sequence. Formally, let $\mathcal{S}(M)$ denote the set (typically infinite) of all possible segments starting from a cut due to marking M, i.e., $\mathcal{S}(M) = \{S(\xi, \sigma) \subseteq N_\pi \mid \ell(\xi) = M, \forall \sigma \in \pi, \forall \pi \in \Pi_\Sigma\}$. Let $\ell' : S(M) \to \{1, 2, \cdots\}$ gives each distinct segment structure a number (label). Function ℓ' maps the sequence of segments of a random execution into a random process of segment labels. These random labels are independent and identically distributed. This simple fact will allow us to reason about an infinite execution by considering all possible finite executions of as little as one segment in length.

9.4.2 Grouped TSEs and their weak ergodicity

Although the structures of segments are independent, as mentioned earlier, multiple TSE instances of a transition pair (s, t) may start in one segment and they can be dependent on each other. As an example, consider the time execution (shown in Figure 9.5) of the STPN model previously shown in Figure 2.4. There are two TSE instances of pair (t_4, t_2) starting from segment 3 and their TSE values not independent. This motivates us to consider all the TSE instances starting from a same segment together as a *group*.

To formally define a TSE group, let us consider a TSE sequence $\{\gamma^{(j)}(s, t, \varepsilon) : j \geq 1\}$ generated by a random timed execution π of $\Sigma = (N, M_0)$. The segment sequence $\{S^{(l)} \in \mathcal{S}(M_0) : l \geq 1\}$ of π partitions the TSE sequence into sub-sequences. Two TSEs $\gamma^{(i)}(s, t, \varepsilon)$ and $\gamma^{(j)}(s, t, \varepsilon)$ are contained in one sub-sequence if their corresponding source events $s^{(i)}$ and $s^{(j)}$ are in a same segment.

As an example, considered the TSEs due to triple $(t_4, t_2, 0)$ of the STPN model in Figure 2.4. For the timed execution of the net shown in Figure 9.5, the first TSE group of the TSE triple is $\{({t_4}^{(1)}, {t_2}^{(1)})\}$ (due to segment 2). Its second TSE group is $\{({t_4}^{(2)}, {t_2}^{(2)}), ({t_4}^{(3)}, {t_2}^{(3)})\}$ since ${t_4}^{(2)}$ and ${t_4}^{(3)}$ are both in the segment 3.

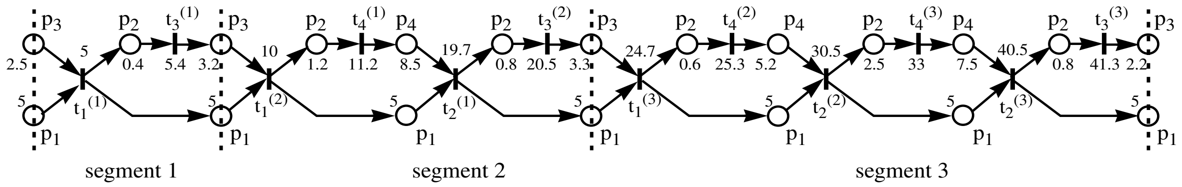


Figure 9.5: A timed execution of the stochastic Petri net in Figure 2.4.

We denote the index of the last TSE in group k by $\eta^{(k)}(s)$. That is,

$$\eta^{(k)}(s) = \max\{i > \eta^{(k-1)}(s) : \\ \exists l, s^{(\eta^{(k-1)}(s)+1)} \in S^{(l)}, s^{(i+1)} \notin S^{(l)}\},$$

where $\eta^{(0)}(s) \triangleq 0$. For convenience, we define the number of TSEs in group k the *length* of the group, denoted by $\alpha^{(k)}(s)$. That is,

$$\alpha^{(k)}(s) = \eta^{(k+1)}(s) - \eta^{(k)}(s).$$

Finally, we define the sum of all TSEs in group k as k-th *grouped TSE* denoted by $\beta^{(k)}(s,t,\varepsilon)$. That is

$$\beta^{(k)}(s,t,\varepsilon) = \sum_{l=\eta^{(k-1)}(s)+1}^{\eta^{(k)}(s)} \gamma^{(l)}(s,t,\varepsilon).$$

In the above example, we have $\alpha^{(1)}(t_4) = 1$, $\alpha^{(2)}(t_4) = 2$, and $\beta^{(1)}(t_4,t_2,0) = \tau({t_2}^{(1)}) - \tau({t_4}^{(1)}) = 19.7 - 11.2 = 8.5$, $\beta^{(2)}(t_4,t_2,0) = (\tau({t_2}^{(2)}) - \tau({t_4}^{(2)})) + (\tau({t_2}^{(3)}) - \tau({t_4}^{(3)})) = (30.5 - 25.3) + (40.5 - 33) = 12.7$.

We will show that the sequence of grouped TSEs satisfies a similar weak ergodic property as the sequence of TSEs, i.e., its average converges to a finite number, which will enable us to estimate the average TSE.

The following theorem verifies that both the group length sequence and the grouped TSE sequence exhibit similar ergodicity property to that of the TSE sequence stated in Theorem 18.

Theorem 19 *Let Σ be a stochastic LS Petri net that has only free-choices and unique-choices, and (s,t) be a transition pair for which Condition 1 holds. Then, for any fixed ε, the average of the grouped TSE sequence (group length sequence)*

$\{\beta^{(k)}(s,t,\varepsilon) : k \geq 1\}$ $(\{\alpha^{(k)}(s) : k \geq 1\})$ *converges to a finite constant* $\overline{\beta}(s,t,\varepsilon)$ $(\overline{\alpha}(s))$. *That is,*

$$\frac{1}{n}\sum_{k=1}^{n}\beta^{(k)}(s,t,\varepsilon) \quad\rightarrow\quad \overline{\beta}(s,t,\varepsilon), \tag{9.9}$$

$$\frac{1}{n}\sum_{k=1}^{n}\alpha^{(k)}(s) \quad\rightarrow\quad \overline{\alpha}(s). \tag{9.10}$$

The convergence in (9.9) and (9.10) takes almost surely and in mean. Moreover, $\overline{\beta}(s,t,\varepsilon) = \overline{\alpha}(s)\overline{\gamma}(s,t,\varepsilon)$ *where* $\overline{\gamma}(s,t,\varepsilon)$ *is the average TSE of the triple* (s,t,ε).

Proof The proof of this theorem is straightforward with the result of Theorem 18 where we showed the similar convergence modes for the (running) average of the TSE sequences.

Let us first show that the average grouped TSE sequence $\{\frac{1}{n}\sum_{k=1}^{n}\beta^{(k)}(s,t,\varepsilon) : n \geq 0\}$ converges almost surely for every triple (s,t,ε) such that (s,t) verifies Condition 1 and ε being finite. Towards this end, let us consider a random timed execution π of Σ. We introduce a random variable $h(n)$ which denotes the total number of TSE pairs that start in any of the first n TSE groups of π, namely, $h(n) = \sum_{k=1}^{n}\alpha^{(k)}(s)$. Apparently, $h(n)\rightarrow\infty$ if $n\rightarrow\infty$ since $\alpha^{(k)}(s) \geq 1$ for every k. Now,

$$\lim_{n\rightarrow\infty}\frac{1}{n}\sum_{k=1}^{n}\beta^{(k)}(s,t,\varepsilon) \quad=\quad \lim_{n\rightarrow\infty}\frac{h(n)}{n}\cdot\frac{1}{h(n)}\sum_{k=1}^{n}\beta^{(k)}(s,t,\varepsilon) \tag{9.11}$$

$$= \quad \lim_{n\rightarrow\infty}\frac{h(n)}{n}\cdot\frac{1}{h(n)}\sum_{l=1}^{h(n)}\gamma^{(l)}(s,t,\varepsilon) \tag{9.12}$$

$$= \quad \lim_{n\rightarrow\infty}\frac{h(n)}{n}\cdot\overline{\gamma}(s,t,\varepsilon),\ \text{almost surely.} \tag{9.13}$$

Equation (9.13) follows (9.12) since $\lim_{h(n)\rightarrow\infty}\frac{1}{h(n)}\sum_{l=1}^{h(n)}\gamma^{(l)}(s,t,\varepsilon)$ *exists* and it equals to the constant $\overline{\gamma}(s,t,\varepsilon)$ almost surely according to Theorem 18.

Notice that $\alpha^{(k)}(s)$ and $\alpha^{(j)}(s)$ are independent and identically distributed (i.i.d) random variables since the labeling of segments $S^{(k)}$ and $S^{(j)}$ are i.i.d variables (as described in Section 9.4.1). According to the Central Limit Theorem [53], we have

$$\lim_{n\rightarrow\infty}\frac{h(n)}{n} = \lim_{n\rightarrow\infty}\frac{1}{n}\sum_{n\rightarrow\infty}\alpha^{(n)} = \mathbf{E}\alpha^{(0)}(s),\ \text{almost surely and in mean,} \tag{9.14}$$

which proves the part of the theorem concerning the group length sequence.

With (9.14), the almost surely convergence of the average grouped TSE sequence follows (9.13). In addition, we see $\overline{\beta}(s,t,\varepsilon) = \overline{\alpha}(s)\overline{\gamma}(s,t,\varepsilon)$ where $\overline{\alpha}(s) = \mathbf{E}\alpha^{(0)}(s)$.

Using similar arguments, we check the grouped TSE sequence also converges in mean by the fact that both the average group length sequence and the average TSE sequence converge in mean. ■

This theorem also states the relationship between the average TSE $\overline{\gamma}(s,t,\varepsilon)$ and the average grouped TSE $\overline{\beta}(s,t,\varepsilon)$. Note that since the labels of segment structures are i.i.d for a random time execution, the average group length $\overline{\alpha}(s)$ is merely the average length of group k (k fixed) over all possible executions, i.e., $\overline{\alpha}(s) = \mathbf{E}\alpha^{(k)}(s)$. In Section 9.8, we use simple statistical method to evaluated $\mathbf{E}\alpha^{(k)}(s)$ as well as the bounds on $\overline{\beta}(s,t,\varepsilon)$, and consequently we get bounds on $\overline{\gamma}(s,t,\varepsilon)$.

9.5 Deriving the bounds on average TSEs

As mentioned earlier, we obtain bounds on the grouped TSE via path analysis of random segments. The intuition behind the bounds is that we ignore the time when tokens in the source places of the segments become available. To do this, we identify a set of *reference events* which can be considered as synchronization points for the targeted event pair, say (e_1, e_2). To obtain an upper bound on the time separation between the two events, we first assume a particular synchronization point, say event f, is critical for e_1, and then compute a time separation of the pair (e_1, e_2) as if all other synchronization points are not critical for e_2. This separation serves an upper bound on the actual TSE of the pair with the assumption that f is critical for e_1. Finally, we get an upper bound on the TSE of the pair which is the largest of all such upper bounds obtained by individually assuming each of the synchronization points is critical for e_1.

9.5.1 The duality of bounds

In many timing analysis problems, finding a lower bound on a delay variable can often be transformed to finding an upper bound on a related delay variable. This is also true in our case.

In particular, for a given ε, and any $j \in \{1, 2, \cdots\}$ s.t. $j > -\varepsilon$, we have $-\gamma^{(j)}(s,t,\varepsilon) = -t^{(j+\varepsilon)} + s^{(j)} \triangleq \gamma^{(j+\varepsilon)}(t,s,-\varepsilon)$ from the definition of γ. Thus, if one finds an upper bound U on $\overline{\gamma}(t,s,-\varepsilon)$, then $-U$ is a lower bound on $\overline{\gamma}(s,t,\varepsilon)$. Because of this duality, we shall be concerned only with the upper bounds from now on.

9.5.2 The $\varepsilon = 0$ case

The desired goal is to find an upper bound on $\overline{\beta}(s,t,\varepsilon)$. To ease the exposition of the technique, we take the occurrence offset-index ε to be 0. Extension of the result to the case where $\varepsilon \neq 0$ is not difficult and is thus omitted.

Our approach first derives an upper bound ${U_\beta}^{(k)}(s,t,\varepsilon)$ on $\beta^{(k)}(s,t,\varepsilon)$ for a fixed $k \geq 1$, which implies that $\mathbf{E}{U_\beta}^{(k)}(s,t,\varepsilon)$ is an upper bound on $\mathbf{E}\beta^{(k)}(s,t,\varepsilon)$. We then show $\mathbf{E}{U_\beta}^{(k)}(s,t,\varepsilon)$ is *independent* of k, which implies it is also an upper bound on $\overline{\beta}(s,t,\varepsilon)$ following the weak ergodicity property of the grouped TSE sequence stated in Theorem 19.

We need the following notations to formulate the upper bound $\mathbf{E}{U_\beta}^{(k)}(s,t,\varepsilon)$. Let $\pi = (N_\pi, d, \ell)$ be a random timed execution of Petri net Σ. For every path $\rho \in N_\pi$, we denote by a random variable $\delta(\rho)$ the sum of random delays assigned to all the places along path ρ, namely, $\delta(\rho) = \sum_{p \in \rho} d(p)$. Since π is considered as a random timed execution, the delay $d(p)$ assigned to every place $p \in P_\pi$ is also a random variable whose distribution function is the same as that of $X(\ell(p))$.

Let D be a particular delay assignment $D \in \otimes_{p \in P_\pi} X(\ell(p))$, $d_D(p)$ denote the delay assigned to place $p \in P_\pi$ under D, and $\delta_D(\rho)$ denote the value of $\delta(\rho)$ under D. For any $x, y \in P_\pi \cup T_\pi$, we denote by $\mathcal{P}(x,y)$ the set of all paths leading from x to y, i.e., $\mathcal{P}(x,y) = \{\rho \in N_\pi | x \rightsquigarrow y\}$. From the timing relation (9.1) (cf., Section 9.2.1), we note that if there is a path from event x to y, y must occur after x by at least the maximum sum of delays on all the paths from x to y. That is, whenever $\mathcal{P}(x,y) \neq \emptyset$, we have

$$\tau(x) + \max_{\rho \in \mathcal{P}(x,y)} \delta(\rho) \leq \tau(y). \tag{9.15}$$

Under a given delay assignment D, we say that x is *critical* for y if (9.15) holds in equality. In that case, there is a path $\rho^* \in \mathcal{P}(x,y)$ such that $\delta_D(\rho^*) = \tau_D(y) - \tau_D(x)$, and we call ρ^* a critical path from x to y.

The criticality of timing along paths can be described using a concept of *reference sets*. Roughly speaking, a reference set for event e in a timed execution π is a subset of events in π such that every path from a source place of N_π to e contains at least one event in R, and every event in R has a path to e.

For example, in Figure 9.5, it can be checked that $\{{t_1}^{(1)}\}$ is a *minimal* reference set for all the events in the execution. On the other hand, $\{{t_3}^{(1)}\}$ is not a reference set for all other events except for ${t_3}^{(1)}$ itself, because there is at least one path from ${t_1}^{(1)}$ to all other events which does not contain ${t_3}^{(1)}$.

The significance of a reference set is that we can determine the occurrence time of an event e by knowing only the occurrence times of the events contained in a reference set of e plus the delay values of places following the events in the reference set. That is, the following timing relation holds for e, where R is a reference set of e.

$$\tau(e) = \max_{e' \in R}[\tau(e') + \max_{\rho \in \mathcal{P}(e',e)} \delta(\rho)]. \tag{9.16}$$

The term $\max_{\rho \in \mathcal{P}(e',e)} \delta(\rho)$ in (9.16) measures the maximum (random) delay on any path from event e' to e which will be repeatedly referred to later. For convenience, we write:

$$\delta^*(e', e) = \max_{\rho \in \mathcal{P}(e',e)} \delta(\rho), \tag{9.17}$$

where $\delta^*(e', e) \triangleq -\infty$ if there is no path from e' to e, i.e., $\mathcal{P}(e', e) = \emptyset$. For completeness, we define an event e itself to be a path of delay 0, and consequently $\delta^*(e, e) = 0$.

Suppose the TSEs of k-th group start in the j-th segment of π, i.e., $S^{(l)}$. Since the set of source places of a segment is a cut of N_π, the poset of the source places of $S^{(l)}$ must contain a reference set for every event e in the segment (in fact, for every event in $S^{(l')} if l' \geq l$). For convenience, if event $e \in S^{(l')}(l' \geq l)$, let us denote by $R(e, l)$ its reference set contained in the poset of the source places of segment $S^{(l)}$.

To give an upper bound on the k-th grouped TSE $\beta^{(k)}$, we first give an upper bound on every TSE in the group. Recall that we used $\alpha^{(k)}$ to denote the number of TSE pairs in group k and the last TSE instance of the group is indexed by $\eta^{(k)}$ in the global TSE sequence. For every TSE $\gamma^{(m)}(s, t, \varepsilon)$ in the group (thus

$\eta^{(k-1)} < m \leq \eta^{(k)}$), Equation (9.18) computes an upper bound $U_\gamma{}^{(m)}(s,t,0)$ on it, which we state as a lemma.

Lemma 20 *Suppose the m-th TSE of the triple $(s,t,0)$, i.e., $\gamma^{(m)}(s,t,0)$, occurs in segment $S^{(l)}$. Then, it is upper bounded by $U_\gamma{}^{(m)}(s,t,0)$ defined below where $R(t^{(m)},l)$ is the reference set for event $t^{(m)}$ contained in the poset of the source places of $S^{(l)}$.*

$$U_\gamma{}^{(m)}(s,t,0) = \max_{e \in R(t^{(m)},l)} [\delta^*(e,t^{(m)}) - \delta^*(e,s^{(m)})] \tag{9.18}$$

Proof The lemma is proved by a common practice of critical path analysis. Let $R(t^{(m)},l)$ be the reference set for event $t^{(m)}$ contained in the poset of the source places of $S^{(l)}$. From Condition 1, we know that $s^{(m)}$ is also in segment l, and denote its reference set contained the poset of $S^{(l)}$ by $R(s^{(m)},l)$.

From timing relation (9.16) and (9.17), we have

$$\begin{aligned}
\tau(t^{(m)}) &= \max_{e \in R(t^{(m)},l)} [\tau(e) + \delta^*(e,t^{(m)})], \text{ and} && (9.19)\\
\tau(s^{(m)}) &= \max_{e \in R(s^{(m)},l)} [\tau(e) + \delta^*(e,t^{(m)})]. && (9.20)
\end{aligned}$$

Thus,

$$\begin{aligned}
&\tau(t^{(m)}) - \tau(s^{(m)}) \\
&= \max_{e \in R(t^{(m)},l)} [\tau(e) + \delta^*(e,t^{(m)})] - \max_{e \in R(s^{(m)},l)} [\tau(e) + \delta^*(e,t^{(m)})] && (9.21)\\
&\leq \max_{e \in R(t^{(m)},l)} [\tau(e) + \delta^*(e,t^{(m)}) - (\tau(e) + \delta^*(e,s^{(m)}))] && (9.22)\\
&= \max_{e \in R(t^{(m)},l)} [\delta^*(e,t^{(m)}) - \delta^*(e,s^{(m)})] && (9.23)
\end{aligned}$$

where (9.22) follows (9.21) properly because $\delta^*(e,e')$ is defined to be $-\infty$ if there is no path leading from e to e' (cf. (9.17)). ■

The critical fact concerning the above upper bound $U_\gamma{}^{(m)}(s,t,0)$ on $\gamma^{(m)}(s,t,0)$ is that it is independent of the occurrence time of the events in the reference set of $s^{(m)}$. In other words, it relies only on the structure of segment $S^{(l)}$ and the delays of the places within the segment. In particular, it does not depend on the history of the timed execution prior to this segment.

Following Lemma 20, an upper bound on the k-th grouped TSE $\beta^{(k)}$ is merely the sum of upper bounds on the TSEs with the group, i.e.,

$$\beta^{(k)}(s,t,0) \le U_\beta{}^{(k)}(s,t,0) = \sum_{m=\eta^{(k-1)}+1}^{\eta^{(k)}} U_\gamma{}^{(m)}(s,t,0). \tag{9.24}$$

By taking the expectation of $U_\beta{}^{(k)}(s,t,0)$, it is now straightforward to show that the average of $\beta^{(k)}(s,t,0)$ over all possible timed executions is upper bounded by $\mathbf{E}U_\beta{}^{(k)}(s,t,0)$. Finally, coupling this upper bound with the strong stationarity of the segment sequence described in Section 9.4.1, we are now ready to state our main result of this subsection which gives the upper bound on the average grouped TSE $\overline{\beta}(s,t,0)$.

Theorem 20 *Let (s,t) a transition pair of a stochastic Petri net Σ for which Condition 1 holds. Then, the average grouped TSE $\overline{\beta}(s,t,\varepsilon=0)$ is upper bounded by $\mathbf{E}U_\beta{}^{(1)}(s,t,0)$. That is,*

$$\overline{\beta}(s,t,0) \le \mathbf{E}U_\beta{}^{(1)}(s,t,0) = \mathbf{E}\sum_{0<m\le\eta^{(1)}} U_\gamma{}^{(m)}(s,t,0), \tag{9.25}$$

where[1] *$U_\gamma{}^{(m)}(s,t,0)$ is given by (9.18).*

Proof Following Lemma 20, an upper bound on the k-th grouped TSE $\beta^{(k)}$ is merely the sum of upper bounds on the TSEs with the group, i.e., $\beta^{(k)}(s,t,0) \le U_\beta{}^{(k)}(s,t,0) = \sum_{m=\eta^{(k-1)}+1}^{\eta^{(k)}} U_\gamma{}^{(m)}(s,t,0)$. By taking the expectation of $U_\beta{}^{(k)}(s,t,0)$, it is now straightforward to show that the average of $\beta^{(k)}(s,t,0)$ over all possible timed executions is upper bounded by $\mathbf{E}U_\beta{}^{(k)}(s,t,0)$. Finally, coupling this upper bound with the strong stationarity of the segment sequence described in Section 9.4.1, we achieve an upper bound on the average grouped TSE $\overline{\beta}(s,t,0)$ as given in the theorem. Specifically, we merely need the convergence *in mean* of the average grouped TSE sequence shown in Theorem 19 in the last statement. ■

Remark 2 It might be useful to note that in a random timed execution, the grouped TSEs $\beta^{(k)}(s,t,\varepsilon)$ and $\beta^{(j)}(s,t,\varepsilon)$ are generally not independent for $j \neq$

[1] Note also that the expectation operator in the rightmost-side of (9.25) does not pass the summation sign since the number of terms under the summation sign, i.e., $\alpha^{(k)} = \eta^{(k)} - \eta^{(k-1)}$, is a random variable.

k. However, their upper bounds $U_{\beta^{(k)}} = \sum_{m=\eta^{(k-1)}+1}^{eta^{(k)}} U_{\gamma}{}^{(m)}(s,t,0)$ and $U_{\beta^{(j)}} = \sum_{m=\eta^{(j-1)}+1}^{eta^{(j)}} U_{\gamma}{}^{(m)}(s,t,0)$ (derived from Lemma 20) are independent (and identically distributed). It is this key fact that makes it possible for us to give bounds on the average (grouped) TSE sequence in a random infinitely long timed execution.

Theorem 20 states the fact that we can give an upper bound on the average grouped TSE $\overline{\beta}(s,t,0)$ by analyzing only one group of a fixed index, say, an index of 0, and over all possible timed executions

9.5.3 The $\varepsilon \neq 0$ case

We now extend our upper bound on $\overline{\beta}(s,t,\varepsilon=0)$ derived in the previous subsection to the case where $\varepsilon \neq 0$. First, we argue that an upper bound on every TSE in group k can be obtained which is similar to the one in the case of $\varepsilon = 0$.

Note that the only difference between the two cases is that when $\varepsilon \neq 0$, the source and destination events of particular TSE pair $(s^{(n)}, t^{(n)})$ may fall into different segments in the segment sequence. Generally, let us consider the triple (s,t,ε) and its k-th group in a random timed execution π. Suppose again that this group starts in segment $S^{(l)}$ of the event graph N_π. Note that all the $\alpha^{(k)}$ source events (i.e., $s^{(m)}, \eta^{(k-1)} < m \leq \eta^{(k)}$) are within the segment $S^{(l)}$. Since transitions s and t satisfy Condition 1, we count that there are $\min\{\alpha^{(k)}, |\varepsilon|\}$ destination events falling outside segment $S^{(k)}$.

Let us first consider the case where $\varepsilon > 0$. In that case, the destination events are either within segment $S^{(l)}$ or in segments following $S^{(l)}$. Therefore, the poset of the source places of segment $S^{(l)}$ constrains a reference set for every destination events (namely, $t^{(m+\varepsilon)}, \eta^{(k-1)} < m \leq \eta^{(k)}$) of the TSE pairs in the k-th group. Following the notation in the previous subsection, we denote such a reference set for destination event $t^{(m+\varepsilon)}$ by $R(t^{(m+\varepsilon)}, l)$. Consequently, similar to the result of Lemma 20 in the $\varepsilon = 0$ case, we again have an upper bound $U_{\gamma}{}^{(m)}(s,t,\varepsilon)$ on the m-th TSE $\gamma^{(m)}(s,t,\varepsilon)$ but using the reference set $R(s^{(m+\varepsilon)}, l)$ (instead of $R(s^{(m)}, l)$) where

$$U_{\gamma}{}^{(m)}(s,t,\varepsilon) = \max_{e \in R(t^{(m+\varepsilon)}, l)} [\delta^*(e, t^{(m+\varepsilon)}) - \delta^*(e, s^{(m)})] \tag{9.26}$$

for every $m \in (\eta^{(k-1)}, \eta^{(k)}]$. Following a similar argument as in the $\varepsilon = 0$ case (Theorem 20), we deduce an upper bound on $\overline{\beta}(s,t,\varepsilon)$. More precisely, we have

$$\overline{\beta}(s,t,\varepsilon) \leq \mathbf{E}{U_\beta}^{(0)}(s,t,\varepsilon) = \mathbf{E} \sum_{0 \leq m \leq \eta^{(0)}} {U_\gamma}^{(m)}(s,t,\varepsilon). \tag{9.27}$$

The case here $\varepsilon < 0$ is no more different than the $\varepsilon > 0$ case except that some of the destinations events now appear in the segments prior to segment $S^{(l)}$. We count that the first destination event of the TSE pairs in k-th group to be $t^{(\eta^{(k-1)}+\varepsilon+1)}$ since the corresponding first source event is $s^{(\eta^{(k-1)}+1)}$. Let $S^{(l')}(l' < l)$ be the segment containing event $t^{(\eta^{(k-1)}+\varepsilon+1)}$. Then, the poset of the source place of segment $S^{(l')}$ contains a reference set for every source and destination events of the TSE pair in k-th group. Following the similar argument to the one made in the $\varepsilon > 0$ case, we upper bound $\overline{\beta}(s,t,\varepsilon < 0)$ by $\mathbf{E}{U_\beta}^{(l)}(s,t,\varepsilon)$. To be precise, we have,

$$\mathbf{E}{U_\beta}^{(l)}(s,t,\varepsilon) = \mathbf{E} \sum_{\eta^{(l-1)} < m \leq \eta^{(l)}} {U_\gamma}^{(m)}(s,t,\varepsilon), \tag{9.28}$$

where l is a random variable determined as $l = \min\{j+1 | \eta^{(j)} \geq -\varepsilon\}$. That is, l indicates the first segment in the segment sequence of π such that all its previous segments contain at least $|\varepsilon|$ occurrences of transition t. In addition, each of the terms ${U_\gamma}^{(m)}(s,t,\varepsilon)$ in (9.28) is now determined by (9.29) below,

$${U_\gamma}^{(m)}(s,t,\varepsilon) = \max_{e \in R(t^{(m+\varepsilon)}, l')} [\delta^*(e, t^{(m+\varepsilon)}) - \delta^*(e, s^{(m)})] \tag{9.29}$$

with l' being the index of the segment that contains the 0-th group of the TSE sequence.

Before closing this section, we discuss several possible algorithms to compute the upper bound on a grouped TSE, i.e., ${U_\beta}^{(k)}(s,t,0)$ in (9.24). Since a segment structure is acyclic, one possible approach is to use the TSE approximate algorithm described in [27] by assigning a delay interval $(-\infty, \infty)$ to every source place of the segment being considered. The algorithm has a complexity no lower than $O(|T(S)|^2)$ where $|T(S)|$ is the number of events in segment S.

However, since all the internal places of a segment in a given timed execution has a fixed delay (only the delay values of source places whose delay values are unknown), the problem can be solved more efficiently. In particular, using a simple longest path analysis from a fixed event e, the term $\delta^*(e, t^{(m)})$ in (9.18) can be determined in $O(|T(S)|+|P(S)|)$ time where $|P(S)|$ is the number of places in segment S. Thus, $U_\gamma{}^{(m)}(s,t,0)$ in (9.18) can be determined in $O((|T(S)|+|P(S)|)*|R|)$ time where $|R|$ is the size of the referent set of e. As a result, $U_\beta{}^{(k)}$ in (9.24) can be determined in $O((|T(S)|+|P(S)|)*|R|)$ time, which is typically much faster than using the method in [27]. Alternatively, the longest path analysis can be conducted from each of the interested events back towards their reference events. This way, $U_\beta{}^{(k)}$ is determined in $O((|T(S)| + |P(S)|) * \eta^{(k)})$ time where $\eta^{(k)}$ is the length of k-th TSE group.

9.6 Bounding TSE variance

The following inequality gives the bounds on $\sigma^2(s,t,\varepsilon)$:

$$\lim_{n\to\infty} \frac{1}{n} \sum_{k=1}^{n} \min\{{U_\gamma{}^{(k)}}^2(s,t,\varepsilon), {L_\gamma{}^{(k)}}^2(s,t,\varepsilon)\} - \max\{\overline{U_\gamma}^2(s,t,\varepsilon), \overline{L_\gamma}^2(s,t,\varepsilon)\}$$
$$\leq \sigma^2_{\gamma(s,t,\varepsilon)} \leq$$
$$\lim_{n\to\infty} \frac{1}{n} \sum_{k=1}^{n} \max\{{U_\gamma{}^{(k)}}^2(s,t,\varepsilon), {L_\gamma{}^{(k)}}^2(s,t,\varepsilon)\} - \min\{\overline{U_\gamma}^2(s,t,\varepsilon), \overline{L_\gamma}^2(s,t,\varepsilon)\}$$

where $\overline{U_\gamma}(s,t,\varepsilon)$ and $\overline{L_\gamma}(s,t,\varepsilon)$ are the means of the upper and lower bounds on the TSE, respectively. Should the above lower bound be negative, we take its trivial lower bound on the variance, zero.

Similar to the technique in bounding the average TSEs, the dependency among upper and lower bounds on the TSEs due to a same segment can be avoided by grouping the squares of the upper and lower bounds on individual TSEs. Consequently, the grouped sum of squares of the TSE bounds are iid due to iid property of the random segments.

9.7 Bounding TSE distributions

For the k-th TSE instance of the separation triple, the following relation holds:

$$\mathbf{1}_{U_\gamma{}^{(k)}(s,t,\varepsilon)\leq x} \leq \mathbf{1}_{\gamma^{(k)}(s,t,\varepsilon)\leq x} \leq \mathbf{1}_{L_\gamma{}^{(k)}(s,t,\varepsilon)\leq x} \tag{9.30}$$

where $U_{\gamma^{(k)}}(s,t,\varepsilon)$ and $L_{\gamma^{(k)}}(s,t,\varepsilon)$ are the upper and lower bounds on the TSE instance $\gamma^{(k)}(s,t,\varepsilon)$, respectively. The quantities $\mathbf{1}_{U_{\gamma^{(k)}}(s,t,\varepsilon)\leq x}$ and $\mathbf{1}_{L_{\gamma^{(k)}}(s,t,\varepsilon)\leq x}$ can be estimated again using longest path analysis as we did with the TSEs. Finally, using a technique similar to the one used in bounding the means of TSEs, we achieve bounds on $F_{\gamma(s,t,\varepsilon)}(x)$ for a fixed x. In practice, upper and lower bounds on $F_{\gamma(s,t,\varepsilon)}$ may be plotted at a sufficiently large number of different values of x, say, 20, yielding a discretized approximation of the TSE distribution.

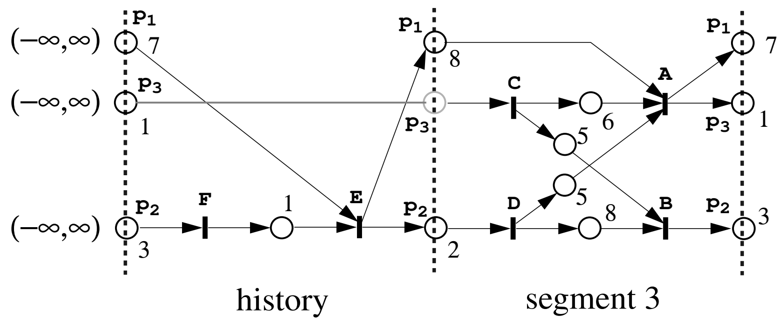
9.8 Evaluating the bounds

In the previous section, closed-form expressions of the bounds on various statistics of TSEs are derived. These expressions take expectations of potentially complicated random variables. In some special cases, analytical methods are available to obtain the exact value of the expectation [129]. In the general case, the formula can be estimated using statistical methods to a sufficiently high accuracy in the probabilistic sense. In our current tool, we use Monte-Carlo sampling (e.g.,[88], see also 8.4.2).

From a theoretical point of view, the validity of the Monte-Carlo approach depends on the finiteness of high order moments of the delays of the STPNs. We do not believe this is a significant restriction because system component delays in real designs are either finite or their moments are finite up to a sufficiently high order.

9.8.1 Improving the bounds

In many cases, the upper and lower bounds on the TSE statistics derived in the previous subsections are very close to each other, which combined yield very good estimates. As one of the distinct features of our technique, the above bounds can be made even closer to each other by slightly increasing the amount of (consecutive) segments used in the derivation of the bound. That is, we can often improve the bounds by slightly pushing the reference set further apart from the TSEs pairs being considered, thereby increasing the amount of history considered. Fig. 9.6 demonstrate this improvement as compared with the bound achieved without extra history in Fig. 9.4.



$$\gamma^{(3)}(\mathbf{A},\mathbf{B},0) \leq \max\{5\text{-}6, (2\text{+}8)\text{-}8\} = 2$$

Figure 9.6: Improving TSE bounds by adding extra segments as history.

9.9 Experiments

The bounding techniques described in this chapter has been fully automated. The resulting tool is named USC-PET. Currently, it applies the simplest statistical method, i.e., the Monte-Carlo approach, to construct the error interval and confidence of the estimates. In this section, we apply USC-PET to analysis of two scalable examples. The first example is a micropipeline where each stage exhibits choice, and the second a Petri net model of Intel's asynchronous instruction length decoding and steering unit called RAPPID [103].

9.9.1 Micropipelines with choice

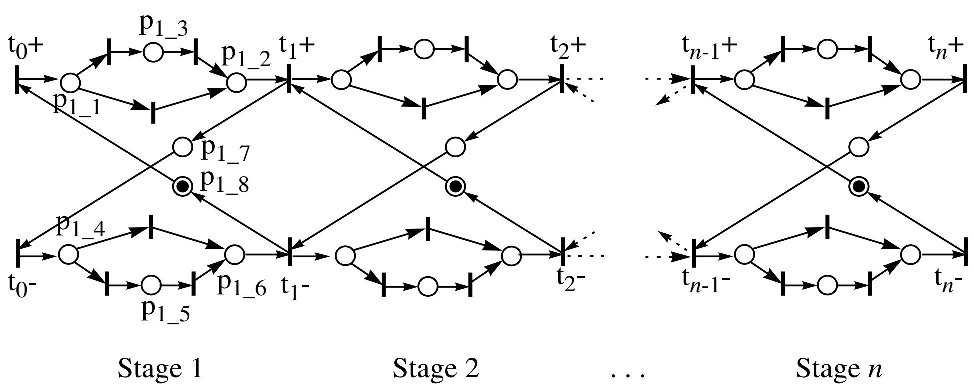


Figure 9.7: A micropipeline where each stage exhibits choice.

Fig. 9.7 depicts the model of the micropipeline with which we experimented. For stage i, places p_{i_1} and p_{i_4} are assigned delays that are uniformly distributed between 0 and 10, and places p_{i_2} and p_{i_6} are assigned delays that are uniformly distributed between 1 and 2. Places labeled by p_{i_3} and p_{i_5} have a fixed of 2. The other two places p_{i_7} and p_{i_8} have a fixed delay of 0.5. Each transition out of all choice places is given a probability of 0.5.

#	#	#	Mean			Variance		
stages	trans.	places	Lower	Upper	CPU sec	Lower	Upper	CPU sec
2	18	16	19.35	19.35	2.8	12.69	12.69	3.9
4	32	32	21.78	21.78	4.3	7.05	7.21	9.2
8	66	64	22.67	22.68	8.1	5.76	5.90	43.5
16	130	128	23.11	23.12	34.0	5.41	5.58	113.2
32	258	256	23.55	23.55	148.8	4.58	5.31	781.8

Table 9.1: Bounds on the mean and variance of cycle time of the micropipeline. The relative error interval and confidence level are set to 1% and 99%, respectively, in the Monte-Carlo sampling.

Table 9.1 lists the estimated bounds on the mean and variance of the cycle time (i.e., $\gamma(\mathtt{t_i}+, \mathtt{t_i}+, 1)$ with i fixed) as a function of pipeline depth. It is interesting to note that the sample size needed in estimating the bounds of variance of the cycle time is remarkably larger ($3 \sim 5$ times larger) than that needed in estimating mean of the cycle time, and thus more CPU time is required. In this table, the amount of segment history is set to approximately 1.5 times the depth of the micropipeline. The results also show that the cycle time of the pipeline increases in mean but decreases in variance as the pipeline depth increases.

Fig. 9.8 shows 4 plots of the lower and upper bounds on the distribution function of the cycle time of an 8-stage micropipeline as the amount of history considered is increased. As in the case of bounding the mean and variance, these plots show that the bounds on the distribution function quickly sharpens as the length of the considered history increases. Fig. 9.9 plots the estimated distribution function of the cycle time for different pipeline depth. As the depth increases, the distribution function shifts towards the right.

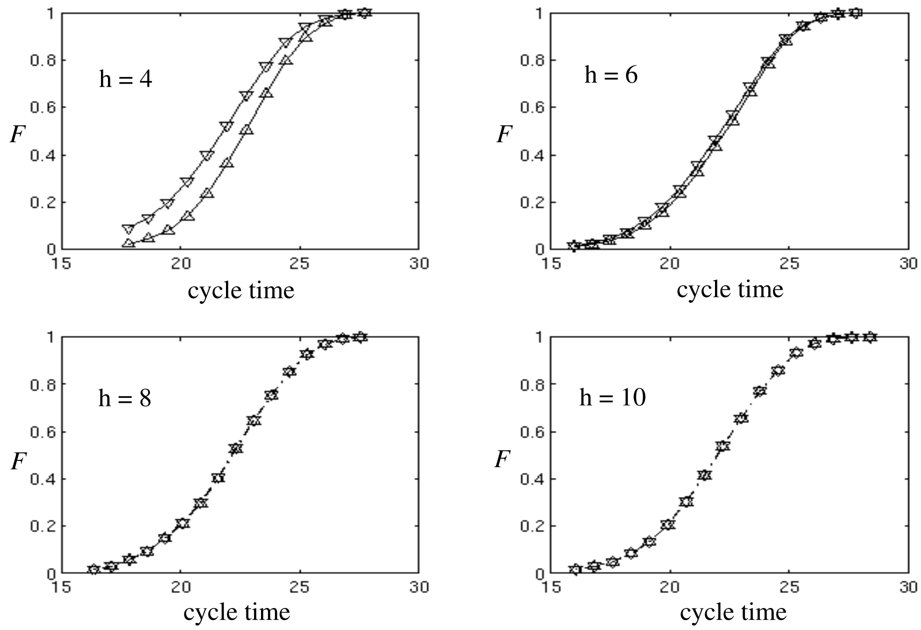


Figure 9.8: Improvement of bounds on the distribution function of cycle time in an 8-stage micropipeline as the amount of the history considered (h is the number of history segments) increases. The relative error interval and confidence level are set to 1% and 99%, respectively, in the Monte-Carlo sampling.

The efficiency of the tool in computing bounds on multiple TSEs is demonstrated by simultaneously estimating bounds on average latencies through all pipeline stages. Table 9.2 lists the required CPU times for both single TSE estimation (estimating the average latency through the whole micropipeline) versus those required for computing bounds on average latencies through all stages. The CPU times are reported by applying the longest path analysis from the poset events of the source places of segments. The result shows the the percentage of extra CPU time required to compute bounds on means of multiple TSEs is negligible. In addition, we observe that this percentage is almost in proportion to the number of transition pairs being analyzed for their TSEs.

9.9.2 A model of RAPPID

The RAPPID design is an aggressive asynchronous design to decode the length of x86 instructions and steer them to the subsequent instruction decoder with a high average throughput. The instructions are fed into the design through a number of

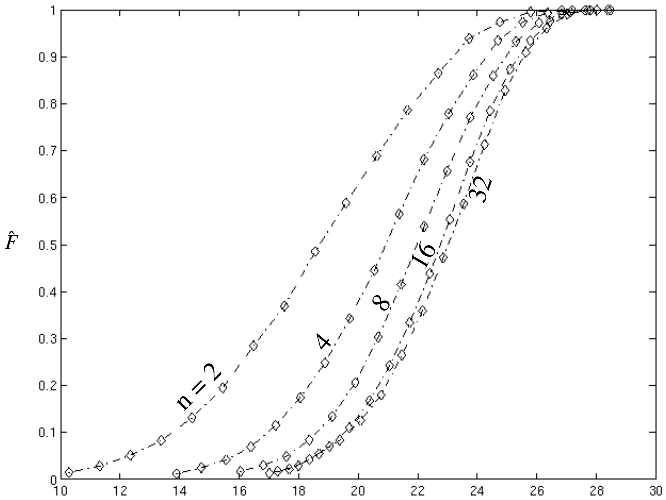


Figure 9.9: The estimated distribution functions (the average of the upper and lower bounds) plotted for pipelines of different depths (i.e., n in number of stages). The relative error interval and confidence level are set to 1% and 99%, respectively, in the Monte-Carlo sampling.

parallel byte units associated with columns. Each column speculatively decodes the byte under the assumption that an instruction starts at its associated byte. Each column has a number of tag units responsible for receiving a token indicating this column is the start of the next instruction and routing the token to a tag unit associated with the column which contains the start of the subsequent instruction. The tag units are arranged in rows each associated with an output buffer for the instruction. To alleviate a bottleneck in the output buffers, tag units in one row send its token to tag units in the next row. Both columns and rows are designed in a ring fashion so that they handle an infinite number of instructions. The real design has 16 columns, 4 rows, and is optimized to decode instructions of up to 7 bytes in length [103].

In our experiment, we examine a live and safe Petri net model of the RAPPID where the delays of places are only loosely related to the real delays in the design. Our model also ignores the handling of very rare instructions (e.g., length changing prefixes) and branches. The central driving units for the model is the actual length decoding logic in the columns. These units are optimized for the common instruction lengths and consequently have longer delay for less common instructions, as

# stages	bounding the mean latency			
	through whole pipeline		through every stage	
	# TSEs	CPU seconds	# TSEs	CPU seconds
2	1	3.29	2	3.44
4	1	9.22	4	11.61
8	1	28.5	8	32.8
16	1	111.4	16	118.5
32	1	446.7	32	460.3

Table 9.2: Computing bounds on multiple TSEs.

previously described in [32]. In our Petri net model, depicted in Fig. 9.10, this logic is modeled with a free-choice place with probability mass function that matches the relative frequency of instruction lengths given in [32]. Even for a given instruction length the decoding time can vary, thereby motivating the use of stochastic delay models for the decoding of each length.

Once decoded, the column broadcasts a signal to multiple tag units associated with its column indicating its instruction is ready to be transmitted to the output buffer associated with the tag unit. In our Petri net model, the tag unit in the top row of the first column is initially marked with a token to indicate that the first instruction will start at the first column and should be routed to the first output buffer. Once all bytes of the first instruction are available, this tag unit sends a token to a downstream tag unit in the second row of the column where the next instruction starts. We use unique-choice places in the model to route tokens to downstream tag units as well as the consumption of tokens representing all available bytes of an instruction.

We specified uniform delays of between 0.1 and 0.6 time units for pGotBuff, between 0.5 and 1.5 time units for pLen1, pLen2, and pLen3, between 1.0 and 2.5 time units for pLen4 and pLen5, and between 2 and 4 time units for pLen6 and pLen7. The pRowReq was given a fixed delay of 0.5 time units and all other places where given fixed delays of 0.1 time units.

We experimented with a number of different RAPPID-based architectures by varying the number of columns, rows, and the maximum instruction lengths handled.

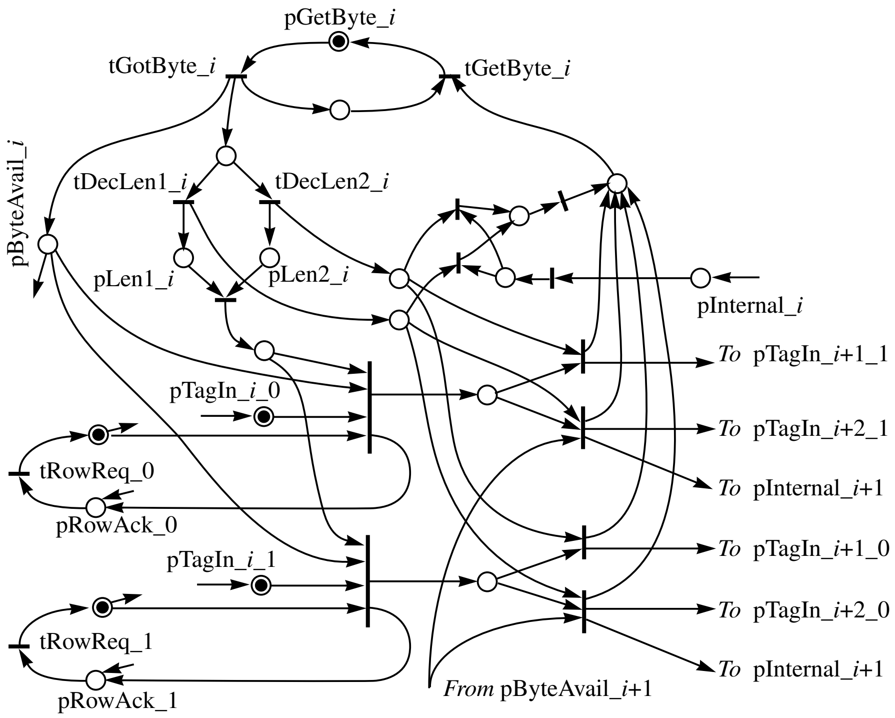


Figure 9.10: A Petri net model of 1 column of the RAPPID instruction length decoding and steering unit. This column has 2 Tag Unit rows and can handle instructions of length up to 2 bytes.

For each architecture, the mean and variance of the time separation between consecutive firings of transition tRowReq0, i.e., $\overline{\gamma}(\texttt{tRowReq0}, \texttt{tRowReq0}, 1)$. The bounds on the mean of this TSE divided by the number of rows provides an estimate of the average decoding cycle time and thus the throughput. The bounds on its variance give an estimate of the performance deviation. The reported variance bounds, when divided by the number of rows, suggest that increasing the the number of rows helps reduce the variance of the overall decoding cycle time. Experimental results are tabulated in Table 9.3. The CPU times listed are for bounding the variance, which includes the times for bounding the averages. The length of the history used in the experiments (i.e., the number of segments prior to the one considered) was set between 8 to 40 segments depending on the sizes of the individual RAPPID models.

Since the delay models do not match with the real design, we would rather not stress the significance of the performance trends. Instead, we prefer to stress the low run-time of the tool and the accuracy of the estimates. In all the examples tested, the width of the derived error bounds are less than 5% and are typically less

# Col.	# Rows	Max Len.	# Trans.	# Places	Mean Lower	Mean Upper	Variance Lower	Variance Upper	CPU secs
2	1	2	29	34	1.315	1.327	0.219	0.249	40.6
2	2	2	36	40	2.643	2.644	0.437	0.442	46.9
2	3	2	43	46	3.987	3.991	0.592	0.594	55.16
2	4	2	50	52	5.295	5.304	0.753	0.754	66.89
4	1	2	57	66	0.684	0.703	0.157	0.214	156.2
4	2	2	70	76	1.374	1.376	0.222	0.231	254.3
4	3	2	83	86	2.062	2.063	0.334	0.337	153.6
4	4	2	96	96	2.744	2.746	0.361	0.363	359.5
8	1	7	273	210	0.957	0.976	0.778	0.871	411.1
8	2	7	338	228	1.925	1.926	1.420	1.428	630.5
8	3	7	403	246	2.893	2.893	2.106	2.106	990.5
8	4	7	468	264	3.843	3.843	2.689	2.690	1,385
16	1	7	545	418	0.548	0.554	0.328	0.348	558.9
16	2	7	674	452	1.056	1.056	0.821	0.824	2,189
16	3	7	803	486	1.588	1.589	1.170	1.175	3,419
16	4	7	932	520	2.122	2.123	1.355	1.358	5,426

Table 9.3: Bounds on the mean and variance of the decoding cycle time of the RAPPID model. The relative error interval and confidence level are set to 5% and 99%, respectively, in Monte-Carlo sampling.

than 1%. The largest model contains over 900 transitions and 500 places and yet can be analyzed with 5% error tolerance and 99% confidence in 1.5 hours. These experiments demonstrate that the tool may be used to trade off the silicon area of the design with the system throughput. If possible, we hope to incorporate more accurate component delay estimates to obtain a more realistic analysis of competing architectures.

Chapter 10

Summary

10.1 Advancements

The work of this thesis has advanced several aspects in the area of performance analysis of asynchronous circuits and systems, especially on the topics of stochastic modeling and analysis.

The thesis proposed two types of modeling formalism, one based on Markov chains and the other based on Petri nets. Both of them allow specification of arbitrarily distributed delays. Traditionally, delay models were often limited to constants and exponential random variables. This advancement in modeling is important in the sense that delays of real system components are often varying but bounded. Modeling such real system delays using merely constants or exponential random variables could result in significant errors of the resulting performance estimates.

In the aspect of model solutions, the thesis improved the state-of-the-art analysis capacity by orders of magnitude in terms of solvable model size. In Markovian analysis, this was achieved by a symbolic state classification method for the structural analysis (Chapters 4 and 5) and a state compression method for the stationary analysis (Chapter 6). Both methods effectively mitigated the state-explosion problem to a large extent. Also in Markovian analysis, the thesis introduced a theoretical framework with the aid of the sojourn time concepts to allow general performance metrics be easily expressed and computed via the unifying notion of time separations of events (TSEs) (Chapter 3). This again was a significant advancement in the sense that the traditional theory allowed only limited class of performance metrics, namely, the average occurrence period of a single event or other metrics that are directly derivatives of such quantities. In practice, designers often need more general performance metrics normally captured by the average time separations between various events.

Perhaps the most important advancements of this thesis, both in theory and practice, were observed in its second part, statistical analysis. The idea of bounding TSE statistics for stochastic timed marked graph (Chapter 8) was a significant methodological advancement in the sense that traditional statistical methods to estimate such statistics had largely been limited to (timed) regenerative systems. For more general models, the thesis characterized a large class of TSE processes that are weakly ergodic for free-choice systems (see Chapter 9). The discovery of weak ergodic property of the grouped TSE process and a simple relation among the statistics of TSE, grouped TSE and group length was a rather surprising result. It is somewhat counter-intuitive in the sense that TSEs of a single group may expand multiple segments yet still possess a similar property to the delay sequence in a regenerative systems (see e.g.,[51]). It is these results that promised a methodological shift from analyzing exact performance metrics to achieving performance bounds for free-choice systems. The subsequent techniques explore these ideas to derive very sharp bounds on not only the means but also the variance and distributions of TSEs with usually polynomial time complexity. These techniques have essentially pushed the state-of-the-art of performance analysis to a stage where industrial-scale system models can be analyzed.

10.2 Applications to systems design

Analyzing system performance is not the end. The techniques developed in this thesis should be explored within the system design flow. For this purpose, our current tool may be augmented with a better user interface (possibly built on some GUI libraries) for model entry and to support translations from other hight-level design specifications such as Verilog and/or VHDL descriptions to Markov chain and/or Petri net models. For instance, the recent work by Blunno and Lavagno [14] may be incorporated for this purpose. This results in a stand-alone tool which may be used to evaluate various architectural choice as well as implementation styles.

Alternatively, the current tool may be integrated into the existing synthesis tools. This will permit fine-grain performance optimization often under the constraint of other design budgets such as silicon area and power consumption. As a result, it is expected that the tool should collect more performance statistics to report quantities such as system performance sensitivity with respect to component delays/speeds. As the targeted performance statistics get higher in their complexity and increase in their amount, the efficiency of the tool may need to be further improved. Towards this end, it may be beneficial to study sophisticated sampling techniques (e.g., importance sampling [102]) to reduce the size of the sample currently required by USC-PET.

10.3 Other applications

The work in this thesis may also have applications in areas other than performance analysis of asynchronous circuits and systems.

In formal verification, the symbolic state space decomposition technique may be explored for the generic bad-cycle-detection problem of model checking (see e.g, [58]). The technique can be easily adapted to obtained an alternative to the Emerson-Lei (EL) algorithm [47]. The time complexity of this new algorithm should be $O(\max\{\mathcal{D}, \mathcal{C}\} \times |\mathcal{A}|)$ while that of the the EL algorithm is no better than $O(\mathcal{D}^2)$ in number of BDD operations, where $\mathcal{D}$, $\mathcal{C}$ and $|\mathcal{A}|$ are the diameter of the state space, the number of cycle sets, and the number of SCCs, respectively. It is expected that when the system reachable state space has a limited number of SCCs (which is usually the case when the design is likely to pass), the resulting algorithm could significantly out-perform the EL algorithm and its variants (e.g., [58]).

Markov chain are widely adopted models in many fields of science, engineering, and business. State explosion has been the major challenge there. The solutions presented in this thesis should be found beneficial in many of these applications. In particular, the state compression technique which was developed for discrete-time Markov (DTMC) chains may be explored for continuous-time Markov chains (CTMC) as well. In CTMCs, a state normally has a self-loop which seems to limit the application of the state compression technique. However, such a self-loop can be eliminated by associating a proper sojourn time with the corresponding state. The elimination could introduce errors and may cause certain stability problem, but it is always possible to perform the elimination selectively over the state space. The result is that after self-loop elimination, the state space of the new Markov chain may have a small feedback vertex set so that the state compression technique can be used to speed up the subsequent stationary analysis.

Finally, the statistical analysis techniques of the work can be used in many other areas where Petri nets are naturally the modeling choice. These areas include, but not limited to, performance analysis/validation of embedded systems, performance and security analysis of computer networks, efficiency analysis of database architectures as well as throughput analysis of manufacturing lines and transportation systems. The idea of statistically bounding performance metrics may be further explored to develop solutions to similar aspects of discrete event system models other than Petri nets.

10.4 Systems with arbitration

This section sketches our recent developments in analysis of systems with arbitration. These developments can be considered as a generalization of our statistical bounding techniques to Petri nets with asymmetric choices. This part of the work also identifies a set of theoretical issues that need to be examined in order to guarantee the correctness of this generalization. Several open questions regarding these theoretical issues will be posted at the end of this section.

10.4.1 STPNs with asymmetric-choice

Arbitration arises naturally in many cases. One example is in resource sharing among processes where the resource can be granted to at most one process at any time. Figure 10.1 shows a Petri net that models an exclusive granting of a resource among two processes.

The net models two processes `a` and `b` each of which forever repeats a sequential procedure. The procedure has two steps, namely, the `thinking` and `using` steps. The availability of the resource is modeled by the marking of place $\mathtt{P_r}$. Initially, places $\mathtt{p_{ta}}$, $\mathtt{p_r}$ and $\mathtt{p_{tb}}$ each has one token as shown in the figure. A token in $\mathtt{p_{ta}}$

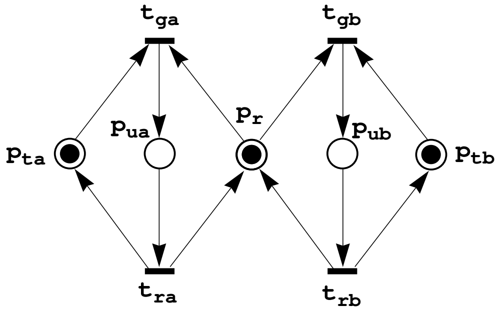


Figure 10.1: A Petri net model of two processes computing for a resource.

(p_{tb}) means process a (b) is in its `thinking` step. A token in p_r means the resource is currently not in use by either of the processes. Note that at the initial marking, both transitions t_ga and t_gb are enabled, resulting in a competition among the two processes for the resource. Suppose the resource is granted to process a, in which case, transition t_{ga} fires. The firing of t_{ga} disables transition t_gb. When process a finishes its `using` step, it releases the resource by firing transition t_{ra}, which brings net to its initial marking and re-enables transition t_{gb}. Note that when process a is `using` the resource, process b gets blocked if has finished its `thinking`. It remains blocked until the resource is released by process a and then competes for the resource.

The choice place p_r in the above PN example is called an *asymmetric-choice* because the transitions in its poset have different sets of input places. Such a structure is essential for a PN to model arbitration (see, e.g., [92]).

As in the case of free-choice PNs, we associate delays to the places of an asymmetric-choice PN in order to model the timed behavior of the system. In the above example, we use random variables $X(p_{ta})$ ($X(p_{ua})$) and $X(p_{tb})$ ($X(p_{ub})$) to denote the `thinking` (`using`) times of processes a and b, respectively. In addition, we assume that each time it is released by a process, the resource experiences a (possibly random) non-negative `resetting` time before it becomes available for further use. This `resetting` time is modeled by a random variable $X(p_r)$.

There are several different resolution policies for an asymmetric choice (see, e.g., [80]). Here, we choose the simple race policy. That is, among the transitions of the poset of the asymmetric choice place, the one that first gets timed enabled receives the token in the choice place. Tokens in the input places of other transition in conflict may remain available if they are available prior to the firing of the timed enabled transition. In case of the simultaneous enabling of multiple transitions in conflict, the asymmetric choice is reduced to a free-choice resolved among the timed enabled transitions in the conflict. In this case, we assume the asymmetric-choice place is associated with a choice probability function that resolves the conflict, similar to that of a free-choice place (see Section 2.2.2).

10.4.2 Bounding TSE statistics

Similar to those of free-choice systems, many important performance metrics of systems with arbitration can be expressed as TSE statistics of the corresponding STPN models. For instance, in the above resource sharing example, the throughput of either of the two processes can be captured by its cycle time. The cycle time of process `a` is the time separation of consecutive firing of transition $\mathtt{t_{ga}}$, which is equal to that of the consecutive firings of transition $\mathtt{t_{ua}}$. The average of such a TSE, i.e., $\overline{\gamma}(\mathtt{t_{ga}}, \mathtt{t_{ga}}, 1)$, captures the average throughput of the process.

One way to compute such a TSE average is to convert the STPN into a Markov chain (using time-discretization) model and then use the techniques developed in Part I to compute the stationary distribution of the chain. However, this hits the state-explosion problem. Below, we sketch a generalization of the statistical bounding techniques in Part II to analyze asymmetric-choice STPNs.

Let us first consider an easy case where all the delays are fixed. Again, we take the STPN model of the above resource sharing example. Since all delays of the net are fixed, it can be checked that all transitions will eventually fire. That is, the net is live.

Suppose now that at some time instant t, process `a` finishes its `thinking`, but finds an empty place $\mathtt{p_r}$. Thus, it gets blocked. The blocking time, however, can not be larger than $X(\mathtt{p_{ub}}) + X(\mathtt{p_r})$ because process `b` must release the resource by the time instant $t + X(\mathtt{p_{ub}})$, and the resource must be available by the time instant $t + X(\mathtt{p_{ub}}) + X(\mathtt{p_r})$. Once process `a` takes over the resource, it finishes its procedure in $X(\mathtt{p_{ua}}) + X(\mathtt{p_{ta}})$ time. Thus, the cycle time of process `a` is upper bounded by $X(\mathtt{p_{ub}}) + X(\mathtt{p_r}) + X(\mathtt{p_{ua}}) + X(\mathtt{p_{ta}})$. Note that this upper bound is also valid in other cases. For instance, let us consider the case where at time instant t, place $\mathtt{p_r}$ has a token available and process `b` also finishes its `thinking`. This is a case where transitions $\mathtt{t_{ga}}$ and $\mathtt{t_{gb}}$ are simultaneously enabled. It can be checked that the blocking time of process a is still upper bounded by $X(\mathtt{p_{ub}}) + X(\mathtt{p_r})$ even if process b is granted the resource at time t. Therefore, we have

$$\overline{\gamma}(\mathtt{t_{ga}}, \mathtt{t_{ga}}, 1) \leq X(\mathtt{p_{ub}}) + X(\mathtt{p_r}) + X(\mathtt{p_{ua}}) + X(\mathtt{p_{ta}}), \tag{10.1}$$

which yields an upper bound on the average cycle time of process a. A trivial lower bound on the average cycle time of process a is $X(\mathtt{p_{ua}}) + X(\mathtt{p_{ta}})$, which is obtained by ignoring the blocking time of the process.

If the above STPN has stochastic delays on places, one may obtain an upper bound on the average cycle time of process a by simply taking the expectation of the right side of Equation 10.1. However, observe that the upper bound in Equation 10.1 can be rather poor if the `using` time of process b is significantly larger than the sum of the `using` and `thinking` times of process a. Moreover, this method does not yield bounds on other statistics of TSEs such as their variance.

To obtain a better upper as well as a better lower bound on TSE statistics, it might be interesting to study a possible generalization of the statistical bounding techniques of Part II to handle arbitration. One of the difficulties in making such a generalization is that in asymmetric-choice STPNs, one cannot partition the underlying event structure of a random time execution into independent identically distributed segments, which was the key to applying the statistical bounding techniques in the free-choice case. The reason is that an asymmetric-choice may be resolved depending on timing. As a result, without the knowledge of the timing of the history, one may not precisely determine the event structure of a portion of a timed execution *a priori*.

Our preliminary idea to overcome the above difficulty is based on a notion of *phases*. Roughly speaking, a phase is a maximum dynamic portion of a random timed execution within which an asymmetric-choice is resolved such that its token is *consecutively* assigned to a particular transition in its poset. In the above resource sharing example, a phase may correspond to a maximum portion of a timed execution where process a is consecutively granted the resource. For convenience, we call it a-active phase. Note that during an a-active phase, process b is *idle* in the sense

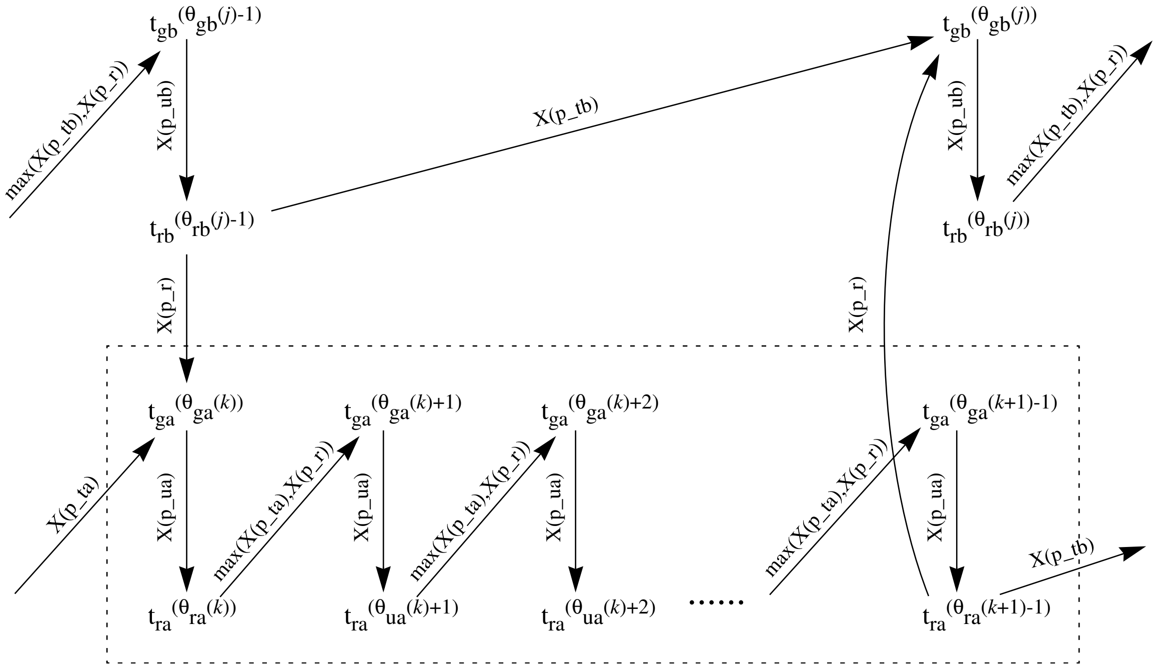


Figure 10.2: An illustration of an a-active phase in the resource sharing example.

that it is not `using` the resource. Thus, an a-active phase corresponds to a b-idle phase. Observe that a-active phases alternate with a-idle phases.

Figure 10.2 shows the k-th a-active phase of the STPN model of the resource sharing example (boxed by the dashed lines). In the figure, we write $\theta_z(n)$ to denote the index of the first firing of transition t_z during its corresponding n-th active phase. As shown in the figure, we have assumed process b is experiencing its j-th idle phase during the k-th a-active phase.

There are a total of $\psi_{\mathtt{a}}(k) = \theta_{\mathtt{ga}}(k+1) - \theta_{\mathtt{ga}}(k)$ cycles of process a during k-th a-active phase. Generally, $\psi_{\mathtt{a}}(k)$ is a random variable, which depends on a set of timing variables. These variables include the `thinking` time of process b during its j-th idle phase, the `thinking` and `using` times of process a within the k-th a-active phase, the resource `resetting` times at the beginning and end of the k-th a-active phase, the resource release time process a prior to the k-th a-active phase (denoted by $h_{\mathtt{ra}}(\theta_{\mathtt{ta}}(k)-1)$), and the resource release time from process b prior to the j-th b-idle phase (denoted by $h_{\mathtt{rb}}(\theta_{\mathtt{rb}}(j)-1)$). Note that all these variables are known without any knowledge of the history prior to the current a-active phase except $h_{\mathtt{ta}}(\theta_{\mathtt{ra}}(k)-1)$ and $h_{\mathtt{rb}}(\theta_{\mathtt{rb}}(j)-1)$. In other words, the difference between

$h_{\mathtt{ta}}(\theta_{\mathtt{ta}}(k)-1)$ and $h_{\mathtt{rb}}(\theta_{\mathtt{rb}}(j)-1)$ carries all the history information needed to determine the event structure of k-th a-active phase.

Give the time difference between $h_{\mathtt{ta}}(\theta_{\mathtt{ta}}(k)-1)$ and $h_{\mathtt{rb}}(\theta_{\mathtt{rb}}(j)-1)$, we can dynamically generate the event structure of k-th a-active as well as j-th b-idle phase by sampling the random delays of the corresponding places. In principle, if we ignore the phase history, for instance, by assuming the above time difference takes values in $(-\infty, \infty)$, we can obtain a set of all possible event structures for the current phase. For a random time execution that contains n a-active phases, we accordingly obtain a sequence of n i.i.d sets d of random event structures.

Within the k-th such set of random event structures, we can compute an upper and a lower bound on the duration of the corresponding b-idle phase. Using the Monte-Carlo sampling approach, we obtain an upper and a lower bound on the average duration of b-idle phases. Similarly, we obtain an upper and a lower bound on the average duration of a-idle phases. In addition, we can obtain an upper and a lower bound on the average duration of a-active phases as well as those on the average duration of b-active phases.

The derivation of bounds on average cycle time of process a is slightly complicated by the fact that an a-active phase may contain multiple cycles of the process. This complication can be resolved by using a similar technique that we developed to handle the similar complication in the analysis of free-choice STPNs where a segment may contain multiple occurrences of the target TSE. Here, we observe that, corresponding to the set of random event structure of k-th a-active phase, there is a set of random lengths of the phase, i.e., $\psi_{\mathtt{a}}(k)$'s, together with a set of random phase durations. In order to obtain an upper bound on the average cycle time of process a, we may choose a phase length and its corresponding phase duration for each Monte-Carlo sample to maximizes the ratio of the sum of phase durations and the sum of the phase lengths sampled so far. Since a-active phases must alternate with a-idle phases and each a-idle phase contains one and only one `thinking` step, we can combine the above Monte-Carlo analysis result with the bounds on the average duration of a-idle phase to obtain an upper and a lower bound on the average cycle time process a. Bounds on the average cycle time of process b can be obtained similarly. Bounds on other statistics such as variance of process cycle times can be achieved by a similar generalization of the techniques describe in Part II. Finally, by incorporating some history of the analyzed phase, the bounds may be sharpened significantly as in the case of free-choice STPNs.

10.4.3 Open questions

The techniques sketched in the previous subsection ignore several important ergodicity issues of related random processes derived from asymmetric-choice STPNs.

One of such random process is the timed marking process of the net. For free-choice STPNs, such a process is weakly ergodic in the sense that there is only one maximal communicating (possibly finite) set of timed markings that are recurrent. It is yet unknown if this is true for a given asymmetric-choice STPN. If an asymmetric-choice STPN has a timed marking process that is not weakly ergodic, the above sketched technique may need some modification and treat the STPNs within each of the recurrent timed marking sets[1]. For further study in this direction, we give following open questions concerning the ergodicity properties of timed marking processes as well as the convergence of TSE processes.

Let N be a live and safe asymmetric-choice Petri net. The STPN family of N, denoted by $\mathcal{N}$, is the set of all possible STPNs whose underlying untimed PN is N.

Question 1 *For what class of $\mathcal{N}$, the timed marking process is weakly ergodic?*

Question 2 *How to determine if an STPN of $\mathcal{N}$ is weakly ergodic, preferably without generating and analyzing its complete timed marking process?*

Question 3 *For what class of event pairs in an ergodic STPN of $\mathcal{N}$, their TSE processes are weakly ergodic?*

10.5 Conclusions

Performance analysis has been an important and challenging topic in the design of competitive asynchronous circuits and systems. Towards this end, the thesis presented two categories of analysis techniques, namely, a set of analytical methods

[1] Recall that in Part I, we did not distinguish a system with arbitration from a systems with only free-choice. However, a state classification step was incorporated in order to restrict the the corresponding Markov chain model into each of the recurrent classes

by modeling systems using Markov chains, and a set of statistical bounding methods by modeling systems using stochastic timed Petri nets. In the first category, the thesis successfully mitigated the state explosion problem in many cases by developing several symbolic techniques using binary and algebraic decision diagrams (BDD/ADDs). In the second category, the thesis focused on a class of systems identified as free-choice systems, and developed several novel techniques combining a structural analysis with a statistical sampling approach to derive bounds on performance metrics, namely, their average, variance and distributions. A generalization of these bounding techniques to systems with arbitration has also be sketched. From the modeling capacity point of view, the thesis was developed under a general timing model where arbitrarily distributed delays can be specified, which has been found important in practical applications. From the analysis capacity point of view, the thesis reduced the computational time of the state-of-the-art by orders of magnitude, and pushed the area of performance analysis to a stage where industry-sized systems may be analyzed in a reasonable amount of time.

The work in this thesis has potential applications in many other areas. For instance, most of the techniques presented can be applied to performance analysis of the general class of discrete event dynamic systems. In addition, the symbolic SCC enumeration technique may be applied to the liveness check in formal verification.

The thesis has identified several directions for future study. A full integration of this work with existing synthesis tools is currently under study to optimize system performance at behavioral, RTL, gate and transition-levels. Several open questions were posted to generalize the statistical bounding techniques to systems with arbitration.

Reference List

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

[2] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley Publishing Company, 1974.

[3] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[4] J. B. Anderson and S. Mohan. Sequential coding algorithms: A survy and cost analysis. *IEEE Transactions on Communications*, 32(2):169–176, February 1984.

[5] F. Baccelli, G. Cohen, G. J. Olsder, and J.-P. Quadrat. *Synchronization and Linearity: An algebra for discrete event systems.* John Wiley & Sons, 1992.

[6] F. Baccelli and J. Mairesse. Ergodic theorems for stochastic operators and discrete event networks. Technical Report No. 2641, INRIA, 1995.

[7] 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)*, pages 188–191, 1993.

[8] P. A. Beerel, C. Hsieh, and S. Wadekar. Energy estimation of speed-independent control circuits. *IEEE Transactions on Computer-Aided Design*, 15(6):672–680, June 1996.

[9] W. Belluomini and C. J. Myers. Verification of timed systems using POSETS. In *Proc. International Workshop on Computer Aided Verification*, pages 403–415, 1998.

[10] E. Best. Some classes of live and safe Petri nets. In *Concurrency and nets,* Special volume in the series, “Advances in Petri nets”, pages 71–94. Springer-Verlage, 1987.

[11] E. Best. Structural theory of Petri nets: The free choice hiatus. In *Lecture Notes in Computer Science, LNCS 254*, pages 168–206. Springer-Verlage, 1987.

[12] E. Best. Partial order behavior and structure of Petri nets. *Formal Aspects of Computing*, 2:123–138, 1990.

[13] E. Best and K. Voss. Free choice systems have home states. *Acta Informatica*, 21:89–100, 1984.

[14] I. Blunno and L. Lavagno. Deriving signal transition graphs from behavioral verilog hdl. In *Proc. of 2nd Workshop on Hardware Design and Petri Nets*, pages 113–130, June 1999.

[15] R. Bolla and F. Davoli. Control of multirate synchronous streams in hybrid TDM access networks. *IEEE/ACM Transactions on Networking*, 5(2):291–304, April 1997.

[16] R. E. Bryant. Graph-based algorithm for boolean function munipulation. *IEEE Transactions on Computers*, 35, August 1986.

[17] P. Buchholz. Exact and ordinary lumpability in finite Markov chains. *Journal of Applied Probability*, 31:59–75, 1994.

[18] P. Buchholz. Lumpability and nearly-lumpability in hierarchical queueing networks. In *Proc. IEEE Int. Computer Performance and Dependability Symp. (IPDS)*, pages 82–91, 1995.

[19] P. Buchholz. Hierarchical structuring of superposed gspns. In *International Workshops on Petri Nets and Performance Modelds*, pages 81–90, June 1997.

[20] J. R. Burch, E. M. Clarke, D. E. Long, K. L. McMillan, and D. L. Dill. Symbolic model checking for sequential circuit verification. *IEEE Transactions on Computer-Aided Design*, 13(4):401–424, April 1994.

[21] R. Burch, F. N. Najm, P. Yang, and T. Trick. A monte carlo approach for power estimation. *IEEE Transactions on VLSI Systems*, 1(1):63–71, March 1993.

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

[23] J. Campos, G. Chiola, J. M. Colom, and M. Silva. Properties and performance bounds for timed marked graphs. *IEEE Transactions on Circuits and Systems–I: Fundamental theory and applications*, 39(5):386–401, May 1992.

[24] J. Campos, G. Chiola, and M. Silva. Properties and performance bounds for closed free choice synchronized monoclass queueing networks. *IEEE Transactions on Automatic Control*, 36(12):1368–1382, Dec. 1991.

[25] G. Casella and R. L. Berger. *Statistical Inference.* Duxbury Press, Wadsworth Publishing Company, California, 1990.

[26] C. G. Cassandras. *Discrete Event Systems: Modeling and Performance analysis.* Irwin Publications, 1993.

[27] S. Chakraborty and D. L. Dill. Approximate algorithms for time separation of events. In *Proc. International Conf. Computer-Aided Design (ICCAD)*, pages 190–194, November 1997.

[28] T. J. Chaney and C. E. Molnar. Anomalous behavior of synchronizer and arbiter circuits. *IEEE Transactions on Computers*, C-22(4):421–422, April 1973.

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

[30] Y.-K. Chong and K. Hwang. Performance analysis of four memory consistency models for multithreaded multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 6(10):1085–1099, 1995.

[31] T.-L. Chou and K. Roy. Accurate power estimation of cmos sequential circuits. *IEEE Transactions on VLSI Systems*, 4(3), Sept. 1996.

[32] 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)*, pages 80–91. IEEE Computer Society Press, March 1998.

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

[34] K. L. Chung. *Markov Chains with Stationary Transition Probabilities.* Springer-Verlag, 1960.

[35] G. Cohen, D. Dubois, J.-P. Quadrat, and M. Viot. A linear-system-theoretic view of discrete-event processes and its use for performance evaluation in manufacturing. *IEEE Transactions on Automatic Control*, 30(3):210–220, March 1985.

[36] F. Commoner, A. W. Holt, S. Even, and A. Pnueli. Marked directed graphs. *Journal of Computer and System Sciences*, 5:511–523, 1971.

[37] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* McGraw-Hill, 1990.

[38] O. Coudert, J. C. Madre, and C. Berthet. Verifying temporal properties of sequential machines without building their state diagrams. In E. M. Clarke and R. P. Kurshan, editors, *Computer-Aided Verification'90*, pages 75–84. American Mathematical Society, June 1990.

[39] A. Dasdan and R. K. Gupta. Faster maximum and minimum mean cycle algorithms for system performance analysis. Technical report, UC Irvine, February 1997.

[40] A. Dasdan, S. S. Irani, and R. K. Gupta. Efficient algorithms for optimum cycle mean and optimum cost to time ratio problems. In *Proc. ACM/IEEE Design Automation Conference*, pages 37–42, 1999.

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

[42] D. L. Dill. Trace theory for automatic hierarchical verification of speed-independent circuits. ACM Distinguished Dissertations, 1988.

[43] D. L. Dill and E. M. Clarke. *Trace theory for automatic hierarchical verification of speed-independent circuits.* Techical Report, Carnegie Mellon University, Computer Science Department, 1988.

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

[45] J. Ebergen and R. berks. Reponse-time of asynchronous linear pipelines. *Proceedings of the IEEE*, 87(02):308–318, February 1999.

[46] J. C. Ebergen. A formal approach to designing delay-insensitive circuits. *Distributed Computing*, 3(5):107–119, 1991.

[47] E. A. Emerson and C. L. Lei. Efficient model checking in fragments of the propositional modal mu-calculus. In *Proceedings of LICS 1986*, pages 267–278, 1986.

[48] Brayton et al. VIS: A system for verfication and synthesis. In *Proc. International Conference on Computer Aided Verification*, pages 428–432, 1996.

[49] G. D. Forney. The viterbi algorithm. *Proc. IEEE*, 61:268–278, 1973.

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

[51] P. W. Glynn and D. L. Iglehart. Simulation output analysis using standardized time series. *Math. Oper. Res.*, 3:1–16, 1990.

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

[53] G. R. Grimmett and D. R. Stirzaker. *Probability and Random Processes (2nd Edition)*. Oxford Science Publications, 1992.

[54] P. J. Haas. Estimation methods for stochastic Petri nets based on standardized time series. In *International Workshops on Petri Nets and Performance Modelds*, pages 194–204, June 1997.

[55] P. J. Haas. *Private communication*, June 1999. Peter Haas is a Research Staff Member at IBM Almaden Research Center, San Jose, California.

[56] G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Markovian analysis of large finite state machines. *IEEE Transactions on Computer-Aided Design*, 15(12):1479–1493, December 1996.

[57] G. D. Hachtel and F. Somenzi. A symbolic algorithm for maximum flow in 0-1 networks. In *Proc. International Conf. Computer-Aided Design (ICCAD)*, pages 403–406, 1993.

[58] R. H. Hardin, R. P. Kurshan, S. K. Shukla, and M. Y. Vardi. A new heuristic for bad cycle detection using bdds. In *Proc. International Workshop on Computer Aided Verification*, pages 268–278, 1997.

[59] H. P. Hillion and A. H. Levis. Timed event-graphs and performance evaluation of systems. In *8th European Workshop on Applied Theory of Petri Nets*, June 1987.

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

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

[62] L. Holden. *Private communication*, December 1997. Lars Holden is the research director of SAND Group within the Norwegian Computing Center at the University of Oslo.

[63] M. A. Holliday and M. Y. Vernon. A generalized timed Petri net model for performance analysis. *IEEE Transactions on Software Engineering*, 13(12):1297–1310, December 1987.

[64] 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)*, pages 2–11. IEEE Computer Society Press, November 1994.

[65] H. Hulgaard and S. M. Burns. Efficient timing analysis of a class of Petri nets. In *Proc. International Workshop on Computer Aided Verification*, pages 423–436, 1995.

[66] R. M. Izquierdo and D. S. Reeves. MPEG VBR slice layer model using linear predictive coding and generalized periodic Markov chains. In *IEEE International Performance, Computing and Communications Conference*, pages 437–443, 1997.

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

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

[69] J. Kingman. Subadditive ergodic theory. *Annals of Probability*, 1:883–909, 1973.

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

[71] J. N. Kozhaya and F. N. Najm. Accurate power estimation for large sequential circuits. In *Proc. International Conf. Computer-Aided Design (ICCAD)*, pages 488–493, 1997.

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

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

[74] 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*, pages 345–357, March 1998.

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

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

[77] M. Malhotra and K. S. Trivedi. Power-hierarchy of dependability-model types. *IEEE Transactions on Reliability*, 43(3):493–502, Sept. 1994.

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

[79] R. Marculescu, D. Marculescu, and M. Pedram. Efficient power estimation for highly correlated input streams. In *Proc. ACM/IEEE Design Automation Conference*, pages 628–634, 1995.

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

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

[82] M. Ajmone Marsan, A. Bobbio, and S. Donatelli. Petri nets in performance analaysis: An introduction. In *Lectures on Petri nets I: Basic Models*, Lecture Notes in Computer Science. Springer-Verlag, 1998.

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

[84] A. J. Martin, S. M. Burns, T. K. Lee, D. Borković, and P. J. Hazewindus. The design of an asynchronous microprocessor. In *Decennial Caltech Conference on VLSI*, pages 226–234, 1989.

[85] Y. Matsunaga, P. C. McGeer, and R. K. Brayton. On computing the transitive closure of a state transition relation. In *Proc. ACM/IEEE Design Automation Conference*, pages 260–265, 1993.

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

[87] K. L. McMillan. A technique for state space search based on unfolding. *Formal Methods in System Design*, 6(1):45–66, January 1995.

[88] I. R. Miller, J. E. Freund, and R. Johnson. *Probability and Statistics for Engineers.* Prentice Hall, 1990.

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

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

[91] T. Murata. Circuit theorectic analysis and synthesis of marked graphs. *IEEE Transactions on Circuits and Systems*, 24:400–405, July 1977.

[92] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77:541–580, April 1989.

[93] S. M. Nowick and D. Dill. "Asynchronous State Machine Synthesis Using a Local Clock". In *International Workshop on Logic Synthesis*, 1991.

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

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

[96] V. D. Ploeg. Preconditioning techniques for large sparse, non-symmetric matrices with arbitrary sparsity patterns. In *Proc. IMACS Symposium on Iterative Methods in Linear Algebra*, pages 173–179, 1991.

[97] S. Qadeer, R. K. Brayton, V. Singhal, and C. Pixley. Latch redundancy removal without global reset. In *Proc. International Conf. Computer Design (ICCD)*, Oct. 1996.

[98] C. V. Ramamoorthy and G. S. Ho. Performance evaluation of asynchronous concurrent systems using Petri nets. *IEEE Transactions on Software Engineering*, 6(5):440–449, September 1980.

[99] R. R. Razouk and C. V. Phelps. Performance analysis using timed Petri nets. In *Proc. of 1984 Int. Conf. Parallel Processing*, pages 126–129, August 1984.

[100] O. Roig, J. Cortadella, and E. Pastor. Verification of asynchronous circuits by BDD-based model checking of Petri nets. In *16th Intl. Conf. on Theory and Application of Petri-Nets*, June 1995.

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

[102] S. M. Ross. *A Course in Simulation.* MacMillian Coll Div., 1990.

[103] S. Rotem, K. Stevens, B. Agapiev, C. Dike, M. Roncken, R. Ginosor, R. Kol, P. A. Beerel, K. Y. Yun, and C. J. Myers. Rappid: An asynchronous instruction length decoding and steering unit. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, April 1999.

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

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

[106] G. Santharam and P.S. Sastry. A reinforcement learning neural network for adaptive control of Markov chains. *IEEE Transactions on Systems, Man & Cybernetics (Part A)*, 27(5):588–600, Sept. 1997.

[107] L. Schrage. *Lindo: User's Manual.* The Scientific Press, 1991.

[108] A. Semenov and A. Yakovlev. Verification of asynchronous circuits using time Petri-net unfolding. In *Proc. ACM/IEEE Design Automation Conference*, pages 59–63, 1996.

[109] G. S. Shedler. *Regenerative Stochastic Simulation.* Academic Press, New York, 1993.

[110] T. Shiple, R. Hojati, A. Sangiovanni-Vincentelli, and R. K. Brayton. Heuristic minimization of bdds using don't cares. In *Proc. ACM/IEEE Design Automation Conference*, pages 225–231, 1994.

[111] M. Silva and J. Campos. Structural performance analysis of stochastic Petri nets. In *IEEE International Computer Performance and Dependability Symposium*, pages 61–70, 1995.

[112] V. Singhal. *Design Replacements for Sequential Circuits.* PhD thesis, University of California at Berkeley, 1996.

[113] 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*, 22(1):9–15, January 1975.

[114] A.M. Stankovic, G.C. Verghese, and D. J. Perreault. Randomized modulation of power converters via Markov chains. *IEEE Transactions on Control Systems Technology*, 5(1):61–73, Jan 1997.

[115] K. Stevens, S. Rotem, S. M. Burns, J. Cortadella, R. Ginosar, M. Kishinevsky, and M. Roncken. Cad directions for high performance asynchronous circuits. In *Proc. ACM/IEEE Design Automation Conference*, pages 116–121, 1999.

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

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

[118] I. E. Sutherland. Micropipelines. *Communications of the ACM*, 32(6):720–738, 1989.

[119] T. Takine, T. Suda, and T. Hasegawa. Cell-loss and output process analysis of a finite-buff discrete-time atm queueing system with correlated arrival. *IEEE Transactions on Communications*, 43(2–4):1022–1037, Feb.-March 1995.

[120] R. E. Tarjan. Depth first search and linear graph algorithms. *ACM Journal on Computing*, 1(2), 1972.

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

[122] V. Vakilotoja and P. A. Beerel. RTL verification of timed asynchronous and homogeneous systems using symbolic model checking. *Integration, the VLSI journal*, pages 473–484, December 1997.

[123] H. J. M. Veendrick. The behavior of flip-flops used as synchronizers and prediction of their falure. *IEEE Journal of Solid-State Circuits*, SC-15(2):169–176, April 1980.

[124] Ted E. Williams. *Self-Timed Rings and their Application to Division.* PhD thesis, Stanford University, June 1991.

[125] J. V. Woods, P. Day, S. B. Furber, J. D. Garside, N. C. Paver, and S. Temple. AMULET1: An asynchronous ARM processor. *IEEE Transactions on Computers*, 46(4):385–398, April 1997.

[126] 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)*, pages 64–75. IEEE Computer Society Press, April 1997.

[127] A. Xie and P. A. Beerel. Accelerating Markovian analysis of asynchronous systems using string-based state compression. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, pages 247–260. IEEE Computer Society Press, March 1998.

[128] A. Xie and P. A. Beerel. Analysis of variance in micropipelines. Technical Report CENG-TR-98-17, EE-Systems Department, University of Southern California, July 1998.

[129] A. Xie and P. A. Beerel. Bounding average time separations of events in stochastic timed marked graphs. Technical report, EE-Systems Department, University of Southern California, Oct 1998.

[130] A. Xie and P. A. Beerel. Efficient state classification of finite state Markov chains. In *Proc. ACM/IEEE Design Automation Conference*, pages 605–610, June 1998.

[131] A. Xie and P. A. Beerel. Efficient state classification of finite state Markov chains. *IEEE Transactions on Computer-Aided Design*, 17(12):1334–1338, December 1998.

[132] A. Xie and P. A. Beerel. Accelerating Markovian analysis of asynchronous systems using state compression. *IEEE Transactions on Computer-Aided Design*, 18(7):869–888, July 1999.

[133] A. Xie and P. A. Beerel. Implicit enumeration of strongly connected components. In *Proc. International Conf. Computer-Aided Design (ICCAD)*, November 1999. (To appear).

[134] A. Xie and P. A. Beerel. Performance analysis of asynchronous circuits and systems using stochastic timed Petri nets (invited paper). In *Proc. of 2nd Workshop on Hardware Design and Petri Nets*, pages 35–62, June 1999.

[135] A. Xie, S. Kim, and P. A. Beerel. Bounding average time separation of events in stochastic timed Petri nets with choice. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, April 1999.

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

[137] K. Y. Yun, D. L. Dill, and S. M. Nowick. Synthesis of 3D asynchronous state machines. In *Proc. International Conf. Computer Design (ICCD)*, pages 346–350. IEEE Computer Society Press, October 1992.

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

[139] O. Daniel Z. Simeu-Abazi and B. Descotes-Genon. Analytical method to evaluate the dependability of manufacturing systems. *Reliability Engineering & Systems Safety*, 55(2):125–130, Feb 1997.

[140] W. M. Zuberek. Timed Petri nets and preliminary performance evaluation. In *Proc. Int. Symp. Computer Architecture (ISCA)*, pages 88–96, 1980.