PROBABILISTIC MODELING AND VERIFICATION OF LARGE SCALE SYSTEMS

BY

YOUNG MIN KWON

B.E., Korea University, 1996
M.E., Korea University, 1998

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2006

Urbana, Illinois

Abstract

Large scale networked embedded systems are becoming increaingly popular with the technology advances in wireless network, energy efficient hardware, and cost effective manufacturing. Parameter tuning of such large scale systems has a significant effect on the performance metrics such as reliability, availability, longevity, and energy consumption. The parameter tuning should be based on a performance evaluation which requires a model that closely represents the real system. In modeling a large scale system, an abstraction of the state space of the system is necessary in order to avoid the state explosion problem. Moreover, users not only, need an expressive and accurate way to describe the performance criteria in terms of the model, they also need a way to evaluate the performance criteria against the model. They can do such evaluation manually for certain well-known properties or they can explore unknown properties with the aid of computers.

Many large scale systems have a stochastic behavior. Such stochastic behavior is the result of the randomness in the systems' operating environments. It can also be the result of the use of randomized protocols that are used to reduce the need for costly synchronization. We abstract a system state as a probability mass function (pmf); a pmf is succinct in representation and is informative enough to describe many useful aggregate behaviors of the system. We model the dynamics of the state transitions as a Discrete Time Markov Chain (DTMC) and develop an on-line model estimation method as well as a goodness of fit test that statistically validates the model. We develop a probabilistic temporal logic called iLTL in order to specify aggregate behaviors of large scale systems. Many important performance criteria on DTMC models can be accurately expressed in iLTL. For the iLTL specifications, performance evaluation is systematically performed through a process of model checking. If a performance criterion is not satisfiable, the model checker returns a sequence of computations that violates the specification in order to help users fix the problem. We show the usefulness of our methodology through an experiment with a wireless sensor network of 90 nodes.

To my wife, for her love and devotion.

Acknowledgments

I would like to thank Professor Gul Agha for his technical guidance and for his personal advice throughout my studies. He has always been a patient, a thoughtful, and a skillful mentor. He taught me the value of life as well as his vision and knowledge. I thank Professor Jose Meseguer for his spiritual guidance and his invaluable advice on forming my initial research directions and on nurturing them with related researches. I thank Professor Mahesh Viswanathan for his advice on complexities of model checking algorithms. I thank Professor William Sanders for his questions and advices on modeling probabilistic systems.

I would like to give special thanks to Kirill Mechitov and Sameer Sundresh for their time and effort on wireless sensor network experiments. Those outdoor experiments were laborious but invaluable experiences. I would like to thank Timo Latvala for the discussions on symbolic model checking and for correcting my papers. I would like to thank OSL members for their advice and help on my research problems and for their effort to make our lab a pleasant place: Amr Ahmed, Tom Brown, Po-Hao Chang, Liping Chen, Joshua Chia, Christo Devaraj, MyungJoo Ham, Nadeem Jamali, Myeong-Wuk Jang, WooYoung Kim, Nirman Kumar, Soham Mazumdar, Mehwish Nagda, Smitha Reddy, Shangping Ren, Koushik Sen, Sudarshan Srinivasan, Prasanna Thati, Predrag Tosic, Sandeep Uttamchandani, Abhay Vardhan, Carlos Varela, Nalini Venkatasubramanian, Andrea Whitesell, and Reza Ziaei.

I would like to thank HeeJun Choi and his wife JiHae Park for their endless hospitality. I would always miss the trips we went together. I also would like to thank all my friends in Urbana-Champaign. I appreciate my cheerful life with them.

I would like to thank my parents, brothers, and parents in law for their constant support and encouragement. Their trust in me has always been my source of belief in my research works. Finally, but most importantly, I owe a lifetime thank to my wife Eunhee Kim. She has always been with me when I were discouraged and when I were delighted. Without her love and devotion this research would not have been possible.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

With advances in network and hardware technologies, networked embedded systems have become more prevalent than ever before. The number of embedded systems outgrow that of personal computers and this trend is projected to be continued in the future. *Wireless sensor networks* (WSNs) have recently emerged as a cost effective way of networking embedded nodes. In this thesis, we develop a method to abstractly model such systems and a test method to statistically validate the models. We also develop a probabilistic temporal logic called iLTL and its model checking algorithm. iLTL is used to evaluate the performances of such system and provides a guideline to improve performance.

A WSN is made up of sensor nodes that collaborate with each other through wireless communication. Each node has its own processor, volatile and nonvolatile memory, one or more sensors, a wireless communication channel, and an independent power source. Because of these unique characteristics, WSNs have been used in many applications such as environmental monitoring [66, 28], structural health monitoring [72, 46], target tracking [2, 45], and so on.

One of the distinguishing characteristics of typical WSNs is their scale. For example, a medium scale application such as shooter localization [44] may involve 60 nodes. Without an appropriate abstraction, it is hard to reason about the global behavior of a WSN, for example to compute its aggregate energy consumption level or availability. If we were to model a system state as a cross-product of each node's state, the standard in concurrency theory, we would end up with the well-known state explosion problem: a WSN of 100 nodes, each with three states, has 3^{100} states. Instead, we take a statistical approach: we abstract the global state of a system as a vector of probabilities, where the size of the vector is the number of states the nodes may be in, and the *i*th element of the vector represents the probability that a randomly picked node is in the *i*th state [35]. With this abstraction we can easily reason about certain expected aggregate properties of the system. Such properties include availability, energy consumption, and throughput.

We model the transition dynamics between states as a *Discrete Time Markov Chain* (DTMC). This is reasonable for many applications running on WSNs as their behavior corresponds to a probabilistic transition systems: nodes go to a sleep mode to save energy with a certain probability and, in order to minimize collisions, nodes send packets based on probabilistic choice. We choose a DTMC as our model instead of a Continuous Time Markov Chain (CTMC) because the interpretation of time is discrete in our performance evaluation logic (see Section 3). The choice of DTMC also enables our method for estimation of the model (see Section 5.1).

A DTMC is defined by a *set of states* and *transition probabilities* between states. Identifying states from a real system can be done rather straightforwardly: depending on the properties we want to evaluate or monitor, we can define a necessary set of states. A developer can easily identify which behavior of an application program belongs to which state. However, assigning transition probabilities between states is not as immediate as defining and identifying states. For example, although a process may go to a sleep mode by a pre-programmed probability, we generally do not know how often a process arrives at the relevant decision point in the code. Thus, we develop a DTMC estimation algorithm based on a sequence of state estimations of an application execution.

Recall that the current state of a Markov process depends only on its immediate past state. Thus, even samples from a single execution have enough previous-next state relations to *estimate* the Markov transition matrix. We also provide a statistical testing method that checks whether the sample is from the estimated DTMC. This test is important because our estimation algorithm would always find optimal parameters that best match with the samples, even if the system were *not* a Markov process. However, if the system is not a Markov process or the parameters are poorly estimated, our test will give users a notification together with any level of desired confidence.

Once we have a DTMC model of a WSN, we can check whether this model satisfies certain performance criteria that should be met. For example, we can evaluate the expected number of message retransmissions before a sender successfully receives an ack message, the expected amount of energy consumed to reliably send a message, or how long it may take to recover from a current unacceptable availability level. The checking is easily and systematically done by checking the model against the specification written in a probabilistic temporal logic called iLTL [33]. The logic is extended so that it can model check concurrent runs of several independent processes [34]. Thus, for independent processes a user can evaluate performances without exponentially blowing system size.

The methodology we are proposing is useful for establishing aggregate expected properties only of sensor networks where the nodes have a sufficient degree of symmetry in the local states, for example, if the nodes implement a randomized algorithm, or if the system is event-driven and the environment exhibits probabilistic behavior over the time interval of interest. In a way, our approach is similar to statistical physics, which allows aggregate properties of a system, such as the entropy or the average temperature, to be estimated. The method described above has limitations when all nodes do not have a uniform probabilistic representation. For example, our method would not be applicable if, for the property of interest, the behavior of the nodes is represented as a tree of join continuations [1]. Note that the goodness of fit test for Markov model estimation we propose should tell us when our method is inapplicable. If some parts of a WSN behave differently than others, we can model them separately and combine them later. For example, nodes at the edges of a WSN may behave differently than nodes in the middle. In this case modeling the entire system as a single Markov process might cause inaccuracies. Instead, we propose to model boundary nodes and central nodes separately and combine them as a *Stochastic Automata Network* (SAN) [53]. In case sub-processes are independent, we can simplify the model greatly by keeping track of transitions of individual sub-system states. We can reconstruct the whole system state using the individual states. Performance evaluation on independent processes occurs commonly: the systems can be physically independent, comparison between several processes requires concurrent and independent runs of those processes, and checking the effect of initial conditions also requires concurrent and independent runs of the same processes.

1.2 Contributions

The contributions of this thesis are as follows. First, we develop a probabilistic temporal logic called iLTL that has different interpretation of computational paths than existing probabilistic temporal logics. As a result it can specify some aggregate properties of large-scale systems which cannot be expressed with conventional logics. Second, we describe a novel way of modeling sensor networks as DTMCs and expressing the aggregate properties of the network, both in its transient states and its steady-state. Third, we provide a way of estimating DTMCs and suggest a statistical test to ensure with high probability that the sensor network obeys the Markov property and that the estimate is sufficiently accurate. Fourth, we suggest a scheme to improve the performance of a system based on probabilistic model checking. Transient and steady-state performances of several systems can be systematically compared through the iLTL model checking process. Finally, we show a novel method for model checking these properties. We use both simulations and experimental results to illustrate and validate our method. Note that standard methods for solving Markov matrices result in efficient solutions for steady-state analysis. However, we capture both the transient behaviors and the steady-state. Moreover, we express our properties in a formal logic and automate the process of checking behaviors.

1.3 Thesis Outline

The thesis is organized as follows. The next chapter provides background materials on Markov processes, probabilistic temporal logics, and motivates our choice of a probabilistic temporal logic. Chapter 3 explains the syntax, the semantics, and a model checking algorithm of iLTL and its extension to multiple DTMC model. In this chapter, we also discuss how the applicability of our method may be extended to some WSNs where not all nodes have the same probabilities over the states of interest. In Chapter 4 we analyze a distance measurement algorithm commonly used in WSN and discuss how to improve the performance of the method through iLTL model checking. In Chapter 5 we propose a DTMC estimation method and a statistical testing method. Chapter 6 experimentally demonstrates the effectiveness of our proposed approaches on a WSN of 90 Mica2 nodes.

1.4 Related Work

In this section, we give a brief overview of related researches. The goal is to put our research in a broader context. Specific details of related work are provided in the background chapter.

Temporal logics are logics on computations over time. Specifically, Linear Temporal Logic (LTL) [41] specifies properties over each computational paths and Computation Tree Logic (CTL) [19] specifies properties about computation trees (unwound Kripke structures) using temporal operators preceded by universal or existential path quantifiers. CTL* [20] is a CTL with the constraint about the path quantifiers relieved.

Probabilistic temporal logics are temporal logics on probabilistic systems such as Markov chains. Probabilistic CTL (PCTL) [25] and PCTL* [4] are probabilistic counterparts of CTL and CTL* where the path quantifiers are replaced by the probabilities that certain properties hold in the sub-computation tree starting from a state. The models of PCTL and PCTL* are Discrete Time Markov Chains (DTMCs), whereas the models of Continuous Stochastic Logic (CSL) [5], a PCTL-like logic with a probability operator on steady state, are Continuous Time Markov Chains (CTMCs).

Inferring the probabilities in the Markov matrix from samples is a problem in learning. A more general form of this problem is inferring the structure of the Markov model itself, given traces from a black box system. This latter problem is called *stochastic grammatical inference*. The most recent work on this is an algorithm for stochastic grammatical inference is presented in [59, 13, 31, 55], where an algorithm and tool is presented to learn Continuous Time Markov Chains. Note that the method does not provide any semantic interpretation for the states inferred; this must be done separately. Because the problem addressed by stochastic grammatical inference is more general, the algorithms are computationally expensive – in case of [59], the running time is cubic. We address a simpler problem in our Markov model estimation method – namely, estimating the probabilities given that the structure of the Markov model is assumed to be known. Assuming a Gaussian distribution of errors, the accuracy of our estimate follows the Central Limit Theorem [51] – it is reduced as a function of the square root of the sample size (where the sample is the number of nodes times the number of runs).

When modeling a complex system and composing several systems as a Markov chain, the resulting state space can be very large. Such large state space often makes checking these systems impractical. If some states can be merged together and be treated as a single state, it would be computationally beneficial. A state lumping technique is a method to partition the state space of a Markov chain and treat each partition as a lumped state [12]. A lumpability condition is that for each state s_i in a partition A_j the probability of moving from s_i to a partition A_k is the same for all states in the partition A_j [30].

For large Markov matrices generated from compositions of Markov processes, symbolic representations have been used to store the matrices efficiently. Multi-valued Decision Diagram (MDD) [47], a generalization of Binary Decision Diagrams (BDD), or Matrix Diagram (MD) [16] reduces space to store the Markov matrix by removing redundancies in the structure.

Model checking is an automatic verification method that checks whether the behaviors of a model are within allowed behaviors of a specification usually expressed in a temporal logic. A model checker such as SPIN [26] has been successfully used to check hardware designs, software testing, security protocol checking, and other computer aidded checking areas.

When systems have probabilistic behaviors, we can quantitatively check the probability that the model satisfies a specification written in a PCTL-like logic. PCTL-like logics build a probability space of paths, and for each sub-formula they compute the probability of the event of satisfactory paths until the truth value of the whole formula is determined. Many probabilistic model checking tools for these logics have been implemented and are used for performance evaluation of probabilisitstic systems [6, 42, 17, 10, 32].

In order to reduce the gap between an analytical model and a real system, a statistical model checking tool called VESTA has been proposed [60]. VESTA provides three interfaces to applications and if they are implemented VESTA runs discrete event simulations until the number of runs meet given statistical parameters. Probabilistic model checking is performed to statistically verify specifications on the recorded runs. VESTA also checks properties expressed as PCTL or CSL.

Much of the material in this thesis has breviously been published in a series of papers. The use of pmf abstraction, definition of iLTL and the basic model checking algorithm was described in [33]. The model estimation method and its application to sensor networks was first described in [35]. The tool is described in [34].

Chapter 2

Background

2.1 Discrete Time Markov Chains

A Markov process is a stochastic process whose past has no influence on the future if its present is specified [51]. Markov processes have been used in many application areas such as communications, computer networks, and reliability theory. For example, many randomized communication protocols, such as *Carrier Sense Multiple Access* (CSMA), are often modeled as a queuing system which is a Markov process [67]. Reliability of a software is also modeled as a probabilistic transition system among modules which also is a Markov process [52, 24, 14]. Many distributed algorithms such as a distributed consensus algorithm can be more practical by using randomized algorithms [9, 68]. A Markov chain is a Markov process with a countable number of states. Queuing system or reliability models are examples of Markov chains. A Markov chain can also be classified as either a *Continuous Time Markov Chain* (CTMC) or a *Discrete Time Markov Chain* (DTMC), depending on the interpretation of time. In this section we will describe properties about DTMCs and in Section 2.3 we will explain CTMCs in detail.

We represent a *Discrete Time Markov Chain* (DTMC) X as a tuple $(S, \mathbf{M})$ where S is a finite set of states $S = \{s_1, s_2, \ldots, s_n\}$ that X can take, and $\mathbf{M} \in \mathbb{R}^{n \times n}$ is a Markov transition matrix that governs the transitions of X's *probability mass function* (pmf). With this representation, the Markov property can be written as follows:

$$\mathrm{P}[X(t) = s_i \mid X(t-1), X(t-2), \ldots, X(0)] = \mathrm{P}[X(t) = s_i \mid X(t-1)].$$

The elements of the Markov transition matrix **M** are conditional probabilities such that

$$\mathbf{M}_{ij} = \mathrm{P}[X(t) = s_i \mid X(t-1) = s_j].$$

So, the pmf of X at time t can be written as

$$\mathrm{P}[X(t) = s_i] = \sum_{j=1}^{n} \mathbf{M}_{ij} \cdot \mathrm{P}[X(t-1) = s_j].$$

Because $\mathbf{M}_{ij}$ is a conditional probability, it is a value in between one and zero, and because

$$\sum_{i=1}^{n} \mathrm{P}[X(t) = s_i \mid X(t-1) = s_j] = 1,$$

each column of **M** sums up to one.

In this paper, we represent the pmf as a column vector as follows:

$$\begin{aligned} \mathbf{x}(t) &= [x_1(t), x_2(t), \ldots, x_n(t)]^T, \\ x_i(t) &= \mathrm{P}[X(t) = s_i]. \end{aligned}$$

With the vector representation of a pmf and the Markov transition matrix **M**, one step prediction of pmf can be written as

$$\mathbf{x}(t+1) = \mathbf{M} \cdot \mathbf{x}(t),$$

and by recursively applying the one step prediction equation, an n step prediction can be written as

$$\mathbf{x}(t+n) = \mathbf{M}^n \cdot \mathbf{x}(t).$$

In this thesis, we consider only those DTMCs which have a unique steady state pmf. The unique steady state pmf is a scalar multiple of the the eigenvector corresponding to an eigenvalue one. Note that every Markov matrix has the eigenvalue one. That is, there is a vector $\mathbf{z} \in \mathbb{R}^n$ such that $(\mathbf{M} - \mathbf{I}) \cdot \mathbf{z} = 0$ where $\mathbf{I} \in \mathbb{R}^{n \times n}$. Because, each column of $\mathbf{M} - \mathbf{I}$ sums up to zero, the rows of $\mathbf{M} - \mathbf{I}$ are not independent and so are the columns of $\mathbf{M} - \mathbf{I}$. That is, there is a linear combination of the columns of $\mathbf{M} - \mathbf{I}$ that makes the sum the zero vector and the linear combination coefficients are the elements of $\mathbf{z}$ [39]. Note that $\mathbf{z}$ is not unique. That is, if $\mathbf{z}$ is an eigenvector of $\mathbf{M}$ then so is $\alpha \cdot \mathbf{z}$ for any $\alpha \in \mathbb{C}$.

However, not every DTMC has a unique steady state. For example, let a Markov matrix $\mathbf{M}$ be an identity matrix. Then, for any $\mathbf{x}(0)$, $\lim_{t\to\infty} \mathbf{x}(t) = \mathbf{x}(0)$. A necessary and sufficient condition for a DTMC to have a unique steady state is that the absolute values of all eigenvalues of $\mathbf{M}$ are strictly less than one, except for the eigenvalue 1.

Other than the steady state condition, we require that all Markov matrices are diagonalizable. The diagonalizability condition is not an unrealistic assumption. In fact most matrices are diagonalizable, especially if the matrices are estimated ones, and the diagonalizability condition will be gained by a slight perturbation of any undiagonalizable matrix.

In summary, we assume the following two conditions in this thesis:

- Let λ_i for $i = 1 \ldots, n$ be the eigenvalues of $\mathbf{M}$ with $\lambda_1 = 1$. Then, $|\lambda_i| < 1$, for $i = 2, \ldots, n$.

- $\mathbf{M}$ is diagonalizable.

With the diagonalizability condition, the relation between the eigenvalue condition and the existence of a unique steady condition can be easily shown as follows. Let

$$\mathbf{M} = \mathbf{Z} \cdot \mathbf{\Lambda} \cdot \mathbf{Z}^{-1},$$

where $\mathbf{\Lambda} = \mathrm{diag}([\lambda_1, \ldots, \lambda_n])$, and $\mathbf{z}_i = [z_{1,i}, \ldots, z_{n,i}]^T$ is the i^{th} column vector of $\mathbf{Z}$ such that $\mathbf{M} \cdot \mathbf{z}_i = \lambda_i \cdot \mathbf{z}_i$. Then,

$$\lim_{t\to\infty} \mathbf{M}^t \cdot \mathbf{x}(0) = \lim_{t\to\infty} \mathbf{Z} \cdot \begin{bmatrix} \lambda_1^t & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & \lambda_n^t \end{bmatrix} \cdot \mathbf{Z}^{-1} \cdot \mathbf{x}(0).$$

- If the eigenvalue condition holds then,

$$\lim_{t\to\infty} \mathbf{x}(t) = \lim_{t\to\infty} \mathbf{M}^t \cdot \mathbf{x}(0) = \left(\sum_{i=1}^{n} Z_{1,i}^{-1} \cdot x_i(0) \right) \cdot \mathbf{z}_1.$$

Note that the constant $\sum_{i=1}^{n} Z_{1,i}^{-1} \cdot x_i(0)$ is determined regardless of $\mathbf{x}(0)$ because $\mathbf{x}(\infty)$ is a pmf and it sums up to 1.

- If there is a λ_i such that $|\lambda_i| = 1$ with $i \neq 1$ and $\lambda_i \neq 1$, then for some $\mathbf{x}(0)$ there is no steady state because λ_i^t will oscillate forever.

- If there is a λ_i such that $\lambda_i = 1$ with $i \neq 1$ then any pmf vector in the linear space spanned by $\mathbf{z}_i$'s can be a steady state pmf.

As is shown above, the unique steady state pmf vector is a scalar multiple of the eigenvector corresponding to the eigenvalue 1. The scalar value can be computed by the fact that pmf vectors sum up to one [65]. That is, $\sum_{i=1}^{n} x_i(\infty) = 1$. Since, $\mathbf{x}(\infty) = \left(\sum_{i=1}^{n} Z_{1,i}^{-1} \cdot x_i(0)\right) \cdot \mathbf{z}_1$,

$$\left(\sum_{i=1}^{n} Z_{1,i}^{-1} \cdot x_i(0)\right) = \frac{1}{\sum_{i=1}^{n} z_{1_i}}.$$

As an example of a DTMC that does not have a unique steady state pmf, let us consider the following Markov matrix:

$$\mathbf{M} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

The eigenvalues of $\mathbf{M}$ are 1 and -1. Thus, it does not satisfy the eigenvalue condition. For any pmf vector $\mathbf{x}(0) = [p, 1-p]^T$, $\mathbf{x}(t) = \mathbf{M}^t \cdot \mathbf{x}(0)$ is $[p, 1-p]^T$ if t is even and $[1-p, p]^T$ otherwise. Thus, there is no steady state pmf.

2.2 Markov Reward Model

Many useful system properties can be expressed by associating a reward to each state of a DTMC. In this model we assume that by visiting a state we earn the reward associated with the state. This extended model can express many interesting features of a system in terms of instant or accumulated expected rewards.

Formally, a Markov Reward Model [15, 56] is a triple $(S, \mathbf{M}, \rho)$ where $S = \{s_1, \ldots, s_n\}$ is a set of states, $\mathbf{M} \in \mathbb{R}^{n \times n}$ is a Markov transition matrix, and $\rho : S \to \mathbb{R}$ is a reward function that maps each state of S to the reward for that state. Note that the tuple $(S, \mathbf{M})$ is a DTMC. Since we consider only constant rewards we represent the reward function as a constant row vector $\mathbf{r} = [\rho(s_1), \ldots, \rho(s_n)]$. With this representation the expected reward of a DTMC at time t is:

$$\begin{aligned}\sum_{i=1}^{n} \rho(s_i) \cdot \mathrm{P}[X(t) = s_i] &= \mathbf{r} \cdot \mathbf{x}(t) \\ &= \mathbf{r} \cdot \mathbf{M}^t \cdot \mathbf{x}(0).\end{aligned}$$

An accumulated reward during a finite time span T can also be represented as a constant vector. That is, the expected reward from time t to $t + T$ is:

$$\begin{aligned}\sum_{\tau=0}^{T} \mathbf{r} \cdot \mathbf{x}(t+\tau) &= \sum_{\tau=0}^{T} \mathbf{r} \cdot \mathbf{M}^{\tau} \cdot \mathbf{x}(t) \\ &= \mathbf{r} \cdot \mathbf{Z} \cdot \left(\sum_{\tau=0}^{T} \mathbf{\Lambda}^{\tau} \right) \cdot \mathbf{Z}^{-1} \cdot \mathbf{x}(t),\end{aligned}$$

where $\mathbf{M} = \mathbf{Z} \cdot \mathbf{\Lambda} \cdot \mathbf{Z}^{-1}$. Note that $\mathbf{r} \cdot \mathbf{M}^{\tau}$ is a constant row vector. So, the accumulated reward can be treated as another instant reward. For large T the second line of the previous equations is useful:

$$\sum_{\tau=0}^{T} \mathbf{\Lambda}^{\tau} = \mathrm{diag}([\sum_{\tau=0}^{T} \lambda_1^{\tau}, \ldots, \sum_{\tau=0}^{T} \lambda_n^{\tau}]),$$

where $\sum_{t=0}^{T} \lambda_i = \lambda_i \cdot \frac{1-\lambda_i^{T+1}}{1-\lambda_i}$. Note also that if $\mathbf{r}$ is orthogonal to the eigenvector $\mathbf{z}_1$ (that is, $\mathbf{r} \cdot \mathbf{z}_1 = 0$) then the expected accumulated reward is finite even though T is infinite.

Example 1 *We give a simple example of a send/ack protocol to illustrate the use of a Markov reward model. In this protocol, a sender sends a packet, on receiving the packet a receiver replies with an ack. If the sender does not receive the ack during a certain period of time, it times out and retransmits the packet. This protocol is represented by the state transition diagram of Figure 2.1, where the numbers at arrows are transition probabilities and numbers in states indicate the reward of the state (energy consumption at that state). The states* `s, ra, rX, XX` *and* `d` *stand for 'sent', 'received and acked', 'received but not acked', 'not received', and 'done'. This state transition diagram can be represented by a DTMC $P = (S, \mathbf{M}, L)$ where*

$$S = \{\, s, ra, rx, XX, d \,\},$$

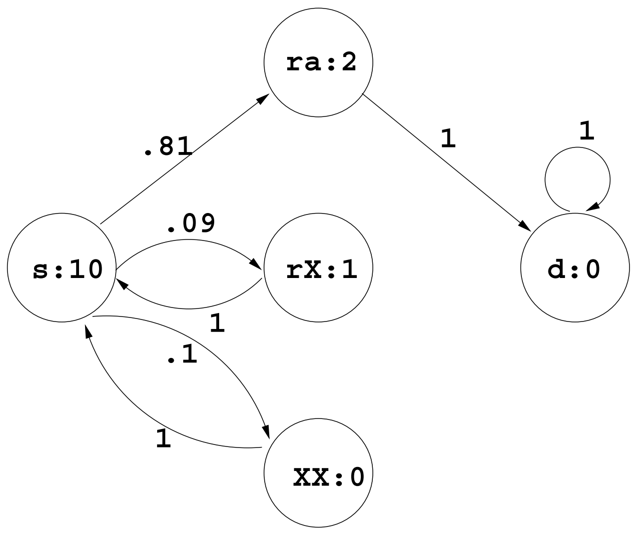


Figure 2.1: A simple send/ack protocol: numbers at arrows are transition probabilities, and numbers in states are required energy at that state.

$$\mathbf{M} = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 \\ 0.81 & 0 & 0 & 0 & 0 \\ 0.09 & 0 & 0 & 0 & 0 \\ 0.1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{bmatrix},$$

and L will be obvious if AP is given. A reward function of expected energy consumption at each step can be represented by a vector $\mathbf{r} = [10, 2, 1, 0, 0]$*. So, the expected energy consumption at time t given an initial pmf* $\mathbf{x}(0)$ *is* $\mathbf{r} \cdot \mathbf{M}^t \cdot \mathbf{x}(0)$*. Note that the reward vector* $\mathbf{r}$ *is orthogonal to the steady state pmf vector* $[0, 0, 0, 0, 1]^T$*. So, we can compute the accumulated reward for infinite period (the expected energy consumption to finish this protocol):*

$$\begin{aligned} \mathbf{r}' &= \mathbf{r} \cdot \mathbf{Z} \cdot \left(\sum_{\tau=0}^{\infty} \mathbf{\Lambda}^{\tau} \right) \cdot \mathbf{Z}^{-1} \\ &= [14.457, 2.0, 15.457, 14.457, 0]. \end{aligned}$$

Hence, the expected energy consumption to finish the send/ack protocol given a current pmf vector $\mathbf{x}$ *is* $\mathbf{r}' \cdot \mathbf{x}$*.*

■

2.3 Continuous-Time Markov Chains

Like a DTMC, a *Continuous Time Markov Chain* (CTMC) is a Markov chain. However a CTMC is defined on continuous time. Thus, a CTMC is defined with a rate matrix instead of the probability transition matrix of a DTMC.

Formally, a CTMC X is a tuple $(S, \mathbf{Q})$, where $S = \{s_1, \ldots, s_n\}$ is a finite set of states X can take, and $\mathbf{Q}$ is a rate matrix such that $Q_{i,j}$ with $i \neq j$ is the rate at which X in the s_j state moves to the s_i state [49]. The rate matrix $\mathbf{Q}$ should satisfy the following conditions:

- $q_{i,j} \geq 0$ for $i \neq j$
- $\displaystyle\sum_{i=1}^{n} q_{i,j} = 0$ for $j = 1, \ldots, n$

The probability transition matrix after time t can be computed by the matrix exponential. That is, let $\mathbf{P}(t) = e^{t \cdot \mathbf{Q}}$, then $\mathrm{P}[X(t) = s_i | X(0) = s_j] = \mathrm{P}(t)_{i,j}$. If $\mathbf{Q}$ is diagonalizable, then the matrix exponential $e^{t \cdot \mathbf{Q}}$ can be computed as follows:

$$\begin{aligned}
e^{t\cdot\mathbf{Q}} &= \sum_{k=0}^{\infty} \frac{(t \cdot \mathbf{Q})^k}{k!} \\
&= \mathbf{Z} \cdot \left(\sum_{k=0}^{\infty} \frac{(t \cdot \mathbf{\Lambda})^k}{k!} \right) \cdot \mathbf{Z}^{-1} \\
&= \mathbf{Z} \cdot \begin{bmatrix} e^{t\cdot\lambda_1} & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & e^{t\cdot\lambda_n} \end{bmatrix} \cdot \mathbf{Z}^{-1},
\end{aligned}$$

where $\mathbf{Z} \cdot \mathbf{\Lambda} \cdot \mathbf{Z}^{-1}$ is the diagonalizaion of the rate matrix $\mathbf{Q}$. We can obtain a DTMC by periodically sampling a CTMC. That is, let T be a sampling period, then

$$\begin{aligned}
\mathbf{x}(n \cdot T) &= \mathbf{P}(n \cdot T) \cdot \mathbf{x}(0) \\
&= \mathbf{P}(T)^n \cdot \mathbf{x}(0).
\end{aligned}$$

Thus, if a sampling period is given, we can convert a CTMC into a DTMC and vice versa.

From the matrix exponential we can derive the forward and backward equations

$$\begin{aligned}\frac{d}{dt}\mathbf{P}(t) &= \sum_{k=0}^{\infty} \frac{t^{k-1}\cdot\mathbf{Q}^k}{(k-1)!} \\ &= \mathbf{P}(t)\cdot\mathbf{Q} \\ &= \mathbf{Q}\cdot\mathbf{P}(t).\end{aligned}$$

The above equations show how the rate matrix, also called as a generator matrix, affects the transitions of pmfs.

2.4 Multiple Markov Process Model

The behaviors of nodes in a WSN may be different depending on the applications parameters and the environments in which they are placed. For example, when flooding or aggregating messages in a WSN, behaviors of nodes may differ depending on their hop-distances to a base station. Also, we often noticed that the nodes at the boundary of a WSN and the nodes in the middle have different behaviors. In these cases, estimating the whole system as a single DTMC can be inappropriate and would fail the goodness of fit test. Instead, we can estimate the system as several separate Markov processes and combine them as a single Markov process.

Kronecker product can be used to combine independent DTMCs [38]. With this operator one can build a joint probability transition matrix when two processes are independent. Let $\mathbf{A} \in \mathbb{R}^{n_A\times m_A}$ and $\mathbf{B} \in \mathbb{R}^{n_B\times m_B}$ then the *Kronecker product* $\mathbf{A}\otimes\mathbf{B} \in \mathbb{R}^{n_A\cdot n_B\times m_A\cdot m_B}$ is

$$\mathbf{A}\otimes\mathbf{B} = \begin{bmatrix} a_{11}\cdot\mathbf{B} & \cdots & a_{1m_A}\cdot\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{n_A1}\cdot\mathbf{B} & \cdots & a_{n_Am_A}\cdot\mathbf{B}\end{bmatrix},$$

where a_{ij} is the $(i, j)^{th}$ element of $\mathbf{A}$. The followings are useful properties of the Kronecker operators

$$\begin{aligned}(\mathbf{A}\otimes\mathbf{B})\otimes\mathbf{C} &= \mathbf{A}\otimes(\mathbf{B}\otimes\mathbf{C}), \\ \mathbf{A}\otimes(\mathbf{B}+\mathbf{C}) &= \mathbf{A}\otimes\mathbf{B}+\mathbf{A}\otimes\mathbf{C},\end{aligned}$$

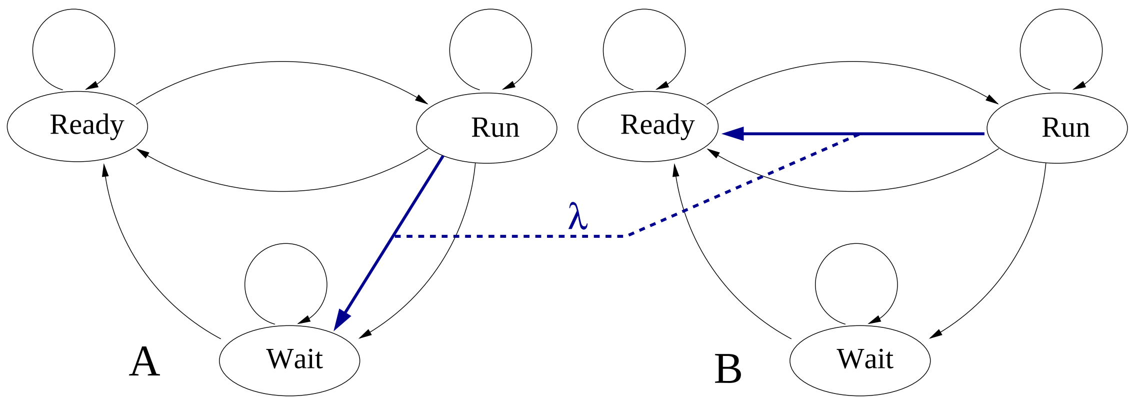


Figure 2.2: Interacting Markov processes. A master process (*A*) sends a command to a slave process (*B*) at a rate of λ. On sending the message A makes a transition from *Run* state to *Wait* state and on receiving the message B makes a transition to *Ready* state if it is in *Run* state. Without the interaction, *A* and *B* are two independent Markov processes.

$$\begin{aligned}(\mathbf{B}+\mathbf{C})\otimes\mathbf{A} &= \mathbf{B}\otimes\mathbf{A}+\mathbf{C}\otimes\mathbf{A},\\ (\mathbf{A}\otimes\mathbf{B})^T &= \mathbf{A}^T\otimes\mathbf{B}^T,\\ (\mathbf{A}\otimes\mathbf{B})\cdot(\mathbf{C}\otimes\mathbf{D}) &= (\mathbf{A}\cdot\mathbf{C})\otimes(\mathbf{B}\cdot\mathbf{D}).\end{aligned}$$

Using the Kronecker product, two DTMCs can be combined as follows: let $A = (S^{(A)}, \mathbf{M}^{(A)})$ and $B = (S^{(B)}, \mathbf{M}^{(B)})$ be two DTMCs, then the composite DTMC is

$$C = (S^{(A)} \times S^{(B)}, \mathbf{M}^{(A)} \otimes \mathbf{M}^{(B)}),$$

where the i^{th} state of the set $S^{(A)} \times S^{(B)}$ is the tuple $(S_q^{(A)}, S_r^{(B)})$, with $q = \lfloor i/|S^{(A)}| \rfloor$ and $r = i \bmod |S^{(A)}|$.

Stochastic Automata Networks (SANs) can be used for combining interacting Markov processes [53, 37]. Specifically, SAN uses synchronized events to model interactions between systems. As an example, Figure 2.2 shows a diagram of interacting processes. There are two types of nodes, *A* and *B*, which are both modeled by the DTMC *PS* of Section 5.3. Suppose that *A* type nodes send commands to *B* type nodes at the rate of λ. When sending the command, *A* changes its state from *Run* to *Wait*. On receiving the command, *B* changes its state from *Run* to *Ready*. The thick arrows of Figure 2.2 show the synchronized event. Because SAN composes CTMCs and generates a CTMC, in order to use SAN, we need to convert estimated DTMCs of sub-systems into CTMCs by taking matrix logs. Then, we compute a composed system and convert it back to a DTMC by taking a matrix exponential. The matrix exponential can be done easily after a matrix diagonalization as follows:

$$\begin{aligned} e^{\mathbf{M}} &= \sum_{n=0}^{\infty} \frac{\mathbf{M}^n}{n!} = \sum_{n=0}^{\infty} \frac{\mathbf{Z} \cdot \mathbf{\Lambda}^n \cdot \mathbf{Z}^{-1}}{n!} \\ &= \mathbf{Z} \cdot e^{\mathbf{\Lambda}} \cdot \mathbf{Z}^{-1}, \end{aligned}$$

where $\mathbf{Z} \cdot \mathbf{\Lambda} \cdot \mathbf{Z}^{-1}$ is a diagonalization of $\mathbf{M}$, and $e^{\mathbf{\Lambda}} = \text{diag}([e^{\lambda_1}, \ldots, e^{\lambda_n}])$.

The Kronecker product counterpart of CTMC is Kronecker sum. For matrices $\mathbf{A} \in \mathbb{R}^{n_A \times n_A}$ and $\mathbf{B} \in \mathbb{R}^{n_B \times n_B}$ the *Kronecker sum* $\mathbf{A} \oplus \mathbf{B} \in \mathbb{R}^{n_A \cdot n_B \times n_A \cdot n_B}$ is

$$\mathbf{A} \oplus \mathbf{B} = \mathbf{A} \otimes \mathbf{I}_{n_B} + \mathbf{I}_{n_A} \otimes \mathbf{B}.$$

The following notations are used for a series of Kronecker products and a series of Kronecker sums

$$\begin{aligned} \bigotimes_{j=1}^{J} \mathbf{A}^{(j)} &= \mathbf{A}^{(1)} \otimes \mathbf{A}^{(2)} \otimes \cdots \otimes \mathbf{A}^{(J)}, \\ \bigoplus_{j=1}^{J} \mathbf{A}^{(j)} &= \sum_{j=1}^{J} \mathbf{I}_{n_1} \otimes \cdots \otimes \mathbf{A}^{(j)} \otimes \cdots \otimes \mathbf{I}_{n_J}, \end{aligned}$$

where n_i is the number of rows of a square matrix $\mathbf{A}^{(i)}$. Let $(S^{(j)}, \mathbf{Q}^{(j)})$ for $j = 1, \ldots, J$ be a set of CTMCs and assume that there are K synchronized events. Then, their composite CTMC is

$$A = (S, \mathbf{Q}), \tag{2.1}$$

where $S = S^{(1)} \times \cdots \times S^{(J)}$ and

$$\mathbf{Q} = \bigoplus_{j=1}^{J} \mathbf{Q}^{(j)} + \sum_{k=1}^{K} \lambda_k \cdot \left(\bigotimes_{j=1}^{J} \mathbf{E}_k^{(j)} - \bigotimes_{j=1}^{J} \mathbf{D}_k^{(j)} \right), \tag{2.2}$$

where λ_k is the rate of the kth event, and $\mathbf{E}_k^{(j)} \in \mathbb{R}^{|S^{(j)}| \times |S^{(j)}|}$ is:

- $\mathbf{I}_{|S^{(j)}|}$: if the jth component is neither a master nor a slave of the kth synchronized event.

- $(\, e_{ab} \,)$[1]: if the jth component is a slave of the kth synchronized event, where $e_{ab} \in \mathbb{R}^{+}$ is a weight such

[1]We use the notation $(\, e_{ij} \,)$ for a matrix whose $(i, j)^{th}$ element is e_{ij}

that the column sum of (e_{ab}) is one if there is a transition from $S_b^{(j)}$ to $S_a^{(j)}$ due to the kth event, and 0 otherwise.

- (e_{ab}): if the jth component is the master of the kth synchronized event, where e_{ab} is 1 if the transition from $S_b^{(j)}$ to $S_a^{(j)}$ is the synchronized event, and 0 otherwise.

The last element $\mathbf{D}_k^{(j)} \in \mathbb{R}^{|S^{(j)}|\times|S^{(j)}|}$ is diag$(\mathbf{1}_{1\times|S^{(j)}|} \cdot \mathbf{E}_k^{(j)})$. That is, the diagonal matrix of the column sums of $\mathbf{E}_k^{(j)}$.

Example 2 *As an example, let us build a composite DTMC of Figure 2.2. In this case, let us assume that the DTMC PS is obtained at the sampling rate of 100/sec and the commands are sent at the rate of* $\lambda = 10/sec$. *The generator matrices* $\mathbf{Q}^{(A)}$ *and* $\mathbf{Q}^{(B)}$ *can be computed from* $e^{\frac{\mathbf{Q}^{(A)}}{100}} = e^{\frac{\mathbf{Q}^{(B)}}{100}} = \mathbf{M}$. *Sending the command moves A from the Run state to the Wait state. Thus,* $\mathbf{E}_{3,2}^{(A)} = 1$ *and* $\mathbf{D}_{2,2}^{(A)} = 1$. *Similarly, because receiving the command moves B from the Run state to the Ready state,* $\mathbf{E}_{3,1}^{(B)} = 1$ *and* $\mathbf{D}_{1,1}^{(B)} = 1$. *Thus, the generator matrix of the composite system is*

$$\mathbf{Q} \quad = \quad \mathbf{Q}^{(A)} \oplus \mathbf{Q}^{(B)} + 10 \cdot \left(\mathbf{E}^{(A)} \otimes \mathbf{E}^{(B)} - \mathbf{D}^{(A)} \otimes \mathbf{D}^{(B)}\right).$$

The resulting DTMC is $(S \times S, e^{\frac{\mathbf{Q}}{100}})$.

■

The stochastic model produced by composing components can be very large and often has redundant states. This model can be represented more compactly by merging those redundant states. Kemeny and Snell showed a condition when a DTMC can be lumped into another simpler DTMC [30].

Definition 1 *Let* $(S, \mathbf{P})$ *be a DTMC with a set of states* $S = \{s_1, \ldots, s_n\}$, *and with a probability transition matrix* $\mathbf{P}$, *and let* $C = \{C_1, \ldots, C_k\}$ *be a partition of* S, *and let* $P_{C_j i} = \sum_{s_k \in C_j} P_{ki}$. *Then, the DTMC is* lumpable *with respect to C if and only if for every* C_i *and* C_j *in* C, $P_{C_j k} = P_{C_j k'}$ *for any* $s_k, s_{k'} \in C_i$.

Definition 2 *Let* $(S, \mathbf{P})$ *be a DTMC with* $S = \{s_1, \ldots, s_n\}$, *and let* $C = \{C_1, \ldots, C_k\}$ *be a partition of* S. $\mathbf{V}_{S,C} \in \mathbb{R}^{k\times n}$ *is a matrix such that* V_{ij} *is 1 if* $s_j \in C_i$ *and 0 otherwise. An* $\mathbb{R}^{n\times k}$ *matrix* $\mathbf{U}_{S,C}$ *is*

$$\mathbf{U}_{S,C} = \left[\frac{\mathbf{V}_{1*}^T}{\sum_{i=1}^n V_{1i}}, \ldots, \frac{\mathbf{V}_{k*}^T}{\sum_{i=1}^n V_{ki}}\right],$$

where $\mathbf{V}_{i*}$ *is the* i^{th} *row of* $\mathbf{V}$.

Given definition 2, Kemeny and Snell showed that a DTMC $(S, \mathbf{P})$ is *strongly lumpable*[2] with respect to a partition C if and only if

$$\mathbf{V} \cdot \mathbf{P} \cdot \mathbf{U} \cdot \mathbf{V} = \mathbf{V} \cdot \mathbf{P}.$$

If a DTMC is lumpable then the lumped DTMC is $(C, \mathbf{V} \cdot \mathbf{P} \cdot \mathbf{U})$.

A large Markov chain made up with many components such as SAN can be simplified by applying the lumping technique. In this thesis we regard a large scale system as a composite system of independent Markov chains. Using the state lumping condition we can reduce the composite system to an individual Markov chain. We also found that a composite system of independent processes with a single leader process can be lumped to the system of the leader process. Proofs for these lumping conditions can be found in Appendix A.

2.5 Temporal Logics

Temporal logics are logics about computations over time. Because of their expressiveness and conciseness, they have been widely used in hardware/software verification methods. The verification is performed in the process model checking that checks whether the behaviors of a model is within the allowed behaviors of specification.

Three logics called *Linear Temporal Logic* (LTL), *Computation Tree Logic* (CTL), and CTL* are the most commonly used temporal logics. These logics have logical connectives such as *not*, *and*, and *or* as well as temporal connectives such as *next*, *eventually*, *always*, *until*, and *release*, so that properties over time can be precisely expressed. Model checking is a process that checks whether a computational model called a Kripke structure [27] satisfies a specification expressed in these logics.

A *Kripke structure* K is a quadruple $K = (S, S_0, R, L)$, where S is a finite set of states, $S_0 \subseteq S$ is a set of initial states, $R \subseteq S \times S$ is a total transition relation, and $L : S \rightarrow 2^{AP}$ is a labeling function that returns the set of atomic propositions that are true in the given state, where AP is the set of atomic propositions. A

[2]Strongly lumpable means lumpable regardless of initial pmf. In this thesis we use the term ‘lumpable’ for ‘strongly lumpable’.

$K, s \models \mathtt{T}$		
$K, s \not\models \mathtt{F}$		
$K, s \models p$	$\Leftrightarrow$	$p \in L(s)$
$K, s \models \neg f$	$\Leftrightarrow$	$K, s \not\models f$
$K, s \models f \vee g$	$\Leftrightarrow$	$K, s \models f$ or $K, s \models g$
$K, s \models f \wedge g$	$\Leftrightarrow$	$K, s \models f$ and $K, s \models g$
$K, s \models \mathtt{E}\psi$	$\Leftrightarrow$	there is a path π with $\pi[0] = s$ such that $K, \pi \models \psi$
$K, s \models \mathtt{A}\psi$	$\Leftrightarrow$	for all path π such that $\pi[0] = s$, $K, \pi \models \psi$
$K, \pi \models f$	$\Leftrightarrow$	$K, s \models \psi$ with $\pi[0] = s$
$K, \pi \models \neg\psi$	$\Leftrightarrow$	$K, \pi \not\models \psi$
$K, \pi \models \psi \vee \phi$	$\Leftrightarrow$	$K, \pi \models \psi$ or $K, \pi \models \phi$
$K, \pi \models \psi \wedge \phi$	$\Leftrightarrow$	$K, \pi \models \psi$ and $K, \pi \models \phi$
$K, \pi \models \mathtt{X}\psi$	$\Leftrightarrow$	$K, \pi^1 \models \psi$
$K, \pi \models \psi \mathtt{U} \phi$	$\Leftrightarrow$	there exists an $i \geq 0$ such that $K, \pi^i \models \phi$ and $K, \pi^j \models \psi$ for $j = 0, \ldots, i-1$
$K, \pi \models \psi \mathtt{R} \phi$	$\Leftrightarrow$	for all $j \geq 0$, if for every $i < j$ $K, \pi^i \not\models \psi$ then $K, \pi^j \models \phi$.

Table 2.1: Semantics of CTL*: a ternary satisfaction relation $\models$.

computational path π of K is an infinite sequence of states $\pi = s_0 s_1 s_2 \cdots$ such that $s_0 \in S_0$ and $(s_i, s_{i+1}) \in R$ for all $i \geq 0$. $\pi[i]$ represents the i^{th} state of π and π^i is the suffix of π beginning from s_i.

The syntax of a CTL* is:

$$
\begin{aligned}
f \quad ::= \quad & \mathtt{T} \mid \mathtt{F} \mid p \in AP \mid \\
& \neg f \mid f \vee g \mid f \wedge g \mid \\
& \mathtt{E}\psi \mid \mathtt{A}\psi, \\
\psi \quad ::= \quad & f \mid \\
& \neg\psi \mid \psi \vee \phi \mid \psi \wedge \phi \mid \\
& \mathtt{X}\psi \mid \psi \mathtt{U} \phi \mid \psi \mathtt{R} \phi \mid \Box\psi \mid \Diamond\psi,
\end{aligned}
$$

where f and g are state formula and ψ and ϕ are path formula. CTL is a restricted CTL* in which the $\mathtt{E}$ and $\mathtt{A}$ path quantifiers are required to precede temporal operators. LTL is another restricted CTL* where formulas are fof the form $\mathtt{A}\psi$, where ψ is a path formula whose only state subformulas are atomic propositions.

The meaning of a CTL* formula with respect to a Kripke structure $K = (S, S_0, R, L)$ and an initial state is defined by a ternary satisfaction relation $\models$ defined in Table 2.1.

The followings are commonly used equivalence relations:

$$\begin{aligned}
\Diamond\psi &\equiv \mathtt{T}\mathrm{U}\psi\\
\Box\psi &\equiv \mathtt{F}\mathrm{R}\psi\\
\psi\mathrm{U}\phi &\equiv \phi \vee (\psi \wedge \mathtt{X}\,(\psi\mathrm{U}\phi))\\
\psi\mathrm{R}\phi &\equiv (\psi \wedge \phi) \vee (\phi \wedge \mathtt{X}\,(\psi\mathrm{R}\phi))\,.
\end{aligned}$$

Model checking a CTL formulae f can be done by labeling each state of S with subformula of f that is true in that state. Because any CTL formula can be expressed in terms of $\neg$, $\vee$, $\mathtt{EX}$, $\mathtt{EU}$, and $\mathtt{E}\Box$, it is enough to describe model checking algorithm for formulas built up this way.

- $\neg f$: $label(s) \leftarrow label(s) \cup \{\neg f\}$ if $f \notin label(s)$.
- $f \vee g$: $label(s) \leftarrow label(s) \cup \{f \vee g\}$ if $f \in label(s)$ or $g \in label(s)$.
- $\mathtt{EX}\psi$: $label(s) \leftarrow label(s) \cup \{\mathtt{EX}\psi\}$ if there is s' such that $(s, s') \in R$ and $\psi \in label(s')$.
- $\mathtt{E}\psi\mathrm{U}\phi$: $label(s) \leftarrow label(s) \cup \{\mathtt{E}\psi\mathrm{U}\phi\}$ if $\phi \in label(s)$ or if $\psi \in label(s)$ and there is s' such that $(s, s') \in R$ and $\mathtt{E}\psi\mathrm{U}\phi \in label(s')$.
- $\mathtt{E}\Box\psi$: $label(s) \leftarrow label(s) \cup \{\mathtt{E}\Box\psi\}$ if there is a cycle in R such that for every state s' in the cycle $\psi \in label(s')$ and there is a path from s to a state in the cycle such that for every state s' in the path $\psi \in label(s')$.

An LTL formulae ψ can be model checked by building a Büchi automaton $\mathcal{A}_{\neg\psi}$ that accepts only those strings that satisfy $\neg\psi$ and checking the emptiness of $\mathcal{L}_{\neg\psi} \cap \mathcal{L}_K$, where $\mathcal{L}_\psi$ is the set of paths that $\mathcal{A}_{\neg\psi}$ accepts and $\mathcal{L}_K$ is the set of paths that K can generate.

A Büchi automaton [11, 69] is a finite automaton that accepts infinite strings. Formally, a *Büchi automaton* $\mathcal{A}$ is a quintuple $\mathcal{A} = (\Sigma, Q, \Delta, Q_0, F)$, where Σ is a finite alphabet, Q is a finite set of states, $\Delta \subseteq Q \times \Sigma \times Q$ is a transition relation, $Q_0 \subseteq Q$ is a set of initial states, and $F \subseteq Q$ is a set of accepting states. Let $v \in \Sigma^\omega$ be a string of length $|\mathbb{N}|$. Then, a *run* of $\mathcal{A}$ over v is a mapping $r : \mathbb{N} \to Q$ such that $r(0) \in Q_0$ and $(r(i), v[i], r(i+1)) \in \Delta$ for $i \geq 0$. Let $inf(r)$ be a set of states occurring infinitely often in r. Then, r is *accepting* if and only if $inf(r) \cap F \neq \emptyset$. The *language* $\mathcal{L}_\mathcal{A} \subseteq \Sigma^\omega$ of $\mathcal{A}$ consists of all paths accepted by $\mathcal{A}$.

A Kripke structure $K = (S, R, S_0, L)$ where $L : S \rightarrow 2^{AP}$ can be directly converted into a Büchi automaton

$$\mathcal{A} = (2^{AP}, S \cup \{i\}, \Delta, \{i\}, S \cup \{i\}),$$

where

- for all $s, s' \in S$, $(s, \alpha, s') \in \Delta$ iff $(s, s') \in R$ and $\alpha = L(s')$.

- for all $s \in S$, $(i, \alpha, s) \in \Delta$ iff $s \in S_0$ and $\alpha = L(s)$.

An LTL formulae can be converted into a Büchi automaton by the *expand* algorithm of Table 2.2 [23]. The expand algorithm uses a structure called *node: <id, I, O, N, X>* where *id* is an identifier of this node, *I* is a set of ids of predecessor nodes of this node, *O* is a set of LTL formulas that is true in this node, *N* is a set of LTL formulas not yet processed, and *X* is a set of LTL formulas that should be true at the next step of this node. Before applying the expand algorithm, an LTL formula is converted to a *negation normal form* so that the ¬ operator appears only at atomic propositions. The conversion is done by recursively applying the following transform:

$$\begin{array}{lcl}
\Diamond\psi & \rightarrow & \mathtt{T}\mathtt{U}\psi \\
\Box\psi & \rightarrow & \mathtt{F}\mathtt{R}\psi \\
\neg\neg\psi & \rightarrow & \psi \\
\neg(\psi \vee \phi) & \rightarrow & \neg\psi \wedge \neg\phi \\
\neg(\psi \wedge \phi) & \rightarrow & \neg\psi \vee \neg\phi \\
\neg\mathtt{X}\psi & \rightarrow & \mathtt{X}\neg\psi \\
\neg(\psi\mathrm{R}\phi) & \rightarrow & \neg\psi\mathtt{U}\neg\phi \\
\neg(\psi\mathtt{U}\phi) & \rightarrow & \neg\psi\mathrm{R}\neg\phi.
\end{array}$$

CTL* model checking can be done by recursively applying CTL or LTL model checking algorithms depending on the type of subformula of CTL*: update the labeling function to include already checked state formulae as a new atomic proposition.

```
expand(n, nodes)
   if n.N=∅
      if ∃m∈nodes s.t.  m.O=n.O and m.X=n.X
         m.In = m.In ∪ n.In
      else
         expand(<newId(),{n.id},∅,n.X,∅>, nodes∪{n})
   else
      let $\eta$ ∈n.N
      n.N=n.N-{$\eta$}
      if $\eta$ ∈n.O
         expand(n, nodes)
      else
         n.O=n.O∪{$\eta$}
         if $\eta$ = F or $\eta \in AP$ and $\neg\eta$ ∈n.O
            return nodes
         if $\eta$ = T or $\eta \in AP$
            expand(n, nodes)
         if $\eta = \psi \wedge \phi$
            expand(<n.I,n.O,n.N,N.X∪{$\psi, \phi$}>, nodes)
         if $\eta = \psi \vee \phi$
            expand(<newId(),n.I,n.O,n.N∪{$\psi$},n.X>,
               expand(<newId(),n.I,n.O,n.N∪{$\phi$},n.X>, nodes)
         if $\eta = \psi \mathrm{U} \phi$
            expand(<newId(),n.I,n.O,n.N∪{$\psi$},n.X∪{$\psi\mathrm{U}\phi$}>,
               expand(<newId(),n.I,n.O,n.N∪{$\phi$},n.X>, nodes)
         if $\eta = \psi \mathrm{R} \phi$
            expand(<newId(),n.I,n.O,n.N∪{$\phi$},n.X∪{$\psi\mathrm{R}\phi$}>,
               expand(<newId(),n.I,n.O,n.N∪{$\psi, \phi$},n.X>, nodes)
         if $\eta = \mathrm{X}\psi$
            expand(<n.I,n.O,n.N,N.X∪{$\psi$}>, nodes)
```

Table 2.2: Expand Algorithm

2.6 Probabilistic Temporal Logics

A number of probabilistic temporal logics such as PCTL, PCTL*, and CSL can be used to specify state transitions of Markov chains [25, 4, 5, 3, 32, 71]. With these logics one can specify the probability that a Markov process follows certain specified paths. For example, in a multi-path communication network, we can express the property "with probability larger than 0.5 a packet will arrive at its destination with path length less than 5." These logics are described on a probability space built on a set of computational paths. That is, in the multi-path example, the sample space is a set of all routing paths and the event is a set of paths whose routing length is less than 5 starting from a certain state.

2.6.1 PCTL* and PCTL

Probabilistic real time computational tree logic (PCTL) [25, 32] is a probabilistic *computational tree logic* (CTL) [19] to model check a DTMC [64]. Differences between PCTL and CTL are: $\mathtt{A}$ (for all) and $\mathtt{E}$ (exists) operators of CTL are replaced by probabilistic operators $P_{<c}$ and $P_{>c}$ and $\mathtt{U}$ operator of CTL is extended to $\mathtt{U}^{\leq t}$ where $t \in \mathbb{R} \cup \{\infty\}$. If $t = \infty$, $\mathtt{U}^{\leq t}$ is equivalent to the $\mathtt{U}$ of CTL. PCTL has also been extended to PCTL* [4]. The relation between PCTL* and PCTL is similar to that of CTL* and CTL [20]: as $\mathtt{X}$ and $\mathtt{U}$ operators are immediately preceded by $\mathtt{A}$ or $\mathtt{E}$ operators in CTL, $\mathtt{X}$ and $\mathtt{U}$ operators should be immediately preceded by probability operators in PCTL.

The syntax of PCTL* is:

$$\begin{aligned} f &::= a \in AP \mid \neg f \mid f \vee g \mid P_{<c}(\phi) \mid P_{>c}(\phi), \\ \phi &::= f \mid \neg\phi \mid \phi \vee \psi \mid \mathtt{X}\phi \mid \phi\mathtt{U}^{\leq t}\psi, \end{aligned}$$

where f and g are state formulas, AP is a finite set of atomic propositions, and ϕ and ψ are path formulas. PCTL* is a set of state formulas.

PCTL can be defined by restricting the path formulas of PCTL* as follows:

$$\begin{aligned} f &::= a \in AP \mid \neg f \mid f \vee g \mid P_{<c}(\phi) \mid P_{>c}(\phi), \\ \phi &::= \mathtt{X}f \mid f\mathtt{U}^{\leq t}g. \end{aligned}$$

PCTL* model checks a DTMC structure $X = (S, s^i, \mathbf{M}, L)$[3], where S is a set of states, $s^i \in S$ is an initial state, $\mathbf{M} \in \mathbb{R}^{|S|\times|S|}$ is a probability transition function such that $M_{i,j} = \mathrm{P}[X(t+1) = s_i | X(t) = s_j]$, and $L : S \to 2^{AP}$ is a labeling function that returns the set of atomic propositions that are true in the given state. For the DTMC structure we can define a probability space $\mathcal{P} = (\Omega, \Delta, \mu)$ [64], where $\Omega = S^\omega$ is a set of all infinite sequence of states, Δ is a Borel σ-field: a set of $\Delta_\pi \subseteq \Omega$ for all finite sequences $\pi \in S^*$ such that

$$\Delta_\pi = \{\, \sigma \in \Omega : \sigma[i] = \pi[i], \text{ for } i = 1, \ldots, |\pi| \,\},$$

[3]Note that this definition of DTMC is used only in this section. For the other sections of this thesis we use the definition of Section 2.1.

where $|\pi|$ is the length of π, and μ is a probability measure such that

$$\mu(\Delta_\pi) = \prod_{i=2}^{|\pi|} T(\pi[i-1], \pi[i]).$$

A key element of the model checking algorithm is to compute the probability that $f\mathrm{U}^{\leq t}g$ is satisfied from each state. For this we assume that the labeling of subformulas f and g is already done. Let $\hat{S} \subseteq S$ be the set

$$\hat{S} = \{s \in S : f \in label(s) \text{ and } g \notin label(s)\},$$

$\mathbf{N} \in \mathbb{R}^{|S|\times|S|}$ be a matrix

$$N_{i,j} = \begin{cases} M_{j,i} & \text{if } s_i \in \hat{S} \\ 1 & \text{if } s_i \notin \hat{S} \text{ and } i = j \\ 0 & \text{otherwise} \end{cases},$$

and $\mathbf{x}(0) \in \mathbb{R}^{|S|}$ be a column vector

$$x_i(0) = \begin{cases} 1 & \text{if } s_i \in label(g) \\ 0 & \text{otherwise} \end{cases}$$

then

$$\mathrm{P}[X, s_i \models f\mathrm{U}^{\leq t}g] = x_i(t), \quad \text{where} \quad \mathbf{x}(t) = \mathbf{N}^t \cdot \mathbf{x}(0).$$

Bianco and de Alfaro [8] introduced nondeterminism to the interpretation of PCTL and PCTL*. The nondeterminism captures the effect of a scheduler: nondeterministic choices of a scheduler give different transition probabilities. For each element Δ_π of a Borel σ-field, they associate two different probabilities: the maximal probability $\mu^+(\Delta_\pi)$, when scheduling is most favorable, and the minimal probability $\mu^-(\Delta_\pi)$, when scheduling is most unfavorable. The meaning of probability operators becomes more conservative:

$$X, s \models P_{\geq a}\phi \quad \Leftrightarrow \quad \mu^-(\{\sigma \in \Omega : X, \sigma \models \phi \text{ and } \sigma[0] = s\}) \geq a$$

$$X, s \models P_{\leq a}\phi \quad \Leftrightarrow \quad \mu^{+}(\{\sigma \in \Omega : \; X, \sigma \models \phi \text{ and } \sigma[0] = s\}) \leq a.$$

Probabilistic reward CTL (PRCTL) is an extension of PCTL so that it can specify properties about Markov Reward Model [15]. PRCTL has operators on rewards such as $\mathcal{E}^n_J(\phi)$, $\mathcal{E}_J(\phi)$, $C^n_J(\phi)$, $\mathcal{Y}^n_J(\phi)$ as well as PCTL operators. Given a state s, the respective meaning of these operators is whether the expected reward rate ($\mathcal{E}$), the instantaneous reward (C), or the expected accumulated reward ($\mathcal{Y}$) are within the interval J if $\mathrm{P}[X(t) = s] = 1$.

2.6.2 Continuous Stochastic Logic (CSL)

A *continuous time Markov chain* (CTMC) is a triple $X = (S, \mathbf{Q}, L)$ [4], where $S = \{s_1, \ldots, s_n\}$ is a finite set of states, $\mathbf{Q} \in \mathbb{R}^{n \times n}$ is a generator matrix, and $L : S \rightarrow 2^{AP}$ is a labeling function. State transition probability has exponential distribution with rates specified by $\mathbf{Q}$. That is, the probability of moving from state s_j to s_i within t time-unit is $1 - e^{-\mathrm{Q}_{i,j} \cdot t}$.

Let a path be an infinite sequence of tuples (s, t), with $s \in S$ and $t \in \mathbb{R}^{+}$ such that

$$\sigma = s_0 \xrightarrow{t_0} s_1 \xrightarrow{t_1} s_2 \xrightarrow{t_2} \cdots,$$

and for a path $\sigma = (s_0, t_0)(s_1, t_2)\cdots$, let $\sigma[i]$ be s_i the i^{th} state X visits in σ, and let $\delta(\sigma, i)$ be t_i, the time X remains in $\sigma[i]$. Let $\mathrm{E}_j = \sum_{i=1}^{n} \mathrm{Q}_{i,j}$ be the rate at which X leaves from state s_j, and let the embedded DTMC $\mathbf{P} \in \mathbb{R}^{n \times n}$ be

$$\mathrm{P}_{i,j} = \begin{cases} \mathrm{Q}_{i,j}/\mathrm{E}_j & \text{if } \mathrm{E}_j \neq 0 \\ 0 & \text{otherwise} \end{cases} .$$

Let $C(s_0, I_0, \ldots, I_{k-1}, s_k)$ be a cylinder set consisting of all paths $\sigma \in (S, \mathbb{R})^{\omega}$ such that $\sigma[i] = s_i$ for $i \leq k$ and $\delta(\sigma, i) \in I_i$ for $i < k$. Then we can define a probability space $\mathcal{P} = (\Omega, F, \mu)$ where Ω is a set of paths, the σ-field F is generated by sets of the form $C(s_0, I_0, \ldots, I_{k-1}, s_k)$, and the probability measure is defined as

$$\mu(C(s_0)) \quad = \quad 1$$

[4]We use this definition of CTMC in this section only. For the other sections we use the definition of Section 2.3.

$$\mu(C(s_0,\ldots,s_k,I_k,s_{k+1})) \quad = \quad \mu(C(s_0,\ldots,s_k)) \cdot \mathrm{P}_{i,j} \cdot \left(e^{-\mathrm{E}_j \cdot a} - e^{-\mathrm{E}_j \cdot b}\right),$$

where i and j are the indexes of s_{k+1} and s_k in S respectively, $a = \inf(I_k)$, and $b = \sup(I_k)$.

Continuous stochastic logic (CSL) [5, 3, 32, 7] is a temporal logic to reason in the probability space defined in the previous paragraph by a CTMC. The syntax of CSL is

$$\begin{aligned} f &::= \mathtt{T} \mid a \in AP \mid f \wedge g \mid \neg f \mid \mathcal{S}_{\bowtie p} f \mid \mathcal{P}_{\bowtie p}\phi \\ \phi &::= \mathtt{X} f \mid f \mathtt{U} g \mid f \mathtt{U}^{\leq t} g, \end{aligned}$$

where $p \in [0,1]$, $\bowtie \in \{\leq, <, \geq, >\}$, f and g are state formula, and ϕ is a path formula. Note that t of $\mathtt{U}^{\leq t}$ is a real number. The $\mathcal{S}_{\bowtie p} f$ operator compares the constant p with the probability that f is true in the steady state, and the $\mathcal{P}_{\bowtie p}\phi$ operator compares a constant p with the probability that ϕ is true in the probability space defined in the previous paragraph.

The key elements of CSL model checking is to compute the probability measure of paths satisfying the path formula. Let ϕ and ψ be state formulas such that labeling on them is already done, then

- $\mathrm{P}[X, s_j \models \mathtt{X}\phi] = \sum_{s_i \in label(\phi)} \mathrm{P}_{i,j}$.
- $\mathrm{P}[X, s_j \models \phi \mathtt{U} \psi]$ is determined by computing the least fixed point of the operator $\Theta : (S \to \mathbb{R}) \to (S \to \mathbb{R})$, where

$$\Theta(F)(s_j) = \begin{cases} 1 & \text{if } s_j \in label(\phi) \\ \sum_{s_i \in S} \mathrm{P}_{i,j} \cdot F(s_j) & \text{if } s_j \in label(\psi) \text{ and } s_j \notin label(\phi) \\ 0 & \text{otherwise} \end{cases} .$$

- $\mathrm{P}[X, s_j \models \phi \mathtt{U}^{\leq t} \psi]$ is determined by computing the least fixed point of the operator $\Omega : (S \times \mathbb{R}^+ \to \mathbb{R}) \to (S \times \mathbb{R}^+ \to \mathbb{R})$, where

$$\Omega(F)(s_j, t) = \begin{cases} 1 & \text{if } s_j \in label(\phi) \\ \sum_{s_i \in S} \mathrm{Q}_{i,j} \cdot \int_0^t e^{-\mathrm{E}_j \cdot \tau} \cdot F(s_j, t-\tau) d\tau & \text{if } s_j \in label(\psi) \text{ and } s_j \notin label(\phi) \\ 0 & \text{otherwise} \end{cases} .$$

For more information please see [5].

Chapter 3

Specification Logic

In this chapter we explain the syntax and the semantics of the probabilistic temporal logic iLTL. We extend iLTL so that it can specify concurrent runs of multiple Markov chains. We also explain the syntax and the semantics of the extended iLTL. Finally, an iLTL model checking algorithm is present in the last section. We show that with certain DTMCs the inequalities about expected rewards become constant and we compute an upper-bound when all of them become constants. Explanations about how to convert iLTL model checking problem into linear programming problem are present at the end.

3.1 Motivations for a New Logic

A model checking algorithm for the PCTL-like logics must repeatedly compute such probabilities from each state for each sub-formulas beginning with the probability operator of a specification, until the probability for the whole specification from the initial state is computed. Although these logics are good for reasoning about transitions of a single process, they are not suitable for DTMCs as abstractions for large scale systems. This is largely due to the fact that they are reasoning on the probability space of computational paths instead of reasoning directly on transitions of the *probability mass function* (pmf).

We illustrate the problem with a DTMC (Figure 3.1). This DTMC has two states: `Run` (*R*) and `Sleep` (*S*). The transition probabilities from *R* to *S* is 1 as well as from *S* to *R*. That is, a node always makes a transition to the other state. Hence, from a single process point of view, there is no consecutive *R* or consecutive *S* states in any computational paths. So, the probability that a specification "*always R*" is satisfied is 0 and PCTL-like logics captures this properly. Now, suppose that there are 100 processes and initially 50 of them are in state *R* and the others are in state *S*. Then, there are always 50 processes in one of the two states and

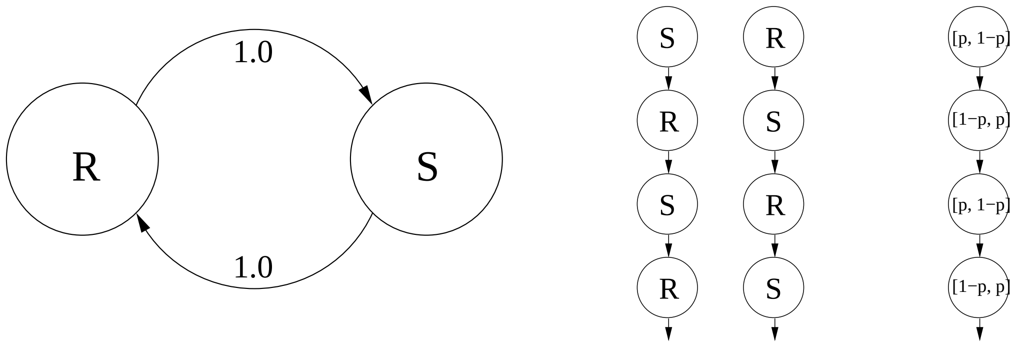


Figure 3.1: An oscillating DTMC (left), and its two different types of computational paths: the paths of states, and the paths of pmfs. Note that there are only two paths of states, but there are uncountably many number of paths of pmfs

a specification like "*always* $P[R] \geq 0.5$" should be true. If all 100 processes are in state R or in state S then the previous specification should be false by the same reason as "*always R*". For this system, because $P[R] \geq 0.5$ is false in S state, PCTL-like logics compute the probability of satisfying the specification to be 0, regardless of the initial condition. Thus, the semantics of PCTL-like logics is not suitable to express properties on transitions of pmf. Furthermore, suppose that a node consumes 10 (mA) in state R and 1 (mA) in state S. The expected energy consumption level at a certain moment is $10 \cdot P[R] + 1 \cdot P[S]$ (mA). Because, PCTL-like logics do not work on the transitions of pmfs, they have limitations in specifying the expected values of performance criteria.

In order to address this problem, we have designed a logic called *iLTL* and have implemented a model checker *iLTLChecker* [33]. iLTL is a *Linear Temporal Logic* (LTL) [41] whose atomic propositions are linear inequalities about pmfs. Those inequalities can be thought of as inequalities about expected "rewards", such as the expected energy consumption or the availability of a system. Instead of working on a probability space of computational paths, iLTL specifies properties based on the pmf transitions. The model checker for iLTL searches for an initial pmf for which trailing transitions of expected rewards violate the specification. This addresses the problem described above. Furthermore, because the iLTL model checker looks for an initial pmf that may lead to a violation of the specification, it can be used as a predictive monitoring tool. For example we can specify the property $(0.1 < P[Ready] \wedge P[Ready] < 0.2) \rightarrow X\,X\,X\,\Box\,(P[Ready] > 0.5)$: if the current interval estimate of the availability of the system is 10% to 20%, three steps later (X X X) the availability of the system always ($\Box$) becomes larger than 50%.

3.2 iLTL

An abstraction mechanism is necessary to make the state representation of a large scale system useful. We represent states of such system by pmf vectors because many useful performance metrics can be expressed in terms of expected values. We have developed a linear temporal logic called iLTL focused on model checking properties about pmf transitions. The model to be checked by iLTL is a DTMC: the atomic propositions of iLTL are linear inequalities about pmf or expected rewards. A path of a DTMC is an infinite sequence of pmf vectors. So, iLTL reasons about a sequence of expected rewards: looking for an initial pmf whose trailing path violates the specification.

Other probabilistic temporal logics such as PCTL or CSL also model check discrete time or continuous time Markov chains (Kripke structures with transition probabilities/rates attached to transition relations). However their primary focus is not on the pmf itself: they define a probability space $(\Omega, \mathcal{F}, P)$ [64] with a set of paths of the Kripke structure as a sample space Ω, sets of paths with common finite prefixes as σ-field $\mathcal{F}$, and the probability measure of an event P is defined as the product of transition probabilities of the common prefixes. So, it is not possible to reason about expected rewards or to compare two probabilities possibly at different times with these logics.

3.2.1 Syntax

The syntax of the probabilistic temporal logic iLTL is as follows:

$$\begin{aligned} \psi \quad &::= \quad T \mid F \mid ineq \mid \\ & \qquad \neg\psi \mid \psi \vee \phi \mid \psi \wedge \phi \mid \\ & \qquad \mathtt{X}\,\psi \mid \psi\,\mathtt{U}\,\phi \mid \psi\,\mathtt{R}\,\phi \\ ineq \quad &::= \quad \textstyle\sum_{i=1}^{n} a_i \cdot \mathrm{P}[X = s_i] \bowtie b, \end{aligned}$$

where $X = (\{s_1, \ldots, s_n\}, \mathbf{M})$, $a_i \in \mathbb{R}$, $b \in \mathbb{R}$, and $\bowtie \in \{=, \neq, <, \leq, >, \geq\}$. As usual, $\Box$ and $\Diamond$ are defined as follows:

$$\begin{aligned} \Box\,\psi \quad &\equiv \quad F\,\mathtt{R}\,\psi, \\ \Diamond\,\psi \quad &\equiv \quad T\,\mathtt{U}\,\psi. \end{aligned}$$

3.2.2 Semantics

In order to explain the semantics of an iLTL formula we first define a computational path of a DTMC $X = (\{s_1, \ldots, s_n\}, \mathbf{M})$ by a function $\sigma_{\mathbf{x}} : \mathbb{N} \rightarrow \mathbb{R}^n$ such that $\sigma_{\mathbf{x}}(t) = \mathbf{M}^t \cdot \mathbf{x}$. Note that $\sigma_{(\sigma_{\mathbf{x}}(t))}$ is the suffix of $\sigma_{\mathbf{x}}$ starting from $\sigma_{\mathbf{x}}(t)$. The semantics of an iLTL formula should be interpreted over these paths. The atomic propositions of iLTL have the form of linear inequalities (*ineq*) over a pmf. The linear inequalities can be regarded as inequalities about expected rewards: we associate a constant reward with each state of a Markov chain and when a process visits a state it earns the reward associated with that state. The expected reward earned at a given point in time is the sum of the products of the rewards at each state and the probability that a process is in that state. Formally, a *Markov Reward Process* (MRP) is a triple $(S, \mathbf{M}, \rho)$ where $(S, \mathbf{M})$ is a DTMC and $\rho : S \rightarrow \mathbb{R}$ is a reward function for each state [15]. Then the expected reward of a MRP $(S, \mathbf{M}, \rho)$ at time t is

$$\sum_{s_i \in S} \rho(s_i) \cdot \mathrm{P}[X(t) = s_i].$$

Thus, a linear inequality over a pmf is an inequality about expected reward: replace a_i with $\rho(s_i)$ and ignore the time index t. The time index is implicitly given from the temporal operators of iLTL: the pmf of *ineq* is replaced by $\sigma_{\mathbf{x}}(0)$.

Many useful properties of a system can be expressed as inequalities about expected rewards. For example, a comparison between two probabilities "$\mathrm{P}[X = A] > \mathrm{P}[X = B]$" can be rewritten in a reward form as "$1 \cdot \mathrm{P}[X = A] - 1 \cdot \mathrm{P}[X = B] > 0$", where 1 and -1 are rewards in states A and B. An expected value of meaningful quantities such as expected queue length can also be expressed in reward form: "$1 \cdot \mathrm{P}[Q = Q_1] + \cdots + n \cdot \mathrm{P}[Q = Q_n] < L$" specifies that the expected queue length of a queuing system of capacity n is less than L. We can also compare two probabilities at different time in the reward form: suppose that $X = (\{s_1, \ldots, s_n\}, \mathbf{M})$ is a DTMC then

$$\begin{aligned} &\mathrm{P}[X(t) = s_i] > \mathrm{P}[X(t+k) = s_j] \ \text{ iff} \\ &\mathrm{P}[X(t) = s_i] - \sum_{j=1}^{n} m_j \cdot \mathrm{P}[X(t) = s_j] > 0, \end{aligned}$$

where $[m_1, \ldots, m_n]$ is the j^{th} row of $\mathbf{M}^k$. One of the reasons iLTL does not explicitly use time-indexes is because it can be written in a reward form as above.

The meaning of logical connectives $\wedge$, $\vee$, and $\neg$ are as usual: $\psi \wedge \phi$ is true if and only if ψ and ϕ are both true, $\psi \vee \phi$ is true if and only if ψ or ϕ are true, and $\neg\psi$ is true if and only if ψ is false.

The semantics of the temporal operators $\mathtt{X}$, $\mathtt{U}$, and $\mathtt{R}$ is explicitly related to a computational path $\sigma_{\mathbf{x}}$. $\mathtt{X}\,\psi$ in $\sigma_{\mathbf{x}}$ is true if and only if ψ is true in $\sigma_{(\sigma_{\mathbf{x}}(1))}$. $\psi\,\mathtt{U}\,\phi$ in $\sigma_{\mathbf{x}}$ is true if and only if ϕ eventually becomes true along the path of $\sigma_{\mathbf{x}}$ and while ϕ is false ψ is true in $\sigma_{\mathbf{x}}$. In other words, $\psi\,\mathtt{U}\,\phi$ in $\sigma_{\mathbf{x}}$ is true if and only if there is $t \geq 0$ such that ϕ is true in $\sigma_{(\sigma_{\mathbf{x}}(t))}$ and for all $s \in [0, t-1]$, ψ is true in $\sigma_{(\sigma_{\mathbf{x}}(s))}$. The until operator ($\mathtt{U}$) can be easily explained by an example: suppose that events are happening in a WSN and one may want to keep the WSN in an alert mode until most of the nodes detect them. Then the specification: $(\mathrm{P}[X = \mathit{Ready}] > 0.5)\,\mathtt{U}\,(\mathrm{P}[X = \mathit{Detect}] > 0.9)$ checks whether the event will be detected by more that 90% of the nodes and until that moment more than 50% of the nodes are in ready state. The *release* operator $\mathtt{R}$ is a dual of the $\mathtt{U}$ operator. $\psi\,\mathtt{R}\,\phi$ in $\sigma_{\mathbf{x}}$ is true if and only if ϕ is true in $\sigma_{\mathbf{x}}$ while ψ is false. The *eventually* operator $\diamond\,\psi \equiv T\,\mathtt{U}\,\psi$ in $\sigma_{\mathbf{x}}$ is true if and only if ψ eventually become true in $\sigma_{\mathbf{x}}$ and the *always* operator $\Box\,\psi \equiv F\,\mathtt{R}\,\psi$ in $\sigma_{\mathbf{x}}$ is true if and only if ψ is always true in $\sigma_{\mathbf{x}}$.

Finally, we can define a satisfaction relation $\models$ between an iLTL formula and a DTMC. We say that a DTMC A is a *model of* an iLTL formula ψ and write $A \models \psi$ if and only if for all pmf transitions $\sigma_{\mathbf{x}}$ of A ψ is true in $\sigma_{\mathbf{x}}$. Note that each initial pmf $\mathbf{x}$ determines a computational path $\sigma_{\mathbf{x}}$. Thus, there are uncountably many computational paths.

Our model checker *iLTLChecker* [34] looks for a computational path in which the negation of an original specification is true. That is, given a specification ψ, *iLTLChecker* looks for an initial pmf $\mathbf{x}$ such that $\neg\psi$ is true in $\sigma_{\mathbf{x}}$.

A ternary satisfaction relation $\models$ over tuples consisting of a Markov matrix, a computational path, and an iLTL formula is recursively defined in Table 3.1. A binary satisfaction relation $\models$, over tuples of a Markov chain and an LTL formula is define as follows:

$$X \models \psi \quad \Leftrightarrow \quad \text{for all initial pmfs } \mathbf{x},\ \ \mathbf{M}, \mathbf{x} \models \psi,$$

where $\mathbf{M}$ is the Markov matrix of X.

The model checking problem is to determine whether a given pair (X, ψ) is in the binary relation $\models$ defined above.

$$
\begin{array}{lll}
\mathbf{M}, \sigma_{\mathbf{x}} \models T & & \\
\mathbf{M}, \sigma_{\mathbf{x}} \not\models F & & \\
\mathbf{M}, \sigma_{\mathbf{x}} \models \Sigma a_i \cdot \mathrm{P}[X = s_i] \bowtie b \;\Leftrightarrow\; \Sigma a_i \cdot x_i \bowtie b & & \\
\mathbf{M}, \sigma_{\mathbf{x}} \models \neg\phi & \Leftrightarrow & \mathbf{M}, \sigma_{\mathbf{x}} \not\models \phi \\
\mathbf{M}, \sigma_{\mathbf{x}} \models \phi \vee \psi & \Leftrightarrow & \mathbf{M}, \sigma_{\mathbf{x}} \models \phi \text{ or } \mathbf{M}, \sigma_{\mathbf{x}} \models \psi \\
\mathbf{M}, \sigma_{\mathbf{x}} \models \phi \wedge \psi & \Leftrightarrow & \mathbf{M}, \sigma_{\mathbf{x}} \models \phi \text{ and } \mathbf{M}, \sigma_{\mathbf{x}} \models \psi \\
\mathbf{M}, \sigma_{\mathbf{x}} \models \mathtt{X}\phi & \Leftrightarrow & \mathbf{M}, \sigma_{(\sigma_{\mathbf{x}}(1))} \models \phi \\
\mathbf{M}, \sigma_{\mathbf{x}} \models \phi \mathtt{U} \psi & \Leftrightarrow & \text{there exists an } i \geq 0 \text{ such that} \\
 & & \mathbf{M}, \sigma_{(\sigma_{\mathbf{x}}(i))} \models \psi \text{ and for all } 0 \leq j < i, \mathbf{M}, \sigma_{(\sigma_{\mathbf{x}}(j))} \models \phi \\
\mathbf{M}, \sigma_{\mathbf{x}} \models \phi \mathtt{R} \psi & \Leftrightarrow & \text{for all } j \geq 0, \text{ if for every } i < j \\
 & & \mathbf{M}, \sigma_{(\sigma_{\mathbf{x}}(i))} \not\models \phi \text{ then } \mathbf{M}, \sigma_{(\sigma_{\mathbf{x}}(j))} \models \psi
\end{array},
$$

where $\bowtie$ is one of $=, \neq, <, \leq, >, \geq$ with their usual meaning.

Table 3.1: Semantics of iLTL: a ternary satisfaction relation $\models$.

3.3 Extended iLTL

In Section 2.4 we have discussed the topics about multiple Markov processes. When these processes have interactions, we can model them with SAN using synchronized events. When they are independent, a composite model can be obtained from individual processes using Kronecker products. As we mentioned earlier, the independent processes are common in performance evaluations, especially for comparisons between different systems and for comparisons between different initial conditions. For example let $A = (S_A, \mathbf{A})$ and $B = (S_B, \mathbf{B})$ be two independent DTMCs, where $S_A = \{a_1, a_2\}$ and $S_B = \{b_1, b_2\}$ are sets of states, and $\mathbf{A}$ and $\mathbf{B}$ are probability transition matrices. As a performance comparison, we may want to check whether *eventually* $\mathrm{P}[A = a_1]$ *becomes larger than* $\mathrm{P}[B = b_1]$ *from any situations*. For this, we first build a composite system $C = (S_A \times S_B, \mathbf{A} \otimes \mathbf{B})$. Figure 3.2 shows the two DTMCs A, B and their combined DTMC C. The performance specification *eventually* $\mathrm{P}[A = a_1] > \mathrm{P}[B = b_1]$ can be written as

$$\diamond\ (\mathrm{P}[C = (a_1, b_1)] + \mathrm{P}[C = (a_1, b_2)] > \mathrm{P}[C = (a_1, b_1)] + \mathrm{P}[C = (a_2, b_1)])\,.$$

One difficulty in this representation is that the number of states grows as the product of the sizes of component systems. Because of the exponential growth in terms of the number of component systems, this method is not scalable. The interacting Markov processes of Figure 2.2 have $3\times3 = 9$ states. If we add another process with 3 states, the number of states become 27. Instead, we have extended iLTL so that it can specify performance criteria on concurrent runs of multiple Markov chains. With the extended logic, we

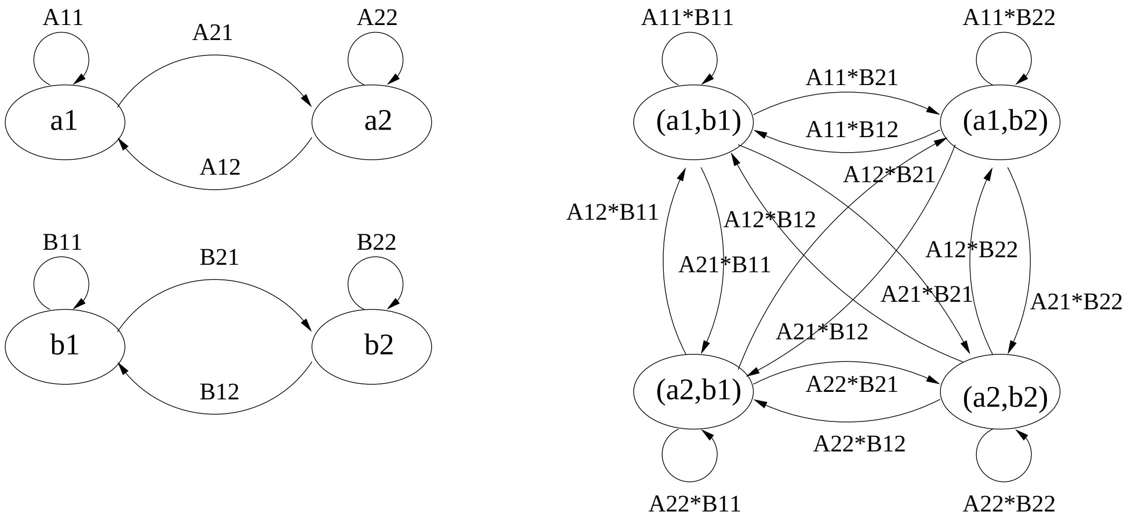


Figure 3.2: Two independent DTMCs $A = (\{a_1, a_2\}, \mathbf{A})$, $B = (\{b_1, b_2\}, \mathbf{B})$, and their combined DTMC $(\{a_1, a_2\} \times \{b_1, b_2\}, \mathbf{A} \otimes \mathbf{B})$.

do not need to build a composite system: the model checker checks every combinations of pmf transitions. Thus, the system size grows linearly as component systems are added. If the two processes of Figure 2.2 are independent, the number of states required is 3+3 = 6. If we add another independent process with 3 states, the necessary number of states becomes 9. In the extended iLTL, a system state is a set of each system's pmfs and the inequality is expanded to range over all system states. That is, for systems $X^{(j)} = (S^{(j)}, \mathbf{M}^{(j)})$, for $j = 1, \ldots, J$, let $\mathbf{x}^{(j)}$ be a pmf vector such that $x_i^{(j)}$ be the probability $\mathrm{P}[X^{(j)} = S_i^{(j)}]$ then the extended state is $\bigcup_{j=1}^{J}\{\mathbf{x}^{(j)}\}$. Extended inequalities are of the form $\sum_{j=1}^{J} \mathbf{a}^{(j)} \cdot \mathbf{x}^{(j)} < b$, where $\mathbf{a}^{(j)}$ is a row vector of size $|S^{(j)}|$. Now, the meaning of an iLTL formula is interpreted on a computational path of the extended states $\sigma_{\cup_{j=1}^{J}\{\mathbf{x}^{(j)}\}}$. With this extension the previous specification can be expressed much more simply as:

$$\diamond\ (\mathrm{P}[A = a_1] > \mathrm{P}[B = b_1])\,.$$

3.3.1 Multiple DTMC model

We represent a multiple DTMC model $\mathcal{X}$ as a set of DTMCs. That is,

$$\mathcal{X} = \bigcup_{j=1}^{J}\{X^{(j)}\},$$

where DTMC $X^{(j)}$ is $(S^{(j)}, \mathbf{M}^{(j)})$, and $S^{(j)} = \{s_1^{(j)}, \dots, s_{n_j}^{(j)}\}$. For simplicity, from this section onwards we use $\bigcup_j A$ for the set $\bigcup_{j=1}^{J}\{A^{(j)}\}$.

iLTL [33] is a probabilistic temporal logic that can be used to specify pmf transitions of a DTMC. In this paper we extend iLTL so that it can specify properties about multiple independent DTMCs together.

3.3.2 Syntax

The syntax of the extended iLTL is

$$\begin{aligned} \psi \quad &::= \quad T \mid F \mid ineq \mid \\ &\quad\quad \neg\psi \mid \psi \vee \phi \mid \psi \wedge \phi \mid \psi \rightarrow \phi \mid \\ &\quad\quad \mathtt{X}\,\psi \mid \psi\,\mathtt{U}\,\phi \mid \psi\,\mathtt{R}\,\phi \mid \Box\,\psi \mid \Diamond\,\psi, \\ ineq \quad &::= \quad a_1 * \mathrm{P}[X_1 = s_1] + \cdots + a_n * \mathrm{P}[X_n = s_n] \bowtie b, \end{aligned}$$

where $a_i, b \in \mathbb{R}$, $X_i \in \mathcal{X}$, $s_i \in S^{(i)}$, and $\bowtie \in \{=, \neq, <, \leq, >, \geq\}$.

3.3.3 Semantics

The semantics of extended iLTL is similar to the iLTL semantics defined in Section 3.2.2. However since extended iLTL is defined over concurrent runs of multiple DTMCs its computational paths should be redefined as functions $\sigma_{\cup_j \mathbf{x}^{(j)}} : \mathbb{N} \rightarrow \bigcup_j \mathbb{R}^{n_j}$ such that

$$\sigma_{\cup_j \mathbf{x}^{(j)}}(t) = \bigcup_j \mathbf{M}^{(j)^t} \cdot \mathbf{x}^{(j)}.$$

Note that $\sigma_{\left(\sigma_{\cup_j \mathbf{x}^{(j)}}(t)\right)}$ is the suffix of $\sigma_{\cup_j \mathbf{x}^{(j)}}$ starting from $\sigma_{\cup_j \mathbf{x}^{(j)}}(t)$. We also extend the reward function so that it can related the computations of different Markov processes. Let the extended Markov Reward Process be a set of Markov Reward Processes:

$$\bigcup_j (S^{(j)}, \mathbf{M}^{(j)}, \rho^{(j)}).$$

Then, an expected reward of the extended MRP at time t is

$$\sum_{j} \sum_{s \in S^{(j)}} \rho^{(j)}(s) \cdot \mathrm{P}[X^{(j)}(t) = s].$$

With the extended MRP we can relate concurrent runs of two Markov Processes. For example, let $A = (\{Ready, Busy\}, \mathbf{M}^{(A)})$ and $B = (\{Ready, Busy\}, \mathbf{M}^{(B)})$ be two DTMCs. Then, a property "the availability of system A is larger than that of system B" can be expressed as

$$\mathrm{P}[A = Ready] > \mathrm{P}[B = Ready].$$

Because the expression can be rewritten as

$$\mathrm{P}[A = Ready] - \mathrm{P}[B = Ready] > 0,$$

the property is whether the expected reward of the extended MRP

$$\left\{\left(\{Ready, Busy\}, \mathbf{M}^{(A)}, \{(Ready, 1), (Busy, 0)\}\right), \left(\{Ready, Busy\}, \mathbf{M}^{(B)}, \{(Ready, -1), (Busy, 0)\}\right)\right\}$$

is larger than zero.

A ternary satisfaction relation $\models$ over pairs consisting of a set of Markov matrices, a computational path, and an extended iLTL formula is recursively defined in Table 3.2.

A binary satisfaction relation $\models$ over a multiple DTMC model and an iLTL formula is define as follows.

$$\mathcal{X} \models \psi \quad \Leftrightarrow \qquad \text{for all } \bigcup_j \mathbf{x}^{(j)}, \quad \bigcup_j \mathbf{M}^{(j)}, \sigma_{\bigcup_j \mathbf{x}^{(j)}} \models \psi. \tag{3.1}$$

The model checking problem is to check whether a given pair $(\mathcal{X}, \psi)$ is in the binary satisfaction relation $\models$ defined above.

$$\begin{array}{lcl}
\bigcup_j \mathbf{M}^{(j)}, \sigma_{\cup_j \mathbf{x}^{(j)}} \models T & & \\
\bigcup_j \mathbf{M}^{(j)}, \sigma_{\cup_j \mathbf{x}^{(j)}} \not\models F & & \\
\bigcup_j \mathbf{M}^{(j)}, \sigma_{\cup_j \mathbf{x}^{(j)}} \models a_1 * \mathrm{P}[X_1 = s_1] + \cdots + a_n * \mathrm{P}[X_n = s_n] \bowtie b & & \\
 & \Leftrightarrow & \sum_{i=1}^{n} a_i \cdot x_k^{(j)} \bowtie b, \text{ where } X_i = X^{(j)} \text{ and } s_i = s_k \in S^{(j)} \\
\bigcup_j \mathbf{M}^{(j)}, \sigma_{\cup_j \mathbf{x}^{(j)}} \models \neg\psi & \Leftrightarrow & \bigcup_j \mathbf{M}^{(j)}, \sigma_{\cup_j \mathbf{x}^{(j)}} \not\models \psi \\
\bigcup_j \mathbf{M}^{(j)}, \sigma_{\cup_j \mathbf{x}^{(j)}} \models \psi \vee \phi & \Leftrightarrow & \bigcup_j \mathbf{M}^{(j)}, \sigma_{\cup_j \mathbf{x}^{(j)}} \models \psi \text{ or } \bigcup_j \mathbf{M}^{(j)}, \sigma_{\cup_j \mathbf{x}^{(j)}} \models \phi \\
\bigcup_j \mathbf{M}^{(j)}, \sigma_{\cup_j \mathbf{x}^{(j)}} \models \psi \wedge \phi & \Leftrightarrow & \bigcup_j \mathbf{M}^{(j)}, \sigma_{\cup_j \mathbf{x}^{(j)}} \models \psi \text{ and } \bigcup_j \mathbf{M}^{(j)}, \sigma_{\cup_j \mathbf{x}^{(j)}} \models \phi \\
\bigcup_j \mathbf{M}^{(j)}, \sigma_{\cup_j \mathbf{x}^{(j)}} \models \mathtt{X}\,\psi & \Leftrightarrow & \bigcup_j \mathbf{M}^{(j)}, \sigma_{\left(\sigma_{\cup_j \mathbf{x}^{(j)}}(1)\right)} \models \psi \\
\bigcup_j \mathbf{M}^{(j)}, \sigma_{\cup_j \mathbf{x}^{(j)}} \models \psi \mathtt{U}\, \phi & \Leftrightarrow & \text{there exists an } a \geq 0 \text{ such that } \bigcup_j \mathbf{M}^{(j)}, \sigma_{\left(\sigma_{\cup_j \mathbf{x}^{(j)}}(a)\right)} \models \phi \text{ and} \\
 & & \bigcup_j \mathbf{M}^{(j)}, \sigma_{\left(\sigma_{\cup_j \mathbf{x}^{(j)}}(b)\right)} \models \psi \text{ for } b = 0, \ldots, a \\
\bigcup_j \mathbf{M}^{(j)}, \sigma_{\cup_j \mathbf{x}^{(j)}} \models \psi \mathtt{R}\, \phi & \Leftrightarrow & \text{for all } b \geq 0, \text{ if for every } a < b \bigcup_j \mathbf{M}^{(j)}, \sigma_{\left(\sigma_{\cup_j \mathbf{x}^{(j)}}(a)\right)} \not\models \psi \\
 & & \text{then } \bigcup_j \mathbf{M}^{(j)}, \sigma_{\left(\sigma_{\cup_j \mathbf{x}^{(j)}}(b)\right)} \models \phi
\end{array}$$

where $\bowtie$ is one of $=, \neq, <, \leq, >, \geq$ with their usual meaning.

Table 3.2: Semantics of extended iLTL: a ternary satisfaction relation $\models$.

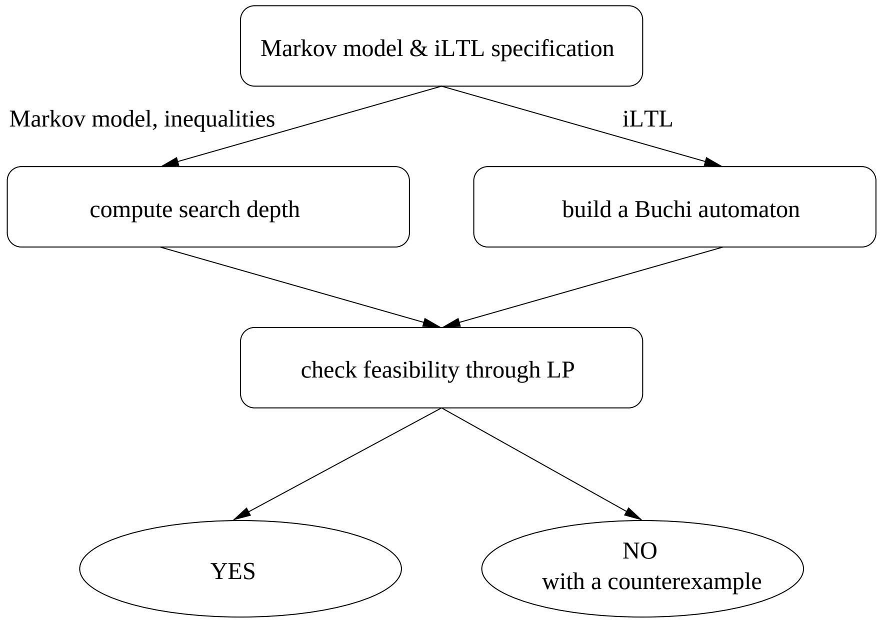

Markov model & iLTL specification
Markov model, inequalities
iLTL
compute search depth
build a Buchi automaton
check feasibility through LP
YES
NO
with a counterexample

Figure 3.3: A block diagram of iLTL model checking algorithm.

3.4 Model Checking Algorithm

Figure 3.3 shows a block diagram of the iLTL model checking algorithm:

Given a multiple DTMC model $\mathcal{X} = \bigcup_j X^{(j)}$, where $X^{(j)} = (S^{(j)}, \mathbf{M}^{(j)})$, we build a big matrix $\mathbf{M}$ and a big state vector $\mathbf{x}$ such that

$$\begin{aligned} \mathbf{M} &= \begin{bmatrix} \mathbf{M}^{(1)} & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & \mathbf{M}^{(J)} \end{bmatrix}, \\ x_{(\sum_{i=1}^{j-1} n_i)+k} &= \mathrm{P}[X^{(j)} = s_k^{(j)}], \end{aligned}$$

where $n_i = |S^{(i)}|$. Because the column sums of $\mathbf{M}$ are one and all elements of $\mathbf{M}$ are non-negative, $\mathbf{M}$ is a Markov transition matrix. However $\mathbf{x}$ is not a pmf vector; it is a concatenation of J pmf vectors. Given an iLTL formula and a multiple DTMC model, an outline of an iLTL model checking algorithm is as follows.

1. Compute the search depth N.
2. Build a Büchi automaton for the iLTL formula.
3. For the graph of the Büchi automaton, search for a counter example by checking the feasibility of linear constraints by linear programming.

We describe the first and the third steps in this section. The second step is a well-known expand algorithm [18] described in Section 2.5.

Before we proceed to describe the whole verification algorithm, we first show a simple illustrative example of a the extended iLTL model checking process.

Example 3 *Let us consider the following multiple DTMC and an iLTL specification:*

- $\mathcal{X} = \{A, B\}$, *where* $A = (\{a_1, a_2\}, \mathbf{M}^{(A)})$, *and* $B = (\{b_1, b_2\}, \mathbf{M}^{(B)})$.
- *A set of atomic propositions* $AP = \{p : 2 \cdot \mathrm{P}[A = a_1] > 3 \cdot \mathrm{P}[B = b_1] + 2\}$,
- *An iLTL formula* $\psi : p \vee \mathtt{X}p \vee \mathtt{XX}p$

We construct the big matrix and the big state vector as:

$$\begin{aligned} \mathbf{M} &= \begin{bmatrix} \mathbf{M}^{(A)} & \mathbf{0} \\ \mathbf{0} & \mathbf{M}^{(B)} \end{bmatrix}, \\ \mathbf{x} &= [\mathrm{P}[A = a_1], \mathrm{P}[A = a_2], \mathrm{P}[B = b_1], \mathrm{P}[B = b_2]]^T . \end{aligned}$$

Because $\mathcal{X} \models \psi$ if and only if $\mathcal{L}_\mathcal{X} \subseteq \mathcal{L}_\psi$, we will check whether $\mathcal{L}_\mathcal{X} \cap \mathcal{L}_{\neg\psi} = \emptyset$. Let q be the negation of the inequality p. That is q: $2 \cdot \mathrm{P}[A = a_1] \leq 3 \cdot \mathrm{P}[B = b_1] + 2$. Then the negation of the specification $\neg\psi$ becomes

$$\neg\psi : q \wedge \mathtt{X}q \wedge \mathtt{XX}q.$$

If there is a pmf $\mathbf{x}$ that satisfies the following constraints, then $\mathcal{X} \not\models \psi$ and $\mathbf{x}$ is the counter example we seek. Otherwise $\mathcal{X} \models \psi$.

$$\begin{aligned} [2, 0, 3, 0] \cdot \sigma_\mathbf{x}(0) &\leq 2 \wedge \\ [2, 0, 3, 0] \cdot \sigma_\mathbf{x}(1) &\leq 2 \wedge \\ [2, 0, 3, 0] \cdot \sigma_\mathbf{x}(2) &\leq 2 \wedge \\ [1, 1, 0, 0] \cdot \sigma_\mathbf{x}(0) &= 1 \wedge \\ [0, 0, 1, 1] \cdot \sigma_\mathbf{x}(0) &= 1 \end{aligned}$$

The constraints can be rewritten as

$$\begin{aligned} [2, 0, 3, 0] \cdot \mathbf{M}^0 \cdot \mathbf{x} &\leq 2 \wedge \\ [2, 0, 3, 0] \cdot \mathbf{M}^1 \cdot \mathbf{x} &\leq 2 \wedge \\ [2, 0, 3, 0] \cdot \mathbf{M}^2 \cdot \mathbf{x} &\leq 2 \wedge \\ [1, 1, 0, 0] \cdot \mathbf{x} &= 1 \wedge \\ [0, 0, 1, 1] \cdot \mathbf{x} &= 1 \end{aligned}$$

Now, the formula is expressed in terms of a state vector $\mathbf{x}$. *The feasibility of this set of linear inequality constraints can be checked by linear programming (LP).*

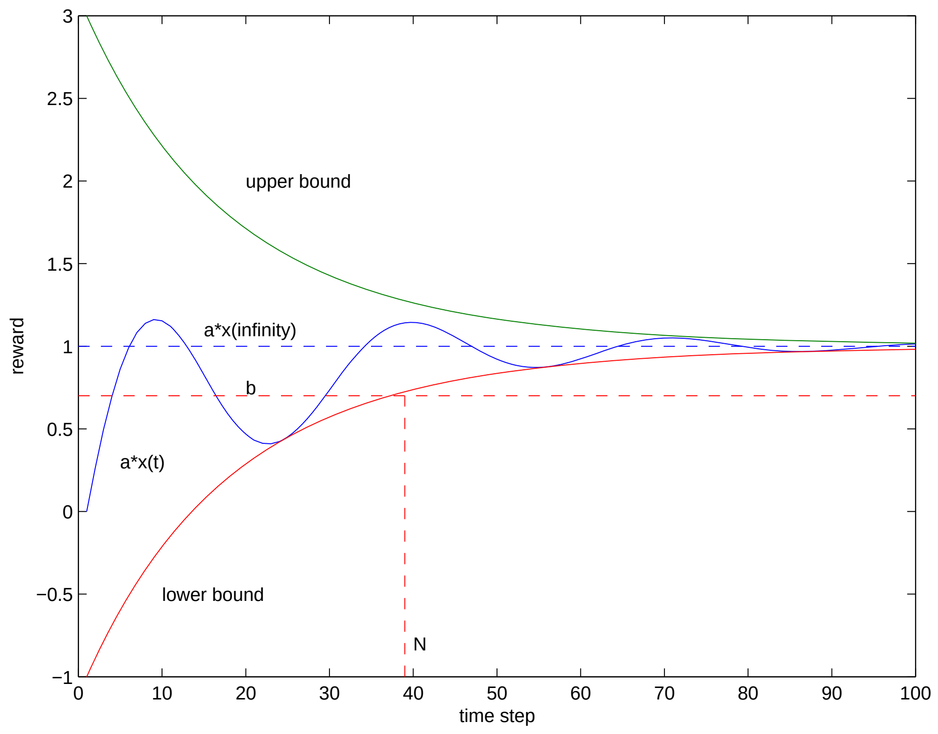


Figure 3.4: Monotonic bounds for expected rewards and a search depth.

■

3.4.1 Computation of Search Depth

Theorem 1 *Let $\lambda_1^{(j)} = 1$ be an eigenvalue of $\mathbf{M}^{(j)}$ for $j = 1, \ldots, J$. If the following three conditions are satisfied then there is an integer N after which the truth values of all inequalities of ψ become constant.*

1. *$|\lambda_i^{(j)}| < 1$ for $j = 1, \ldots, J$ and $i = 2, \ldots, |S^{(j)}|$,*
2. *$\mathbf{M}^{(j)}$ is diagonalizable for $j = 1, \ldots, J$, and*
3. *the* right hand side *(RHS) of each inequality of ψ is not equal to its* left hand side *(LHS) in the steady state.*

Proof We will show that for each inequality α of ψ, there is a bound N_α. Then the maximum of all N_α's is the bound N we are seeking. Let α be $\mathbf{a} \cdot \mathbf{x} < b$, where $\mathbf{a} = [\mathbf{a}^{(1)}, \ldots, \mathbf{a}^{(J)}]$ and $\mathbf{x} = [\mathbf{x}^{(1)^T}, \ldots, \mathbf{x}^{(J)^T}]^T$.

First, we will show the existence of N_α and then we will actually compute it. Because

$$\mathbf{M}^t = \begin{bmatrix} \mathbf{M}^{(1)t} & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & \mathbf{M}^{(J)t} \end{bmatrix},$$

if each $\mathbf{M}^{(j)}$ has a unique steady state value, $\mathbf{M}$ has a unique steady state value. Although, the first condition of Theorem 1 is enough to guarantee the existence of the unique steady state value [29], together with the second condition we can easily show it. Let $\mathbf{M}^{(j)} = \mathbf{Z}^{(j)} \cdot \mathbf{\Lambda}^{(j)} \cdot \mathbf{Z}^{(j)-1}$, where $\mathbf{\Lambda}^{(j)}$ =diag($[\lambda_1^{(j)}, \ldots, \lambda_{n_j}^{(j)}]$) is a diagonal matrix of eigenvalues, and $\mathbf{Z}^{(j)} = [\mathbf{z}_1^{(j)}, \ldots, \mathbf{z}_{n_j}^{(j)}]$) is a matrix of eigenvectors. Because $\mathbf{M}^{(j)t} = \mathbf{Z}^{(j)} \cdot \mathbf{\Lambda}^{(j)t} \cdot \mathbf{Z}^{(j)-1}$, and $\mathbf{\Lambda}^{(j)\infty} = \text{diag}([1, 0, \ldots, 0])$, for any initial pmf $\mathbf{x}^{(j)}$, $\sigma_{\mathbf{x}^{(j)}}(\infty) = \mathbf{M}^{(j)\infty} \cdot \mathbf{x}^{(j)} = \mathbf{z}_1^{(j)}/c$, where the normalizing constant $c = [1, \ldots, 1] \cdot \mathbf{z}_1^{(j)}$. Let b^∞ be $\mathbf{a} \cdot \sigma_{\mathbf{x}}(\infty)$, then

$$\mathbf{a} \cdot \mathbf{M}^t \cdot \mathbf{x} < b \quad \Leftrightarrow \quad \mathbf{a} \cdot (\mathbf{M}^t \cdot \mathbf{x} - \sigma_{\mathbf{x}}(\infty)) < b - b^\infty.$$

Since the LHS of the second inequality tends to 0 as $t \to \infty$ and $b \neq b^\infty$, for a given $\mathbf{x}$ there is a time n' such that

$$\mathbf{a} \cdot \sigma_{\mathbf{x}}(t) < b \quad \Leftrightarrow \quad \mathbf{a} \cdot \sigma_{\mathbf{x}}(n') < b \quad \text{for} \quad t \geq n'$$

Now we will compute a upper bound N_α after which the inequality α becomes a constant. As one can see from Figure 3.4, the bound N_α, which may not be the least upper bound, is an integer t such that

$$\left|\mathbf{a} \cdot \left(\mathbf{M}^t \cdot \mathbf{x} - \sigma_{\mathbf{x}}(\infty)\right)\right| < \left|b - b^\infty\right|.$$

The fact that $\mathbf{x}$ is a concatenation of pmfs $\mathbf{x}^{(j)}$ for $j = 1, \ldots, J$ leads to a monotonically decreasing function which is larger than the LHS of the above inequality for all $\mathbf{x}$.

$$\begin{aligned} &\left|\mathbf{a} \cdot \left(\mathbf{M}^t \cdot \mathbf{x} - \sigma_{\mathbf{x}}(\infty)\right)\right| \\ &\qquad = \left|\mathbf{a} \cdot \left(\mathbf{Z} \cdot \mathbf{\Lambda}^t \cdot \mathbf{Z}^{-1} \cdot \mathbf{x} - \sigma_{\mathbf{x}}(\infty)\right)\right| \end{aligned}$$

$$\begin{aligned}
&= \left| \sum_{j=1}^{J} \mathbf{a}^{(j)} \cdot \left(\mathbf{Z}^{(j)} \cdot \mathbf{\Lambda}^{(j)^t} \cdot \mathbf{Z}^{(j)^{-1}} \cdot \mathbf{x}^{(j)} - \sigma_{\mathbf{x}^{(j)}}(\infty) \right) \right| \\
&= \left| \sum_{j=1}^{J} \left(\sum_{i=1}^{n_j} c_i^{(j)} \cdot \lambda_i^{(j)^t} \cdot \left(\sum_{k=1}^{n_j} \mathbf{Z}_{ik}^{(j)^{-1}} \cdot x_k^{(j)} \right) - a_i^{(j)} \cdot (\sigma_{x^{(j)}}(\infty))_i \right) \right| \\
&= \left| \sum_{j=1}^{J} \left(\sum_{i=2}^{n_j} c_i^{(j)} \cdot \lambda_i^{(j)^t} \cdot \left(\sum_{k=1}^{n_j} \mathbf{Z}_{ik}^{(j)^{-1}} \cdot x_k^{(j)} \right) \right) \right| \\
&\le \sum_{j=1}^{J} \left(\sum_{i=2}^{n_j} |c_i^{(j)}| \cdot |\lambda_i^{(j)}|^t \cdot \left(\sum_{k=1}^{n_j} |\mathbf{Z}_{ik}^{(j)^{-1}}| \cdot x_k^{(j)} \right) \right) \\
&\le \sum_{j=1}^{J} \left(\sum_{i=2}^{n_j} |c_i^{(j)}| \cdot |\lambda_i^{(j)}|^t \cdot \max_{k \in [1, n_j]} |\mathbf{Z}_{ik}^{(j)^{-1}}| \right),
\end{aligned}$$

where $c_i^{(j)} = \mathbf{a}^{(j)} \cdot \mathbf{z}_i^{(j)}$. The last inequality is derived from the facts that $\sum_{k=1}^{n_j} x_k^{(j)}(0) = 1$ and $x_k^{(j)}(0) \ge 0$ for $k = 1, \ldots, n_j$.

Let $u(t)$ be the right-hand side of the last inequality,

$$u(t) = \sum_{j=1}^{J} \sum_{i=2}^{n_j} c'_{ij} \cdot |\lambda_i^{(j)}|^t, \quad \text{where } c'_{ij} = |c_i^{(j)}| \cdot \max_{k \in [1, n_j]} |\mathbf{Z}_{ik}^{(j)^{-1}}|.$$

Because $|\lambda_i^{(j)}| < 1$ for $j = 1, \ldots, J$ and $i = 2, \ldots, n_j$, $u(t)$ is a monotonically decreasing function of t which is larger than $|\mathbf{a} \cdot (\mathbf{M}^t \cdot \mathbf{x} - \sigma_{\mathbf{x}}(\infty))|$ and $\lim_{t \to 0} u(t) = 0$. The upper bound and the lower bound of the expected reward $\mathbf{a} \cdot \sigma_{\mathbf{x}}(t)$ are $\sigma_{\mathbf{x}}(\infty) + u(t)$ and $\sigma_{\mathbf{x}}(\infty) - u(t)$ respectively.

Thus, $N_\alpha = \lceil t \rceil$ such that $u(t) = |b - b^\infty|$. The bound N after which all inequalities of an iLTL formula ψ become constant is the maximum of the time bounds N_α of all inequalities of ψ.

■

3.4.2 Model Checking as Feasibility Checking

Let $X = (\{s_1, \ldots, s_n\}, \mathbf{M})$ be the big DTMC made from a multiple DTMC model $\mathcal{X} = \bigcup_j (S^{(j)}, \mathbf{M}^{(j)})$. In order to check X against an iLTL specification ψ, we first build a Büchi automaton $\mathcal{A}_{\neg\psi}$ by the expand algorithm [23] as explained in section 2.5. One small extra thing to do that is not in the expand algorithm is the negation of atomic propositions. Note that after converting an LTL formula to a negation normal form, negation operators are only at the atomic propositions. For iLTL formulas this negation operators are explicitly removed by changing the compare relation. We can further replace $>$ and $\ge$ relations with $<$ and

$\le$ respectively by multiplying by -1 on both sides of the inequalities. The following equations summarize the conversion.

$$\begin{aligned}
\neg(\mathbf{r}\cdot\mathbf{x} = b) &\Leftrightarrow \mathbf{r}\cdot\mathbf{x} \neq b \\
\neg(\mathbf{r}\cdot\mathbf{x} \neq b) &\Leftrightarrow \mathbf{r}\cdot\mathbf{x} = b \\
\neg(\mathbf{r}\cdot\mathbf{x} > b) &\Leftrightarrow \mathbf{r}\cdot\mathbf{x} \le b \\
\neg(\mathbf{r}\cdot\mathbf{x} \ge b) &\Leftrightarrow \mathbf{r}\cdot\mathbf{x} < b \\
\neg(\mathbf{r}\cdot\mathbf{x} < b) &\Leftrightarrow -\mathbf{r}\cdot\mathbf{x} \le -b \\
\neg(\mathbf{r}\cdot\mathbf{x} \le b) &\Leftrightarrow -\mathbf{r}\cdot\mathbf{x} < -b.
\end{aligned}$$

The compare relation $\neq$ will be replaced by a disjunction of two inequalites as follows

$$\mathbf{r}\cdot\mathbf{x} \neq b \quad \Leftrightarrow \quad (\mathbf{r}\cdot\mathbf{x} < b) \vee (-\mathbf{r}\cdot\mathbf{x} < -b)\,.$$

Note that there are strict comparision relation $<$ in the converted inequalities which cannot be used as a constraint for linear programming directly. This constraint can be checked by doing a secondary linear programming with a slack variable [43] for the inequality and associate a negative cost to the new variable. The strict comparision relation is satisfiable if and only if the minimized cost is less than 0. For example, if $\mathbf{r}\cdot\mathbf{x} < b$ is an inequality to check, then

- Check the feasibility of $\mathbf{r}\cdot\mathbf{x} \le b$ by a linear programming.
- If the inequality above is feasible then minimize the following cost function

$$\begin{aligned}
c &= \min_{\mathbf{x},a} -a \\
&\qquad \text{such that } \mathbf{r}\cdot\mathbf{x} + a = b, \\
&\qquad\qquad a \ge 0
\end{aligned}$$

  possibly with other constraints. If the cost c is less than 0 than the strict comparision is satisfiable.

For convenience, we first define the following terms for an iLTL formula ψ and a DTMC $X = (\{s_1,\ldots,s_n\},\mathbf{M})$. Let AP be the set of atomic sub-formulas of ψ and let $\mathcal{A}_\psi = (\Sigma, Q, \Delta, Q_0, F)$, where

$\Sigma = 2^{AP}$ be the Büchi automaton of ψ.

- $L : (\mathbb{N} \to \mathbb{R}^n) \to 2^{AP}$ is a *labeling function* such that $L(\sigma_{\mathbf{x}}(t)) = \{a \in AP : a(\sigma_{\mathbf{x}}(t))\}$. We also write $L(\sigma_{\mathbf{x}})$ for the infinite string of Σ such that $L(\sigma_{\mathbf{x}})(t) = L(\sigma_{\mathbf{x}}(t))$ for $t \in \mathbb{N}$.

- $G_\psi = (Q, \Delta_G)$ is a *graph associated with* $\mathcal{A}_\psi$, where Q is a set of nodes and $\Delta_G = \{(q_i, q_j) : (q_i, \alpha, q_j) \in \Delta$ for any $alpha \in \Sigma\}$ is a set of edges.

- $G_\psi^\infty = (Q, \Delta_G^\infty)$ is a *final graph associated with* $\mathcal{A}_\psi$, where Q is a set of nodes and $\Delta_G^\infty = \{(q_i, q_j) : (q_i, L(\sigma_{\mathbf{x}}(\infty)), q_j) \in \Delta\}$ is a set of edges. Note that $L(\sigma_{\mathbf{x}}(\infty))$ is constant regardless of the choices of $\mathbf{x}$.

- Let $F^{SC} \subseteq Q$ be the set of nodes of strongly connected components of G_ψ^∞, and let $F^+ \subseteq Q$ be a set of nodes from which a node in F^{SC} is reachable along the edges of G_ψ^∞.

- An *inequality transform function* $\tau_{\mathbf{M}} : ineq \to ineq$ such that $\tau_{\mathbf{M}}(a_1 \cdot x_1 + \cdots + a_n \cdot x_n < b) = a_1' \cdot x_1 + \cdots + a_n' \cdot x_n < b$ where $[a_1', \ldots, a_n'] = [a_1, \ldots, a_n] \cdot \mathbf{M}$. With a little abuse of notion for any $I \subseteq AP$, we define $\tau_{\mathbf{M}}(I) = \{\tau_{\mathbf{M}}(a) : a \in I\}$.

- Let $\pi : \mathbb{N} \to Q$ be an infinite path of G_ψ then a *constraint set* C_π^N is

$$\begin{aligned} C_\pi^N &= D \cup \bigcup_{t=0}^{N} \tau_{\mathbf{M}^t}(I_t) \\ D &= \bigcup_{j=1}^{J} \left\{ x_{(\sum_{i=1}^{j-1} n_i)+1} + \cdots + x_{(\sum_{i=1}^{j-1} n_i)+n_j} = 1 \right\} \cup \bigcup_{i=1}^{\left(\sum_{j=1}^{J} n_j\right)} \{x_i \geq 0\}, \end{aligned}$$

where $I_t = \bigcap_{(\pi(t),\alpha,\pi(t+1))\in\Delta} \alpha$[1] and D is a necessary and sufficient condition for $\mathbf{x}$ to be a state of multiple DTMC $\mathcal{X}$.

Theorem 2 *Let $\mathcal{A}_\psi = (\Sigma, Q, \Delta, Q_0, F)$ be a Büchi automaton of an iLTL ψ, $X = (\{s_1, \ldots, s_n\}, \mathbf{M})$ be a DTMC, and N be the search depth for X and ψ by the Theorem 1. Then, there is an initial state $\mathbf{x}$ such that the string $L(\sigma_{\mathbf{x}})$ is accepted by $\mathcal{A}_\psi$ if and only if there is a feasible point $\mathbf{y}$ of the constraint set C_π^N where $\pi : \mathbb{N} \to Q$ is an infinite path of G_ψ such that $\pi(0) \in Q_0$, $\pi(N) \in F^+$, and $(\pi(t), L(\mathbf{M}^t \cdot \mathbf{y}), \pi(t+1)) \in \Delta$ for $t \geq 0$.*

[1]The transition label of the expand algorithm is exactly I_t. However its implicit meaning is a set of transition labels that satisfy I_t.

Proof $\Rightarrow$: Let $L(\sigma_{\mathbf{x}})$ be a string accepted by $\mathcal{A}_{\psi}$, then there is an accepting path π such that $(\pi(t), L(\sigma_{\mathbf{x}}(t)), \pi(t+1)) \in \Delta$. For the constraints D of C_{π}^{N}, because $\mathbf{x}$ is an initial state $\mathbf{x}$ satisfies D. Because $C_{\pi}^{N} \subseteq \bigcup_{t=0}^{N} \tau_{\mathbf{M}^t}(L(\sigma_{\mathbf{x}}(t)))$ and $\mathbf{x}$ is a feasible point of $\tau_{\mathbf{M}^t}(L(\sigma_{\mathbf{x}}(t)))$ for $t = 0, \ldots, N$, $\mathbf{x}$ is a feasible point of C_{π}^{N}.

Let $\inf(\pi)$ be the set of states that occur infinitely often in π, then because $\inf(\pi) \cap F \neq \emptyset$, there is a cycle including at least one state in $\inf(\pi) \cap F$. Thus, $\inf(\pi) \cap F^{SC} \neq \emptyset$. Because $L(\sigma_{\mathbf{x}}(n)) = L(\sigma_{\mathbf{x}}(\infty))$ for $n \geq N$, there is a state in F^{SC} reachable from $\pi(N)$ along a path of G_{ψ}^{∞}. Thus, $\pi(N) \in F^{+}$.

$\Leftarrow$: Let $\mathbf{x}$ be a feasible point of C_{π}^{N}. Then $\mathbf{x}$ is a feasible point of $\tau_{\mathbf{M}^t}(I_t)$ for $t = 0, \ldots, N$, where $I_t = \bigcap_{(\pi(t),\alpha,\pi(t+1))\in\Delta} \alpha$. Because $\mathbf{x}$ is a feasible point of $\tau_{\mathbf{M}^t}(I_t)$, there is an $\alpha_t \subseteq \Sigma$ such that $(\pi(t), \alpha_t, \pi(t+1)) \in \Delta$ and $L(\mathbf{M}^t \cdot \mathbf{x}) = \alpha_t$. And because $L(\sigma_{\mathbf{x}}(t)) = L(\sigma_{\mathbf{x}}(\infty))$ for $t \geq N$ and $\pi(N) \in F^{+}$, $L(\sigma_{\mathbf{x}})$ is accepted by $\mathcal{A}_{\psi}$. Also, because $\mathbf{x}$ satisfies constraints D, $\mathbf{x}$ is a valid state of X.

■

By Theorem 2, the problem of checking $X \models \psi$ is reduced to checking the feasibility of constraints C_{π}^{N} for paths π of the automaton $\mathcal{A}_{\neg\psi}$. Note that each C_{π}^{N} corresponds to a conjunctive subterm of the DNF form of ψ. So, the number of constraint sets to be checked is still exponential in N. However, C_{π}^{N} can be built while doing depth first traversal (multiple visit of states is allowed) of the graph $G_{\neg\psi}$: $C_{\pi}^{n} = \bigcup_{t=0}^{n} \tau_{\mathbf{M}^t}(\alpha_t)$ where $(\pi(t), \alpha_t, \pi(t+1)) \in \Delta$. At every step of depth first traversal we check the feasibility of C_{π}^{n}, and if an infeasibility is found early in the traversing process, we can skip checking the subtree. This is equivalent to checking the common prefixes of conjunctive subterms of DNF form and discarding the subterms if the prefixes are not satisfiable. The worst time complexity of the algorithm is $O\Big((\text{max out degree of } \mathrm{G}_{\neg\psi})^{N}\Big)$. But in many cases the checking terminates in reasonable amount of time even for large N. The feasibility can be checked by solving linear programming problem with artificial variables [43]. One small but very effective optimization is to check the feasibility of a new constraint set against the feasible point of the previous step before doing an LP. Since we add just one inequality constraint at each step, in many cases, we can skip the LP phase.

Example 4 *Figure 3.5 shows a Büchi automaton for the negation of* $\Box \mathrm{a}$. *Let us assume that $\neg a$ is true for the final pmf $\sigma_{\mathbf{x}}(\infty)$. Then all three states are in F^{+} and after N transitions the model checker will announce* $\Box \mathrm{a}$ *as false. If $\neg a$ is false by the final pmf then the only state in F^{+} is the final state with an empty set of atomic propositions. So, if the F^{+} is reachable by N transitions then the answer is false, otherwise it is true.*

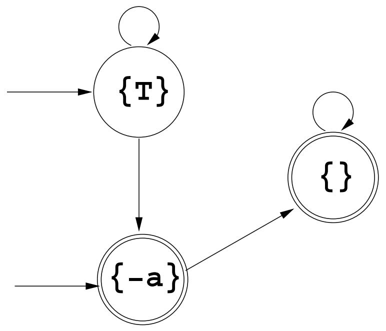


Figure 3.5: A Büchi automaton for an LTL formula ¬□a

Let us assume that N is 3 and ¬a is false by the final pmf $\sigma_{\mathbf{x}}(\infty)$. *Then* F^+ *is the set of states* $\{\{\neg a\}, \{\}\}$. *All paths of length 3 that end with a state in* F^+ *are:*

$$(\neg a, \phi, \phi),\ (\mathrm{T}, \neg a, \phi),\ (\mathrm{T}, \mathrm{T}, \neg a).$$

So, the set of inequalities to check is (let ¬a is $\mathbf{c} \cdot \mathbf{x} < b$*):*

$$\{\mathbf{c} \cdot \mathbf{x} < b\},\ \{\mathbf{c} \cdot \mathbf{M} \cdot \mathbf{x} < b\},\ \{\mathbf{c} \cdot \mathbf{M}^2 \cdot \mathbf{x} < b\}.$$

If there is a feasible solution vector $\mathbf{x}$ *that satisfies any of the three sets of inequalities and the constraints D, then* $\mathbf{x}$ *is a counterexample that satisfies the negated specification* ¬□a. *That is, the initial pmf* $\mathbf{x}$ *violates the specification* □a.

■

Chapter 4

Reliability Assessment

The goal of software reliability assessment is to quantify the robustness of software and to provide guidance in testing the software. However, often the software can be so large that checking it as a whole is be impractical. In order to cope with this difficulty, a modular testing method has been proposed where the software is divided into several modules and the reliability of the software is assessed based on each module's reliability and a program transition statistics between modules obtained from user profiles. This process can be modeled as a DTMC whose states are the modules with two additional states representing the failure and the success states, and whose transition probabilities between states are estimated from the user profiles. The reliability of a software is the probability that the success state will eventually be arrived which can be computed by a simple steady state analysis on the DTMC model is achieved [52, 54, 73].

There are many useful results about steady state analysis of a Markov process that can be applied directly. However, the standard analytical techniques do not address transient behaviors of a system. Thus, without automated or semi-automated verification methods, a general purpose transient analysis of a system is difficult. Moreover, as before we want to understand if there are some inital conditions under which a property may be violated. In this chapter, we analyze a *Time Difference Of Arrival* (TDOA) distance measurement method commonly used in WSN [57] and assess the reliability of the method using iLTL model checking. We build a probabilistic model of the method from our experience and we demonstrate how iLTL model checking can be used to assess a protocol reliability and how it helps in designing the protocol.

4.1 Probabilistic Model Of Time Difference Of Arrival Method

In Wireless Sensor Network each node's position information is crucial for many applications such as environmental monitoring, tracking, or location guided routing mechanism [66, 2, 40]. Most non-GPS based localization algorithms compute each node's position based on measured distances between nodes [36, 62, 63, 48]. TDOA is an acoustic distance measurement method commonly used in WSNs. In general, indoor localization methods suffer from echos, whereas outdoor methods suffer from environmental noise and signal attenuation due to large inter-node spacing. In this section, we decompose the outdoor TDOA distance measurement method into multiple phases which have possible sources of errors. With the multi-phase probabilistic model, we evaluate the performance of the method through iLTL model checking.

TDOA method uses the propagation speed difference between sound and radio: the time difference between the arrivals of the two signals will be proportional to the distances between the sender of the signals and the receiver of them. However, the signal attenuation in an outdoor environment and environmental noise affects the detection probability as well as the probability of wrong measurement. If the radio or the sound signal is undetected, then the whole process can be repeated after timeout. However, a wrong measurement due to noises cannot be undone. Thus, we put many integrity checking mechanisms for the measured distances before we accepting the measurement.

Scheduling is also an important issue for medium to large scale localization to prevent interferences between different measurements. For medium scale localization, centralized scheduling schemes often provide enough reliability and efficiency. In the centralized scheme, a central server sends a start command to a node to send the radio and the sound signals. If a node receives the two signals it reports the measured distance back to the central server. Because many WSN communication protocols do not guarantee reliable message delivery to save energy, there is a probability of message loss. The central server repeats this process until all or a reasonable number of measurements are made.

Figure 4.1 shows a flow diagram of the TDOA distance measurement method. First, a central server sends a *start* command to a sender node. This command usually follows along a pre-formed spanning tree based routing path to the sender node. The probability that the sender node actually receives the command is *pStart*. If the command fails to be delivered to the sender, the server will timeout and will try the measurement later again with probability *1 - pStart*. On receiving the command, the sender node sends the radio and the sound signals —multiple sounds at pre-defined intervals are actually sent in order to increase the

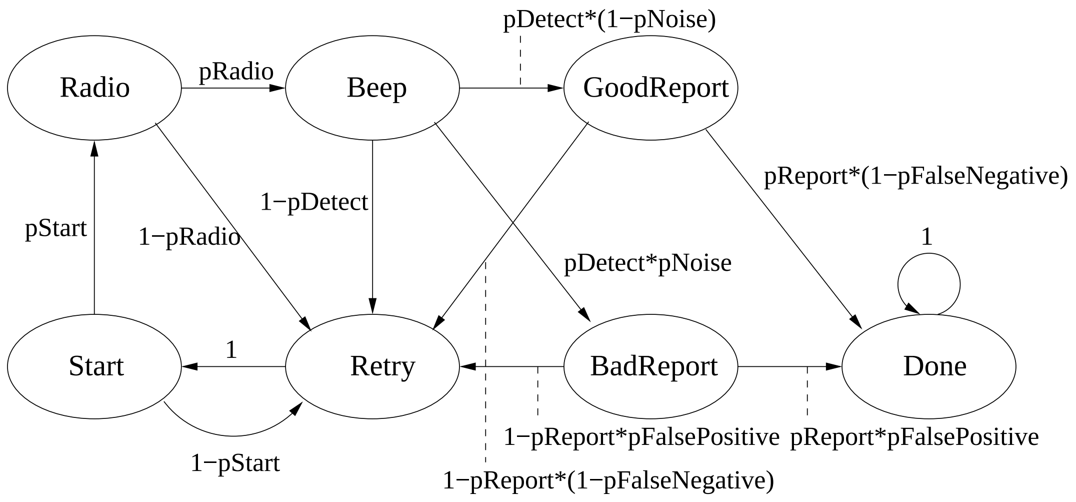


Figure 4.1: A flow diagram of Time Difference Of Arrival (TDOA) distance measurement.

Transition probability		Energy Consumption	
Parameter	Probability	State	Energy
pStart	0.9	Start	5
pRadio	0.95	Radio	1
pDetect	0.9	Beep	3
pNoise	0.3	GoodReport	5
pReport	0.9	BadReport	5
pFlaseNegative	0.2	Done	0
pFalsePositice	0.2	Retry	0

Table 4.1: Assumed transition probabilities and energy consumptions ($e : S \rightarrow \mathbb{R}$) at each state of the TDOA flow diagram of Figure 4.1.

detection probability. If any of the two signal is missed, the server will timeout and will repeat the process. Another problem is that a detection in the receiver does not mean that the real signal is detected: it can be due to environmental noise. Thus, we model the probability of *good measurement* as $pDetect \cdot (1 - pNoise)$ and the probability of *bad measurement* as $pDetect \cdot pNoise$. In order to filter out noisy measurements, each receiver checks the integrity of the received signal and discards suspicious measurements. Thus, for the *good measurements* there is a probability *pFlaseNegative* of false negative from the integrity checking method and there is *pFlasePositive* probability of false positive for the *bad measurements*. The result is also reported back along the spanning tree routing path with the probability of success *pReport*. We assume that each state of Figure 4.1 consumes the amount of energy shown in Table 4.1. We represent the table by a reward function $e : S \rightarrow \mathbb{R}$.

For the flow diagram of Figure 4.1 we can build a DTMC: $A = (S, \mathbf{M})$, where S={ *Start, Radio, Beep,* *GoodReport*, *BadReport*, *Done*, *Retry*}, and

$$\mathbf{M} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ pSta & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & pRad & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & pDet \cdot (1 - pNoi) & 0 & 0 & 0 & 0 \\ 0 & 0 & pDet \cdot pNoi & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & pRep \cdot (1 - pFN) & pRep \cdot pFP & 1 & 0 \\ 1 - pSta & 1 - pRad & 1 - pDet & 1 - pRep \cdot (1 - pFN) & 1 - pRep \cdot pFP & 0 & 0 \end{bmatrix}.$$

Let the reliability of the TDOA method be the probability that a good measurement is reported[1]. Then the reliability R from a pmf $\mathbf{x}(t)$ is:

$$\begin{aligned} R &= \sum_{s \in S} \left(\sum_{\tau = t}^{\infty} pRep \cdot (1 - pFN) \cdot \mathrm{P}[A(\tau) = GoodReport | A(t) = s] \right) \cdot \mathrm{P}[A(t) = s] \\ &= \left([0, 0, 0, pRep \cdot (1 - pFN), 0, 0, 0] \cdot \sum_{\tau = 0}^{\infty} \mathbf{M}^{\tau} \right) \cdot \mathbf{x}(t). \end{aligned}$$

Note that if the steady state pmf $\mathbf{x}(\infty)$ is orthogonal to the reward vector $[0, 0, 0, pRep \cdot (1 - pFN), 0, 0, 0]$, R is finite. Because the *Done* state is reachable from every state and it is the only sink state of the DTMC A, the steady state pmf is

$$\mathrm{P}[A(\infty) = s)] = \begin{cases} 1 & \text{if s} = Done \\ 0 & \text{Otherwise.} \end{cases}$$

Thus, $\mathbf{x}(\infty)$ is orthogonal to the reward vector and R has finite value.

Expected energy consumption level to finish the method is another important performance criterion in energy limited computing environment. The expected energy consumption level at each moment is

$$\Delta E(t) = \sum_{s \in S} e(s) \cdot \mathrm{P}[A(t) = s],$$

[1]The reliability of software is the probability that a process of that software eventually terminates successfully [52, 54].

and the expected energy to finish the whole process from a pmf $\mathbf{x}(t)$ is

$$\begin{aligned} E(t) &= \sum_{s \in S} \left(\sum_{\tau=t}^{\infty} \sum_{s' \in S} e(s') \cdot \mathrm{P}[A(\tau) = s' | A(t) = s] \right) \cdot \mathrm{P}[A(t) = s] \\ &= \left([5, 1, 3, 5, 5, 0, 0] \cdot \sum_{\tau=0}^{\infty} \mathbf{M}^{\tau} \right) \cdot \mathbf{x}(t). \end{aligned}$$

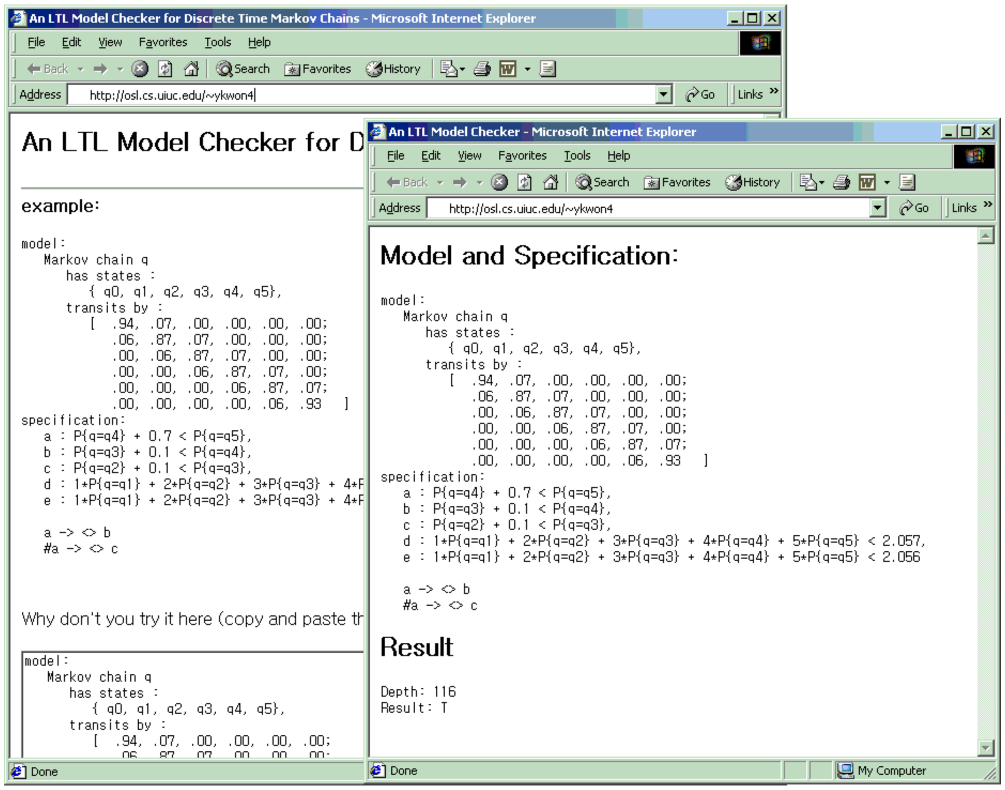
Again, because the reward vector $[5, 1, 3, 5, 5, 0, 0]$ is orthogonal to the steady state pmf $[0, 0, 0, 0, 0, 1, 0]^T$, $E(t)$ has a finite value.

4.2 Reliability Assessment using iLTL Model Checking

In this example, we examine the effect of parameter changes on the performance of the system. Figure 4.7 and Figure 4.8 show an iLTLChecker description for performance evaluation of the DTMC A. The description begins with an optional variable declaration block. After the variable declaration, a block for a list of DTMC definitions follows. Each DTMC definition has the name of the DTMC, a set of states that the DTMC can be at, and a probability transition matrix. Finally, a specification block is defined. The specification block begins with an optional inequality definition block followed by an iLTL specification. Also, any line after a `#` mark is regarded as a comment. A complete description of the iLTLChecker syntax is in Appendix B. Figure 4.2 is a snapshot of the model checker which is available from `http://osl.cs.uiuc.edu/~ykwon4`.

The inequalities `a` and `b` of Figure 4.8 restrict that the TDOA method begins from the *Start* state. The inequality `c` specifies that the availability of A is less than that of B. Note that if the reliabilities of the two systems are the same, then `c` from a state $\{\mathbf{x}^{(A)}(t), \mathbf{x}^{(B)}(t)\}$ means that during time 0 to $t-1$, the reliability of A is larger than that of B because from time t the reliability of A is smaller than that of B. That is, up to time $t-1$ A has more possibility of correct measurements than B does. The inequality `d` is similar to the inequality `c`, except that it uses a time index–*6* in this example. In order to clarify the meaning of the time index let $\mathbf{Z} \cdot \mathbf{\Lambda} \cdot \mathbf{Z}^{-1}$ be the diagonalization of $\mathbf{M}$ and let the matrices $\mathbf{P}(t)$ and $\mathbf{Q}(t)$ be

$$\mathbf{P}(t) = \mathbf{M}^t = \mathbf{Z} \cdot \mathbf{\Lambda}^t \cdot \mathbf{Z}^{-1},$$



An LTL Model Checker for Discrete Time Markov Chains - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://osl.cs.uiuc.edu/~ykwon4
Go Links
Done

An LTL Model Checker for D

example:

```
model:
   Markov chain q
      has states :
         { q0, q1, q2, q3, q4, q5},
      transits by :
         [  .94, .07, .00, .00, .00, .00;
            .06, .87, .07, .00, .00, .00;
            .00, .06, .87, .07, .00, .00;
            .00, .00, .06, .87, .07, .00;
            .00, .00, .00, .06, .87, .07;
            .00, .00, .00, .00, .06, .93   ]
specification:
   a : P{q=q4} + 0.7 < P{q=q5},
   b : P{q=q3} + 0.1 < P{q=q4},
   c : P{q=q2} + 0.1 < P{q=q3},
   d : 1*P{q=q1} + 2*P{q=q2} + 3*P{q=q3} + 4*F
   e : 1*P{q=q1} + 2*P{q=q2} + 3*P{q=q3} + 4*F

   a -> <> b
   #a -> <> c
```

Why don't you try it here (copy and paste th

```
model:
   Markov chain q
      has states :
         { q0, q1, q2, q3, q4, q5},
      transits by :
         [  .94, .07, .00, .00, .00, .00;
```

An LTL Model Checker - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://osl.cs.uiuc.edu/~ykwon4
Go Links
Done
My Computer

Model and Specification:

```
model:
   Markov chain q
      has states :
         { q0, q1, q2, q3, q4, q5},
      transits by :
         [  .94, .07, .00, .00, .00, .00;
            .06, .87, .07, .00, .00, .00;
            .00, .06, .87, .07, .00, .00;
            .00, .00, .06, .87, .07, .00;
            .00, .00, .00, .06, .87, .07;
            .00, .00, .00, .00, .06, .93   ]
specification:
   a : P{q=q4} + 0.7 < P{q=q5},
   b : P{q=q3} + 0.1 < P{q=q4},
   c : P{q=q2} + 0.1 < P{q=q3},
   d : 1*P{q=q1} + 2*P{q=q2} + 3*P{q=q3} + 4*P{q=q4} + 5*P{q=q5} < 2.057,
   e : 1*P{q=q1} + 2*P{q=q2} + 3*P{q=q3} + 4*P{q=q4} + 5*P{q=q5} < 2.056

   a -> <> b
   #a -> <> c
```

Result

```
Depth: 116
Result: T
```

Figure 4.2: A snapshot of iLTLChecker for M/M/1 queuing system.

$$\mathbf{Q}(t) = \sum_{\tau=t}^{\infty} \mathbf{Z} \cdot \begin{bmatrix} 0 & 0 & & 0 \\ 0 & \lambda_2^{\tau} & & \\ & & \ddots & \\ 0 & & & \lambda_n^{\tau} \end{bmatrix} \cdot \mathbf{Z}^{-1} = \mathbf{Q}(0) \cdot \mathbf{P}(t),$$

where $\lambda_1 = 1$ of $\mathbf{\Lambda}$ is replaced by 0. Note that we assume that the reward vectors for accumulated rewards are orthogonal to $\mathbf{z}_1$. With this assumption, we can safely replace the λ_1 with 0. Then,

$$\begin{aligned} \mathrm{P}[A(t) = s_i] &= \sum_{j=1}^{n} \mathbf{P}(t)_{i,j} \cdot \mathrm{P}[A(0) = s_j], \\ \mathrm{Q}[A(t) = s_i] &= \sum_{j=1}^{n} \mathbf{Q}(t)_{i,j} \cdot \mathrm{P}[A(0) = s_j]. \end{aligned}$$

Thus, d compares the two reliabilities after 6 steps. The inequality e compares the expected energy ($E(t)$)to

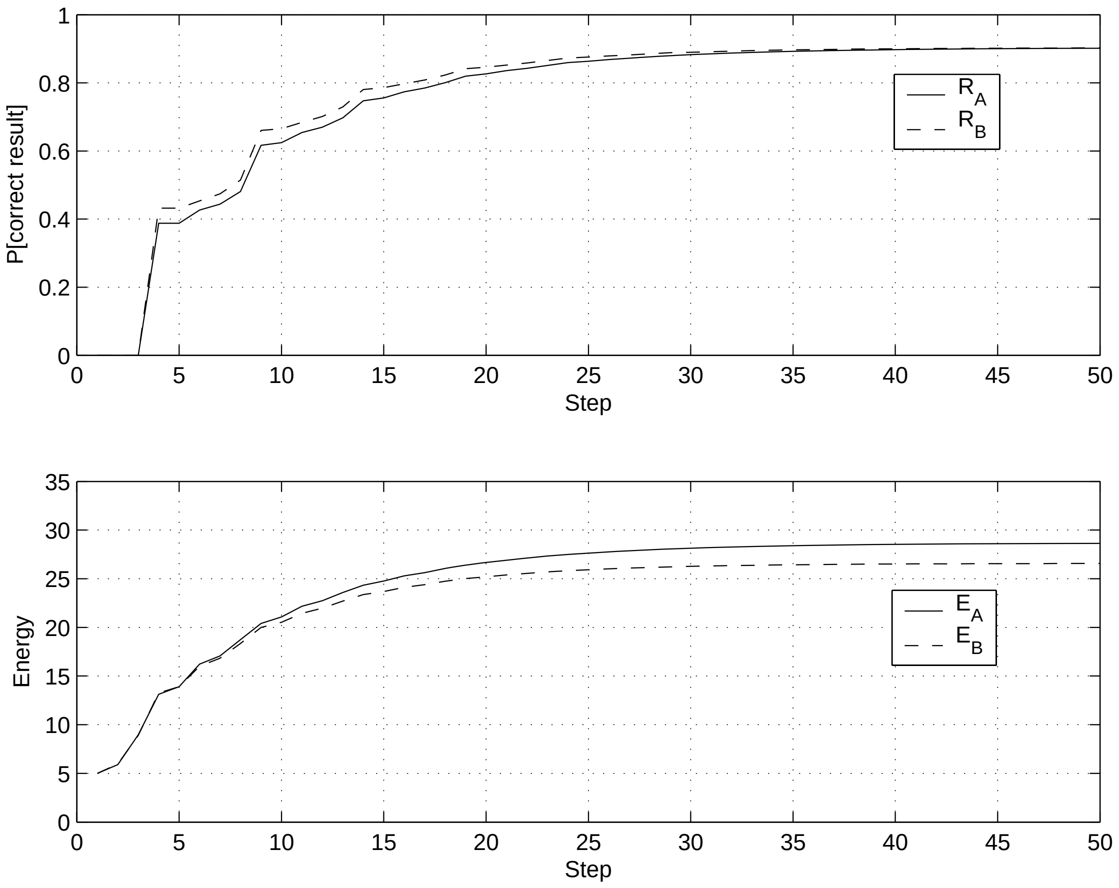


Figure 4.3: The effect of increased reliabilities in command sending module and result reporting module. It does not increase an overall reliability of the TDOA operation. However, it speeds up the measurement process. Furthermore, it reduces the expected energy to finish the measurement.

finish the TDOA operation in the two systems.

$$E(t) \quad = \quad \sum_{i=1}^{n} e(s_i) \cdot \mathrm{Q}[A(t) = s_i]$$

Inequalities `i` and `j` specify the probability that the TDOA operation is finished. That is, the inequality `i` specifies that the probability that the operation is done is at least 10%.

The model checking result with Specification 1.1 with the modified parameters *pSta2=0.95* and *pRep2 = 0.95* is as follows:

```
Depth:  5646
Result: F
counterexample:
    pmf(A(0)): [ 1.0 0.0 0.0 0.0 0.0 0.0 0.0 ]
    pmf(B(0)): [ 1.0 0.0 0.0 0.0 0.0 0.0 0.0 ]
```

The first line *Depth: 5646* shows the required search depth to check the specification. The second line *Result: F* means that the model does not satisfy the specification. That is, the increased reliabilities in start command forwarding and result reporting mechanisms do not increase the reliability of the whole operation. The last two lines are the counterexample: the initial pmfs such that the computational path from them violates the specification. Specification 1.2 describes that the increased reliability on command sending module decreases the whole system reliability. The result is false in this case also. From the model checking results of 1.1 and 1.2, we can conclude that those module reliabilities do not affect the overall reliability of the system. Thus, we checked the transient reliabilities of the system. Specification 1.3 checks whether in the early stages of the operation, the improved system has more possibility of correct result. In other words, whether it speeds up the measurement process. The model checking result confirms it to be true. The first graph of Figure 4.3 shows the result. The dashed line is above the solid line although they overlap in the steady state. Specification 1.4 checks whether the improvement reduces the expected energy consumption to finish the operation. Model checking result is true and the second graph of Figure 4.3, the expected energy consumption level, shows it.

The model checking results of the previous experiments show that improving the module reliabilities of sending command and of reporting results do not improve the overall reliability of the system. However, they speed up the process and save energy. Now, we check improving which of the two modules is more beneficial. For this experiment we use the module reliabilities of *pSta=0.9, pRep=0.95* for system *A* and *pSta2=0.95, pRep2=0.2* for system *B*. After several trials of model checking, we found that from step 6 and before the steady state, system *A* has slightly higher probability of getting the correct result than *B*. That is, improving the result reporting module speeds up the process more than improving the command sending, module. Specification 2.1 specifies this property and is model checked to be true. Specification 2.2 is a simpler expression of Specification 2.1 and has the same result. The first graph of Figure 4.4 shows the result. The reliabilities of the two systems are the same, but system *A* is faster than system *B*. Specification 2.3 checks whether system *A* saves more energy than system *B*. The model checker reports it to be true and the second graph of Figure 4.4 shows the result.

We tested the effect of integrity checking filter on the reliability. First, we compare the system with the

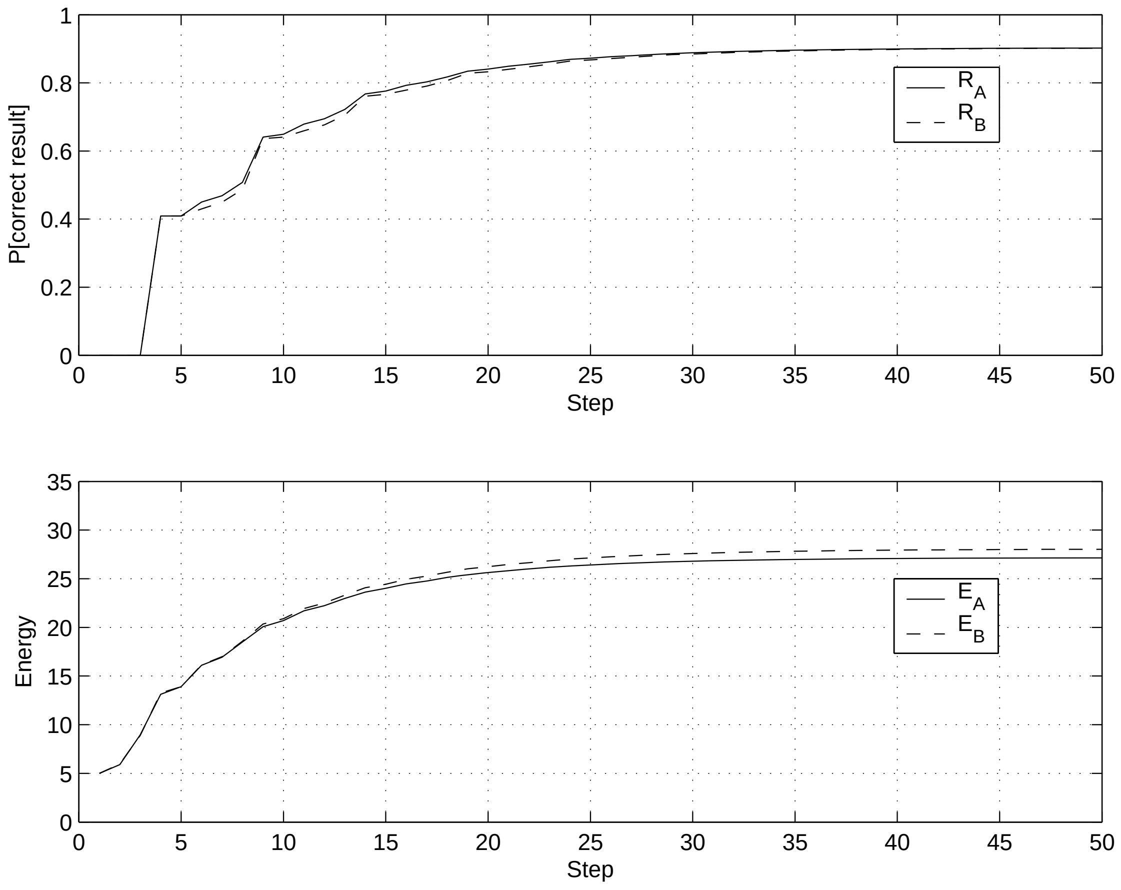


Figure 4.4: Comparisons between increased reliability in command sending module versus increased reliability in result reporting module. If they are both improved by the same amount (5.56%), improving the reporting module speeds up and saves energy as well.

original filter and the system with a more optimistic filter. That is, the filter accepts more noisy measurement and has more possibility of false positives and less possibility of false negatives. The formula 3.1 specifies whether this change increases the reliability, and Formula 3.2 specifies whether it decreases the reliability. The model checking results on the specification verify that the change reduces the reliability. iLTL formulas 3.3 and 3.4 specify whether the change increases or decreases the expected energy to finish the measurement and the model checking result shows that the change saves the expected energy. We also tested the effect of pessimistic filter. That is a filter with higher possibility of false negatives and lower possibility of false positives. iLTL formulas 4.1 and 4.2 specify whether the pessimistic filter increases or decreases the reliability and the model checking results verify that the change increases the reliability. Formulas 4.3 and 4.4 are about the effect on the expected energy consumption level and the model checking result verifies that the pessimistic filter increases the expected energy consumption level.

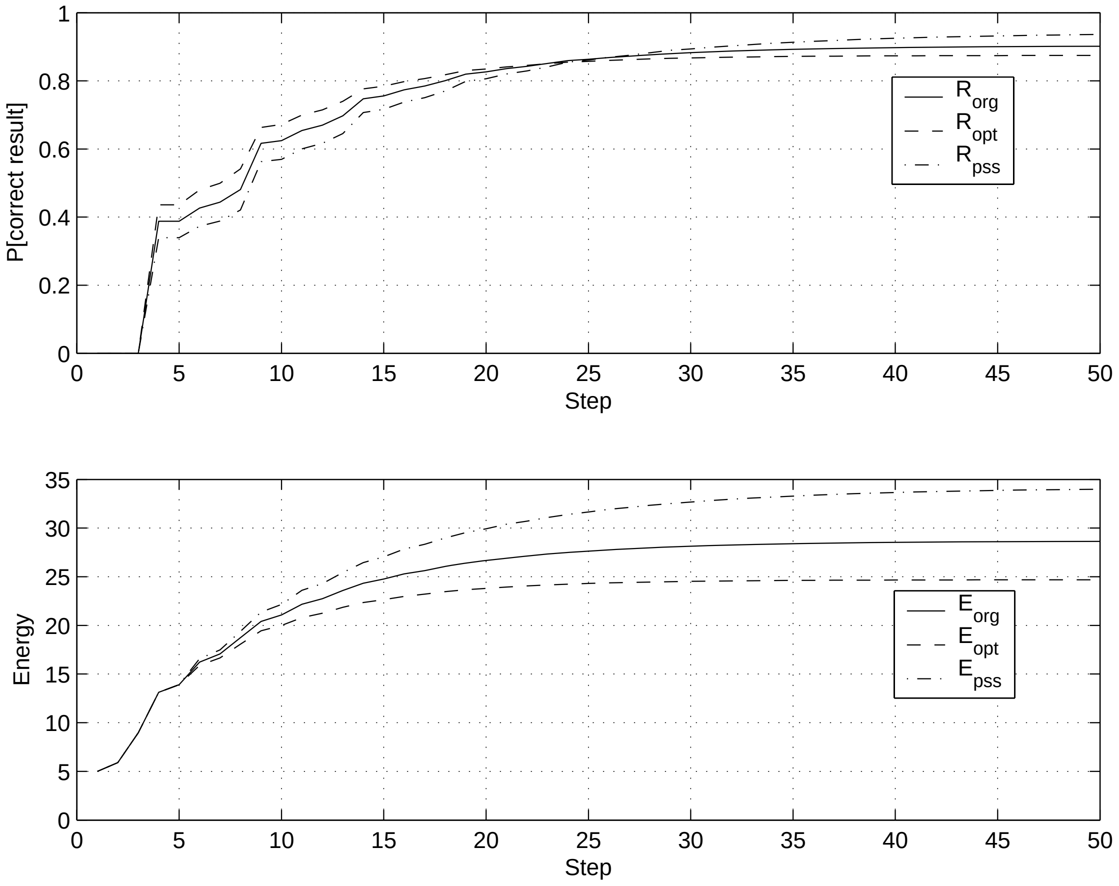


Figure 4.5: The effect of the noisy measurement filter. Optimistic filter reduces the overall reliability. However, it speeds up the measurement process and saves energy. Pessimistic filter increases the overall reliability. However, it slows down the process and uses more energy.

Figure 4.5 shows the effect of filter changes on the reliability of the system and on the expected energy consumption level. The first graph of Figure 4.5 is about the reliability: the solid line shows the reliability with the original filter, the dashed line shows the reliability with the optimistic filter, and the dot-dashed line shows the reliability with the pessimistic filter. Although the optimistic filter is faster, it reduces the reliability. The pessimistic filter is slower but it increases the reliability. The second graph of Figure 4.5 shows the expected energy consumption level: the optimistic filter saves energy and the pessimistic filter consumes more energy.

Finally, we checked a system characteristics about the expected energy consumption level. The formulae 5.1 of Figure 4.8 specifies that five consecutive observations of expected energy consumption level less than 1.5 unit of energy is enough evidence that the expected energy consumption level of the system is always less than 1.5 unit. The model checking result is true. The formulae 5.2 of Figure 4.8 specifies whether

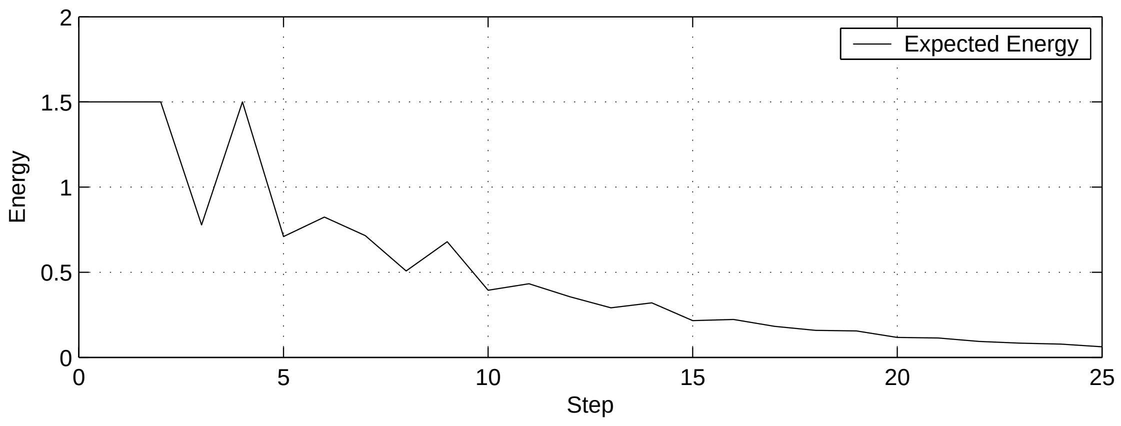


Figure 4.6: A counter example of Specification 5.2: a transition of expected energy consumption level.

four consecutive observation of expected energy consumption level less than 1.5 unit is sufficient evidence for the same condition. For this specification the model checker returns false with a counter example of $\mathbf{x}^{(A)}(0)$ = [0, 0.186, 0, 0.220, 0.043, 0.357, 0.194]. Figure 4.6 shows how the expected energy changes from the counter example. Note that from step zero to step three the expected energy consumption level was less than 1.5. However, on step four, it becomes 1.5 again.

4.3 Discussion

The reliability of a system in an environment with randomness can be increased by adding redundancies to the system such as repeated trials of a task. However, the increased reliability are achieved at the cost of other performance metrics such as the longevity or the throughput of the system. In this section, we demonstrated the usefulness of our methodology in tunning multiple parameters of a system through a case study on TDoA distance measurement protocol. We evaluated the effects of the parameter changes on various system performance criteria. Our method is more useful than the traditional steady state analysis based approaches in that it can systematically check transient as well as steady state behaviors of a system. Furthermore, the specification logic is general enough to evaluate other system properties, such as energy consumption level, than the reliability of a system.

```
var:
    eps   = 0.0000001,
    pSta  = 0.9,  pRep  = 0.9,
    pRad  = 0.95, pDet  = 0.9,  pNoi = 0.3,
    pFP   = 0.2,  pFN   = 0.2,
    pSta2 = pSta, pRep2 = pRep, pFP2 = pFP, pFN2 = pFN,

    pSta2 = 0.95, pRep2 = 0.95
#   pSta2 = 0.95, pRep  = 0.95
#   pFP2  = 0.3,  pFN2  = 0.1
#   pFP2  = 0.1,  pFN2  = 0.3

model:
  Markov chain A
    has states :
      { Start, Radio, Beep, GoodReport, BadReport, Done, Retry },
    transits by :
       [   0,       0,      0,              0,                 0,             0, 1;
           pSta,    0,      0,              0,                 0,             0, 0;
           0,       pRad,   0,              0,                 0,             0, 0;
           0,       0,      pDet*(1-pNoi),  0,                 0,             0, 0;
           0,       0,      pDet*pNoi,      0,                 0,             0, 0;
           0,       0,      0,              pRep*(1-pFN),      pRep*pFP,      1, 0;
           1-pSta,  1-pRad, 1-pDet,         1-pRep*(1-pFN),    1-pRep*pFP,    0, 0     ],

  Markov chain B
    has states :
      { Start, Radio, Beep, GoodReport, BadReport, Done, Retry },
    transits by :
       [   0,       0,      0,              0,                 0,             0, 1;
           pSta2,   0,      0,              0,                 0,             0, 0;
           0,       pRad,   0,              0,                 0,             0, 0;
           0,       0,      pDet*(1-pNoi),  0,                 0,             0, 0;
           0,       0,      pDet*pNoi,      0,                 0,             0, 0;
           0,       0,      0,              pRep2*(1-pFN2),    pRep2*pFP2,    1, 0;
           1-pSta2, 1-pRad, 1-pDet,         1-pRep2*(1-pFN2), 1-pRep2*pFP2, 0, 0     ]
```

Figure 4.7: An iLTLChecker description: optional variable declaration block and Markov model definition block. A specification will continue in Figure 4.8.

```
specification:
  a : P[A=Start] > 1-eps,
  b : P[B=Start] > 1-eps,
  c : pRep*(1-pFN)*Q[A   =GoodReport] < pRep2*(1-pFN2)*Q[B   =GoodReport],
  d : pRep*(1-pFN)*Q[A(6)=GoodReport] < pRep2*(1-pFN2)*Q[B(6)=GoodReport],
  e : 5*Q[A=Start]+1*Q[A=Radio]+3*Q[A=Beep]+5*Q[A=GoodReport]+5*Q[A=BadReport] >
      5*Q[B=Start]+1*Q[B=Radio]+3*Q[B=Beep]+5*Q[B=GoodReport]+5*Q[B=BadReport],
  f : 5*P[A=Start]+1*P[A=Radio]+3*P[A=Beep]+5*P[A=GoodReport]+5*P[A=BadReport] < 1.5,
  i : P[A=Done] > .1,
  j : P[A=Done] < .99

  ##################################################
  # when pSta2 + and pRep2 +
  ##################################################
  (a /\ b) -> c              # 1.1) F: no reliability change
  #(a /\ b) -> ~ c           # 1.2) F
  #(a /\ b) -> [] ((i /\ j) -> ~ c) # 1.3) T: more correct result in early stage
  #(a /\ b) -> e             # 1.4) T: less energy consumption

  ##################################################
  # when pSta2 + and pRep +  (not pRep2)
  ##################################################
  #(a /\ b) -> X X X X X X [] (j -> c) # 2.1) T: less correct result in early stage
  #(a /\ b) -> [] (j -> d)  # 2.2) T: pRep + is better in transient reliability than pSta2 +
  #(a /\ b) -> ~e           # 2.3) T: pRep + is better in energy consumption than pSta2 +

  ##################################################
  # optimistic filter: pFP2 +, pFN -
  ##################################################
  #(a /\ b) -> c            # 3.1) F
  #(a /\ b) -> ~ c          # 3.2) T: less reliable
  #(a /\ b) -> e            # 3.3) T: less energy consumption
  #(a /\ b) -> ~e           # 3.4) F

  ##################################################
  # pessimistic filter: pFP2 -, pFN +
  ##################################################
  #(a /\ b) -> c            # 4.1) T: more reliable
  #(a /\ b) -> ~ c          # 4.2) F
  #(a /\ b) -> e            # 4.3) F
  #(a /\ b) -> ~e           # 4.4) T: more energy consumption

  ##################################################
  # instant expected energy consumption
  ##################################################
  #(f /\ X f /\ X X f /\ X X X f /\ X X X X f) -> [] f  # 5.1)
  #(f /\ X f /\ X X f /\ X X X f             ) -> [] f  # 5.2)
```

Figure 4.8: An iLTLChecker description: specification block has optional inequality definitions followed by an iLTL specification.

Chapter 5

Markov Model Estimation

Recall that a DTMC is finite state automation with transition probabilities between the states. As discussed earlier, we assume that identifying the states of interest from application programs is straightforward. In this section, we propose a Markov transition matrix estimation method based on *Quadratic Programming* (QP) [43]. We also provide a statistical test so that we can discard the estimated DTMC model if the real system does not exhibit Markov behavior.

5.1 DTMC Model Estimation

Assume that the property of interest is over states satisfying the Markov property, and that every node of the WSN is independent and identical. These assumptions are valid in many WSN applications: a class of such applications is systems where each node makes a probabilistic choice to change its state and in many cases this decision is made independent of the states of the other nodes. The ability to make such independent decision saves large amounts of energy, because synchronization requires information exchange among nodes. For example, in many applications a sensor node often goes into a sleep mode with certain probability regardless the states of other nodes. However, we can expect with high confidence that at least certain number of nodes are awake.

Even in a medium scale WSN, there are hundreds of nodes; this number provides a sufficiently large population for robust pmf estimation. If we estimate the pmfs periodically, we can estimate the underlying Markov transition matrix by comparing two consecutive estimations. In the experimental case study described in Section 6, each time-synchronized node periodically stores its state in its memory. We subsequently collect these sequences from all nodes and estimate a sequence of pmfs. Note that this state sampling code can be used once for Markov chain estimation and then disabled so that it does not affect the performance of a system.

Let $X = (S, \mathbf{M})$ be the true Markov process of the system. We compute an estimated Markov chain $(S, \hat{\mathbf{M}})$ such that $\hat{\mathbf{M}}$ minimizes a squared sum of differences between a one step predicted pmf $\hat{\mathbf{x}}(t+1)$ and a sampled pmf $\overline{\mathbf{x}}(t+1)$. We estimate the probability $x_i(t) = \mathrm{P}[X(t) = s_i]$ by the fraction of nodes in state s_i over the whole population size. We describe this process in more detail in order to drive a goodness of fit test. We first introduce a one-zero *Random Variable* (RV) Y_i for each state s_i such that $Y_i(X(t)) = 1$ if $X(t) = s_i$ and 0 otherwise. Let ns be the population size at each sample, and let an RV $K_i(t)$ be

$$K_i(t) = \sum_{t=1}^{ns} Y_i(X(t)).$$

Then $K_i(t)$ has a binomial distribution: $\mathrm{P}[K_i(t) \le n] = \mathrm{Bin}(n, ns, x_i(t))$, where

$$\mathrm{Bin}(n, m, p) = \sum_{i=0}^{n} \left(\begin{array}{c} m \\ i \end{array} \right) \cdot p^i \cdot (1-p)^{m-i}.$$

We use a normalized frequency $\overline{x_i}(t) = K_i(t)/ns$ as our point estimator for $x_i(t)$. Since each sample at a given point in time is independent, by the Central Limit Theorem (CLT) for large ns our point estimator $\overline{x_i}(t)$ has a normal distribution $N(\mu = x_i(t), \sigma = \sqrt{\frac{x_i(t)\cdot(1-x_i(t))}{ns}})$ [51].

Assume that we have m pmf samples of an n state Markov process and let $\mathbf{P} \in \mathbb{R}^{m\times n}$ be a matrix of these point estimates: $P_{ij} = \overline{x}_j(i)$. We estimate a Markov matrix $\hat{\mathbf{M}} \in \mathbb{R}^{n\times n}$ such that it minimizes the differences between a sampled pmf $\overline{\mathbf{x}}(t+1)$ and a one step predicted pmf $\hat{\mathbf{x}}(t+1 \mid t) = \hat{\mathbf{M}} \cdot \overline{\mathbf{x}}(t)$:

$$E = \min_{\hat{\mathbf{M}}} \sum_{t=1}^{m-1} \left| \overline{\mathbf{x}}(t+1) - \hat{\mathbf{M}} \cdot \overline{\mathbf{x}}(t) \right|^2.$$

Let $\mathbf{P}_c$ and $\mathbf{P}_f$ be $\mathbf{P}$'s sub-matrices of the first and the last $m-1$ rows. Then the matrix $\hat{\mathbf{M}}$ that minimizes E is:

$$\hat{\mathbf{M}} = (\mathbf{P}_c^T \cdot \mathbf{P}_c)^{-1} \cdot \mathbf{P}_c^T \cdot \mathbf{P}_f.$$

However, since $\hat{\mathbf{M}}$ is an estimate of a Markov matrix, there are constraints on $\hat{\mathbf{M}}$. These are $0 \le \hat{M}_{ij} \le 1$ for all $1 \le i, j \le n$ and $\sum_{i=1}^{n} \hat{M}_{ij} = 1$ for all $1 \le j \le n$. We can minimize E subject to those constraints by

Quadratic Programming [43].

Let $\mathbf{z} \in \mathbb{R}^{n^2 \times 1} = \left[\hat{\mathbf{M}}_{*1}^T, \hat{\mathbf{M}}_{*2}^T, \ldots, \hat{\mathbf{M}}_{*n}^T\right]^T$ and

$$\begin{aligned}
\mathbf{H} \in \mathbb{R}^{n(m-1) \times n^2} &= \begin{bmatrix} \mathbf{P}_c & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \mathbf{P}_c \end{bmatrix}, \\
\mathbf{f} \in \mathbb{R}^{n(m-1) \times 1} &= \left[\mathbf{P}_{f*1}^T, \ldots, \mathbf{P}_{f*n}^T\right]^T, \\
\mathbf{A} \in \mathbb{R}^{2n^2 \times n^2} &= [\mathbf{I}_{n^2}, -\mathbf{I}_{n^2}]^T, \\
\mathbf{b} \in \mathbb{R}^{2n^2 \times 1} &= \left[\mathbf{1}_{1,n^2}, \mathbf{0}_{1,n^2}\right]^T, \\
\mathbf{C} \in \mathbb{R}^{n \times n^2} &= \begin{bmatrix} \mathbf{1}_{1,n} & \cdots & \mathbf{0}_{1,n} \\ \vdots & \ddots & \vdots \\ \mathbf{0}_{1,n} & \cdots & \mathbf{1}_{1,n} \end{bmatrix}, \\
\mathbf{d} \in \mathbb{R}^{n \times 1} &= \mathbf{1}_{n,1}, \\
\mathbf{I}_n \in \mathbb{R}^{n \times n} &= \begin{bmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{bmatrix}, \\
\mathbf{1}_{a,b} \in \mathbb{R}^{a \times b} &= \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix}, \\
\mathbf{0}_{a,b} \in \mathbb{R}^{a \times b} &= \begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix},
\end{aligned}$$

where $\hat{\mathbf{M}}_{*i}$ and $\mathbf{P}_{f*i}$ are the i^{th} columns of $\hat{\mathbf{M}}$ and $\mathbf{P}_f$. Then $\hat{\mathbf{M}}$ can be obtained by

$$\begin{aligned}
&\min_{\mathbf{z}}\ 0.5\mathbf{z}^T\mathbf{H}^T\mathbf{H}\mathbf{z} - \mathbf{f}^T\mathbf{H}\mathbf{z} \\
&\quad \text{subject to} \\
&\quad \mathbf{A}\mathbf{z} \le \mathbf{b}, \\
&\quad \mathbf{C}\mathbf{z} = \mathbf{d}
\end{aligned}$$

5.2 Goodness of Fit Test for Estimated DTMCs

Although we can always define a set of states and estimate transition probabilities between the states through the estimation algorithm described in Section 5.1, we still do not know whether the real system has the Markov property. In this section, we propose a goodness of fit test against the estimated model. This test method statistically rejects the estimated model with a given level of confidence if the sequence of pmf samples does not agree with the model.

For this test, we first build a null hypothesis and an alternative hypothesis:

H_0**:** The actual process is a Markov chain whose transition matrix is the same as the estimated matrix.

H_a**:** The actual process is not a Markov chain or the Markov transition matrix is different from the estimated one.

Under the null hypothesis H_0, we can compute one step predicted pmf using a current sampled pmf. Using a χ^2 test between the predicted pmf and a sampled next step pmf, we can compare how well they fit together. More specifically, under the null hypothesis, the RV $q(t)$, called Pearson's test statistic, has χ^2 distribution with $n-1$ degrees of freedom:

$$q(t) = \sum_{i=1}^{n} \frac{(K_i(t) - ns \cdot \hat{x}_i(t \mid t-1))^2}{ns \cdot \hat{x}_i(t \mid t-1)},$$

where $K_i(t)$ is the RV we defined in Section 5.1 and $\hat{\mathbf{x}}(t \mid t-1) = \hat{\mathbf{M}} \cdot \overline{\mathbf{x}}(t-1)$ is a one step predicted pmf from the previous sample $\overline{\mathbf{x}}(t-1)$. We omit the degrees of freedom in χ^2 distribution for simplicity in notation. Thus, unless otherwise specified, χ^2 means χ^2 with $n-1$ degrees of freedom. We write χ^2_α for the value y such that $\chi^2(y) = \alpha$.

Now, we introduce a significance level γ, and a random variable B_γ such that

$$B_\gamma = \sum_{t=2}^{m} \delta_{\chi^2_{1-\gamma}}(q(t)),$$

where $\delta_\theta(x) = 1$ if $x \geq \theta$ and 0 otherwise. Note that, under the null hypothesis H_0, for the probability estimation at time t, $\mathrm{P}[q(t) > \chi^2_{1-\gamma}] = \gamma$ and the one-zero random variable $\delta_{\chi^2_{1-\gamma}}(q(t))$ has an expected value γ which is the probability of bad sampling under H_0. Since, there are m samples B_γ has a binomial distribution of order $m-1$ and probability γ. Let n_γ be a least integer such that $\mathrm{Bin}\left(n_\gamma, m-1, \gamma\right) \geq 1-\alpha$. We accept H_0 if for all $\gamma \in (0,1)$, $B_\gamma \leq n_\gamma$. Otherwise we reject H_0 with the probability of type I error α.

The problem is that γ is in $\mathbb{R}$, which we cannot enumerate. However, because B_γ takes finite number of values, we can still perform the goodness of fit test. Let s be the index of $q(t)$ when $q(t)$'s are sorted in the increasing order and let γ_s be $1-\chi^2(q(t))$. Note that B_{γ_s} is equal to $m-s$. We divide the interval for γ into several sub-intervals and do test on each of the m intervals: $(1,\gamma_1]$, $(\gamma_s, \gamma_{s+1}]$ for $1 \leq s \leq m-2$, and $(\gamma_{m-1}, 0)$. Because n_γ is m-1 and 0 for the intervals $1 < \gamma \leq \gamma_1$ and $\gamma_{m-1} < \gamma < 0$ respectively and so is B_γ, the test is trivially true in these intervals. Thus, $B_\gamma \leq n_\gamma$ for $0 < \gamma < 1$ if and only if $n_{\gamma_s} < m-s$ for $1 \leq s \leq m-1$.

5.3 Simulation Study

We illustrate the accuracy and usefulness of our DTMC estimation algorithm and goodness of fit test by means of a simulation study. Specifically, we model a set of processes running on WSN nodes by a three state automaton with *Ready, Run, Wait* states. When a process is initiated, it is in a *Ready* state. It then transitions to a *Run* state. If the process performs an I/O operation, it will transition to a *Wait* state, and after the I/O operation is completed, it will return to the *Ready* state. A process in the *Run* state moves back to the *Ready* state so that other processes running on the same node can proceed together. We assume that the process does not terminate; this is usual in WSN applications–typically, a WSN application monitors the environment and reports the result to a base station through the life of its platform. Assume that the system is a DTMC $PS = (S, \mathbf{M})$ where $S = \{Ready,\ Run,\ Wait\}$ and

$$\mathbf{M} = \begin{bmatrix} 0.95 & 0.07 & 0.05 \\ 0.05 & 0.90 & 0.0 \\ 0.0 & 0.03 & 0.95 \end{bmatrix}.$$

Figure 5.1 shows a block diagram of the DTMC PS; the ellipses represent states and the numbers represent transition probabilities.

We ran a simulation with 3000 independent and identical DTMC nodes of PS. Beginning from a state S, each node changes its state based on the transition probability matrix $\mathbf{M}$ at each step. We measured the frequencies of each state at each step and normalized these frequencies so that the sum of the fractions was

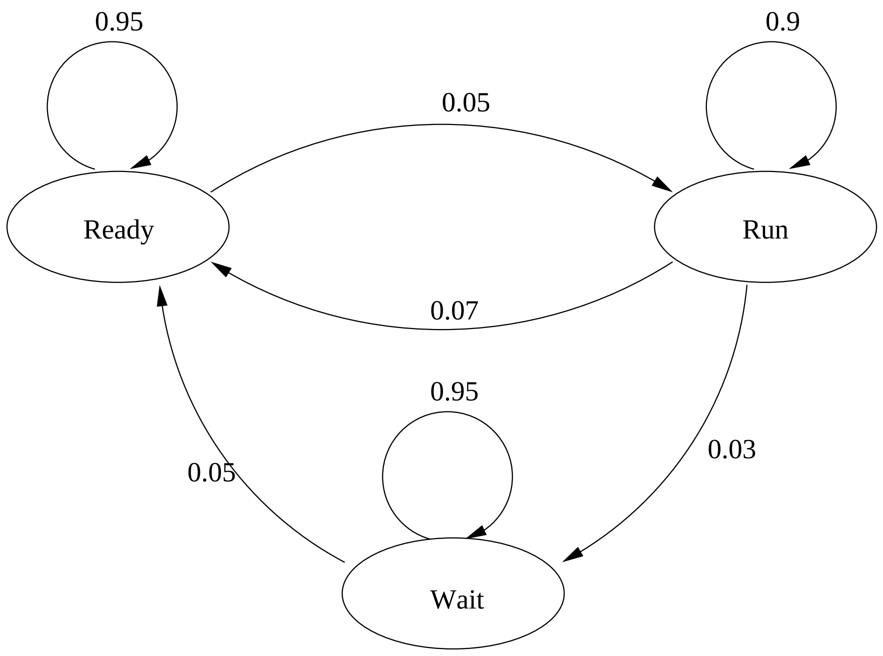


Figure 5.1: A process state diagram: the ellipses are states and the numbers at arrows are transition probabilities between states.

1. Recall that these normalized frequencies are the point estimates of the probabilities.

The first graph of Figure 5.2 shows the sampled pmf transitions of the 3000 nodes. Note that, there are some jitters in the graph. These jitters are due to the small sample population size. Based on the CLT, we hypothesize that the variance of the jitters will be inversely proportional to the number of nodes. The number of states, which is the degrees of freedom of $q(t)$ plus one, will relate inversely to the jitters in the observations. In order to confirm the cause of the jitter, we ran the same simulation again with 10^5 nodes. The second graph of Figure 5.2 shows the sampled pmf transitions. One can see that most of the jitters of the first graph have been removed. The thick lines of the third graph are the computed pmf transitions of the real system PS. We can see that the sampled pmf transitions of the 10^5 nodes looks very close to the computed pmf transitions. In order to check the effect of the number nodes on the sensitivity of the pmf estimation, we run the simulation with different numbers of nodes, varying from 10 to 100,000, and checked the variances of the pmf samples. The first graph of Figure 5.3 shows the variances with different numbers of nodes. Note that x-axis and y-axis are both log scaled. As we had expected, the variances are inversely proportional to the number of nodes.

In order to check the effect of the number of states on the sensitivity of the estimation, we build M/M/1 queuing systems [64] of different capacities and run several simulations with them. The reason we choose

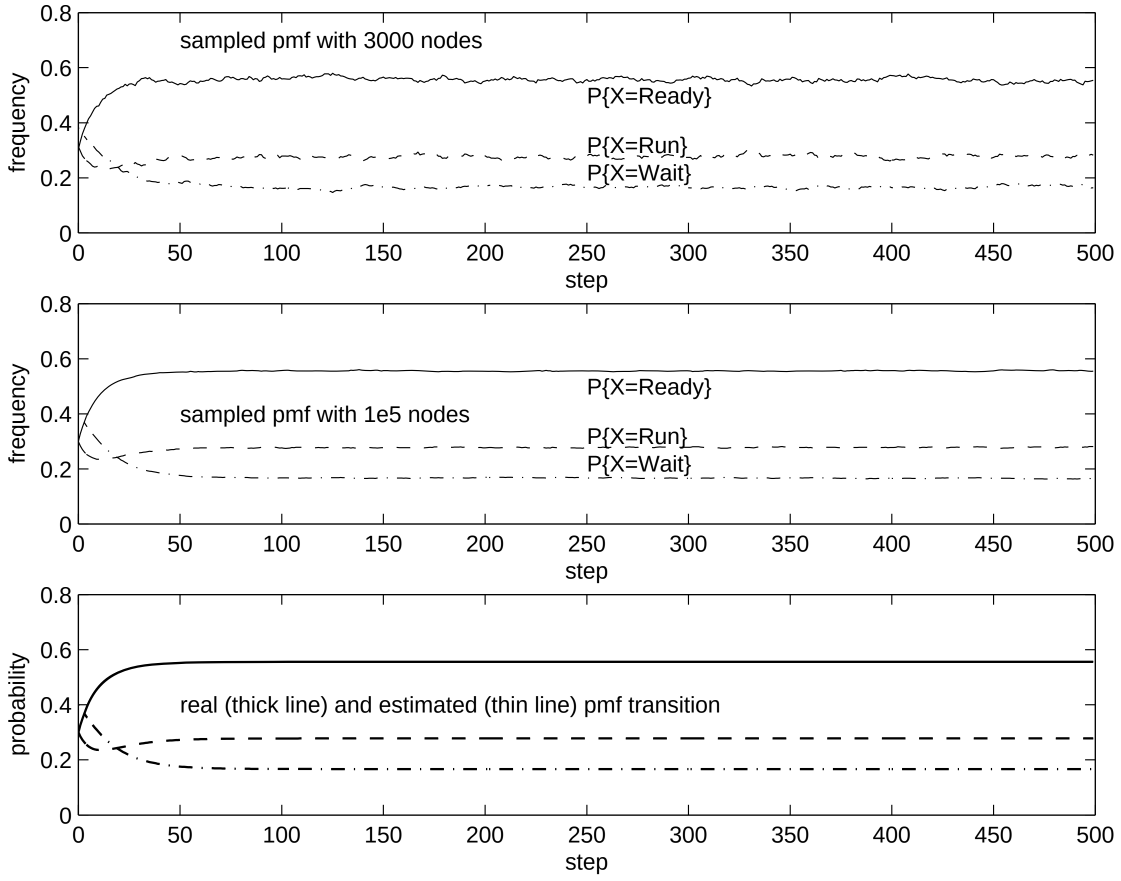
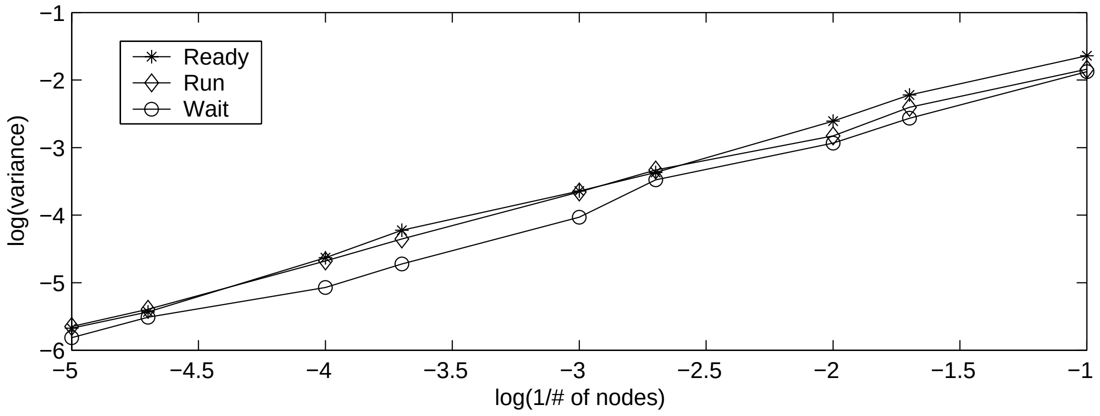


Figure 5.2: Top: pmf samples of 3000 node simulation. Middle: pmf samples of 10^5 node simulation. Bottom: predicted pmf transitions of a real (thick lines) and those of an estimated Markov chains (thin lines). The second graph shows that the cause of large jitters in the first graph is the small sample population size. The last graph shows how well the process is estimated.

queuing systems is because the number of states we want to check can be directly interpreted as the capacity of the queue. We set $\mu = 0.105/T$, $\lambda = 0.1/T$, number of nodes to be 5000, and capacity of the queues to be varying from 5 to 45 at a regular interval of 5, where T is the sampling period. We first checked the effect of the number of states on the Pearson's test statistic q. The second graph of Figure 5.3 shows a histogram of q with the number of states varying from 5 to 45. As we had expected, the histograms move toward larger values and become flat as the number of states increases. However, the relation between the number of states and the variance is not as significant as we had expected. We think that this is because the increase in the number of states distributes the nodes over a larger space, and this may counteract the effect of increased q value.



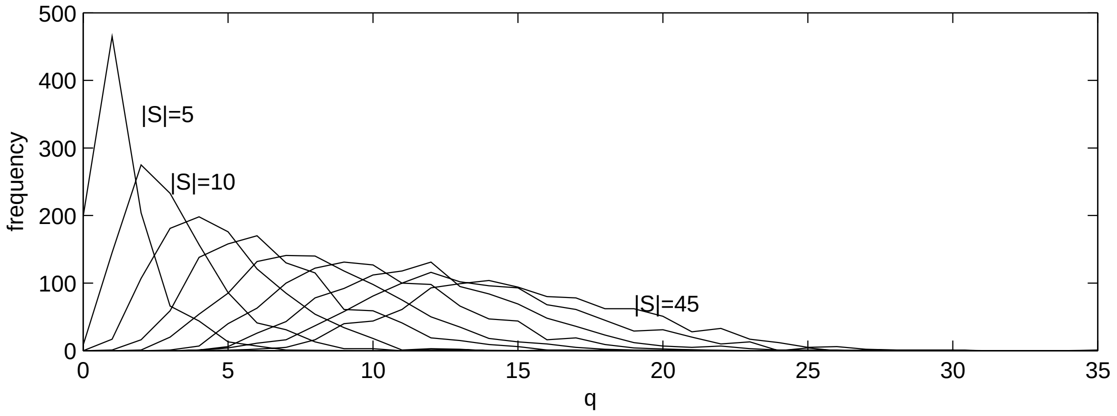


Figure 5.3: Top: Variance versus the number of nodes. As is expected from the CLT, the variance of sample is inversely proportional to the number of nodes. Bottom: Histogram of q values (Pearson's test statistic) versus the number of states from simulations of M/M/1 queuing system of different capacities. The q values have χ^2 distribution with the number of states minus 1 degrees of freedom.

From the simulation results of 3000 nodes, we estimate a DTMC $\hat{PS} = (S, \hat{\mathbf{M}})$, where

$$\hat{\mathbf{M}} = \begin{bmatrix} 0.9441 & 0.0823 & 0.0503 \\ 0.0532 & 0.8930 & 0.0000 \\ 0.0028 & 0.0246 & 0.9497 \end{bmatrix}.$$

For simplicity, the numbers in $\hat{\mathbf{M}}$ are truncated to five decimal numbers. We present a computed pmf transitions of $\hat{PS}$ as thin lines of the last graph of Figure 5.2. We can roughly see how close the estimated responses are to the real ones. The estimated model passes the goodness of fit test with a significance level of 0.01.

In order to demonstrate the effectiveness of our goodness of fit test, we modify $\mathbf{M}$ slightly (changing

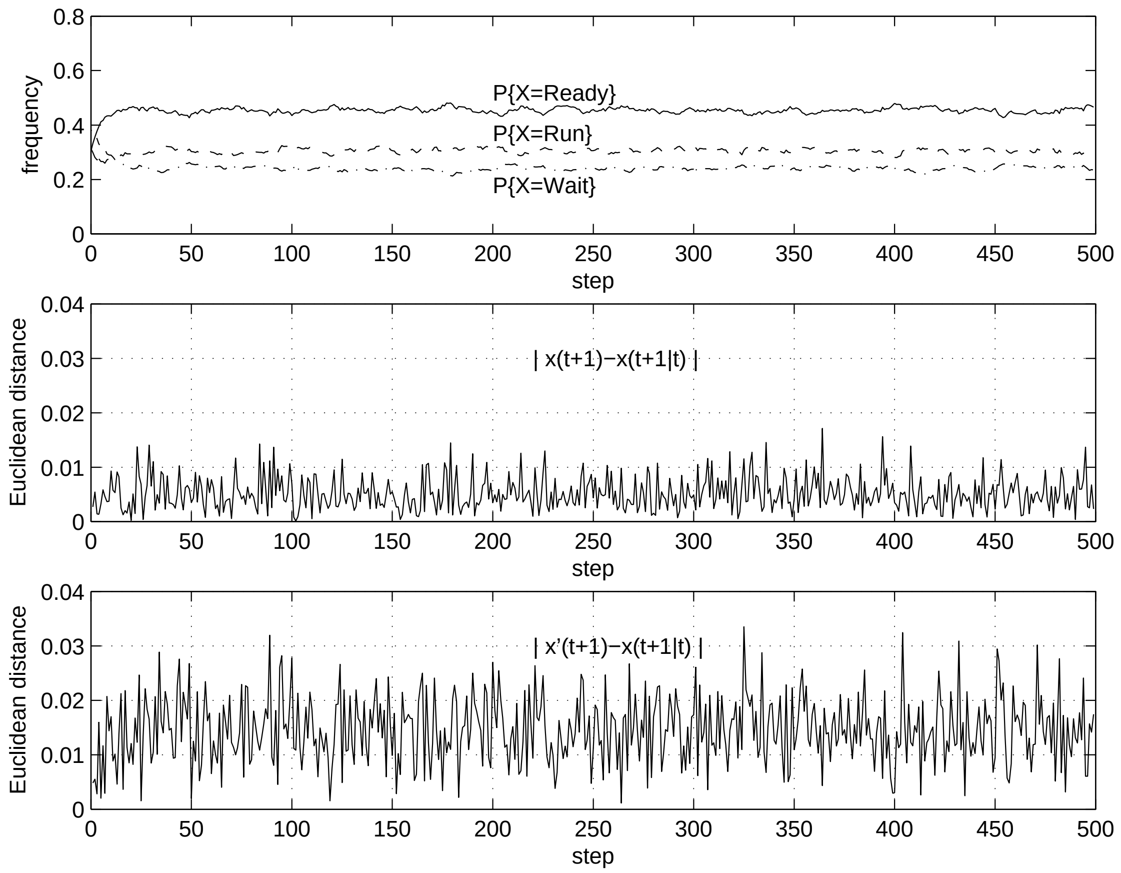


Figure 5.4: Top: pmf samples of 3000 node simulation with modified Markov chain PS'. Middle: difference between expected pmf ($\hat{PS}$) and sampled pmf (PS). Bottom: difference between expected pmf ($\hat{PS}$) and sampled pmf (PS'). The third graph shows larger error than the second graph: if the pmfs are sampled from a different Markov process then the differences between expected pmf and sampled pmf become large.

some of the entries by 0.05) to get $\mathbf{M}'$ below:

$$\mathbf{M}' = \begin{bmatrix} 0.9 & 0.07 & 0.1 \\ 0.1 & 0.85 & 0.0 \\ 0.0 & 0.08 & 0.9 \end{bmatrix}.$$

We then ran the same simulation with 3000 nodes with a DTMC $PS' = (S, \mathbf{M}')$. The first graph in Figure 5.4 shows how the sampled pmf transits with the modified Markov transition matrix $\mathbf{M}'$. Comparing with the first graph of Figure 5.2, the probability P[$PS = Ready$] is decreased and the others are increased. The second and the third graphs in Figure 5.4 show Euclidean distances between sampled pmfs and one step prediction by $\hat{PS}$ for the simulations of Figure 5.2 and Figure 5.4. That is, $|\mathbf{x}(t+1) - \hat{\mathbf{x}}(t+1 \mid t)|$

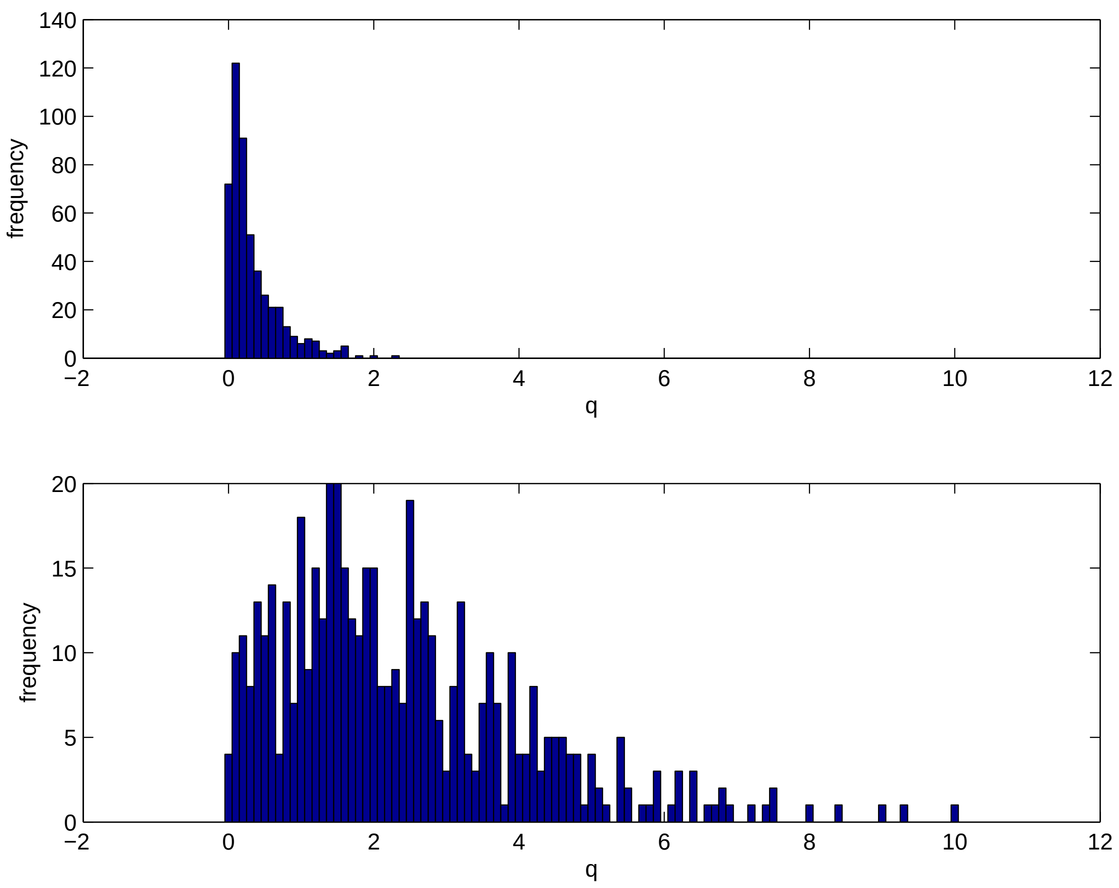


Figure 5.5: Histograms of q values (Pearson's test statistic). Top: samples from *PS* (real process), Bottom: samples from *PS'* (modified process). If samples are made from expected pmfs, the q values have a χ^2 distribution (Top). Otherwise they do not show the χ^2 distribution (Bottom).

and $|\overline{\mathbf{x}}'(t+1) - \hat{\mathbf{x}}(t+1 \mid t)|$ where, $\overline{\mathbf{x}}(t)$ is a sampled pmf vector of PS at step t, $\overline{\mathbf{x}}'(t)$ are that of PS' and $\hat{\mathbf{x}}(t+1 \mid t) = \hat{\mathbf{M}} \cdot \overline{\mathbf{x}}(t)$. As expected, the prediction error of the second graph is less than that of the third graph. Figure 5.5 shows histograms of Pearson's test statistic, q values, for the samples from the original Markov process *PS* and the samples from the modified Markov process *PS'*. The first graph shows much smaller q values than the second graph.

We checked the estimated DTMC $\hat{PS}$ against the simulation results of modified DTMC PS'. When $\gamma = 0.5099$, there are 271 samples out of 499 that do not satisfy $q < \chi^2_{0.4901}$. With the significance level of 0.01 and $\gamma = 0.5099$, the confidence interval is [0,335]. Thus, we reject the null hypothesis H_0 with the probability of type I error 0.01 and accept H_a.

Simulations suggest that the estimation error is reduced more significantly if we use a larger number of nodes than if we increase the number of trials. However, we do not have a formal proof of this. For each

χ^2-test, one more observation should be made. We are testing our estimates against a sample that is included in the overall estimation ($\hat{\mathbf{M}}$). This creates a bias in our test. However, this bias is slight, given that we are using 500 samples – removing a single sample will not significantly change the result. Therefore, we avoid the unncecessary recalculations.

Chapter 6

Experiments

We now describe a case study on a WSN platform. This case study proceeds along the same steps in Section 5.3: we estimate the Markov transition matrix, check it with the goodness of fit test of Section 5.2, and evaluate several performance criteria by iLTL model checking. Our experimental platform comprises of 90 Mica-2 motes. Each mote has an ATmega128L low power 8 bit CPU working at 8MHz clock speed, 4Kbyte of SRAM, 128Kbyte of program flash, and 512Kbyte of serial flash memory. Although, a mote has relatively large serial flash memory, this memory is slow especially for write operation. Mica-2 is also equipped with a CC1000 radio transceiver. At the lower level communication layer, the Manchester encoding is used, and we can achieve a theoretical throughput of 38.4Kbps. In order to save energy, an application can go to a sleep mode. The amount of energy spent in various modes is summarized in Table 6.1 [21].

TinyOS is an operating system running on Mica-2 nodes [70]. When developing an application, TinyOS provides a programming framework and library functions to support I/O operations. TinyOS is linked with application codes and loaded on Mica-2 nodes. TinyOS has three different program blocks: I/O operations are made by calling a *command* block, the result of an I/O operation can be passed to an application through an *event* block in the form of a signal, and application programs that run a long time should be written in a *task* block [22]. TinyOS schedules tasks by managing a queue of non-preemptable task blocks. Thus, spin-looping in a task block will block the entire operations of TinyOS.

6.1 Application scenario and program

We implemented the program in Figure 6.1. This application program samples a microphone and stores the result in a buffer (sampling code not shown). Whenever the `run` task is scheduled, it computes a Fourier

	Processor	Radio	
		transmit	receive
Active	8 mA	25 mA	8 mA
Sleep	$< 15\ \mu$A	$< 1\ \mu$A	

Table 6.1: Energy consumption level of a Mica-2 mote.

```
task run() {
  state=Run;
  DFT(mic);
  if(rand()%100<5)
    call SendMsg.send(),
    state=Wait;
  else
    post run(),
    state=Ready;
}
task dummy() {
  int i,n=rand()%4;
  for(i=0;i<n;i++)
    DFT(mic);
  post dummy();
}
```

```
event SendMsg.sendDone(){
  post run();
  state=Ready;
}
double DFT(int* smp) {
  s=c=0;
  for(i=0;i<T;i++)
    s+=sinV[i]*smp[i],
    c+=cosV[i]*smp[i];
  return sqrt(s*s+c*c);
}
```

Figure 6.1: Abbreviated experimental program. The `state` variable keeps track of the state of the system. There is a timer interrupt routine that samples the `state` value at a regular interval.

transform coefficient of the mic sample [50]. After the coefficient is computed, in order to reduce collisions, the application sends the result to a base station with a 5% probability. To make the experiment more realistic, we created a `dummy` task which simulates other applications running on the same node. We track the states of a process by recording them in the variable `state`. A timer interrupt routine is called at every 1/256 sec and samples the `state` variable. The state sample data is recorded on the SRAM because the serial flash is too slow and would affect the sampling. In view of the small SRAM, we compress the sample data by a run length encoding algorithm [58].

As one can see from the $\hat{\mathbf{M}}$ matrix (Section 6.2), when a process is in the *Wait* state it remains in this state with 95% probability which gives a high compression ratio to the encoding algorithm. Note that as sampling speed increases so does the probability that a process remains in its current state. That means that the run length encoding algorithm performs better at high frequency sampling. For this experiment we configure every node within a radio communication range of a base station. Because we do not need multi-hop message forwarding, we turn off the radio communication channel except when a node is sending

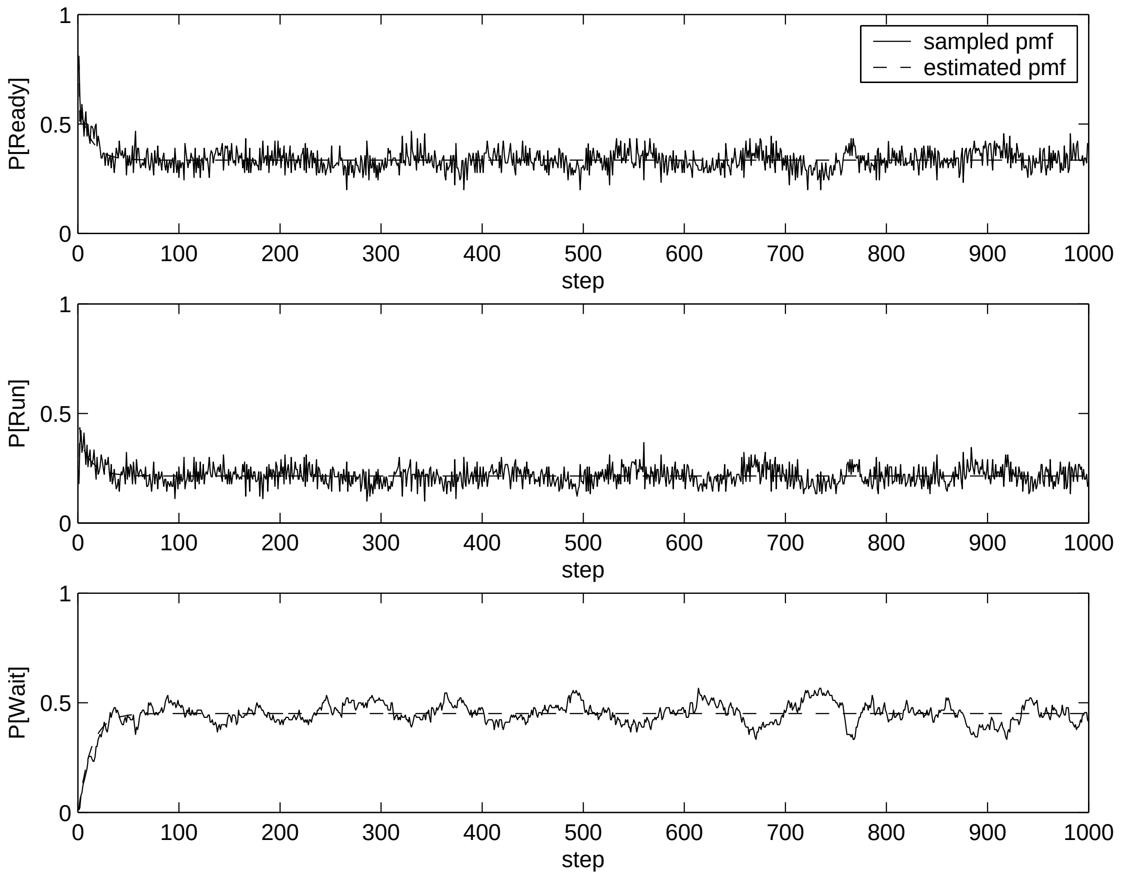


Figure 6.2: Sampled pmf transitions from 90 nodes experiment (solid lines) and predicted pmf transitions of an estimated Markov chain (dashed lines).

a message. This saves energy which might be consumed by unnecessary messages.

6.2 DTMC Estimation

The 90 nodes were programmed identically, except for their ids; the ids are used as seed values for a random number generator. All nodes are initially time synchronized by a start message from a base station: on receiving the start message, each node starts executing the code of Figure 6.1. A process state is sampled and compressed at the rate of 256 times per sec. We connected the sampling routine directly to a timer interrupt so that sampling accuracy is not compromised by the task scheduling of TinyOS. The sample data is retrieved from each node one by one. We aligned the 90 sequences of samples so that they begin at the same moment and compute a sequence of estimated pmfs.

The solid lines of Figure 6.2 show the transitions of sampled probability distribution. As we have seen

```
model:
   Markov chain A
      has states:
         { Ready, Run, Wait},
      transits by :
         [  .4691, .7383, .0435;
            .4827, .2455, .0000;
            .0482, .0162, .9565  ],
   Markov chain B
      has states:
         { Ready, Run, Wait},
      transits by :
         [  .4691, .7383, .0435;
            .4827, .2455, .0000;
            .0482, .0162, .9565  ],
    Markov chain C
      has states:
         { Ready, Run, Wait},
      transits by :
         [  .4691, .7383, .0435;
            .1827, .2455, .9000;
            .3482, .0162, .0565  ]
```

```
specification:
   a : P[A=Ready] > 0.5,
   b : P[A=Ready] > 0.3,
   c : P[A=Wait] > 0.27,
   d : 8*P[A=Ready] + 8*P[A=Run] + 33*P[A=Wait] < 25,
   e : 8*P[A=Ready] + 8*P[A=Run] + 33*P[A=Wait] > 15,
   f : P[A=Run] > P[B=Run] + 0.3,
   g : P[A=Ready] - P[B=Ready] < 0.05,
   h : P[A=Ready] - P[B=Ready] > -0.05,
   i : P[A=Ready] > P[B=Ready] > 0.1,
   j : 8*P[A=Ready] + 8*P[A=Run] + 33*P[A=Wait] <
       8*P[C=Ready] + 8*P[C=Run] + 33*P[C=Wait] + 5,
   k : 8*P[A=Ready] + 8*P[A=Run] + 33*P[A=Wait] <
       8*P[C=Ready] + 8*P[C=Run] + 33*P[C=Wait] + 1.3

   a -> X X [] b       # 1.
   #<> [] (d /\ e)     # 2.
   #(~e) U c           # 3.
   #(f /\ g /\ h) -> (X d /\ X X d /\ X X X d /\
   #                                 X X X X d) # 4.
   #j -> X X X k       # 5.
```

Figure 6.3: An iLTLChecker description of the estimated DTMC model and specifications.

in Figure 5.2, the jitters are due to small sample population size. From this sequence of pmf estimations, we estimate a DTMC $A = (S, \hat{\mathbf{M}})$, where $S = \{Ready, Run, Wait\}$ and

$$\hat{\mathbf{M}} = \begin{bmatrix} 0.4691 & 0.7383 & 0.0435 \\ 0.4827 & 0.2455 & 0.0000 \\ 0.0482 & 0.0162 & 0.9565 \end{bmatrix}.$$

The dashed lines of Figure 6.2 show pmf transitions of the estimated DTMC A. We plotted the estimated pmf transition graphs starting from the initial sample pmf. We applied the goodness of fit test of Section 5.2 to the estimated DTMC A and the pmf samples. Our estimated model passed the test with a significance level of 0.05.

6.3 Performance Evaluation

Figure 6.3 shows an iLTLChecker description of the estimated DTMC *A* and an iLTL specification. The inequalities `a` and `b` in Figure 6.3 describe whether the availability of DTMC *A* is larger than 0.5 and 0.3 respectively. Similarly, the inequality `c` specifies that the probability that a process is in the `Wait` state is larger than 0.27. Inequalities `d` and `e` are about expected energy consumption levels. From Table 6.1 we know that if a process is in the `Run` state or in the `Ready` state it consumes 8 mA of energy and if it is in the `Wait` state (sending messages) it consumes 33 mA (25 for radio + 8 for process) of energy. Thus, `e` specifies that the expected energy consumption level of a node is larger than 15 mA.

The first iLTL formula `a -> X X [] b` specifies that if the availability of the system is larger than 50%, then from two steps onward the availability never goes below 30%. The model checking result is true: iLTLChecker shows the following result.

```
Depth:  40
Result: T
```

The first line `Depth:40` is the maximum number of time step iLTL needs to search in order to check the specification. iLTLChecker shows the depth first before it begins search. This number gives a hint about the model checking time: if it is too large, one may need to change the specification. The second line `Result:T` means that the DTMC is a model of the specification. In other words, all pmf transitions of the DTMC *A* satisfy the specification.

If we slightly modify the specification to `a -> X [] b` or replace the RHS of the inequality `a` with 0.45, the DTMC *A* does not satisfy the specifications: availability of 50% now does not guarantee the minimum availability of 30% from the next step, and an availability of 45% now does not guarantee a minimum availability of 30% beginning from two steps later. The following is the output of iLTLChecker for the specification "P[A = *Ready*] > 0.45 → `X X` □ (P[A = *ready*] > 0.3)":

```
Depth:  40
Result: F
counterexample:
  pmf(A(0)): [ 0.461 0.000 0.539]
  pmf(B(0)): [ 1.000 0.000 0.000]
```

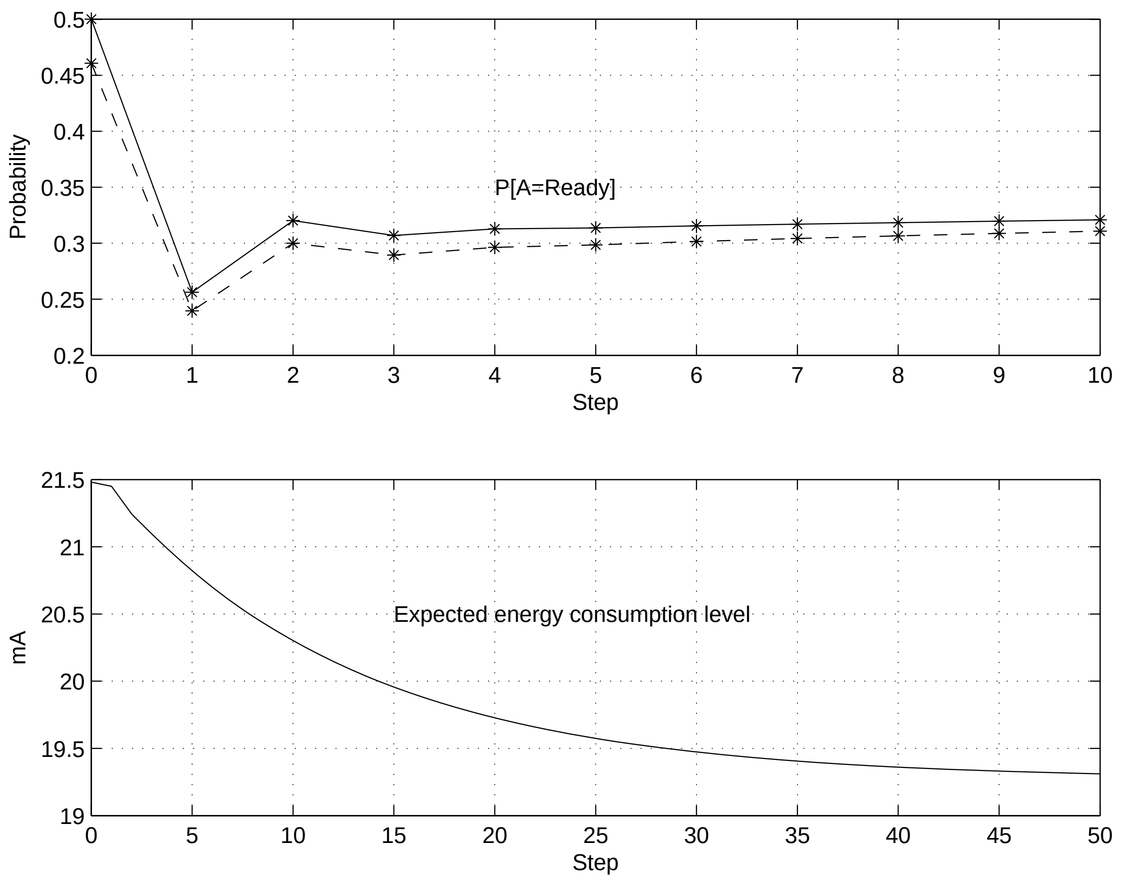


Figure 6.4: Top: Availability of system `A`. The dashed line shows that the availability become less that 30% at step 3. Bottom: expected energy consumption level of system `A`. In the steady state it will be 19.2 (mA).

```
pmf(C(0)): [ 1.000 0.000 0.000]
```

The second line indicates that the DTMC does not satisfy the specification and the 4th line shows a counterexample: an initial pmf for the DTMC *A* that leads to a violation of the specification.

We can check the result from the first graph of Figure 6.4. This graph shows how the availability of the system is changing over time from two different initial pmfs: $[0.5, 0, 0.5]^T$ for the solid graph and $[0.461, 0, 0.539]^T$ for the dashed graph. From the dashed graph, we can see an example run that satisfies `a -> X X [] b` but does not satisfy `a -> X [] b`: the availability is less than 30% at step 1 but it is always larger than 30% from step 2. The dashed line shows a counterexample of the modified specification: the availability is less than 30% at step 3 and step 4.

The second formula `<> [] (d ∧ e)` specifies that in the steady state the expected energy consumption level is in between 15 mA and 25 mA. From an eigenvalue analysis, we can compute that the steady state

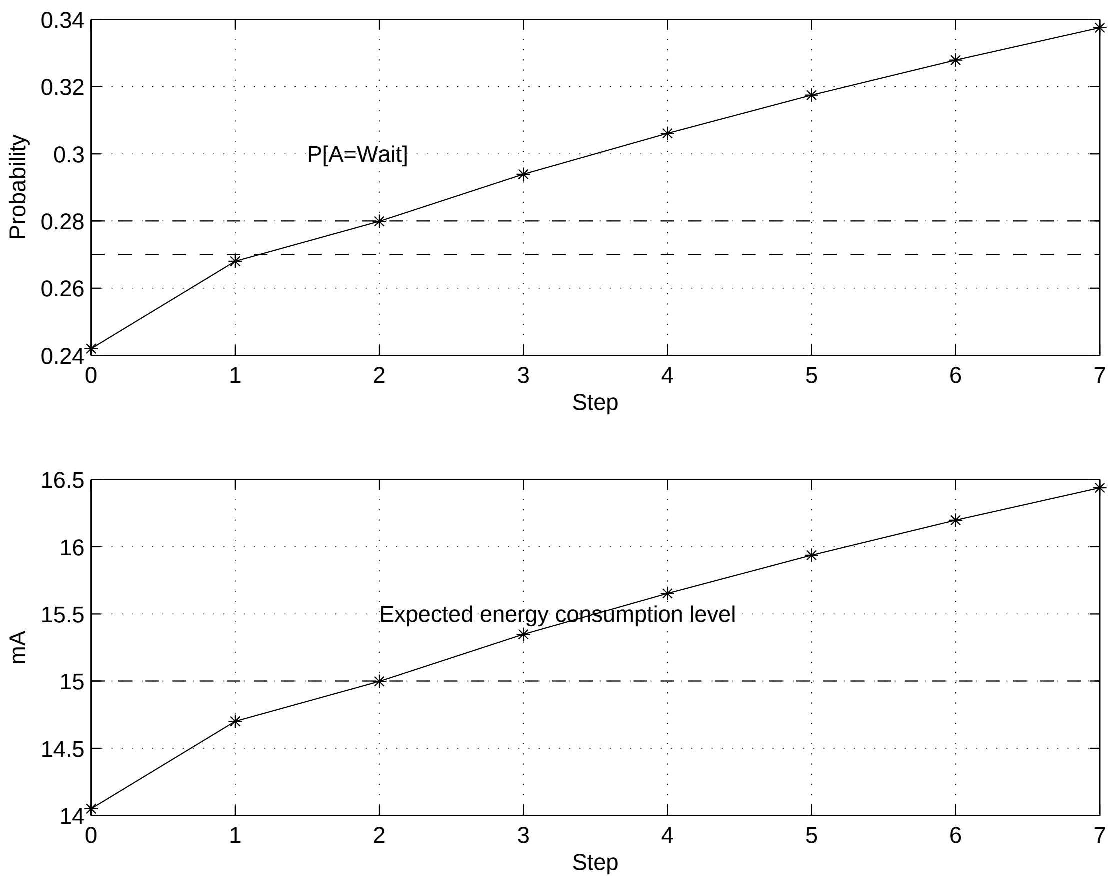


Figure 6.5: Expected energy consumption level and P[A = $Wait$]. At step 2, P[A = $Wait$] is 0.28 and the expected energy consumption level is 15 (mA), which is a counterexample of the modified third specification.

pmf of `A` is $\mathbf{x}(\infty) = [0.34, 0.21, 0.45]^T$ and the expected energy consumption level in the steady state is 19.2 mA. Thus, we expect the model checking result to be true; indeed, iLTLChecker returns true. The second graph of Figure 6.4 shows the transition of expected energy consumption level starting from the counterexample of the previous paragraph.

The third specification `(¬ e) U c` describes that the probability P[A = $Wait$] eventually becomes larger than 0.27 and while it is less than 0.27 the expected energy consumption level is less than 15 mA. iLTLChecker returns true for this statement. However, if we change the RHS of the inequality `c` to 0.28 it returns false with a counterexample of $[0.72, 0, 0.28]^T$. We can immediately check the result at step 0: P[A = $Wait$] > 0.28 is false and the expected energy consumption level is 15 mA. In order to check the transitions over time we draw Figure 6.5 from an initial pmf $[0.758, 0, 0.242]^T$. From the first graph of Figure 6.5, we can see that P[$A(t)$ = $wait$] > 0.27 is true from step 2 onward and the expected energy is less than 15 mA until step 1. Thus the original specification of Figure 6.3 is true. However since $\mathrm{P}[A(t) = \mathit{wait}] > 0.28$ is true from step 3 onward, the modified specification is false.

The fourth specification is about the effect of different initial conditions. The specification `(a ∧ b ∧ c) → (X d ∧ X X d ∧ X X X d ∧ X X X X d)` checks whether the availability of the system will be improved by at least 10% during the next four steps if 30% more nodes are in the *Run* state than there are now. The model checking result is true with the search depth 44. However, if we modify the specification as `(a ∧ b ∧ c) → (X d ∧ X X d ∧ X X X d ∧ X X X X d ∧ X X X X X d)`, the model checker returns false. That is, the 30% more nodes in the *Run* state do not guarantee a 10% increase in availability durning the next *five* steps. The following is the output of the iLTLChecker:

```
Depth:  44
Result: F
counterexample:
   pmf(A(0)): [ 0.698 0.302 0.000]
   pmf(B(0)): [ 0.748 0.000 0.252]
   pmf(C(0)): [ 1.000 0.000 0.000]
```

The last two lines of the output give a counterexample. That is, the computation $\sigma_{\{[0.698,0.302,0]^T,\ [0.748,0,0.252]^T,\ [1,0,0]^T\}}$ violates the specification. The first graph of Figure 6.6 shows the transitions of the availability from the counterexample. The solid line shows how the availability changes when 30% more nodes are in the *Run* state, and the dashed line shows the original availability transitions. The second graph of Figure 6.6 shows the difference between them. Note that at step 5 the difference becomes less than 10%, which violates the specification. However, as is verified by the initial specification, the difference is larger than 10% from the first step to the fourth step.

The last specification is about the performance comparisons between two systems. The formula `i → X X X j` compares the energy consumption levels of two different DTMCs *A* and *C*. The specification states that if system *A* consumes at least 5 (mA) more energy than system *B* does now, then three steps later *A* will consume at least 1.3 (mA) more energy than *B* will. The model checker proves this to be true. However, if we modify the specification as `i → X X j`, the model surprisingly does not satisfy the specification: the difference can be less than 1.3 (mA) at the second step, but it is always larger than 1.3 (mA) at the third step. The counterexample from the model checker is: $\sigma_{\{[0.735,0,0.265]^T,[1,0,0]^T,[0.935,0,0.065]^T\}}$. The first graph

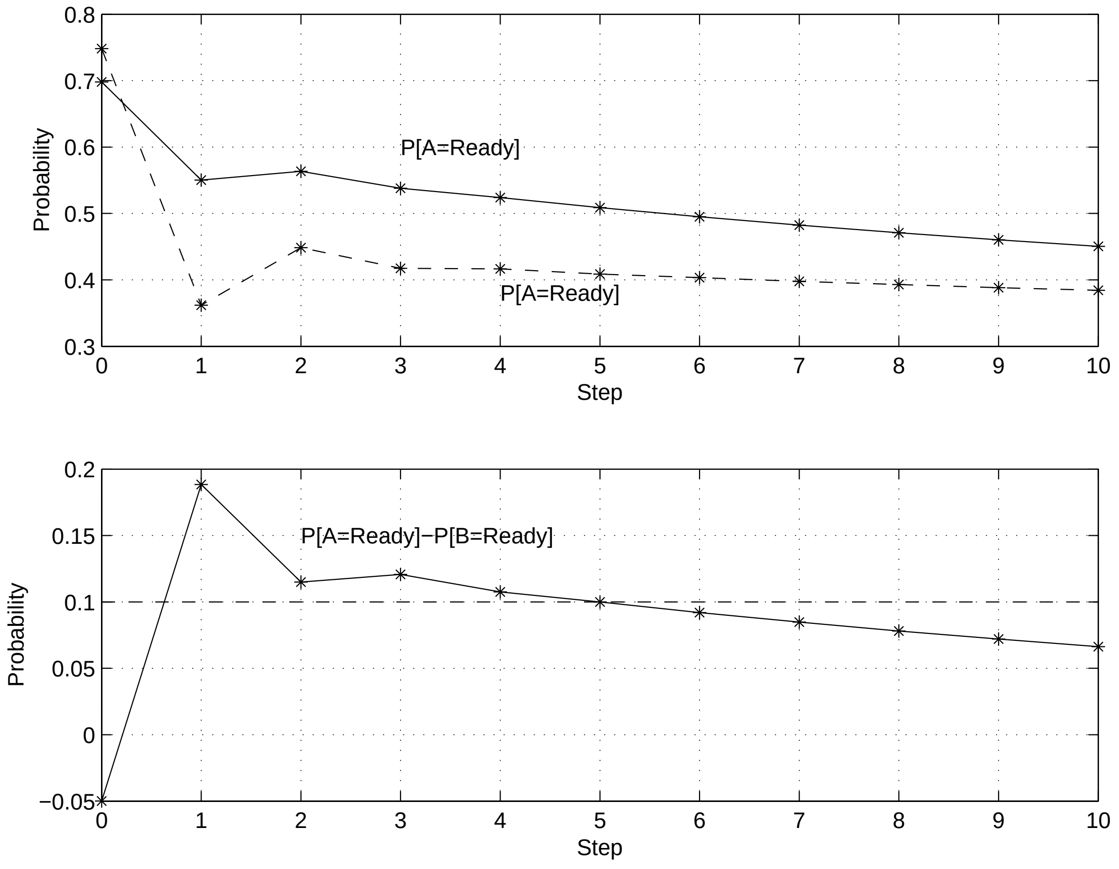


Figure 6.6: Availability transitions from two different initial conditions (top) and their differences (bottom). From step 5 the difference becomes less than 10%, which violates the modified fourth specification.

of Figure 6.7 shows how the expected energy consumption level changes from the counterexample, and the second graph shows the difference between them. At the second step of the second graph, the difference becomes smaller than or equal to 1.3 (mA) and hence violates the specification. However, at step three the difference again becomes larger than 1.3 (mA) as is checked by the original specification.

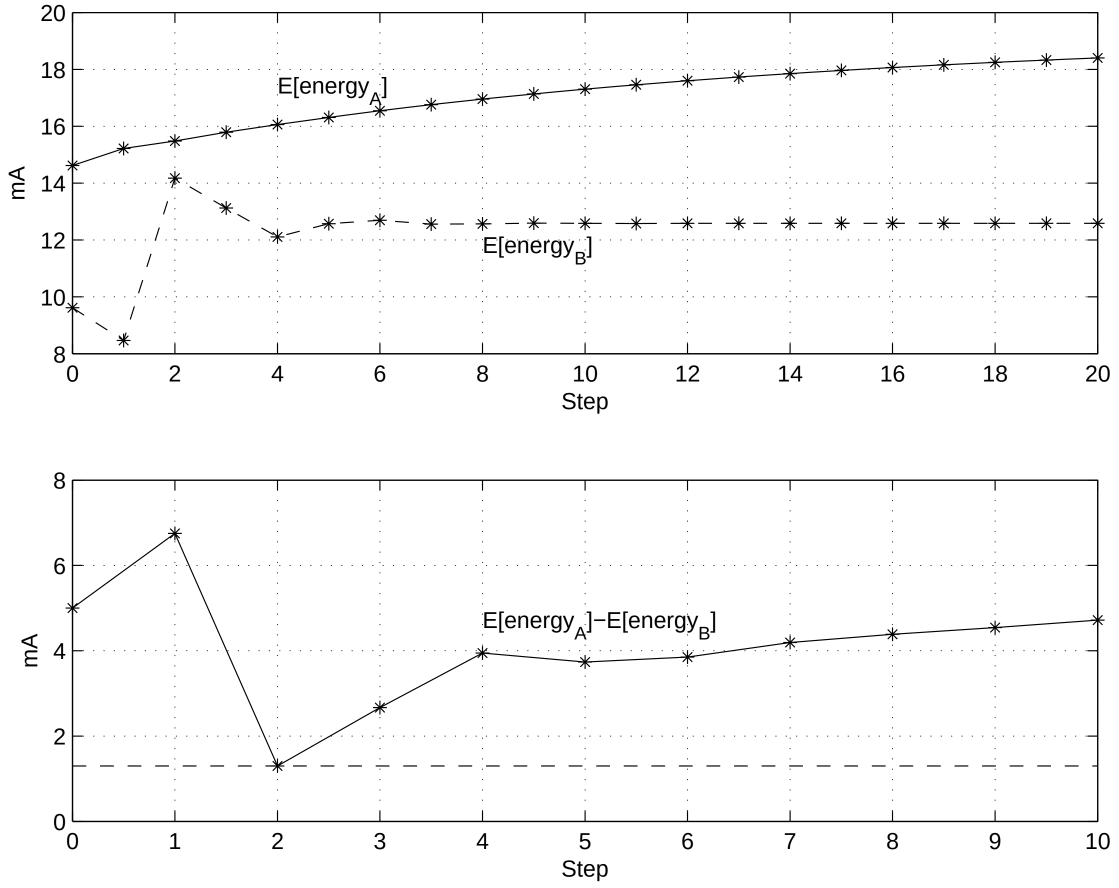


Figure 6.7: Expected energy consumption levels of two different systems (top) and their differences (bottom). The difference becomes 1.3 (mA) at step 2, which violates the modified fifth specification.

Chapter 7

Discussion

Our simulations and experimental case study suggest that model estimation based on statistical methods and performance evaluation based on model checking methods, may be useful for establishing some aggregate properties of certain large-scale sensor networks when a sensor network can be modeled as a DTMC. In fact, given that many applications on sensor networks aim to localize sensing, computation and actuation in order to reduce synchronization costs, we believe that many sensor networks will be appropriately modeled as DTMCs.

It is possible that there are more than one DTMCs present in a WSN. For example, a node's behavior for broadcasting or aggregating messages may depend on its hop distance from the center of the network. As another example, the nodes that are at the boundary of a sensor network may behave differently than the nodes that are in the center. In these case, multiple DTMCs are present in a WSN. There are two cases which arise in modeling a system with multiple DTMCS. The first case is one in which there are interactions between the DTMCS. In this case, we have to combine the individual DTMCs into a single large DTMC. The second case arises when the DTMCs are independent. This case can arise because of the structure of the WSN. Moreover, if our goal is to compare the performance of different Markov processes, we can consider these processes as independent DTMCs. In case of independent DTMCs, we can execute concurrent runs of the independent DTMCs in the model checker.

In some cases, the classification of nodes into DTMCs is obvious. For example, in the case of hop distances, we can easily map the node to the DTMC to which it belongs. In other cases, the membership may not be so clear. For example, the different DTMCs may be the result of differences in the environment in which the subsystems are laid out. We are currently working on an algorithm to classify nodes in such cases. Another problem is that the Markov process may not be stationary, i.e., the DTMCs may be constantly changing. In this case, we can reuse our goodness of fit test to determine that the model is no longer valid. It is also possible that a system swings between several modes of DTMCs. For example, a WSN in a normal mode and the system in an alert mode may have different behaviors. We are currently working on the Bayesian estimation methods for this case. However, we need ways of telling if there is a trend in the updates, and efficient ways of updating the probabilities rather than re-estimating them from scratch.

Another open problem in using our method is determining the granularity of time steps. Note that the time period corresponds to the sampling period. Although faster sampling will provide more accurate estimation, it also results in a slow varying Markov matrix and such matrices are harder to model check: if pmfs change slowly then there should be more steps before changes are detected. For purposes of our study, we chose the time steps to be about 1/4th of the possible maximum sampling rate. This was based on the need to do some computations between samples and not to allow the sampling to significantly modify the behavior of the system. However, relevant research in control theory suggests a rule of thumb which would have suggested an even lower sampling rate – the rule of thumb requires the sampling rate to be one tenth the rising time (the time the system takes to respond to input or sensing change).

One concern with our approach is that we are using best estimates of probabilities. However, any statistical method of estimating probabilities results in error bounds for the probabilities. Thus we really need to work with the infinite family of Markov matrices defined by the error in estimates of the probabilities. This problem has been addressed in [61]. While the precise complexity of this algorithm is not known, it has been shown to be at least NP-hard. Moreover, the algorithm in [61] provides model checking for a PCTL logic; checking iLTL specifications is computationally much more expensive. Given the locally continuous nature of the behaviors we are interested in, we conjecture that working with best estimates is likely to be sufficient for WSNs. However, further research will be needed to establish the limitations of our approach.

Appendix A

Lumping Independent Systems

In this chapter we show that a state space of a SAN with independent components can be lumped into any component's state space and that of a SAN with a single leader component that generates all synchronized events can be lumped into the leader's state space. By repeatedly applying these facts, we can lump a hierarchical system into its top-most leader process. With this simplification, one can reduce a million state composite system down to a hundred state system that is tractable for our probabilistic model checker.

Since our model checking logic works for DTMCs and a SAN model is given as a CTMC, we build a DTMC from the given CTMC and model check on the resulting DTMC. Let $A = (S, \mathbf{Q})$ be a CTMC and let $\mathbf{x}(t)$ be a pmf vector at time t. Then $\mathbf{x}(t) = e^{t\cdot\mathbf{Q}} \cdot \mathbf{x}(0)$. Suppose that we are sampling transitions of pmf vector with a period T. Let $\mathbf{P} = e^{T\cdot\mathbf{Q}}$, then $\mathbf{x}(n+1) = \mathbf{P} \cdot \mathbf{x}(n)$, where $\mathbf{x}(n)$ is the n^{th} sample of $\mathbf{x}(t)$. Note that the sampled process is a DTMC, $(S, \mathbf{P})$. Without loss of generality, we assume that the sampling period T is 1 unit time in this paper.

Proposition 1 *Let $(S, \mathbf{Q})$ be a CTMC with a set of states $S = \{s_1, \ldots, s_n\}$, and with a generator matrix $\mathbf{Q}$, and let $C = \{C_1, \ldots, C_k\}$ be a partition of S. Then a DTMC $(S, e^{\mathbf{Q}})$ is lumpable with respect to the partition C if $\mathbf{V} \cdot \mathbf{Q} \cdot \mathbf{U} \cdot \mathbf{V} = \mathbf{V} \cdot \mathbf{Q}$, where $\mathbf{V}$ and $\mathbf{U}$ are matrices defined in definition 2.*

Proof We first prove that $\mathbf{V} \cdot \mathbf{Q}^n \cdot \mathbf{U} \cdot \mathbf{V} = \mathbf{V} \cdot \mathbf{Q}^n$ for any non-negative integer n by induction. Let $\mathbf{V} \cdot \mathbf{Q}^i \cdot \mathbf{U} \cdot \mathbf{V} = \mathbf{V} \cdot \mathbf{Q}^i$ be the induction hypothesis, and let $\mathbf{V} \cdot \mathbf{Q}^0 \cdot \mathbf{U} \cdot \mathbf{V} = \mathbf{V} \cdot \mathbf{Q}^0$ be the base case. The base case is true because $\mathbf{V} \cdot \mathbf{U}$ is an identity matrix. If we multiply $\mathbf{Q}$ on both sides of the induction hypothesis then

$$\mathbf{V} \cdot \mathbf{Q}^{i+1} = \mathbf{V} \cdot \mathbf{Q}^i \cdot \mathbf{U} \cdot \mathbf{V} \cdot \mathbf{Q}$$

$$\begin{aligned} &= \mathbf{V} \cdot \mathbf{Q}^i \cdot \mathbf{U} \cdot \mathbf{V} \cdot \mathbf{Q} \cdot \mathbf{U} \cdot \mathbf{V} \\ &= \mathbf{V} \cdot \mathbf{Q}^i \cdot \mathbf{Q} \cdot \mathbf{U} \cdot \mathbf{V} \\ &= \mathbf{V} \cdot \mathbf{Q}^{i+1} \cdot \mathbf{U} \cdot \mathbf{V}. \end{aligned}$$

The third line is derived from the induction hypothesis. So, $\mathbf{V} \cdot \mathbf{Q}^n \cdot \mathbf{U} \cdot \mathbf{V} = \mathbf{V} \cdot \mathbf{Q}^n$ for any non-negative integer n.

Because $e^{\mathbf{Q}} = \sum_{k=0}^{\infty} \frac{\mathbf{Q}^k}{k!}$, if $\mathbf{V} \cdot \mathbf{Q} \cdot \mathbf{U} \cdot \mathbf{V} = \mathbf{V} \cdot \mathbf{Q}$ then

$$\mathbf{V} \cdot e^{\mathbf{Q}} = \mathbf{V} \cdot e^{\mathbf{Q}} \cdot \mathbf{U} \cdot \mathbf{V}.$$

Proposition 2 *Let* $(S, \mathbf{Q})$ *be a CTMC and* $\mathbf{V}, \mathbf{U}$ *be matrices as in Definition 2. If* $\mathbf{V} \cdot \mathbf{Q} \cdot \mathbf{U} \cdot \mathbf{V} = \mathbf{V} \cdot \mathbf{Q}$ *then* $\mathbf{V} \cdot e^{\mathbf{Q}} \cdot \mathbf{U} = e^{\mathbf{V} \cdot \mathbf{Q} \cdot \mathbf{U}}$.

Proof Because $(\mathbf{V} \cdot \mathbf{Q} \cdot \mathbf{U}) \cdot (\mathbf{V} \cdot \mathbf{Q}^k \cdot \mathbf{U}) = \mathbf{V} \cdot \mathbf{Q}^{k+1} \cdot \mathbf{U}$, one can easily show that $(\mathbf{V} \cdot \mathbf{Q} \cdot \mathbf{U})^k = \mathbf{V} \cdot \mathbf{Q}^k \cdot \mathbf{U}$ by induction on k. With this property, $\mathbf{V} \cdot e^{\mathbf{Q}} \cdot \mathbf{U} = \sum_{k=0}^{\infty} \mathbf{V} \cdot \frac{\mathbf{Q}^k}{k!} \cdot \mathbf{U} = \sum_{k=0}^{\infty} \frac{(\mathbf{V} \cdot \mathbf{Q} \cdot \mathbf{U})^k}{k!} = e^{\mathbf{V} \cdot \mathbf{Q} \cdot \mathbf{U}}$.

Let the matrix $\mathbf{Q}$ of Equation (2.1) be the generator matrix of a composite system of Equation (2.2). We will show that if there are no synchronized events then the composite system state space can be lumped into the state space of a single local component. If there is a single master of all synchronized events, then the composite system state space can be lumped into the state space of the master component.

Definition 3 *For simplicity, let* $n_j = |S^{(j)}|$. *Then* $\mathbf{V}$ *and* $\mathbf{U}$ *matrices as in Definition 2 corresponding to the partitions of the* i^{th} *component's state space are as follows:*

$$\begin{aligned} \mathbf{V}_i &= \bigotimes_{j=1}^{i-1} \mathbf{1}_{1\times n_j} \otimes \mathbf{I}_{n_i} \otimes \bigotimes_{j=i+1}^{J} \mathbf{1}_{1\times n_j}, \\ \mathbf{U}_i &= \bigotimes_{j=1}^{i-1} \frac{\mathbf{1}_{n_j\times 1}}{n_j} \otimes \mathbf{I}_{n_i} \otimes \bigotimes_{j=i+1}^{J} \frac{\mathbf{1}_{n_j\times 1}}{n_j}. \end{aligned}$$

Proposition 3 $e^{\mathbf{Q}}$ *is lumpable with respect to the partition associated with* $\mathbf{V}_i$ *for any* $i = 1, \ldots, J$ *if there are no synchronized events.*

Proof Because of Proposition 1, we need only show that

$$\mathbf{V}_i \cdot \mathbf{Q} \cdot \mathbf{U}_i \cdot \mathbf{V}_i \quad = \quad \mathbf{V}_i \cdot \mathbf{Q}.$$

Since there are no synchronized events, $\mathbf{E}_k^{(j)}$ and $\mathbf{D}_k^{(j)}$ are zero matrices. We check the lumpability by checking the lumpability of sub-terms of the following summation:

$$\bigoplus_{j=1}^{J} \mathbf{A}_j \quad = \quad \sum_{j=1}^{J} \mathbf{I}_{n_1} \otimes \cdots \otimes \mathbf{A}_j \otimes \cdots \otimes \mathbf{I}_{n_J}.$$

case 1: if $i \neq j$

$$\begin{aligned}
&\mathbf{V}_i \cdot \left(\bigotimes_{k=1}^{j-1} \mathbf{I}_{n_k} \otimes \mathbf{Q}^{(j)} \otimes \bigotimes_{k=j+1}^{J} \mathbf{I}_{n_k} \right) \\
&\quad = \left(\bigotimes_{k=1}^{i-1} \mathbf{1}_{1\times n_k} \otimes \mathbf{I}_{n_i} \otimes \bigotimes_{k=i+1}^{J} \mathbf{1}_{1\times n_k} \right) \cdot \left(\bigotimes_{k=1}^{j-1} \mathbf{I}_{n_k} \otimes \mathbf{Q}^{(j)} \otimes \bigotimes_{k=j+1}^{J} \mathbf{I}_{n_k} \right) \\
&\quad = \left(\bigotimes_{k=1}^{j-1} \mathbf{1}_{1\times n_k} \cdot \mathbf{I}_{n_k} \right) \otimes \mathbf{I}_{n_i} \otimes \left(\bigotimes_{k=i+1}^{j-1} \mathbf{1}_{1\times n_k} \cdot \mathbf{I}_{n_k} \right) \otimes \left(\mathbf{1}_{1\times n_j} \cdot \mathbf{Q}^{(j)} \right) \otimes \left(\bigotimes_{k=j+1}^{J} \mathbf{1}_{1\times n_k} \cdot \mathbf{I}_{n_k} \right) \\
&\quad = \mathbf{0}.
\end{aligned}$$

The last equality is derived because the column sums of the generator matrix $\mathbf{Q}^{(j)}$ are zero. Since $\mathbf{V} \cdot \mathbf{Q}^{(j)} = \mathbf{0}$, the equality holds.

case 2: if $i = j$

The right hand side of the equality is:

$$\begin{aligned}
&\mathbf{V}_i \cdot \left(\bigotimes_{k=1}^{i-1} \mathbf{I}_{n_k} \otimes \mathbf{Q}^{(i)} \otimes \bigotimes_{k=i+1}^{J} \mathbf{I}_{n_k} \right) \\
&\quad = \left(\bigotimes_{k=1}^{i-1} \mathbf{1}_{1\times n_k} \otimes \mathbf{I}_{n_i} \otimes \bigotimes_{k=i+1}^{J} \mathbf{1}_{1\times n_k} \right) \cdot \left(\bigotimes_{k=1}^{i-1} \mathbf{I}_{n_k} \otimes \mathbf{Q}^{(i)} \otimes \bigotimes_{k=i+1}^{J} \mathbf{I}_{n_k} \right) \\
&\quad = \bigotimes_{k=1}^{i-1} \mathbf{1}_{1\times n_k} \cdot \mathbf{I}_{n_k} \otimes \mathbf{Q}^{(i)} \otimes \bigotimes_{k=i+1}^{J} \mathbf{1}_{1\times n_k} \cdot \mathbf{I}_{n_k} \\
&\quad = \bigotimes_{k=1}^{i-1} \mathbf{1}_{1\times n_k} \otimes \mathbf{Q}^{(i)} \otimes \bigotimes_{k=i+1}^{J} \mathbf{1}_{1\times n_k}.
\end{aligned}$$

The left hand side of the equality is:

$$\begin{aligned}
&\left(\bigotimes_{k=1}^{i-1} \mathbf{1}_{1\times n_k} \otimes \mathbf{Q}^{(i)} \otimes \bigotimes_{k=i+1}^{J} \mathbf{1}_{1\times n_k}\right) \cdot \mathbf{U}_i \cdot \mathbf{V}_i \\
&= \left(\bigotimes_{k=1}^{i-1} \mathbf{1}_{1\times n_k} \otimes \mathbf{Q}^{(i)} \otimes \bigotimes_{k=i+1}^{J} \mathbf{1}_{1\times n_k}\right) \cdot \left(\bigotimes_{k=1}^{i-1} \frac{\mathbf{1}_{n_k\times 1}}{n_k} \otimes \mathbf{I}_{n_i} \otimes \bigotimes_{k=i+1}^{J} \frac{\mathbf{1}_{n_k\times 1}}{n_k}\right) \cdot \mathbf{V}_i \\
&= \left(\bigotimes_{k=1}^{i-1} 1 \otimes \mathbf{Q}^{(i)} \otimes \bigotimes_{k=i+1}^{J} 1\right) \cdot \mathbf{V}_i \\
&= \left(\bigotimes_{k=1}^{i-1} 1 \otimes \mathbf{Q}^{(i)} \otimes \bigotimes_{k=i+1}^{J} 1\right) \cdot \left(\bigotimes_{k=1}^{i-1} \mathbf{1}_{1\times n_k} \otimes \mathbf{I}_{n_i} \otimes \bigotimes_{k=i+1}^{J} \mathbf{1}_{1\times n_k}\right) \\
&= \bigotimes_{k=1}^{i-1} \mathbf{1}_{1\times n_k} \otimes \mathbf{Q}^{(i)} \otimes \bigotimes_{k=i+1}^{J} \mathbf{1}_{1\times n_k}.
\end{aligned}$$

Because of Proposition 2, the probability transition matrix for the lumped state with respect to the partition associated with $\mathbf{V}_i$ is $\mathbf{P} = e^{\hat{\mathbf{Q}}}$, where

$$\hat{\mathbf{Q}} = \mathbf{V}_i \cdot \left(\bigoplus_{j=1}^{J} \mathbf{Q}^{(j)}\right) \cdot \mathbf{U}_i = \mathbf{Q}^{(i)}.$$

Proposition 4 $e^{\mathbf{Q}}$ *is lumpable with respect to the partition associated with* $\mathbf{V}_m$ *if the following conditions hold:*

1. $A^{(m)}$ *is a single master of all synchronized events, and*
2. *for every component, if it is a slave of a synchronized event then it has transitions from every state in response to that event.*

Proof Again, because of Proposition 1, we need only show that

$$\mathbf{V}_m \cdot \mathbf{Q} \cdot \mathbf{U}_m \cdot \mathbf{V}_m = \mathbf{V}_m \cdot \mathbf{Q}.$$

From the proof of Theorem 3, equality holds for the Kronecker summation term. What we need to show is that equality holds for the summation of Kronecker product terms of Equation (2.1). Again, we check this for each sub-term of the summation.

The right hand side of the equality is

$$\begin{aligned}
&\mathbf{V}_m \cdot \bigotimes_{j=1}^{J} \mathbf{E}_k^{(j)} \\
&= \left(\bigotimes_{j=1}^{m-1} \mathbf{1}_{1\times n_j} \otimes \mathbf{I}_{n_m} \otimes \bigotimes_{j=m+1}^{J} \mathbf{1}_{1\times n_j} \right) \cdot \bigotimes_{j=1}^{J} \mathbf{E}_k^{(j)} \\
&= \bigotimes_{j=1}^{m-1} \mathbf{1}_{1\times n_j} \otimes \mathbf{E}_k^{(m)} \otimes \bigotimes_{j=m+1}^{J} \mathbf{1}_{1\times n_j}.
\end{aligned}$$

The last equality is derived because of the second condition: the column sums of $\mathbf{E}_k^{(j)}$ are one if $j \neq m$.

The left hand side of the equality is

$$\begin{aligned}
&\left(\bigotimes_{j=1}^{m-1} \mathbf{1}_{1\times n_j} \otimes \mathbf{E}_k^{(m)} \otimes \bigotimes_{j=m+1}^{J} \mathbf{1}_{1\times n_j} \right) \cdot \mathbf{U}_m \cdot \mathbf{V}_m \\
&= \left(\bigotimes_{j=1}^{m-1} 1 \otimes \mathbf{E}_k^{(m)} \otimes \bigotimes_{j=m+1}^{J} 1 \right) \cdot \mathbf{V}_m \\
&= \bigotimes_{j=1}^{m-1} \mathbf{1}_{1\times n_j} \otimes \mathbf{E}_k^{(m)} \otimes \bigotimes_{j=m+1}^{J} \mathbf{1}_{1\times n_j}.
\end{aligned}$$

$\mathbf{V}_m \cdot \bigotimes_{j=1}^{J} \mathbf{D}_k^{(j)} = \mathbf{V}_m \cdot \left(\bigotimes_{j=1}^{J} \mathbf{D}_j \right) \cdot \mathbf{U}_m \cdot \mathbf{V}_m$ can be shown similarly.

The probability transition matrix for the lumped state with respect to the partition associated with $\mathbf{V}_m$ is $\mathbf{P} = e^{\hat{\mathbf{Q}}}$, where $\hat{\mathbf{Q}}$ is

$$\begin{aligned}
&\mathbf{V}_m \cdot \left(\bigoplus_{j=1}^{J} \mathbf{Q}^{(j)} + \sum_{k=1}^{K} \lambda_k \cdot \left(\bigotimes_{j=1}^{J} \mathbf{E}_k^{(j)} - \bigotimes_{j=1}^{J} \mathbf{D}_k^{(j)} \right) \right) \cdot \mathbf{U}_m \\
&= \mathbf{Q}^{(m)} + \sum_{k=1}^{K} \lambda_k \cdot \left(\mathbf{E}_k^{(m)} - \mathbf{D}_k^{(m)} \right).
\end{aligned}$$

Suppose that a hierarchical system is built in such a way that the first-level leaders partition the nodes and coordinate them independently. For the first-level leaders, there are the second-level leaders that partition them and coordinate them independently and so on. For these systems by recursively applying Proposition 3 (think of leaders as independent systems) and Proposition 4 (higher level single leader coordinate systems), we can lump the state space of the whole system into that of the top most leader's.

Appendix B

iLTLChecker Syntax

```
program
    : opt-var-def-list
        model : dtmc-decl-list
        specification : inequality-list iltl

dtmc-decl-list
    : dtmc-decl
    | decl-list : dtmc-decl

dtmc-decl
    : markov chain ID
        has states : { id-list } ,
        transits by : matrix

id-list
    : ID
    | id-list , ID

matrix
    : [ num-list-list ]

num-list-list
    : num-list
    | num-list-list ; num-list

num-list
    : add-expr
    | num-list , add-expr

inequality-list
    : inequality
    | inequality-list , inequality
```

```
inequality
    : ID : term-list comp term-list

comp
    : =
    | ~=
    | <
    | <=
    | >
    | >=

term-list
    : term
    | term-list + term
    | term-list - term

term
    : mul-expr
    | probability
    | mul-expr * probability
    | accumulation
    | mul-expr * accumulation

accumulation
    : Q [ ID opt-time = ID ]

probability
    : P [ ID opt-time = ID ]

opt-time
    :
    | ( add-expr )

iltl
    : binary

binary
    : unary
    | binary /\ unary
    | binary \/ unary
    | binary -> unary
    | binary U  unary
    | binary R  unary

unary
    : atomic
    | ˜ unary
    | X unary
    | [] unary
    | <> unary

atomic
    : T
    | F
    | ID
    | ( iltl )

opt-var-def-list
    :
    | var : var-def-list

var-def-list
    : var-def
    | var-def-list , var-def

var-def
    : ID = add-expr

add-expr
    : mul-expr
    | add-expr + mul-expr
    | add-expr - mul-expr

mul-expr
    : uni-expr
    | mul-expr * uni-expr
    | mul-expr / uni-expr

uni-expr
    : term-expr
    | - term-expr
    | + term-expr

term-expr
    : NUM
    | ID
    | ( add-expr )
```

Appendix C

Notations

The notations used in this thesis are as follows

- $\mathbb{R}, \mathbb{N}, \mathbb{C}$: The set of real numbers, the set of natural numbers, and the set of complex numbers
- $\mathbb{R}^n$: The set of n dimensional real vectors.
- $\mathbf{M}$: A Markov transition matrix.
- ρ: A reward function of states.
- $\mathbf{Q}$: A generator matrix of CTMC.
- $e^{\mathbf{Q}}$: A matrix exponential of $\mathbf{Q}$. That is, $e^{\mathbf{Q}} = \sum_{n=0}^{\infty} \frac{\mathbf{Q}^n}{n!}$.
- $\mathrm{Q}[X(t) = s]$: An accumulated probability such that $\mathrm{Q}[X(t) = s] = \sum_{\tau=t}^{\infty} \mathrm{P}[X(\tau) = s]$.
- $\mathbf{x}$: A probability mass function in a vector form.
- $\mathbf{Z} \cdot \mathbf{\Lambda} \cdot \mathbf{Z}^{-1}$: A diagonalization of a Markov matrix $\mathbf{M}$.
- λ_i: The i^{th} eigenvalue of $\mathbf{M}$. We assume that $\lambda_1 = 1$.
- $\mathbf{z}_i$: The i^{th} eigenvector of $\mathbf{M}$.
- $\mathcal{X}$: A multiple DTMC model which is a set of independent DTMCs. That is, $\mathcal{X} = \bigcup_{j=1}^{J} X^{(j)}$.
- $\mathbf{M}^{(j)}$: The j^{th} Markov matrix in a multiple DTMC model.
- $\mathbf{x}^{(j)}$: The j^{th} pmf in a multiple DTMC model.

- π: A infinite path $s_0 s_1 \cdots \in S^\omega$ such that $\pi(t) = s_t$, where S is a set of states of a DTMC.

- $\sigma_{\mathbf{x}}$: A computational path of a DTMC such that $\sigma_{\mathbf{x}}(t) = \mathbf{M}^t \cdot \mathbf{x}$.

- $\sigma_{\cup_j \mathbf{x}^{(j)}}$: A computational path of a multiple DTMC such that $\sigma_{\cup_j \mathbf{x}^{(j)}}(t) = \bigcup_{j=1}^{J} \mathbf{M}^{(j)^t} \cdot \mathbf{x}^{(j)}$.

- AP: A set of atomic propositions.

- L: A labeling function. $L(\sigma_{\mathbf{x}})(t)$ is a set of atomic propositions that is true in time t in the computational path $\sigma_{\mathbf{x}}$.

- $\otimes$: Kronecker product operator.

- $\oplus$: Kronecker sum operator.

- ϕ, ψ: iLTL formula.

- $\mathcal{A}_\phi$: A Büchi automaton corresponding to ϕ.

- $\mathcal{L}_\phi$: A set of strings accepted by $\mathcal{A}_\phi$.

References

[1] Gul A. Agha, Ian A. Mason, Scott F. Smith, and Carolyn L. Talcott. A foundation for actor computation. In *Journal of Functional Programming*, volume 7, pages 1–72. Cambridge University Press, 1997.

[2] Javed Aslam, Zack Butler, Florin Constantin, Valentino Crepsi, George Cybenko, and Daniela Rus. Tracking a moving object with a binary sensor network. In *Sensys*, 2003.

[3] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Verifying continuous time markov chains. In *CAV*, pages 269–276. LNCS 1102, 1996.

[4] Adnan Aziz, Vigyan Singhal, and Felice Balarin. It usually works: The temporal logic of stochastic systems. In *CAV*, pages 155–165. LNCS 939, 1995.

[5] C. Baier, J.P. Katoen, and H. Hermanns. Approximate symbolic model checking of continuous-time markov chains. In *CONCUR*, pages 146–162. LNCS 1664, 1999.

[6] Christel Baier, Boudewijn R. Haverkort, Holger Hermanns, and Joost-Pieter Katoen. Model checking meets performance evaluation. *SIGMETRICS Perform. Eval. Rev.*, 32(4):10–15, 2005.

[7] Christel Baier, Joost-Pieter Katoen, Holger Hermanns, and Boudewijn Haverkort. Simulation for continuous-time markov chains. pages 338–354. LNCS 2421, 2002.

[8] Andrea Bianco and Luca de Alfaro. Model checking of probabilistic and nondeterministic systems. In *Proceedings of Conferenco on Foundations of Software Technology and Theoretical Computer Science*, volume 1026, pages 499–513, 1995.

[9] Gabriel Bracha and Sam Toueg. Asynchronous consensus and broadcast protocols. *J. ACM*, 32(4):824–840, 1985.

[10] Peter Buchholz, Joost-Pieter Katoen, Peter Kemper, and Carsten Tepper. Model-checking large structured markov chains. In *Journal of Logic and Algebraic Programming*, volume 56, pages 69–97, 2003.

[11] J.R. Büchi. On a decision method in restricted second order arthmetic. In *Proc. of the Int. Conf. on Logic, Methodology and Philosophy of Science*, pages 1–11. Stanford University Press, 1960.

[12] Peter Bucholz. Exact and ordinary lumpability in finite markov chains. In *Journal of Applied Probability*, volume 31, pages 59–74, 1994.

[13] Rafael C. Carrasco and Jose Oncina. Learning stochastic regular grammars by means of a state merging method. In *Proceedings of the Second International Colloquium on Grammatical Inference and Applications*, pages 139–152. Springer-Verlag, 1994.

[14] R.C. Cheung. A user-oriented software reliability model. In *IEEE Transactions on Software Engineering*, volume SE-6, pages 118–125, March 1980.

[15] Gianfranco Ciardo, Raymond A. Marie, Bruno Sericola, and Kishor S. Trivedi. Performability analysis using semi-markov reward process. In *IEEE Transactions on Computers*, volume 39, pages 1251–1264, October 1990.

[16] Gianfranco Ciardo and Andrew S. Miner. A data structure for the efficient kronecker solution of gspns. In *International Workshop Petri Nets and Performance Models*, pages 22–31, 1999.

[17] Gianfranco Ciardo and Andrew S. Miner. Smart: The stochastic model checking analyzer for reliability and timing. In *International Conference on Quantitative Evaluation of Systems (QEST)*, pages 338–339. IEEE Computer Society, 2004.

[18] Edmund Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. MIT Press, 2000.

[19] E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Logic of programs*. LNCS 131, Springer-Verlag, 1981.

[20] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logics specification: A practical approach. In *Proc. 10th Int. ACM Symposium on Principles of Programming Languages*, pages 117–126, 1983.

[21] Crossbow Technology, Inc. http://www.xbow.com.

[22] David Gay, Phil Levis, Rob von Behren, Matt Welsh, Eric Brewer, and David Culler. The nesc language: A holistic approach to networked embedded systems. In *In Proceedings of Programming Language Design and Implementation (PLDI) 2003*, June 2003.

[23] R. Gerth, D. Peled, M.Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *IFIP/WG*, volume 6.1, pages 3–18, 1995.

[24] A.L. Goel and K. Okumoto. Time-dependent error-detection rate models for software reliability and other performance measures. In *IEEE Transactions on Reliability*, volume R-28, pages 206–211, August 1979.

[25] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. In *Formal Aspects of Computing*, volume 6, pages 512–535, 1994.

[26] Gerard J. Holzmann. The model checker spin. In *IEEE Transactions on Software Engineering*, volume 23, pages 279–295, May 1997.

[27] G.E. Hughes and M.J. Creswell. *Introduction to Modal Logic*. Methuen, 1997.

[28] Philo Juang, Hidekazu Oki, Yong Wang, Margaret Martonosi, Li Shiuan Peh, and Daniel Rubenstein. Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with zebranet. volume 36, pages 96–107. ACM Press, 2002.

[29] Julian Keilson. *Markov Chain Models-Rarity and Exponentiality*. Springer-Verlag, 1979.

[30] John G. Kemeny and J. Laurie Snell. *Finite Markov Chains*. Springer-Verlag, 1976.

[31] Christopher Kermorvant and Pierre Dupont. Stochastic grammatical inference with multinomial tests. In *Proceedings of the 6th International Colloquium on Grammatical Inference*, pages 149–160. Springer-Verlag, 2002.

[32] Marta Kwiatkowska, Gethin Norman, and David Parker. Prism 2.0: A tool for probabilistic model checking. In *International Conference on Quantitative Evaluation of Systems (QEST)*, pages 322–323. IEEE Computer Society, 2004.

[33] YoungMin Kwon and Gul Agha. Linear inequality LTL (iLTL): A model checker for discrete time markov chains. In *International Conference on Formal Engineering Methods*, pages 194–208. LNCS 3308, 2004.

[34] YoungMin Kwon and Gul Agha. iLTLChecker: A probabilistic model checker for multiple DTMCs. In *International Conference on the Quantitative Evaluation of Systems (QEST)*. IEEE Computer Society, 2005.

[35] YoungMin Kwon and Gul Agha. Scalable modeling and performance evaluation of wireless sensor networks. In *Real-Time and Embedded Technology and Applications Symposium (to appear)*. IEEE, 2006.

[36] YoungMin Kwon, Kirill Mechitov, Sameer Sundresh, WooYoung Kim, and Gul Agha. Resilient localization for sensor networks in outdoor environments. In *International Conference on Distributed Computing Systems*, pages 643–652. IEEE Computer Society, 2005.

[37] Vinh V. Lam, Peter Buchholz, and William H. Sanders. A structured path-based approach for computing transient rewards for large ctmcs. In *International Conference on Quantitative Evaluation of Systems (QEST)*, pages 136–145. IEEE Computer Society, 2004.

[38] Amy N. Langville and William J. Stewart. The kronecker product and stochastic automata networks. In *Journal of Computational and Applied Mathematics*, pages 429–447. Elsevier B.V., 2004.

[39] David C. Lay. *Linear Algebra and Its Applications*. Addison Wesley, 1994.

[40] Wen-Hwa Liao, Jang-Ping Sheu, and Yu-Chee Tseng. GRID: A fully location-aware routing protocol for mobile ad hoc networks. *Telecommunication Systems*, 18(1-3):37–60, 2001.

[41] Orna Lichtenstein and Amir Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Proc of 12th ACM Symposium on Principles of Programming Languages*, pages 97–107, 1985.

[42] Gabriel G. Infante Lopez, Holger Hermanns, and Joost-Pieter Katoen. Beyond memoryless distributions: Model checking semi-markov chains. In *Proceedings of the Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification*, pages 57–70. Springer-Verlag, 2001.

[43] David G. Luenberger. *Linear and Nonlinear Programming*. Addison Wesley, 2nd edition, 1989.

[44] Miklos Maroti, Gyula Simon, Akos Ledeczi, and Janos Sztipanovits. Shooter localization in urban terrain. In *Computer*, pages 60–61. IEEE Computer Society Press, 2004.

[45] Kirill Mechitov, Sameer Sundresh, YoungMin Kwon, and Gul Agha. Cooperative tracking with binary-detection sensor networks. In *International Conference on Embedded Networked Sensor Systems (Sensys)*, pages 332–333. ACM Press, 2003.

[46] Kirill Mechitov, Sameer Sundresh, YoungMin Kwon, and Gul Agha. High-frequency distributed sensing for structure monitoring. In *International Workshop on Networked Sensing Systems*, 2004.

[47] Andrew S. Miner and Gianfranco Ciardo. Efficient reachability set generation and storage using decision diagrams. In *International Conference Application and Theory of Petri Nets*, pages 6–25. LNCS 1639, Springer-Verlag, 1999.

[48] Dragos Niculescu and Badri Nath. Ad hoc positioning system (APS). In *GLOBECOM*, November 2001.

[49] J.R. Norris. *Markov Chains*. Cambridge University Press, 1997.

[50] Alan V. Oppenheim and Alan S. Willsky. *Signals and Systems*. Prentice Hall, 1983.

[51] Athanasios Papoulis. *Probability, Random Variables, and Stochastic Processes*. McGraw-Hill, 3rd edition, 1991.

[52] Hoang Pham. *Software Reliability*. Springer, 2000.

[53] Brigitte Plateau and Karim Atif. Stochastic automata network for modeling parallel systems. In *IEEE Transactions on Software Engineering*, volume 17, pages 1093–1108, 1991.

[54] Jayant Rajgopal and Mainak Mazumdar. Modular operational test plans for inference on software reliability based on a markov model. In *IEEE Transactions on Software Engineering*, volume 28, pages 358–363, April 2002.

[55] Dana Ron, Yoram Singer, and Naftali Tishby. On the learnability and usage of acyclic probabilistic finite automata. In *COLT '95: Proceedings of the eighth annual conference on Computational learning theory*, pages 31–40. ACM Press, 1995.

[56] Stuart Russell and Peter Norvig. *Artificial Intelligence A Modern Approach*. Prentice Hall, 1995.

[57] Janos Sallai, Gyorgy Balogh, Miklos Maroti, and Akos Ledeczi. Acoustic ranging in resource constrained sensor networks. In *Technical Report ISIS-04-504*. Vanderbilt University.

[58] Robert Sedgewick. *Algorithms in C*. Addison Wesley, 1990.

[59] Koushik Sen, Mahesh Viswanathan, and Gul Agha. Learning continuous time markov chains from sample executions. In *International Conference on Quantitative Evaluation of Systems (QEST)*, pages 146–155. IEEE, 2004.

[60] Koushik Sen, Mahesh Viswanathan, and Gul Agha. Vesta: A statistical model checker and analyzer for probabilistic systems. In *International Conference on Quantitative Evaluation of Systems (QEST)*. IEEE Computer Society, 2005.

[61] Koushik Sen, Mahesh Viswanathan, and Gul Agha. Model-checking markov chains in the presence of uncertainties. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Lecture Notes in Computer Science, Springer-Verlag, 2006. (To Appear), 2006.

[62] Yi Shang and Wheeler Ruml. Improved MDS-based localization. In *INFOCOM*, volume 7, pages 2460–2651. IEEE, 2004.

[63] Yi Shang, Wheeler Ruml, and Ying Zhang. Localization from mere connectivity. In *MobiHoc*, pages 201–212, June 2003.

[64] Henry Start and John W. Woods. *Probability and Random Processes with Applications to Signal Processing*. Prentice-Hall, 3rd edition, 2002.

[65] Gilbert Strang. *Linear Algebra and Its Applications*. Harcourt Brace Jovanovich, 3rd edition, 1988.

[66] Robert Szewczyk, Alan Mainwaring, Joseph Polastre, John Anderson, and David Culler. An analysis of a large scale habitat monitoring application. In *Sensys*, 2004.

[67] Andrew S. Tanenbaum. *Computer Networks*. Prentice Hall, 4th edition, 2003.

[68] Gerard Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, 2nd edition, 2001.

[69] Wolfgang Thomas. Automata on infinite objects. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 133–192. 1990.

[70] TinyOS. http://www.tinyos.net.

[71] Moshe Y. Vardi. Probabilistic linear-time model checking: an overview of the automata-theoretic approach. In *Proc. 5th Int. AMAST Workshop Formal Methods for Real-Time and Probabilistic Systems*, volume 1601, May 1999.

[72] Ning Xu, Sumit Rangwala, Krishna Kant Chintalapudi, Deepak Ganesan, Alan Broad, Ramesh Govindan, and Deborah Estrin. A wireless sensor-network for structural monitoring. In *Sensys*, 2004.

[73] Sherif M. Yacoub, Bojan Cukic, and Hany H. Ammar. Scenario-based reliability analysis of component-based software. In *Proceedings of the 10th International Symposium on Software Reliability Engineering (ISSRE)*, pages 22–31. IEEE Computer Society, 1999.

Author’s Biography

YoungMin Kwon received his B.E. degree in Electrical Engineering from Korea University in 1996. He received his M.E. degree in Mechatronics from Korea University in 1998 under the guidance of Professor Tae-Woong Yoon. During his studies, his research focus was on developing a controller for a missile autopilot. He then worked at Hyundai Electronics, and Damdammicro Systems as a system engineer. In 2001, he entered University of Illinois at Urbana Champaign for graduate studies in Computer Science under the guidance of Professor Gul Agha. During his studies, he developed a mobile agent platform called ActorNet for Wireless Sensor Networks (WSNs), and a probabilistic temporal logic called iLTL for evaluating performances of large scale systems such as WSNs.