RELIABILITY ANALYSIS OF NON-DETERMINISTIC SYSTEMS

LIN GUI

NATIONAL UNIVERSITY OF SINGAPORE

2014

RELIABILITY ANALYSIS OF NON-DETERMINISTIC SYSTEMS

LIN GUI

(B.Eng.(Hons.), Nanyang Technological University, Singapore, 2010)

A THESIS SUBMITTED FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

NUS GRADUATE SCHOOL FOR INTEGRATIVE SCIENCES AND ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

2014

Declaration

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

[illegible]

Lin Gui
01 August 2014

Acknowledgements

I am deeply indebted to my supervisor, Dr. Dong Jin Song. Without his encouragement, understanding and persistent guidance, this dissertation would not have been possible. He is a considerate advisor who always puts students' supervision and welfare as a top priority.

I would like to thanks my thesis advisory committee: Dr. P. S. Thiagarajan and Dr. Sun Jun for their involvement and constructive comments on my research. I have special thanks to Dr. Sun Jun, who acts like my co-supervisor and gives me valuable instructions and friendly assistance during my whole Ph.D. journey. I also am grateful to my mentor Dr. Liu Yang for numerous helpful advice and inspiring discussions.

There are also friends in SE lab who share my joy and pain, and make my graduate study a colorful and enriching journey. I would like to acknowledge my seniors: Dr. Chen Chunqing, Dr. Zhang Shaojie, Dr. Zheng Manchun, Dr. Song Songzheng, Dr. Tan Tian Huat, Liu Yan, Shi Ling; and fellow students: Khanh, Liu Shuang, Bai Guang Dong, Li Li, Chen Manman.

This study is supported by the scholarship from NUS National Graduate School. The School of Computing has also provided excellent research facilities. Additionally, I have been encouraged by receiving the Research Achievement Award 2013. For all of these, I am very grateful.

I would like to show my deepest gratitude and love to my parents. A special thank to my mother Youping, a strong and cheerful lady, who never puts any pressure on me and always encourages me. I would also like to thank to my husband, Dr. Zhao Bing, for lighting up my life. I will never forget the countless weekends he companied me in the lab and the every moment he cheered me up.

Contents

Summary

Many industries are highly dependent on computers for their automated functioning. With higher dependencies on software, the possibilities of crises due to system failures also increase. System failures would potentially lead to significant losses in capital or even human lives. To prevent those losses, assessing the system reliability before its deployment is highly desirable.

Concurrent or distributed systems like cloud-based services are ever more popular these days. Assessing the reliability of such systems is highly non-trivial. Particularly, the order of executions among different components adds a dimension of non-determinism, which invalidates existing reliability analysis methods based on Markov chains. Moreover, reliability analysis of such non-deterministic systems is also challenged by the state explosion issue. This thesis proposes to analyze the reliabilities of non-deterministic systems via probabilistic model checking on Markov decision processes (MDPs), a well-known automatic verification technique dealing with both probabilistic and non-deterministic behaviors. On top of that, various techniques (e.g., statistical, numerical and graphical methods) are incorporated to enhance the scalability and efficiency of reliability analysis. The Ph.D. work is summarized into the following three aspects.

First, to support the reliability analysis of non-deterministic systems, we propose a method combining hypothesis testing and probabilistic model checking. The idea is to apply hypothesis testing to deterministic system components and use probabilistic model checking techniques to lift the results through non-determinism. Furthermore, if a requirement on the system level reliability is given, we apply probabilistic model checking techniques to push down the requirement through non-determinism to individual components so that they can be verified using hypothesis testing. Based on the proposed framework, a toolkit RaPiD has been developed to support automated software reliability analysis including reliability prediction, reliability distribution and sensitivity analysis. Case studies have been carried out on real world systems including a stock trading system, a therapy control system and an ambient assisted living system.

The second part is on improving the efficiency of the proposed approach, in particular, the fundamental part that calculates the probability of reaching certain system states (i.e., reachability analysis). It is known that existing approaches on reachability analysis for Markov models are often inefficient when a given model contains a large number of states and loops. In this work, we propose a divide-and-conquer algorithm to eliminate strongly connected components (SCCs), and actively remove redundant probability distributions based on convex property. With the removal of loops and part of probability distributions, the reachability analysis can be accelerated as evidenced by our experimental results.

Last but not the least, the scalability of the proposed approach has been improved, in particular, for distributed systems. Traditional probabilistic model checking is limited to small scale distributed systems as it works by exhaustively exploring the global state space, which is a product of the state spaces of all components and often huge. In this work, we improve the probabilistic model checking through a method of abstraction and reduction, which controls the communications among different system components and actively reduces the size of each system component. We formally prove the soundness and completeness of the proposed approach. Through the evaluations with several systems, we show that our approach often reduces the size of the state space by several orders of magnitude while still producing sound and accurate assessment.

Key words: **Reliability Analysis, Non-determinism, Markov Decision Process, Probabilistic Model Checking, Hypothesis Testing**

List of Tables

List of Figures

Chapter 1

Introduction and Overview

1.1 Motivation and Goals

1.1.1 Reliability Analysis

Nowadays, software becomes unprecedented popular and permeates everywhere in our daily life, from wristwatches, mobile phones to automobiles and aircraft. Virtually any industry, e.g., automotive, avionics, oil, semiconductors, pharmaceuticals, telecommunications and banking, is highly dependent on the computers for their automated functioning. With higher dependencies on software, the possibility of crises due to computer failures also increases. Failures would damage the reputation of the system operators, and potentially lead to losses in capitals or even human lives. The probability of failure-free software operation within a specific period and environment is referred to as reliability [81]. To prevent those losses due to software failures, it is desirable to have the reliability of the system analyzed before its deployment.

Existing approaches on reliability analysis problems fall into two categories: black-box approaches [70, 132] and white-box approaches [26, 81, 77, 70]. The black-box approaches treat a system as a monolith and evaluate its reliability using testing techniques. They use the observed failure information to predict the reliability of software based on several mathematical models. On the contrary, the white-box approaches assume reliability of system components are known and evaluate software reliability analytically based on the model of the system architecture. Typical reliability models include discrete time Markov chains (DTMCs) [26], continuous time Markov chains (CTMCs) [81], or semi-Markov processes (SMPs) [77]. Those approaches assume the system is deterministic, i.e., given the same inputs, the outputs of the system are always the same. Thus, in the corresponding reliability model, the probabilities of transitions among components are assumed to be known. For instance, the transition probability is assumed to be a constant in DTMC-based approaches or a function of time in CTMC/SMP-based approaches. All those approaches assume that there is a unique probability distribution for the possible usages of a component.

1.1.2 Non-deterministic Systems

As software becomes more complex and often operates in a distributed or dynamic environment, the execution orders among or the usage of certain software components are hard to be measured prior to the software deployment. We consider such systems as non-deterministic systems. In non-deterministic systems, there exist some states that have more than one outgoing transitions that are engaged in a purely non-deterministic fashion. That is, the outcome of this selection process is not known a priori, and hence, no statement can be made about the likelihood with which transition is selected. Non-determinism exists in many modern softwares, e.g., a cloud computing system within which multiple processes aim to access a shared resource and a pervasive system within which the software intensively interfaces with environments or human behaviors.

1.1.3 Research Targets

The requirements of reliability analysis approaches for such non-deterministic system are summarized as follows.

- **Support for non-determinism**. The approaches should be able to perform reliability analysis on non-deterministic systems. That is, even for a system operating in complex and dynamic environments, its reliability can still be analyzed as close to the 'true' reliability as possible, before software deployment.
- **Efficiency**. The approach should be efficient, i.e., the computation process should be as fast as possible.
- **Scalability**. To handle large scale software system, the approaches should have good scalability.

Although there are many related works on improving reliability analysis in terms of its efficiency and scalability, insofar, none of them can work for non-deterministic system. In this work, we are motivated to propose an approach to meet the first requirement, on top of which to satisfy the last two requirements.

1.2 Summary of This Thesis

Existing reliability analysis approaches only apply to deterministic systems. In this thesis, we propose to analyze the reliability of non-deterministic system via probabilistic model checking on Markov decision processes (MDP), a well-known automatic verification technique dealing with both probabilistic and non-deterministic behaviors. In addition, we integrate various techniques, e.g., statistical, numerical and graphical methods, to make the reliability analysis much more scalable and efficient. The Ph.D. work is summarized into the following three main aspects.

Reliability analysis via combining model checking and testing. Testing provides a probabilistic assurance of system correctness. In general, testing relies on the assumptions that the system being examined is deterministic so that test cases can be sampled. However, a challenge arises when the system behaves non-deterministically in a dynamic operating environment because it will be unknown how to sample the test cases. In this work, we propose a method to combine hypothesis testing and probabilistic model checking to analyze the reliability of non-deterministic systems. The idea is to apply hypothesis testing to deterministic system components and use probabilistic model checking techniques to lift the results through non-determinism. Furthermore, if a requirement on the level of assurance is given, we apply probabilistic model checking techniques to push down the requirement through non-determinism to individual components so that they can be verified using hypothesis testing. Based on the proposed framework, a toolkit RaPiD has been developed to support automated software reliability analysis including reliability prediction, reliability distribution and sensitivity analysis. Case studies have been carried out on real world systems including a stock trading system, a hospital therapy control system and an ambient assisted living system.

Improved probabilistic reachability analysis via SCC reduction. The second part is on improving the efficiency of the proposed approach, in particular, the fundamental part that calculates the probabilities of reaching certain system states (i.e., reachability analysis). It is known that existing approaches on reachability analysis for DTMCs or MDPs are often inefficient when a given model contains loops or formally called strongly connected components (SCCs). In this work, we propose divide-and-conquer algorithms to eliminate SCCs in DTMCs and MDPs respectively. For MDPs, the proposed algorithm can actively remove redundant probability distributions based on the convex property. With the removal of loops and parts of probability distributions, the probabilistic reachability analysis can be accelerated, as evidenced by our experimental results.

Reliability analysis on distributed systems based on abstraction and refinement. Distributed systems like cloud-based services are ever more popular these days. Traditional probabilistic model checking is limited to small scale distributed systems as it works by exhaustively exploring the global state space, which is the product of the state spaces of all components and often huge. As a result, reliability analysis of distributed system using probabilistic model checking is particularly difficult and even impossible. In this part of the work, we improve the probabilistic model checking through a process of abstraction and reduction, which controls the communications among different system components and actively reduces the size of each system component. We prove the soundness and completeness of the proposed approach. Through the evaluations with several systems, we show that our approach often reduces the size of the state space by several orders, while still producing sound and accurate assessment.

1.3 Thesis Outline and Overview

The thesis is structured in 7 chapters in total. In the following, we briefly present the outline of the thesis and overviews of the rest of chapters.

Chapter 2 recalls the background knowledge that are fundamental in this thesis. In this chapter, we first introduce two typical models that are widely employed in probabilistic systems: discrete time Markov chain (DTMC) and Markov decision process (MDP). DTMC models a fully probabilistic system, while MDP can model both non-deterministic and probabilistic systems. Calculating the probability of reaching certain system states is a vital part in analyzing the quantitative aspects of the system such as reliability. Therefore, in the second part of this chapter, we introduce two methods for calculating this probability including linear programming and value iteration.

Chapters 3-6 present the main contributions of the thesis and structured in the following manner. At the beginning of each chapter, an introduction is given, followed by details of our technical contributions and experimental evaluations. Each chapter ends with a separate discussion of related work.

Chapter 3 presents our proposed reliability analysis framework that combines probabilistic model checking with testing. The chapter starts with a running example to demonstrate why testing alone is not enough for reliability analysis. Next, it presents our approach on combining model checking and testing for two reliability analysis activities: reliability prediction that is to calculate the overall system reliability, and reliability distribution that is to distribute the overall reliability to individual system components.

Chapter 4 introduces our reliability analysis toolkit called RaPiD (Reli*a*bility Pred*i*ction and Distribution) and then presents an application of RaPiD in analyzing an ambient assisted living (AAL) system. This system involves a variety of sensors, networks and remind infrastructures, which interact with unpredicted human behaviors. Thus, the reliability analysis is highly challenging. This chapter gives the details on how to construct a reliability model of an AAL system from its usage scenarios. Based on the reliability model, reliability analysis and sensitivity analysis are conducted with our toolkit.

Chapter 5 introduces the divide-and-conquer approaches to improve the efficiency of reachability analysis in Markov models. Reachability analysis is a fundamental step in probabilistic model checking and therefore it is a critical part in our reliability analysis. This chapter first shows that the main method (i.e., value iteration) has slow convergence problem due to the existence of loops. It then presents our two reduction algorithms for DTMCs and

MDPs respectively. The main idea is to partition the state space of a DTMC or MDP into small groups, and remove loops in each group. After iteratively removing loops, the resulting DTMC or MDP is acyclic that reachability can be calculated efficiently.

Chapter 6 introduces the abstraction and refinement approach to improve the scalability of reliability assessment for distributed systems. This approach works by controlling the communications among different components and actively reducing the size of each component. The resulting state space of the distributed systems can thereby be reduced by several orders of magnitude. To further improve the accuracy of the reliability assessment, two heuristics are introduced to systematically refine the communications.

Chapter 7 concludes this thesis with a summary of contributions and an outlook on future research directions.

1.4 Acknowledgment of Published Work

Most of the work presented in this thesis has been published in international conference proceedings.

- Chapter 3 was published at the 13th International Symposium on Software Testing and Analysis (ISSTA 2013) [58].
- The case study in Chapter 4 was published at the 19th International Symposium on Formal Methods (FM 2014) on industry track [88].
- In Chapter 5, Section 5.3 was published at the 10th International Conference on integrated Formal Methods (iFM 2013) [123].
- In Chapter 5, Section 5.4 is accepted for the publication at the 16th International Conference on Formal Engineering Methods (ICFEM 2014) [59].

In addition, the idea of this thesis has been presented on the doctor symposium of the 19th International Symposium on Formal Methods (FM 2014) [57]. The reliability analysis toolkit presented in Chapter 4 is accepted for the publication in the 22nd ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE 2014). The work in Chapter 6 is currently under submission to the 37th International Conference on Software Engineering (ICSE 2015). For all the publications mentioned above, I have contributed substantially in both theory development and tool implementation.

Chapter 2

Background

In this chapter, we define some general and fundamental notations and concepts used in our work. In the first part, two typical formalisms for probabilistic systems, including discrete time Markov chain and Markov decision process are introduced. In discrete time Markov chains (DTMCs), all transitions are probabilistic. Markov chains are the most popular operational model for the evaluation of performance and dependability of software systems [15, 64]. Over the past few decades, it has been also widely used as the models for software reliability analysis [26, 70, 50, 55, 51]. However, Markov chains are not suitable to model interleaving behavior of distributed systems or the system that interacts with an unknown environment. For this purpose, Markov decision processes (MDPs) [109] are employed. MDPs can model both non-deterministic and probabilistic systems.

The problem of calculating the probability of reaching certain system states is central to the probabilistic system analysis. In fact, it is a fundamental task in probabilistic model checking [16]. In the second part, two main methods for calculating the reachability probabilities are introduced. Other concepts will be introduced in later chapters where they are relevant.

2.1 Modeling Formalisms

A stochastic system has the Markov property if the conditional probability distribution of future states of the system depends only on the present state, not upon the sequence of events that leads to this state [16]. This is also known as the memoryless property. A model with this property is called a Markov model. In this section, two typical Markov models will be introduced, i.e., discrete time Markov chain and Markov decision process. Discrete time Markov chains can only model purely probabilistic systems and Markov decision processes can not only model probabilistic but also non-deterministic systems. Both models assume the underlying time domain of the system operation is discrete, and each transition is assumed to take a single time unit, which are reasonable abstractions for most software systems operating on digital computers.

Given a set of states S, a probability distribution is a function $u : S \to [0, 1]$ such that $\Sigma_{s \in S} u(s) = 1$. The probability distribution can also be expressed in vector form as $\mathbf{u}$, and $Distr(S)$ denotes the set of all discrete probability distributions over S. In the following part, we introduce the details on Discrete time Markov chain and Markov decision process.

2.1.1 Discrete Time Markov Chain

Definition 2.1.1 *A discrete time Markov chain is a tuple $\mathcal{D} = (S, init, Pr)$ where S is a set of states; $init \in S$ is the initial state; $Pr : S \to Distr(S)$ is a transition function.* □

A discrete time Markov chain (DTMC) is a fully probabilistic transition system where S represent possible states of the system; transitions among states occur at discrete time and follow a probability distribution. In this thesis, we focus on the finite model, i.e., a DTMC or an MDP that has a finite number of states and transitions. Moreover, we consider a

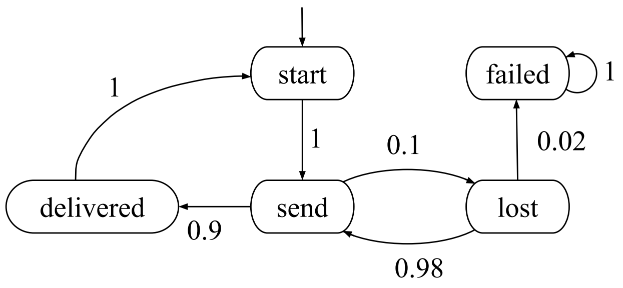


Figure 2.1: An example of a discrete time Markov chain

single initial state, which can be easily generalized to several initial states with a certain probability distribution.

Formally, a DTMC model can be expressed by a stochastic matrix $P : S \times S \to [0, 1]$ such that $\sum_{s' \in S} P(s, s') = 1$. An element $P(s_i, s_j)$ represents the transition probability from state s_i to state s_j. The row $P(s, \cdot)$ for state s in this matrix contains the probabilities of moving from s to its successors, while the column $P(\cdot, s)$ for state s specifies the probabilities of entering state s from any other state. A state is an absorbing state if it has only self-looping outgoing transitions, i.e., $P(s_i, s_i) = 1$. A path of π, which gives one possible evolution of the Markov chain, is a sequence of states $s_0 s_1 s_2 \ldots$ such that $s_0 = \mathit{init}$ and $P(s_i, s_{i+1}) > 0$ for all $i \geq 0$.

An example of DTMC is depicted in Figure 2.1, which models a simple error-prone communication protocol with an unreliable channel. Here, state *start* is the initial state, at which the transition will go to state *send* with probability of 1. In the state *send*, a message can be successfully delivered with a probability of 0.9; otherwise, it will be lost. If there is a message lost, with a probability of 0.98, it will send an alert to re-deliver the message; and with a probability of 0.02, it fails to do so. This DTMC models a purely probabilistic system which has exact one probability distribution at each state. Using the enumeration *start*, *send*, *lost*, *delivered*, *failed* for the states, the stochastic matrix P is a 5×5 matrix as follow.

$$P = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0.1 & 0.9 & 0 \\ 0 & 0.98 & 0 & 0 & 0.02 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

An example of a path is

$$\pi = (start\ send\ lost\ send\ lost\ send\ delivered)^{\omega}.$$

Along this path, each message has to be retransmitted twice before successfully delivered.

2.1.2 Markov Decision Process

Discrete time Markov chains (DTMCs) represent a fully probabilistic model of a system, i.e., in each state of the model, the exact probability of moving to each other state is always known. In DTMCs, the probabilistic choices may serve to model and quantify the possible outcomes of randomized actions such as sending a message over a lossy communication channel, tossing a coin, or modeling the interface of a system with its environment. For instance, for an error-prone communication protocol, it might be reasonable to assign a probability of 0.9 for successfully delivering a message, and a probability of 0.1 for losing the message. This, however, requires statistical experiments to obtain adequate probability distributions that model the average behavior of the environment, i.e., the media for the message channel. In cases where this information is not available, or where it is needed to analyze the system in all potential environments, a natural choice is to model the interface with the environment by non-determinism.

Another great need for non-determinism is in modeling distributed systems. Due to the interleaving of the behavior of the distributed processes involved, the non-deterministic choice is used to determine which of the concurrent processes performs the next step. Finally, non-determinism is also crucial for the situations that involve underspecification of certain system actions or control strategies, or abstraction of a complex system using a simpler one. For example, in the case of data abstraction, one might replace probabilistic branching by a non-deterministic choice.

As a result, to model such systems exhibiting both probabilistic and non-deterministic behavior, Markov decision processes (MDPs) are more favored in those systems [16]. The formal definition of MDP is introduced as follows.

Definition 2.1.2 (Markov Decision Process) *A Markov decision process is a tuple* $\mathcal{M} = (S, init, Act, Pr)$ *where* S *is a set of states;* $init \in S$ *is the initial state;* Act *is an alphabet; and* $Pr : S \times Act \to Distr(S)$ *is a labeled transition relation.* $\square$

An action α is *enabled* in state s if and only if $\sum_{s' \in S} \Pr(s, \alpha)(s') = 1$. Let $Act(s)$ denote the set of enabled actions in s. Given a state s, we denote the set of probability distributions of s as $\mathbf{U}_s$, s.t., $\mathbf{U}_s = \{Pr(s, a) \mid a \in Act\}$. A state without any outgoing transitions to other states is called an *absorbing* state, which has only a self-loop with a probability of 1. Without loss of generality, in this work, we assume that MDP has only one initial state and is always deadlock-free, i.e., for any state $s \in S$, $Act(s) \neq \varnothing$. It is known that we can add a self-looping transition with a probability of 1 to a deadlock state without affecting the calculation result [16].

An infinite or a finite *path* in $\mathcal{M}$ defined as a sequence of states $\pi = s_0, s_1, \cdots$ or $\pi = s_0, s_1, \cdots, s_n$, respectively, such that $\forall\, i \geq 0$ (for finite paths, $i \in [0, n-1]$), $\exists\, a \in Act, Pr(s_i, a)(s_{i+1}) > 0$. An MDP is *non-deterministic* if any state has more than one probability distribution. A DTMC can be interpreted as a special MDP that has only one event (and one probability distribution) at each state, and thus is deterministic.

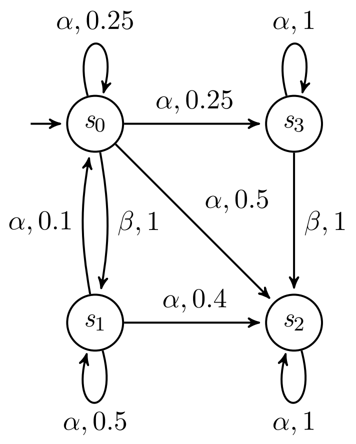


Figure 2.2: An example of a Markov decision process

An example of MDP is shown in Figure 2.2, where state s_0 is the initial state, i..e., $\mathit{init} = s_0$. In the figure, transitions labeled with the same action belong to the same distribution. The set of enabled actions at, for instance, state s_0, is $Act(s_0) = \{\alpha, \beta\}$ with $P(s_0, \alpha)(s_0) = P(s_0, \alpha)(s_3) = 0.25$, $P(s_0, \alpha)(s_2) = 0.5$, and $P(s_0, \beta)(s_1) = 1$. On selecting action β, the next state is s_1; on selecting action α, the successor states s_0, s_2 and s_3 are all possible. Without information about the frequency of actions α and β in at state s_0, the selection between these two actions is purely non-deterministic.

Schedulers A scheduler is used to resolve the non-determinism in each state in an MDP. Intuitively, given a state s, an action is first selected by a scheduler. Once an action is selected, the respective probability distribution is also determined; and then one of the successive states is reached according to the probability distribution. In this thesis, we focus on a subclass of schedulers that are called *memoryless schedulers*, as the maximal and minimal reachability probabilities can be obtained by schedulers of this simple subclass. Formally, a memoryless scheduler for an MDP $\mathcal{M}$ is a function $\sigma : S \to Act$. At each state, a memoryless scheduler always selects the same action in a given state. This choice is independent of the path that leads to the current state. In the following, unless otherwise specified, the terms ‘schedulers’ and ‘memoryless schedulers’ are used interchangeably. An induced MDP, $\mathcal{M}_\sigma$, is the DTMC defined by an MDP $\mathcal{M}$ and a scheduler σ. A non-memoryless scheduler is the scheduler that can select different action in a given state according to the execution history. An MDP $\mathcal{M}$ can be viewed as a group of DTMCs, each of which is obtained with a different scheduler.

2.2 Probabilistic Reachability Analysis

In this thesis, the reliability model is an MDP, as it can model both probabilistic and non-deterministic behavior of a system. One fundamental question in quantitative analysis of MDPs is to compute the probability of reaching target states G from the initial state (hereafter, reachability probabilities). Noted that with different schedulers, the result may be different. The measurement of interest is thus the maximum and minimum reachability probabilities. The maximum probability of reaching any state in G in an MDP $\mathcal{M}$ is denoted as $P^{max}(\mathcal{M} \models \Diamond G)$, which is defined as:

$$P^{max}(\mathcal{M} \models \Diamond G) = \sup_\sigma P(\mathcal{M}_\sigma \models \Diamond G)$$

Similarly, the minimum probability of reaching any state in G is defined as:

$$P^{min}(\mathcal{M} \models \Diamond G) = \inf_\sigma P(\mathcal{M}_\sigma \models \Diamond G),$$

The supremum/infimum ranges over all and potentially infinitely many schedulers. Rather than considering all schedulers, it suffices to consider only memoryless schedulers, in order to obtain maximum and minimum reachability probabilities [16].

Theorem 2.2.1 (Equation System for Max Reachability Probabilities) *Given a finite MDP $\mathcal{M}$ with state space S, and target states $G \subseteq S$, the vector $V_{s\in S}$ with $V(s) = P^{max}(s \models \Diamond G)$ yields the unique solution of the following equation system:*

- *If $s \in G$, then $V(s) = 1$.*
- *If $s \not\models \Diamond G$, then $V(s) = 0$.*
- *If $s \notin G$ and $s \models \Diamond G$, then*

$$V(s) = max\left\{\sum\nolimits_{t\in S} \Pr(s,\alpha)(t) \cdot V(t)\,|\alpha \in Act(s)\right\}$$

□

The maximum reachability probability of reaching target states from a state in an MDP can be transformed into an equation system based on Theorem 2.2.1. Here, we use $P^{max}(s \models \Diamond G)$ to denote the probability for reaching G from a given state s in an MDP. The minimum reachability probability can be obtained in a similar manner.

In the following, with the MDP in Figure 2.2 in Page 14, we demonstrate how to numerically calculate the maximum probability of reaching state s_2 from the initial state. Let V be a vector such that, given a state s, $V(s) = P^{max}(\mathcal{M} \models \Diamond G)$ is the maximum probability of reaching G from a state s. Here, state s_0 is the initial state, and G contains a single target state s_2, i.e., $V(s_2) = 1$. For instance, $V(s_0)$ is the maximum probability of reaching G from the initial state. State s_2 is the target state, so $V(s_2) = 1$. Using backward reachability analysis, we can identify the set of states $X = \{s_0, s_1, s_2, s_3\}$ such that G is reachable from any state in X, i.e., $\forall\, s \in X, s \models \Diamond G$; and a set of states Y from where G is unreachable, i.e., $\forall\, s \in Y, s \not\models \Diamond G$, hence, $V(s) = 0$. In this case, all the states can reach s_2, hence $Y = \varnothing$. With memoryless schedulers, there are two main approaches on calculating the reachability probabilities for states $X \setminus G$, i.e., $\{s_0, s_1, s_2, s_3\}$.

2.2.1 Linear Programming

The method encodes each probability distribution for a state in $X \setminus G$ into a linear inequality. This is defined as,

$$V(s) \geqslant \sum\nolimits_{t \in S} P(s, \alpha)(t) \cdot V(t), \quad \text{for } s \in X \setminus G \tag{2.1}$$

with an additional constraint $V(s) \in [0, 1]$, and the goal is to minimize the sum of V.

Taking state s_0 in Figure 2.2 in Page 14 for example, there are two actions α, β, each attached with a probability distribution, i.e., $Pr(s_0, \alpha) = \{0.25 \mapsto s_0, 0.5 \mapsto s_2, 0.25 \mapsto s_3\}$ and $Pr(s_0, \beta) = \{1 \mapsto s_1\}$. These can be encoded to

$$\begin{aligned} V(s_0) &\geqslant 0.25\,V(s_0) + 0.5\,V(s_2) + 0.25\,V(s_3), \\ V(s_0) &\geqslant V(s_1). \end{aligned}$$

We can obtain the inequalities using the probability distributions from all the other states in a similar way. The unique solution of this set of linear inequalities is $V = (1, 1, 1, 1)$. V can be automatically obtained by solving such linear programming using standard algorithms.

2.2.2 Value Iteration

This method iteratively builds an approximation of V based on the previous approximation. Let V^i be the i-th approximation. For $\forall\, s \in X \setminus G$, we have

$$\begin{aligned} V^0(s) &= 0, \\ V^{i+1}(s) &= max\{\sum_{t \in S} Pr(s, a)(t) \cdot V^i(t) \mid a \in Act(s)\}. \end{aligned} \tag{2.2}$$

Table 2.1: List of values V^i at each iteration i

Iteration i	0	1	2	3	4	5	6	7
$V^i(s_0)$	0	0.5	0.875	0.96875	0.9921875	0.998046875	0.999511719	0.99987793
$V^i(s_1)$	0	0.4	0.65	0.8125	0.903125	0.95078125	0.975195313	0.987548828
$V^i(s_2)$	1	1	1	1	1	1	1	1
$V^i(s_3)$	0	1	1	1	1	1	1	1
diff	Inf	1	0.375	0.1625	0.090625	0.04765625	0.024414063	0.012353516

It can be shown that for every state s, $V^{i+1}(s) \geqslant V^i(s)$ and we can obtain V in the limit, $\lim_{i\to\infty} V^i = V$. In reality, it may take many iterations before V^i converges and thus value iteration is often stopped when the absolute/relative difference between two successive iterations falls below a certain threshold ϵ. The number of iterations required is related to the subdominant eigenvalue of the transition matrix [119]. Each iteration involves a series of matrix-vector multiplications, with a complexity of $\mathcal{O}(n^2 \cdot m)$ in the worst case, where n is the number of states in S and m is the maximum number of actions/distributions from a state.

Applying value iteration to the MDP in Figure 2.2 in Page 14, we have $V^i(s_2) = 1$ for any i and

$$
\begin{aligned}
V^{i+1}(s_3) &= max\{V^i(s_3), 1\} = 1, \\
V^{i+1}(s_1) &= 0.1\,V^i(s_0) + 0.5\,V^i(s_1) + 0.4, \\
V^{i+1}(s_0) &= max\{0.25\,V^i(s_0) + 0.5 + 0.25\,V^i(s_3),\, V^i(s_1)\}.
\end{aligned}
$$

With the value iteration method, it is then easy to get $V^0 = (0, 0, 1, 0)$; $V^1 = (0.5, 0.4, 1, 1)$; $V^2 = (0.875, 0.65, 1, 1)$; $V^3 = (0.96875, 0.8125, 1, 1)$; etc. The evaluation continues until difference $max_{s\in S} \mid {x_s}^{(n+1)} - {x_s}^{(n)} \mid$ is below certain predefined threshold. The values V^i and its corresponding value difference for the first eight iterations are listed in Table 2.1. As your can see, if the stopping criterion is set as maximum difference is within 0.05, i.e., $max_{s\in S} \mid {x_s}^{(n+1)} - {x_s}^{(n)} \mid \leq 0.05$, the program stops at iteration 5, and reports final reachability results as $V = \{0.998046875, 0.95078125, 1, 1\}$. This result indicates that the maximum probability of reaching to s_2 from state s_0 is 0.998046875, from state s_1, 0.95078125, from state s_2, s_3, 1. As we can also observed from Table 2.1, more iterations are required if there is an even smaller threshold requirement for the maximum difference.

Chapter 3

Reliability Analysis via Combining Model Checking and Testing

Testing is useful because it provides a certain level of assurance of system correctness/reliability. The more testing is conducted, the more likely a system behavior (being a bug or not) is demonstrated. When the system under test is deterministic, the level of "assurance" can be precisely captured through hypothesis testing, which is a statistical process determining whether to reject a null hypothesis based on tests generated according to the probability distribution in a model [11]. However, the testing method for quantifying the level of "assurance" remains unknown if the system is non-deterministic (or equivalently that the probability distributions of certain events are unknown or hard to predict). In this chapter, a probabilistic "assurance" for non-deterministic system is achieved through combining hypothesis testing and probabilistic model checking, and the underlying principle are demonstrated through an application of system reliability analysis.

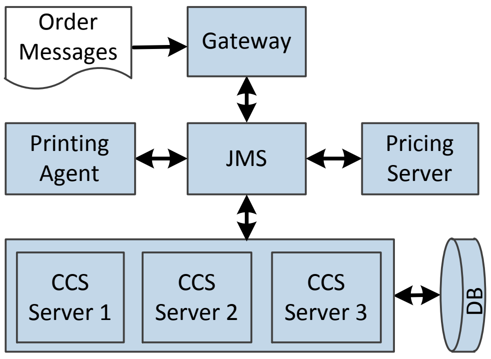


Figure 3.1: Architecture of the CCS System

3.1 Introduction

A motivating example One product of our industrial collaborator, a financial software solution provider, is the Call Cross System (CCS), which is a stock trading system accepting order flow in a global operating environment. It operates on a 24 hours basis for 6 days per week. It has operating successfully since 2005 and playing a crucial part in the core business of a financial institute in Boston. The CCS system is required to be highly reliable, and quantitatively 99.99% of the transactions must be correctly handled. With such a requirement, an immediate question is: how do we calculate the system reliability and present our calculation as a formal evidence to show that the delivered system will meet the requirement? A related question is: given the system level reliability requirement, what is the reliability requirement on each of the system components, so that the component development teams can carry out reliability measures on their own? The former is known as reliability prediction problem and we term the latter as reliability distribution problem.

In order to answer the questions, we must understand the architecture of the CCS. Figure 3.1 shows the high-level architecture of the CCS, where arrows represent the directions of dataflow. At the top level, the system consists of six components. *Gateway* serves as a linkage between peripheral applications and the CCS. It receives order messages in the data batch manner and dispatches them to different symbol partitions in the CCS Servers via Java Message Service (JMS) server. *JMS* serves as the messaging engine for order flows, executions, pricing requests and responses, printing trade reports in several markets. *CCS Servers* are the core of the system to perform the business logics.They consist of a cluster of Websphere AppServer nodes with WebSphere Partitioning Facility (WPF) enabled. The workload is dynamically distributed to the server nodes based on the partitioning policies. The status of each partition is monitored in real-time. If one node fails, partitions in the failed node are reloaded to other healthy nodes. The partitions communicate with the external objects via a JMS server. *Pricing Server* is a JMS client that processes pricing requests and responds with pricing events to CCS servers. *Printing Agent* is a JMS client that receives printing requests and responses to JMS after printing. *DB* is the database server for the CCS system. A stock trading transaction is accomplished through a series of steps involving multiple system components. First, the Gateway sends order messages to its inbound queue through the JMS. The CCS servers receive order messages from inbound queue, process and store them into the database through underlying service framework. Afterwards, the Pricing Server provides the current price to CCS Servers. After the transactions are finished, the trading information is shown by the Gateway or printed by the Printing Agent. The transaction completes after displaying out. Furthermore, components are often duplicated in the CCS system to achieve high reliability.

Why existing approaches are not enough? Software reliability is defined as the probability of failure-free software operation for a specified period of time in a specified environment [81]. In this work, we consider reliability of a system based on the probability of failure of the system. Existing approaches on the reliability prediction problem fall into two categories: black-box approaches [70, 132] and white-box approaches [26]. The black-box approaches treat a system as a monolith and evaluate its reliability using testing techniques. They use the observed failure information to predict the reliability of software based on several models such as Jelinski-Moranda model [71], Musa Okumoto model [100], Littlewood-Verrall model [86], etc. On the contrary, the white-box approaches assume reliability of system components are known and evaluate software reliability analytically based on a model of the system architecture including discrete time Markov chains (DTMCs) [26], continuous time Markov chains (CTMCs) [81], or semi-Markov processes (SMPs) [77]. In these approaches, the probabilistic transfer of control among components is assumed to be known. For instance, the probability is assumed to be a constant in DTMC-based approaches or a function of time in CTMC/SMP-based approaches.

In the following, we argue that because the CCS system's behavior relies on the run-time environment, the existing approaches are not ideal for it, nor for non-deterministic systems in general. The white-box approaches rely on modeling systems in DTMCs, CTMCs or SMPs, which imply that there is only one probability distribution out of any system component. In other words, if the system's behavior is hard to predict, the assumption that the probability distribution of transitions among system components is known should be problematic. For instance, if we model the CCS system using a DTMC, one probability distribution is required to capture the probability that an order is processed by different CCS servers. Obtaining this probability distribution is highly non-trivial as the target CCS server is chosen at run-time using a sophisticated dynamic load balancing algorithm. A probability distribution obtained in a testing environment is likely to be different from that of the real system. As a result, the estimated system reliability may lose its accuracy. A "safer" (and more convincing to the stakeholder) prediction is to assume no knowledge on the distribution and assume that an order may be *non-deterministically* assigned to any CCS server. The existing black-box approaches rely on testing the overall systems. However, with a non-deterministic system under test, it is unclear how test cases should be generated systematically so that testing can provide a quantifiable level of "assurance".

There is another issue with the existing white-box approaches. Two inputs are required

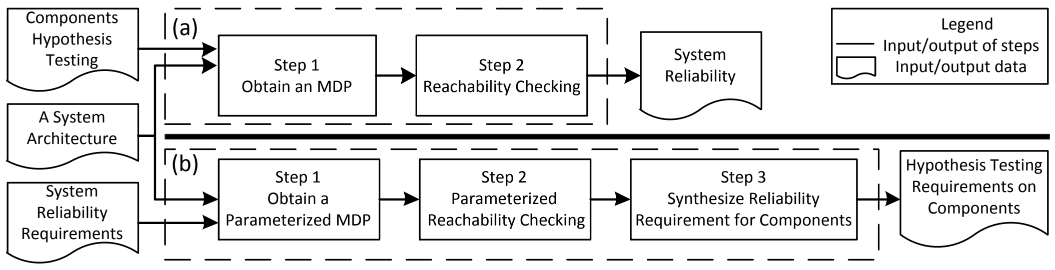

Components Hypothesis Testing
A System Architecture
System Reliability Requirements
(a)
Step 1 Obtain an MDP
Step 2 Reachability Checking
System Reliability
Legend
Input/output of steps
Input/output data
(b)
Step 1 Obtain a Parameterized MDP
Step 2 Parameterized Reachability Checking
Step 3 Synthesize Reliability Requirement for Components
Hypothesis Testing Requirements on Components

Figure 3.2: Workflow: (a) reliability prediction; (b) reliability distribution

including the reliability of the system components and a model of the system architecture. The former is usually obtained simply through component-based testing, which could be misleading. For instance, a component which failed 2 out of 50 test cases and a component which failed 40 out of 1000 test cases would have the same reliability of 96%. It is, however, obvious that 96% for the second component is more accurate. In the CCS system, we have indeed discovered that the number of tests for different components varies significantly. Mixing these semantically different data in calculating the system reliability gives inaccurate results.

Combining testing and model checking From the above analysis, testing is ineffective in non-deterministic systems. On the contrary, model checking is well-known to be able to handle non-deterministic systems systematically [32, 16]. We thus propose to combine testing (in particular, hypothesis testing) and model checking (in particular, probabilistic model checking) for non-deterministic systems. The idea is to apply hypothesis testing to system components which are deterministic and use probabilistic model checking to lift the results through non-determinism.

In the example of reliability prediction and distribution, we propose to apply hypothesis testing to measure component reliability with error bounds, and to use MDP-based probabilistic model checking to obtain system-level reliability. Hypothesis testing is one of the testing methods [114], and can be used to bound the number of test cases by indicating when the test can be stopped [132]. With hypothesis testing, users can quantify the accuracy of a test result by giving error bounds, i.e., the probability of false positive and false negative testing conclusions. MDPs are used in our probabilistic model checking. Compared to DTMCs, MDPs support non-determinism, i.e., there may be multiple probability distributions from a state in the model. With its expressiveness, we can then properly model complicated systems like the CCS. However, compared to DTMC-based reliability prediction or distribution, MDP-based algorithms are more challenging, as we show later.

Figure 3.2-a shows our workflow of solving the reliability prediction problem. Firstly, hypothesis testing is applied to obtain the reliability of system components. The result is a probability, i.e., the reliability of a component being larger or equal to this probability, with error bounds defined by users. Next, MDP-based reachability analysis is used to compute the overall system reliability, which is the probability that the system reaches the success state. Notice that existing algorithms on probabilistic reachability checking must be extended to handle the error bounds obtained with hypothesis testing. Figure 3.2-b shows the workflow of solving the reliability distribution problem. Given a reliability requirement for the system with error bounds, we solve the problem in three steps. Firstly, we construct a parameterized MDP model within which each component is associated with variables representing its reliability measurement. Next, we develop a parameterized probabilistic reachability checking algorithm to obtain the minimum constraints on the variables. Lastly, we synthesize concrete reliability requirement for each component based on the constraints. We develop a toolkit named RaPiD to fully automate our approach and apply it to investigate two real-world systems.

Organization The rest of this chapter is organized as follows. Section 5.2 reviews background on MDP-based probabilistic model checking and hypothesis testing. Section 3.3 presents our approach on combining probabilistic model checking and hypothesis testing.

Section 3.4 shows an application of our approach to reliability prediction and distribution. Section 3.5 evaluates our approach. Section 6.6 surveys related works.

3.2 Background on Hypothesis Testing

This work requires some background on both probabilistic model checking and hypothesis testing. Since relevant details on probabilistic model checking, including discrete time Markov chain, Markov decision process and reachability analysis, have already been introduced in Chapter 2. In this section, we briefly introduce the background on hypothesis testing.

Hypothesis testing is a statistical process to decide the truthfulness of two mutual exclusive statements: H_0 and H_1, where H_0 is the hypothesis that the probability of a given event is larger than or equal to a given value p_0, and H_1 is the alternative hypothesis (i.e., the probability of the event is less than or equal to a given value p_1). Besides, two parameters are required from users. One is the targeted assurance level (θ) over the system, and the other is the indifference region (2δ). Indifference region refers to the region (p_1, p_0), used to avoid exhaustive sampling and obtain the desired control over the precision [135]. With the input θ and δ, $p_0 = \theta + \delta$, $p_1 = \theta - \delta$. The probability of accepting H_1 given that H_0 holds is required to be at most α, called false negative, and the probability of accepting H_0 if H_1 holds should be no more than β, called false positive. In practice, the error bounds (i.e., α, β), and δ can often be decided by how much testing resource the component developer has. In general, it would require more resource for a smaller error bounds or a smaller indifference region.

Hypothesis testing has been applied for reliability estimation [132]. Let R be the reliability of a module. Suppose that we wish to test the hypothesis that the reliability R is at least θ. With a user defined δ, we have hypothesis $H_0 : R \geq \theta+\delta$ and $H_1 : R \leq \theta-\delta$. We remark that hypothesis testing requires a way of sampling system executions according to its operational usage. Many sampling methods have been developed and applied for software demonstrating testing [124, 115]. There are two main acceptance sampling methods to decide when testing can be stopped. One is fixed-size sampling test, which often results in a large number of tests [135]. The other one is sequential probability ratio test (SPRT), which yields a variable sample size. SPRT is faster than fix-sampling methods as the testing process ends as soon as a conclusion is made. The basic idea of SPRT is to calculate the probability ratio, after observing a test result and comparing with two stopping conditions [12]. If either of the conditions is satisfied, the testing stops and returns which hypothesis is accepted. Readers can refer to [135] for details.

Example In the CCS system, to verify whether the reliability of a CCS server is at least 0.8, users should define the test parameters (i.e., δ, α and β). Assuming $\alpha = 0.01$, $\beta = 0.01$, and $\delta = 0.1$, the parameters define the goal for the testing, i.e., whether to accept H_0 "the reliability of the CCS Server is at least 0.9" or H_1 "its reliability is at most 0.7". If the "true" reliability is at most 0.8, it is guaranteed that the probability of wrongly accepting H_0 is less or equal to 0.01. By the stopping criterion, we have $x_m \geq 0.8138m + 3.4040$ (to accept H_0) or $x_m \leq 0.8138m - 3.4040$ (to accept H_1). We start the testing with $m = 0$; $x_m = 0$. After executing a test case (which is chosen randomly according to the user profile), m increases by 1. x_m either increases by 1 if the test finishes without failure or remains the same otherwise. Next, the updated m and x_m are used for stopping criteria. If any one of the stopping criteria is fulfilled, H_0 or H_1 is accepted accordingly; otherwise, sampling continues with another test case and the above steps will be repeated. $\square$

The error bounds quantify the reliability measurement. They can differentiate the above-mentioned case, i.e., two different test cases which have concluded the same reliability of 96% with different number of tests. Assuming that a fixed sampling plan is adopted with θ

set to 0.96 and δ set to 0.01, the minimum error bounds are: $\alpha = 0.5262$ and $\beta = 0.3357$ for 2 failures out of 50 tests case; and $\alpha = 0.2161$ and $\beta = 0.2026$ for the case of 40 failures out of 1000 tests. Therefore, the result based on the larger sample size is more accurate in terms of smaller error bounds.

SPRT is guaranteed to terminate [12], while the expected sample size is hard to determine. Wald [11] has provided a good approximation. The expected sample size increases from 0 to p_1 and decreases from p_0 to 1. The worst case is when the "true" probability is within the indifference region. If $a = 0.01, b = 0.01, p_0 = 0.99$, and $p_1 = 0.98$, the expected sample size will be 3.0005×10^3 by Wald's approximation. If considering the hypothesis testing parameters with high precision, which is normally the case in practice, e.g., $a = 0.001, b = 0.001, p_0 = 0.9999, p_1 = 0.9998$, the expected sample size will be 6.8811×10^5 in this case.

3.3 Combining Model Checking and Hypothesis Testing

Hypothesis testing enables directly sampling on systems, but is not suited to non-deterministic systems. On the contrary, probabilistic model checking can handle non-determinism easily with exact solutions, but suffers from state explosion problem. The combination of both is proposed in such a way that hypothesis testing is conducted on each subsystem separately and probabilistic model checking method is performed on the system level modeled in an MDP. This can be formally presented as follows.

Let $\mathcal{M}$ be an MDP, ϕ be a property (which can be in LTL [107], PCTL [63], etc.). By probabilistic model checking, it can calculate the probability of the set of paths in $\mathcal{M}_\sigma$ that satisfy the ϕ for all schedulers σ, denoted as $P(\mathcal{M} \models \phi)$. It can also check whether a property holds with probability at least θ, i.e., $P_{\geq\theta}(\mathcal{M} \models \phi)$, which returns a Boolean value. The model is assumed to be composed of several components, denoted as $\mathcal{M}(\mathcal{D}_0, \mathcal{D}_1, \ldots, \mathcal{D}_n)$, where each $\mathcal{D}_i$ is a deterministic system component. We connect the probability of satisfying global property ϕ with the probability of satisfying the local properties for each component with the following function.

$$P(\mathcal{M} \models \phi) = f\left(P\left(\mathcal{D}_0 \models \phi_0\right), P\left(\mathcal{D}_1 \models \phi_1\right), \ldots, P\left(\mathcal{D}_n \models \phi_n\right)\right)$$

where ϕ_i is the local property for component i and f is a function that takes in the probability of satisfying the local properties in each component and outputs the probability of satisfying the global property in the whole model.

Similarly, the verification task of comparing the probability of satisfying a global property with a bound relates to that of local properties in each components is as follows.

$$P_{\geq\theta}(\mathcal{M} \models \phi) = \left(P_{\geq\theta_0}\left(\mathcal{D}_0 \models \phi_0\right)\right) \wedge \left(P_{\geq\theta_1}\left(\mathcal{D}_1 \models \phi_1\right)\right) \wedge \ldots$$

where $\theta = f(\theta_0, \theta_1, \ldots, \theta_n)$. In each deterministic component $\mathcal{D}_i$, the probability of satisfying a local property ϕ_i being larger than a given value θ, denoted as $P_{\geq\theta_i}\left(\mathcal{D}_i \models \phi_i\right)$, can be verified by hypothesis testing.

With the setting above, we show how to solve two different problems as explained below. Firstly, if the objective is to obtain a probability of a system satisfying a global property, we first perform hypothesis testing to obtain the probability of each component satisfying the corresponding local property. Notice that we need to specify certain discrete levels from high to low, e.g., $l1 = 1$, $l2 = 0.99$, etc. Hypothesis testing is performed against those discrete levels (e.g., l_i) sequentially, until $P_{\geq l_i}(\mathcal{D}_i \models \phi_i)$ is true. The l_i is the approximated maximum probability of satisfying the local property. Afterwards, we perform probabilistic model checking based on the obtained results to calculate $P(\mathcal{M} \models \phi)$. Second, a global verification task, i.e., $P_{\geq\theta}(\mathcal{M} \models \phi)$, can be distributed into several local verification tasks, i.e., $P_{\geq\theta_i}\left(\mathcal{D}_i \models \phi_i\right)$ for each component $\mathcal{D}_i$. This relies on the assumptions that there is an inverse function of the given f, denoted by f^{-1}, such that each θ_i is calculated from $f^{-1}(\theta)$;

whenever the result of $P_{\geq\theta_i}(\mathcal{D}_i \models \phi_i)$ is false for a component $\mathcal{D}_i$, $P_{\geq\theta}(\mathcal{M} \models \phi)$ is also false, and vise versa.

Besides, it is still necessary to analyze how error bounds at system level are related to the ones at components.

Lemma 3.3.1 *Let α_c and β_c be the error bounds of verifying components $\mathcal{D}_c$. Let $\mathcal{C}$ be the set of all components of a system. Let the error bounds for the overall system be (α, β). α is bounded by $max\{\alpha_c \mid c \in \mathcal{C}\}$ and β is bounded by $\sum_{c\in\mathcal{C}} \beta_c$.*

Proof Let $\mathcal{M}$ be a system. We define the following hypothesis: Φ ("Accept $P_{\geq\theta}(\mathcal{M} \models \phi)$ is true"); Φ_c ("Accept $P_{\geq\theta_c}(\mathcal{D}_c \models \phi_c)$ is true"); $\neg\Phi$ ("Accept $P_{\geq\theta}(\mathcal{M} \models \phi)$ is false"); $\neg\Phi_c$ ("Accept $P_{\geq\theta_c}(\mathcal{D}_c \models \phi_c)$ is false"); Ψ ("In fact, $P_{\geq\theta}(\mathcal{M} \models \phi)$ is true"); Ψ_c ("In fact, $P_{\geq\theta_c}(\mathcal{D}_c \models \phi_c)$ is true"); $\neg\Psi$ ("In fact, $P_{\geq\theta}(\mathcal{M} \models \phi)$ is false"); and $\neg\Psi_c$ ("In fact, $P_{\geq\theta_c}(\mathcal{D}_c \models \phi_c)$ is true"). Let (α, β) be the error bounds for verifying $P_{\geq\theta}(\mathcal{M} \models \phi)$. By definition, α is $\Pr(\Psi \mid \neg\Phi)$, i.e., the probability of false negative.

$$\begin{aligned}
\alpha &= \Pr(\Psi \mid \exists\, c \in \mathcal{C}. \neg\Phi_c) \qquad - \star \\
&\leq max\{\Pr(\Psi \mid \neg\Phi_c) \mid c \in \mathcal{C}\} \\
&\leq max\{\Pr(\Psi_c \mid \neg\Phi_c) \mid c \in \mathcal{C}\} \\
&\leq max\{\alpha_c \mid c \in \mathcal{C}\}
\end{aligned}$$

($\star$) holds because "accepting that $P_{\geq\theta}(\mathcal{M} \models \phi)$ false" is equivalent to "accepting that there exists a $\mathcal{D}_c$ such that $P_{\geq\theta_c}(\mathcal{D}_c \models \phi_c)$ is false".

Similarly, β is $\Pr(\neg\Psi \mid \Phi)$, i.e., the probability of false positive.

$$\begin{aligned}
\beta &= \Pr(\neg\Psi \mid \Phi) = \Pr(\neg\Psi \mid \forall\, c.\ \Phi_c) \\
&= \Pr(\exists\, c \in \mathcal{C}.\ \neg\Psi_c \mid \forall\, c \in S.\ \Phi_c)
\end{aligned}$$

$$\begin{aligned} &\leq \sum_{i \in \mathcal{C}} \Pr(\neg\Psi_c \mid \forall\, c \in \mathcal{C}.\ \Phi_c) \\ &= \sum_{c \in \mathcal{C}} \Pr(\neg\Psi_c \mid \Phi_c) = \sum_{c \in \mathcal{C}} \beta_c \end{aligned}$$

□

The setting above is beneficial in terms of alleviating state space explosion problem and easily handling system non-determinism. However, the general form above depends on a few assumptions which are not easy to be satisfied in general. We assume verification of a global property ϕ can be divided into the verification against local properties $\phi_0, \phi_1, \ldots, \phi_n$; and the probability of satisfaction in the system is related to that of components by a function f. In general, such function is hard to attain, not to mention its inverse function f^{-1}.

Nonetheless, f can be obtained in some cases. A special case is when all local properties are the same as a global property, i.e., $\phi_i = \phi$ for the component $\mathcal{D}_i$. The function f is an evaluation of the MDP model which can be done by numerical methods e.g., value iteration. In the following, we show that the problem of reliability analysis is exactly such a special case and thus can be solved efficiently.

3.4 Reliability Analysis

The special case can readily apply to the software reliability analysis, i.e., reliability prediction and reliability distribution, by setting local properties and modeling function "f" explicitly. The property of interest in reliability analysis is the probability that a software has no failure. This is a global property, denoted by $P(\mathcal{M} \models \Box\neg \mathit{failure})$. The global property is actually composed by a set of local properties, i.e., probability of each module running successfully without any failure. An MDP model is built from the system architecture and users environment. Each module in the system can be treated as a component

$\mathcal{D}$ in the MDP. The transition probability between components, e.g., P_{ij}, is the probability from component i to component j, given that component i does not fail, i.e., P_{ij} is conditional on $P(\mathcal{D}_i \models \Box\neg\mathit{failure})$. Therefore, the transition probability of the MDP, e.g., from component i to component j, is $P(\mathcal{D}_i \models \Box\neg\mathit{failure})P_{ij}$. Here, conditional probability p_{ij} can be estimated from operational profile, which is the set of independent operations that a software system performs and their associated probabilities [99, 70]. By reachability checking on the MDP (e.g., via value iterations), the relationship between $P(\mathcal{M} \models \Box\neg\mathit{failure})$ and $P(\mathcal{D}_i \models \Box\neg\mathit{failure})$ for each component i can be established.

In the following, we present the details of applying the combination of hypothesis testing with probabilistic model checking to software reliability analysis. We first set up assumptions of our reliability analysis, followed by the construction of an MDP from system architecture and operational environment. The methodologies for reliability prediction and distribution are then introduced, respectively.

3.4.1 Assumptions and Threads to Validity

Considering reliability analysis as a special application, there are some underlying assumptions in terms of system reliability model and component failure behavior.

We use an MDP to model a system. Each state represents the execution of a single component of the application. Same as other Markov models, our model relies on the assumption of Markovian transfer of control among components, i.e., the probability distribution of future executing components depends only upon the present components.

In the standard set up of MDP, probabilities are used to model environment while non-determinism is used to model user decisions (e.g., available options) [108]. In our setting, non-determinism is used to resolve uncertain situations where the exact probability distributions are not available. This can be treated as a more general understanding on MDP by treating environment and user in the standard set up as a whole system. In fact, using MDP to resolve uncertain situations is not new. Nondeterminism has been used to model concurrency between processes by means of interleaving in probabilistic model checking [16].

Similar to [26, 70, 55, 50], our model also assumes that there is statistical independence among failures of the components. More specifically, the failure occurring within one component is neither the result of a failure occurring within another component, nor able to cause any other component to fail. However, in practice, different component may be heavily dependent, which may be a result of data exchange occurring through parameters or messages passing. In this work, we limit ourself to the applications whose components are failure independent. It appears to be a strict condition. However, the present assumption can be well satisfied considering many up-to-date large systems (e.g. CCS), within which the components are designed, implemented and tested independently. If any failure dependent components exist and can be grouped into one, the model would still work.

Moreover, in our reliability model, we assume that failure of any component will eventually lead to the failure of the system. For a system consisting of self-recovery or self-correction mechanism, there are some executions that end successfully after recovery. These scenarios are not considered as failure cases. Nonetheless they can be modeled in an MDP.

3.4.2 System Level Modeling

When to use non-deterministic choices? Compared to DTMCs, MDPs allow us to capture both probabilistic and non-deterministic behavior. A central issue is: when to use non-deterministic choices and when to use probabilistic choices. In general, probabilistic choices can be viewed as informed non-deterministic choices. That is, we use a non-deterministic choice when we have no definitive information on how the choice is resolved. For instance, if all we know is that there are two different outgoing transitions after executing a component C, we model the two transitions using a non-deterministic choice. If the choice is made locally, after testing C systematically, we learn the frequency of each outgoing transition and we can model C with a probabilistic choice. However, if the result of executing C is correlated to its inputs, there are two cases. If the inputs are the result of executing some other component K in the system, we may either model it as a non-deterministic choice conservatively; or we calculate the probability distribution of C's results based on the probability distribution of K's results. Notice that if we systematically test C and K as a whole, we may obtain a probability distribution of C's results. However, if the inputs of C are from an external environment which is difficult to predict (e.g., like the traffic of stock transactions), a non-deterministic choice would deliver a "safer" model.

In a nutshell, testing helps to turn non-deterministic choices into probabilistic choices. Ideally, we would like to learn probability distribution of all actions in the system. Nonetheless, due to the limited resources for testing or knowledge of the external environment, we often have to employ non-deterministic choices.

Example In the following, we illustrate the difference between non-deterministic choices and probabilistic choices using a simple example. Figure 3.3 presents a simplified fragment of the CCS system. There are two components $S1$ and $S2$, with reliability 0.8 and 0.9, respectively. The components execute simultaneously and independently. Assume that $S1$ is chosen 30% of the time. The corresponding DTMC model is shown on the left of the figure. The system reliability is then estimated as $0.3 \times 0.8 + 0.7 \times 0.9$, which is 0.87. There are two potential problems with the above prediction. First, the two components are running in parallel and hence a DTMC cannot truthfully model the system. Second, the transition probability is decided through operational profiles, which can only be obtained within limited tests in a testing environment before the system is deployed. The transition probability is hardly accurate since it is determined by the dynamic tasks loading at run-

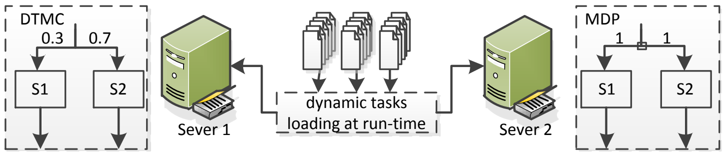


Figure 3.3: A system with run-time tasks distribution

time. If an MDP is used to model the system, as shown on the right, the reliability is calculated as 0.8 if $S1$ is chosen and 0.9 if $S2$ is chosen. The result shows that in the worst case, the system is only as reliable as $S1$. We can see that the result based on the MDP model is less dependent on the external environment. □

In the CCS example, depending on the run-time traffic of the transaction, a sophisticated dynamic load balancing algorithm is used to distribute transactions to the three CCS servers. There are 15 different combinations of choices, i.e., any of the three is chosen; two of them are chosen in a particular order; or all three of them are chosen in a particular order. It is challenging, if not impossible, to predict the probability distribution of the choices. Modeled as non-deterministic choices, these choices can be distinguished by different schedulers and the system reliability for each choice can be calculated and compared during value iteration.

Reliability Modeling The model of a system in our setting is an MDP $\mathcal{M} = (S, \mathit{init}, \mathit{Act}, \mathit{Pr})$. For each system component C (i.e., a self-contained piece of codes that can be independently designed, implemented, and tested), there is a pair of states, C and xC, in S which represent the state of C executing and the state right after C terminates, respectively. Further, S contains two absorbing states: a state of *Success* and a state of *Failure*. Here, the probability of a system always not getting *failure* state is the same as the probability of a system eventually reaching *success* state, denoted as $P(\mathcal{M} \models \diamond \mathit{Success})$. If the reliability of C is R_C, there is one probability distribution from C such that there is probability R_C to reach xC and probability $1 - R_C$ to reach *Failure*. Notice that if there is certain failure handling mechanism in the system (like WPF in the CCS system), the transition with probability $1 - R_C$ leads to a failure recovering state instead of the *Failure* state.

A simplified model of the CCS system is shown in Figure 3.4. The data in this figure are obtained based on test results on an early version of the system. Only the non-deterministic choices among the three CSS servers are shown and we further ignore the ordering among the three servers so as to save space. For compact presentation, we skip the *Failure* state. Instead, a node labelled as $C(R)$ is used to denote that the name of the component is C and the probability of reaching *Failure* is $1 - R$ and probability of R to transit to the successive components. The transition probability at each edge represents the usage information. Taking $Server1$ as an example, its reliability can be read off from the graph as 0.9972 and it has three outgoing transitions labeled with action η. If $Server1$ terminates successfully, it has a probability 0.584 of going to $Exit$, and a probability 0.416 of going to DB. If $Server1$ fails, it goes to $Server2$, which serves as a backup server for $Server1$. A backup transition is denoted by a dash line in the figure. In the corresponding MDP, the three transitions are labeled with $(\eta, 0.9972 \times 0.584)$, $(\eta, 0.9972 \times 0.416)$, and $(\eta, 1 - 0.9972)$, respectively. Note that if action $(\epsilon, 1)$ is chosen at $Enter$, there is a backup server available for $Server1_a$ or $Server2_a$. If action $(\delta, 1)$ or $(\omega, 1)$ is selected instead, there are no backup servers available, i.e., both servers are failed or the three servers are all running to finish a job.

3.4.3 Reliability Prediction

Given the hypothesis testing results of each component and an MDP, we then calculate the overall system reliability.

Based on Lemma 3.3.1, if all components are tested with the same error bounds (α_c, β_c), the error bounds of the overall system are $(\alpha_c, \beta_c \times N)$, where N is the number of components. Notice that system-level false positive β could be N times larger than the one at component level. This implies that the confidence of system level measurement is lower than that of

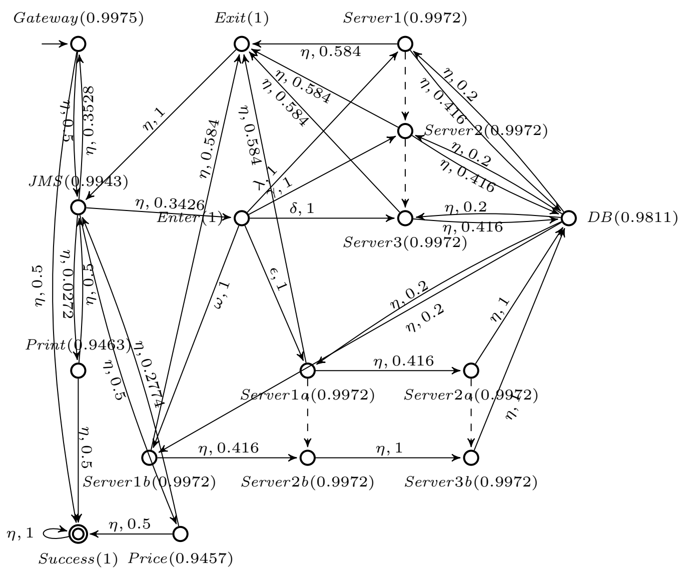


Figure 3.4: A simplified model for the CCS system

the components.

The maximum and minimum system reliabilities can by calculated by $P^{max}(\mathcal{M} \models \diamond \mathit{Success})$ and $P^{min}(\mathcal{M} \models \diamond \mathit{Success})$, respectively, and can be calculated using the value iteration method. Given the CCS model[1] in Figure 3.4, we obtain that the system reliability ranges from 0.95505 to 0.95729. The worst reliability is obtained with a scheduler such that three servers are running together, whereas the best reliability is obtained when only one is running. Detailed analysis is discussed in Section 3.5.

[1]The reliability are relatively low as they are obtained from test environment before the software released. The data for released version is confidential from the company. We have demonstrated that our method can still work and is accurate to a certain level by assuming relatively high reliability of each component (e.g., 99.999%), which is often the case after software released, available at [4].

3.4.4 Reliability Distribution

Our approach on distributing the overall system reliability requires two inputs: (1) a reliability requirement R on the overall system and a pair of error bounds (α, β); (2) a system model in the form of an MDP. The goal is to find a reliability requirement on some components so that the overall reliability requirement is satisfied. The resultant requirement on the components (e.g. component c), is in the form of a reliability probability R_c and a pair (α_c, β_c), which can be established using hypothesis testing on the smaller-scale component. In the following, we first show how to identify R_c and then how to obtain (α_c, β_c).

Given an MDP $\mathcal{M}$ and a scheduler δ, we can obtain a DTMC $\mathcal{M}_\delta$. The probability of reaching the *Success* state, $P(\mathcal{M}_\delta \models \diamond Success)$, is a polynomial function constituted by multiple variables (i.e., R_c for all relevant components). The constraint $P_{\geq R}(\mathcal{M}_\delta \models \diamond Success)$ then gives us the reliability requirement on each component, under the scheduling of δ. However, such a constraint is hardly useful in practice as the reliability of the components constraint each other. For simplification and making the results useful in practice, we assign different weights for the components participating in reliability distribution, by considering testing costs, e.g., testing time, and effort. In practice, the software can make use of some readily developed components. The components whose reliability is already known and rarely changes (e.g., a legacy component), will not participate in reliability distribution.

As a result, $P_{\geq R}(\mathcal{M}_\delta \models \diamond Success)$ becomes a polynomial inequality constituted by a variable x only. Using numerical methods, we can obtain a lower bound on x, which is the reliability requirement we need. Multiplying x with assigned weights, the reliability requirement for the components participating in reliability distribution can be obtained. Take the model in Figure 3.4 for example. Assume R is 0.98 and the scheduler δ_1 resolves the non-deterministic choice at state *Enter* by selecting action η. We further assume a unit weight assigned to all components, the calculated polynomial using our algorithm above is

$0.5x^1 + 0.16435x^3 + 0.05402x^5 + 0.11641x^7 - 0.09865x^8 \cdots \geq R$. When the iterations stop, the polynomial is accumulated up to the term of x^{160}. We omit the result of the terms here. By Newton's method [10], we obtain that the lower bound on x is 0.99601. This is the reliability requirement for every component since we assuming the same weight.

The above concerns only one scheduler. In general, there are multiple schedulers and we need to guarantee that the system reliability requirement is satisfied with any scheduler. Applying value iteration directly is very challenging as we need to compare polynomial functions representing the probability of reaching *Success* through different distributions from a state in each iteration. We thus adopt an alternative approach, i.e., we compute a lower bound on x for every scheduler and the maximum of the lower bounds gives us the minimum requirement on component reliability. Based on [16], only finitely many memoryless schedulers need to be considered. Our algorithm works as follows. First, an unvisited memoryless scheduler δ is selected. Next, we perform the value iteration method on $\mathcal{M}_\delta$. The following shows how the result vector V is updated. Assume scheduler δ chooses a distribution μ_s at state s: $V^{(n+1)}(s) = \sum_{t \in S} x \times \mu_s(t) \times V^{(n)}(t)$. Once a stopping condition is satisfied, we obtain a constraint $V(init) \geq R$ and solve the equation $V(init) - R = 0$ using Newton's method to obtain a lower bound on x so that $V(init) \geq R$ is true. The steps above are repeated for all memoryless schedulers.

The upper bound of memoryless schedulers equals to the product of the numbers of distributions for each state. If there are ten states and two of them both have 3 distributions and the rest has one, the number of schedulers is bounded by 9. Essentially, the more non-determinism there is, the more schedulers are to be considered. In practice, the number of schedulers is manageable as we are dealing with a high-level system model. Further, since schedulers are independent, we can parallelize the computation.

Next, we distribute the system error bounds (α, β). We assume that error bounds of each component are the same, denoted as (α', β'). Based on Lemma 3.1, we can deduce the following constraints: $\alpha' \leq \alpha$ and $\beta' \leq \frac{\beta}{N}$. Therefore, the two error bounds for each component are α and $\frac{\beta}{N}$, respectively. Given the model in Figure 3.4, we assume the system error bounds are $(0.02, 0.04)$. Since N is 8, the error bounds for each component are $(0.02, 0.005)$. β' might be considerably smaller than β if N is large. A very small error bound for hypothesis testing may lead to a large number of tests. The practical implication is that one can estimate system reliability by testing either larger components with larger error bounds (which is harder to test but needs less tests) or smaller components with smaller error bounds (which is easier to test but needs more tests).

3.5 Implementation and Evaluation

The proposed approach has been realized in a toolkit named RaPiD (Reli*a*bility Pred*i*ction and Distribution). RaPiD is a self-contained toolkit for reliability prediction and distribution, and it is publicly available at [4], with all case studies. It provides a user friendly interface to draw MDP models as well as fully automated methods to solve the reliability prediction and distribution problems. RaPiD is implemented with 4K lines of C# code. It uses a number of MATLAB (version 2009a) libraries to support powerful mathematical calculations as well as graph plotting functions. In the following, we apply RaPiD to study two real-world systems and obtain interesting results.

3.5.1 Reliability Prediction for Call Cross System

The CCS system has more than 300K lines of code. To predict the reliability of the CCS system, we first build an MDP model (as partly shown in Figure 3.4) and then test each of the components. Next, we apply RaPiD with a stopping criterion of 1E-5 and obtain the minimum/maximum system reliability of 0.95505/0.95729, respectively.

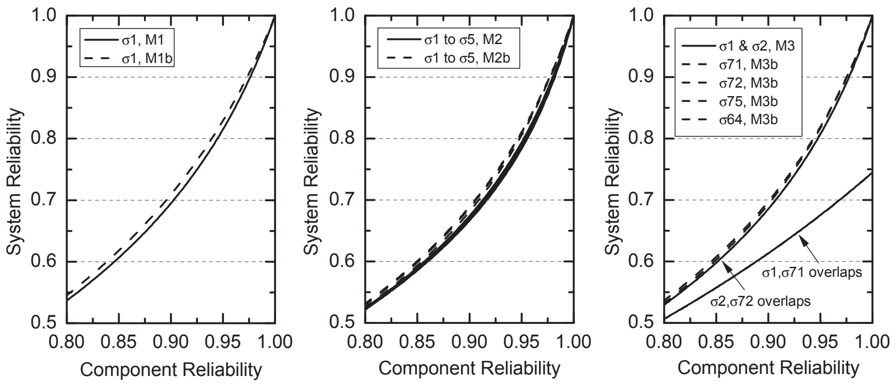


Figure 3.5: System reliability vs. component reliability for: (a) M1 and M1b; (b) M2 and M2b; (c) M3 and M3b

To compare the effectiveness of different models, i.e., the effect of non-deterministic choices for system reliability prediction, we build three models in total. Model $M1$ is a DTMC model, assuming that the probability distribution of all transitions among system components are known; $M2$ is as shown in Figure 3.4 where non-deterministic choices are used to model the run-time choice of the CCS servers, and $M3$ further introduces non-determinism by modeling the choices of going back to JMS or DB from a CCS server non-deterministically. To investigate the usefulness of the back-up servers in terms of system reliability, we modify these three models to incorporate transitions leading to a backup server. The resultant models are denoted as $M1b$, $M2b$ and $M3b$, respectively.

As presented, if we assume all components have the same reliability x, we can obtain system reliability as a polynomial function of x for each scheduler. Using RaPiD, we can plot the functions, as shown in Figure 3.5. Different from $M1$, $M2$ has 5 schedulers, and $M3$ has 160 schedulers in total since it has a state with 5 non-deterministic choices and another five states each with 2 non-deterministic choices. The dash lines are the corresponding functions for $M1b$, $M2b$ and $M3b$, respectively. Notice that many of functions are identical (e.g., for $M3$) and their plots overlap with each other.

Table 3.1: Reliability prediction for the three models

Name (#Schedulers)	**M1 (1)**	**M2 (5)**	**M3 (160)**
Min. Reliability	0.95568	0.95401	0.73149
Max. Reliability	0.95568	0.95568	0.96257

The following observations can be made based on the results. First, the difference between maximum reliability and minimum reliability becomes larger when the number of non-deterministic choices increases. For instance, Table 3.1 shows the differences for the three models.

Assuming that we need to show the system's reliability is at least 0.95, the result based on $M3$ is not conclusive, which suggests further testing is necessary so that we can learn the probability distribution of the non-deterministic choice (i.e., the transitions from CCS servers to JMS or DB) and make more accurate prediction. The result based on $M2$ on the other hand shows that we can make fairly accurate prediction without making any assumption on the run-time dynamic loading decisions, and this serves as a strong argument that the system is robust in the open dynamic stock market. If we superimpose the results for $M1$, $M2$ and $M3$ (i.e., the solid curves in Figure 3.5 (a), (b), (c)), we can find that the curve in graph (a) resides between the curves in graph (b), all the curves in (a) and (b) reside between the curves in graph (c). Second, in all three graphs, the dashed curves are higher than the corresponding solid curves. It implies that the system reliability indeed becomes higher by introducing a backup server. Nonetheless, it should be noticed that with the increase of component reliability, the gain of system reliability by introducing the backup server decreases. The results confirm (and quantify) our intuition.

3.5.2 Reliability Distribution for Call Cross System

RaPiD solves the reliability distribution problem using the approach documented in Section 3.4.4. There are totally 15 different choices of servers operation modes for the CCS,

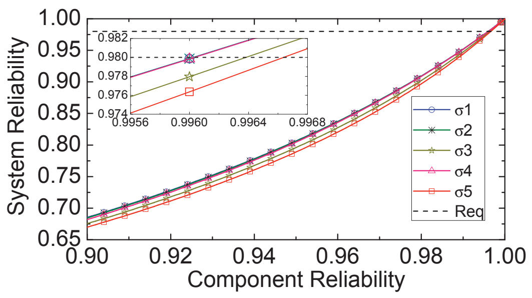


Figure 3.6: Reliability analysis result for the CCS

which indicates 15 schedulers existing in the model. Figure 3.6 visualizes the results for those five schedulers for the model shown in Figure 3.4, where scheduler $\sigma_1/\sigma_2/\sigma_3$ chooses action $\eta/\lambda/\gamma$ (i.e., to run $Server1/Server2/Server3$ only, respectively) at state $Enter$; scheduler σ_4 leads to state $Server1_a$ and possibly state $Server2_a$ subsequently (i.e., to run $Server1$ and $Server2$ at the same time); and scheduler σ_5 leads to a state where all three servers are running.

Assume that the system reliability requirement is 0.98 (i.e., the horizontal dash line in Figure 3.6). When all curves are above the dash line, we have sufficient component reliability to guarantee the system is reliable with any scheduler. Given the same component reliability, a scheduler is “better” if its corresponding system reliability is higher. Similarly, given the same system reliability, a scheduler is “better” if its corresponding component reliability is lower. Notice that when component reliability is low, the less servers are chosen to work simultaneously (e.g., scheduler σ_1 and σ_2), the higher the system reliability is achieved; and as component reliability becomes higher (e.g., > 0.95), the schedulers (e.g., σ_4) leading to more servers running outperform the others.

3.5.3 Reliability Distribution for Therapy Control System

The Burr Proton Therapy Center is a radiation therapy facility associated with the Massachusetts General Hospital in Boston. Proton therapy is a treatment controlling the dose of radiation delivered to the patients. High precision radiation therapy enables reduced dose to healthy tissue. Reliability assurance on such system is of utmost importance. One software component of the system, called the Therapy Control System (TCS), provides the users with all the control functions necessary. It is written primarily in 250K lines of C code. The TCS handles the storage and retrieval of patient data entry of prescriptions, scheduling of treatments, patient positioning and beam delivery.

A high-level view of the system is shown in Figure 3.7. The *Human/Computer Interface Layer* is a graphical user interface. The *Application Layer* is the core of the system. It consists of four modules: System Manager (SM), which controls operational modes, and event reporting; Beam Manager (BM), which handles allocation and operation of the proton beam transport; Treatment Manager (TM), which handles the patient treatment sequence from prescription to irradiation; and Database Manager (DM), which provides functions to allow the other modules to access to the database. The *Control Unit Layer* contains drivers for the physical devices, including Accelerator Control Unit (ACU), Energy Selection and Beam Transport Control Unit (ECU-BTCU), Positioning Control Unit (PCU), Treatment Control Unit (TCU), and Safety Control Unit (SCU). These are implemented in a table-driven fashion as low level state machines. RTServer is the information distribution server. It manages all communication among client processes, freeing all low-level network coding. RTH1 and DataDAQ are two data acquisition interfaces. RTH1 is in charge of ACU and SCU, while DataDAQ is in charge of the rest of control units. The service starts with any beam service requirements sent via Human/Computer Interface Layer, and completes after the BM generating the irradiation summary.

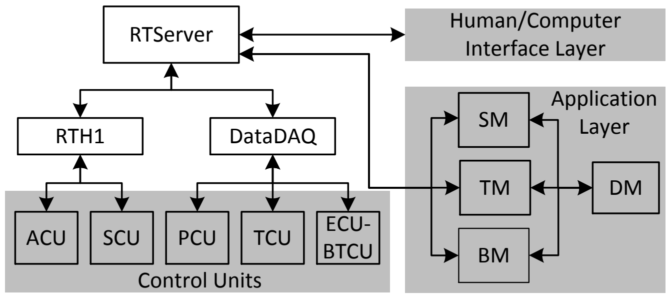


Figure 3.7: Architecture of a Therapy Control System

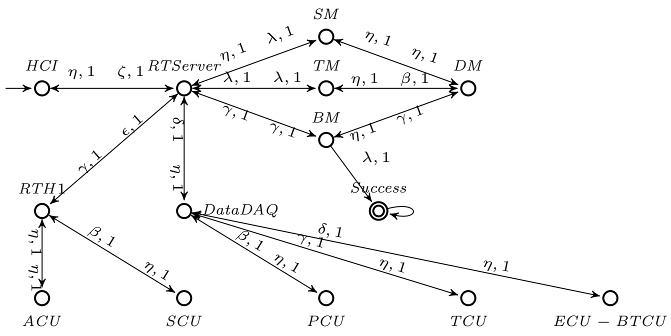


Figure 3.8: A reliability model for the TCS

The TCS system serves as an excellent case study for our reliability distribution method as it is presently undergoing a software upgrade. Some components are to be revised or replaced. Given the requirement on system level reliability, it is desirable to generate concrete reliability requirement for newly developed components so that they are contracted properly. The challenge in applying RaPiD is that there is no precise information on transition probabilities. The reason is that testing the system is highly complicated, as there are 5 concurrent machines and many interrupting events generated by hardware control units. However less complex safety mechanisms are in place to mitigate any error. As a result, transition probability and the system are modeled by non-deterministic choices only. Nonetheless, we show that we can still obtain some useful results.

A simplified MDP model of the system is shown in Figure 3.8. As the reliability of each

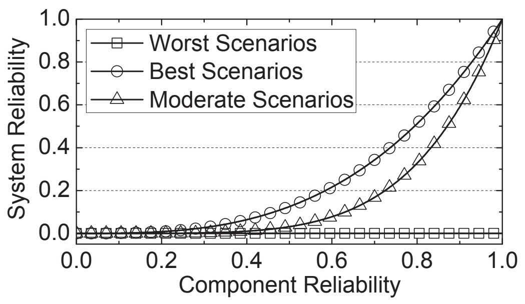


Figure 3.9: Reliability analysis result for the TCS

component is not available, they are omitted. Although there are 2,592 schedulers, only three different system reliability functions of component reliability exist. By further analyzing the corresponding schedulers reported by RaPiD, we can identify three typical workflows of the system that result in the three scenarios, respectively.

In Figure 3.9, the plot of the worst scenarios is a horizontal line of zeros. It implies that for any component reliability, the system level reliability is zero, i.e., the system cannot reach the *Success* state. This set of schedulers always chooses $RTH1$ or $DataDAQ$ from *RTServer* and hence state *Success* is never reached. The best scenarios include the cases within which the transition directly goes from *RTServer* to BM and then reaches *Success*. In real situation, this is an extreme case that the system sends beam treatments request directly to the BM, which completes the job, and then the whole transition finishes successfully. Moderate scenarios contain cases within which *RTServer* goes to SM, TM, or BM and then goes to DM; and afterwards, DM reports data back to BM and lastly the *Success* state is reached. As we can see, without any testing results on the system, we are able to find out the worst/best system reliability in respect to components reliability. This information is particularly helpful in the early stage of software development, as the system developers can use the results as a guideline on how to test the system or how to improve the system reliability, e.g., by improving the feedback communication from control unit layer to

RTServer.

3.5.4 Scalability

RaPiD is efficient in our case studies. The reliability prediction took 0.03 seconds for the CCS, and the reliability distribution took 42 seconds for the CCS (with 160 schedulers) and 628 seconds for the TCS (with 2,592 schedulers). To further test the scalability of RaPiD, we evaluate RaPiD's reliability prediction and distribution using 5 benchmark MDP models from [78] as well as randomly generated models (with 1K to 50K states and the number of states for having multiple transitions are sampled from a uniform distribution). The results show that RaPiD is able to handle 14K states per second on average (with termination threshold as relative difference 1.0E-6) in calculating reachability probability. Reliability distribution (with a bound 600 on the number of terms in the obtained polynomial) is slightly slower due to maintaining/updating/solving the polynomial functions. The data is obtained using a PC with Intel(R) Core(TM) i7CPU at 2.80 GHz and 8 GB of RAM. This evaluation is for a single MDP. However, when we use parallel composition of a set of MDPs to model distributed systems communication, the state space is the product of its local system state spaces, which is challenged by scalability issue. We have proposed a solution to reduce the state space, as detailed in Chapter 6.

3.6 Related Work

Hypothesis testing has gained its popularity in probabilistic model checking [29, 136, 84] as it can overcome state space explosion problem. Its applications were limited to deterministic systems in the early stage. In recent years, it has been extended to non-deterministic systems [19, 66, 82]. [19] provides an approach that only limits to spurious non-determinism that introduced by the commutativity of concurrently executed transition in compositional setting. [66, 82] apply learning techniques to search for near optimal schedulers so as to convert MDP to an induced Markov Chain. The effectiveness in searching for the near-optimal schedulers is decided by several parameters for controlling the maximum number of schedulers to evaluate each time and the effectiveness of learning process. All those parameters shall be tuned by users. Instead, our approach lifts up non-determinism to a level that exact methods can be used.

Our work can also be viewed as performance analysis of programs with probabilities. Geldenhuys et al. [45] have provided an approach to calculate the probabilities of code executions quantitatively. This work can be seen as a form of profiling. In the context of weakest preconditions, McIver and Morgan have several work in studying probabilities and non-determinism in programs, e.g., in [97]. In our work, the probability of transition is known as a priori.

Our framework has been applied to reliability prediction and distribution. For reliability prediction, it is related to software architecture based reliability evaluation [26, 70, 55, 50] and software scenario based reliability evaluation [112, 134]. Compared to the above work, our approach handles systems with model parameters which are hard to obtain. Furthermore, it can quantify the accuracy of component reliability with the help of hypothesis testing. Some recent studies focus on dynamically changing parameters in reliability models and updating parameters based on run-time data [95, 43, 38]. [95] develops a Δ evaluation method to update system reliability if one component reliability changes in run-time by reusing previous evaluation results. This method is limited to systems where only one component reliability changes at once. [43] assesses the reliability inquiries expressed in Probabilistic Computation Tree Logic at pre-calculation stage, whereas DTMCs' transition parameters are substituted into pre-obtained formulae at run-time. [38] continuously corrects and updates DTMCs' parameters in a run-time fashion based on a Bayesian technique. These are not applicable until the software is released. Our reliability model tackles the issue on missing run-time information before system deployment. In addition, our remedy relies on modeling hard-to-predict run-time behaviors as non-deterministic choices so as to obtain reliability measurement which is independent of the dynamic environment. Hypothesis testing has also been used in estimating system reliability [132], which treats programs as black boxes.

Our reliability distribution problem is similar but slightly different from the reliability allocation problem by solving an optimization problem, e.g., in [106, 69, 92]. The optimization goals are to minimize the amount of testing time while ensuring that a system is sufficiently reliable. [92] also discusses a way to minimize the number of remaining faults given a fixed amount of testing efforts. Our method on reliability distribution focuses on the minimization of component reliability requirement. To the best of authors' knowledge, our work is the first on applying the combination of probabilistic model checking with hypothesis testing to reliability prediction and distribution. Moreover, we have established the system error bounds from components error bounds and vice versa.

3.7 Summary

In this chapter, a framework that combines both hypothesis testing and probabilistic model checking is proposed for the reliability analysis of non-deterministic systems. This method applies hypothesis testing to deterministic system components and uses the probabilistic model checking techniques to resolve non-determinism. We have performed case studies on a stock trading system and a hospital therapy control system to demonstrate its usefulness.

Chapter 4

Reliability Analysis of an Ambient Assisted Living System with RaPiD

The rapid increase of aging population in all industrialized societies has raised serious problems, e.g., creating enormous costs for the intensive care of elderly people. The Ambient Assisted Living (AAL) system, as a promising solution is designed to assist their independent living [101, 130]. In such systems, sensors and inference engines are widely used to perceive environment changes and user intentions. Applications and actuators are triggered accordingly to provide necessary assistance to the user. However, lack of reliability guarantees prevents AAL systems to be widely utilized. For instance, a failure of prompting a reminder could harm the user's life, e.g., a call-for-help alert failed to prompt when the elder falls may leave him/her unattended for a long and eventually leading him/her to death. Thus, it is essential to conduct reliability analysis and provide quantitative guarantees on the system before deployment. In this chapter, we introduce our reliability analysis toolkit called RaPiD (Reli*a*bility Pred*i*ction and Distribution), which is an implementation based on the techniques proposed in Chapter 3, and then demonstrate the usage and usefulness of RaPiD on ALL systems by studying on a real smart healthcare system [18].

4.1 Introduction

An Ambient Assisted Living (AAL) system is considered reliable if the reminder service is correctly delivered to the right person at the right usage scenario. It is a challenging task to analyze the reliability of such systems for the following reasons. First of all, AAL systems are inherently complex. They are usually composed of multiple layers of software and hardware components which have limited capability and accuracy. Previous research [102, 111] reported that the inherent inaccurate and unreliable low-level sensors are often used to detect context information from the environment. This is probably on the cost-efficiency considerations, i.e., low-capability but cheap sensors are often selected due to budget constraints. Furthermore, AAL systems rely heavily on different types of wireless networks with different reliability. For instance, sensors transmit signals via Zigbee, a low-cost low-power wireless mesh network, while software components and actuators exchange messages using more reliable networks such as WLAN. Moreover, human errors, e.g., a user may forget to wear the RFID tags, could also cause the failure of the system.

Though failures of such systems are unavoidable, it is critical to manage the reliability in an acceptable level. Systematic reliability analysis of such AAL systems is thus in great need. AAL systems are highly user-oriented such that the system can automatically react according to users' behaviors. However, due to unpredictable users' activities (i.e., non-deterministic), their system behaviors are often too complicated to be analyzed before its deployment. This nature makes our techniques introduced in Chapter 3 good candidates in analyzing ALL systems, i.e., reliability analysis can be conducted based on a system model in the form of a Markov decision process (MDP). We develop a toolkit called RaPiD to assist automatic reliability analysis. Using probabilistic model checking techniques, e.g., (parametric) reachability checking, three fundamental and highly important reliability issues can be investigated in RaPiD for non-deterministic systems: (1) RaPiD can synthesize the overall system reliability given the reliability values of system components; (2) given a reliability requirement on the overall system, RaPiD can distribute the reliability requirement to each component; and (3) RaPiD can identify the component that affects the system reliability most significantly.

With RaPiD, we have performed reliability analysis on a smart health care system, called AMUPADH. It is a typical AAL system designed for the healthcare of elderly dementia people. Our evaluation shows that the overall system reliability is below 0.4. For the system to reach a reliability of 0.4, each Wi-Fi network related node should have a reliability of at least 0.9. Our analysis also concludes that it is impossible for the system reliability to reach 0.5 based on the current design. There is such a scenario that the system always fails to identify the person that is performing an abnormal activity. Thus, half of the chances the reminder will be sent to a wrong person. Lastly, the sensitivity analysis suggests that the overal system reliability can be improved most efficiently by improving the Wi-Fi network. In the end, the analysis results are reported to AMUPADH designers who confirmed their consistency with the real data collected from the hospital. They redesigned the activity recognition rules and added more nodes in the Wi-Fi network to increase its reliability. As a result, we show that our reliability analysis approach can provide good estimation of system reliability and is particularly helpful in identifying the critical component inside the system.

Organization The remainder of the chapter reports the details of this case study. It is organized as follows. Section 4.2 introduces our reliability analysis toolkit RaPiD. Section 4.3 presents the case study system, AMUPADH. Sections 4.4 reports details on constructing reliability models from AMUPADH. Section 4.5 presents the reliability analysis results for AMUPADH. Section 4.6 discusses the related work.

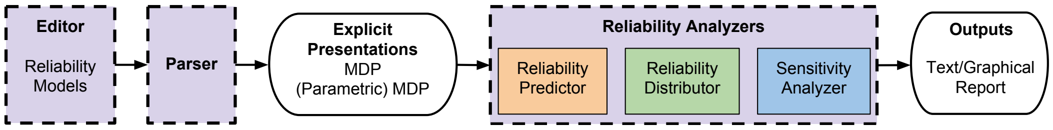


Figure 4.1: RaPiD architecture

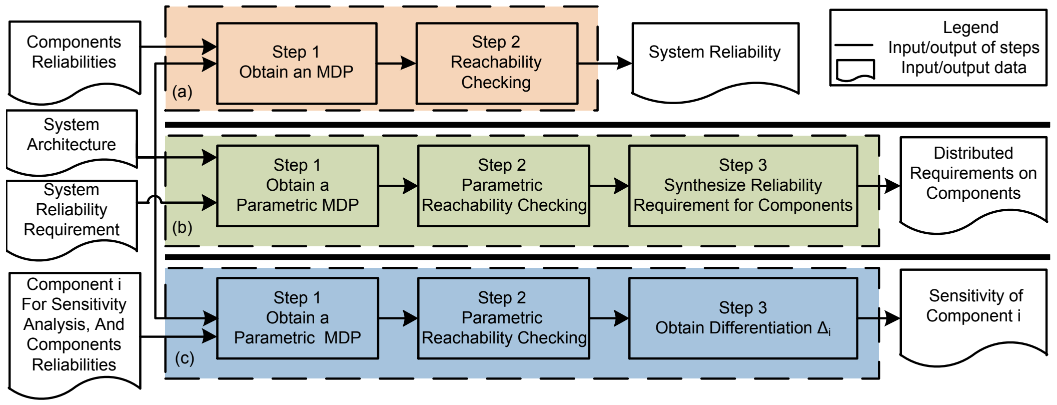

Components Reliabilities
System Architecture
System Reliability Requirement
Component i For Sensitivity Analysis, And Components Reliabilities
Step 1 Obtain an MDP
Step 2 Reachability Checking
(a)
System Reliability
Legend
Input/output of steps
Input/output data
Step 1 Obtain a Parametric MDP
Step 2 Parametric Reachability Checking
Step 3 Synthesize Reliability Requirement for Components
(b)
Distributed Requirements on Components
Step 1 Obtain a Parametric MDP
Step 2 Parametric Reachability Checking
Step 3 Obtain Differentiation Δi
(c)
Sensitivity of Component i

Figure 4.2: RaPiD overview: (a) reliability prediction; (b) reliability distribution; (c) sensitivity analysis

4.2 RaPiD: A Toolkit for Reliability Analysis

RaPiD is implemented to provide a friendly user interface to draw reliability models and fully automated methods to the reliability analysis (i.e., reliability prediction, reliability distribution, and sensitivity analysis) for non-determistic systems. Starting from 2012, RaPiD has come to a stable stage with solid testing and has been applied to analyze several real-world systems. The tool (along with a screencast illustrating its usage) is available at [4].

RaPiD consists of three main components, i.e., Editor, Parser, and Reliability Analyzers. Figure 4.1 shows the architecture of RaPiD. In the graphical editor, a system model for reliability analysis (i.e., reliability model) is first created, from which the explicit model, i.e., MDP, is then automatically obtained by the parser. The core algorithms for reliability analysis are in the reliability analyzers including reliability predictor, reliability distributor and sensitivity analyzer. After the analysis, RaPiD presents results in terms of a text report

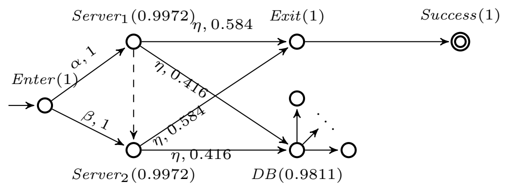


Figure 4.3: An example of reliability model

or a graphical plot. With different input knowledge and requirements, RaPiD can readily perform three different software reliability with its respective analyzers. More detailed technical backgrounds of RaPiD are referred to Chapter 3. In the following, we introduce the input reliability model of RaPiD and then present the workflow for its three reliability analyzers.

4.2.1 Reliability Model

Extended from Cheung's model [26], the model for reliability analysis in our setting is an MDP $\mathcal{M}$, which can be built from the system architecture and user environments. It can support the modeling of both probabilistic and non-deterministic behaviors. In this model, states and transitions are two key elements, which can be constructed as follows.

States Each system component C is a self-contained piece of codes that can be independently designed, implemented, and tested. Each system component represents a state in MDP. In addition, there are two absorbing states: a state of *Success* and a state of *Failure*. A simple model is demonstrated in Figure 4.3. For compact reliability model presentation, we skip the *Failure* state. Instead, a node labelled as $C(R_c)$ is used to denote a component C with a probability of R_c to transit to the successive components, and a probability of $1 - R$ to reach *Failure* state.

Transitions The transition probability in a probability distribution at each edge represents the usage information, e.g., P_{ij}, is the probability from component i to component j, given that component i does not fail. An MDP can have more than one probability distributions at a state. This feature enables RaPiD to model all possible operating environments/situations explicitly. RaPiD supports the modeling of failure handling mechanism in the system, where the transition with probability $1 - R_C$ leads to a failure recovering state instead of the *Failure* state. Taking $Server_1$ in Figure 4.3 as an example, its reliability can be read off from the graph as 0.9972 and it has two outgoing transitions labeled with action η. If $Server_1$ terminates successfully, it will have a probability 0.584 of going to *Exit*, and a probability 0.416 of going to database DB. If $Server_1$ fails, it goes to $Server_2$, which serves as a backup server for $Server_1$. This backup transition is denoted by the dash line in the figure.

4.2.2 Reliability Analysis

Depending on the different input knowledge and requirements, RaPiD can readily address three different questions on software reliability.

- "What is the overall system reliability if the reliability of each component is known, considering all possible user behaviors and unreliable factors?"

This is the problem of ***reliability prediction***. This question is to be answered necessarily before system deployment since end users would prefer to know how reliable the system is. The reliability value of each component and an MDP model of system are required for predicting the overall system reliability. Reliability prediction is equivalent to checking the probability that the system never fails. It is then transformed into a problem of calculating the probability of reaching accepting nodes from an initial state to a goal state s on an MDP model $\mathcal{M}$, denoted as $Pr(\mathcal{M}, s)$. Here, RaPiD performs value iteration approach [16] to compute the reachability probabilities. Unlike DTMC approaches, the result here is a probability range due to the non-deterministic behaviors. The upper bound is the system reliability corresponding to the best scenario in the systems; whereas, the lower bound corresponds to the worst scenario.

- "What is the reliability required on a certain component if there is a requirement on overall system reliability?"

This is the problem of ***reliability distribution***. Addressing this issue is useful because we can have specific quantitative requirements on the selection of software and hardware components, whose qualities are often cost-sensitive. The reliability distribution analysis shown in Figure 4.2 (b) needs two inputs: (1) a reliability requirement R on the overall system; (2) a parametric MDP model $\mathcal{M}$. RaPiD considers only memoryless schedulers that have already been proven to be enough for probability reachability analysis in MDPs [16]. Given a scheduler σ, we can obtain the system reliability (i.e., $Pr(\mathcal{M}_\sigma, s)$) as a polynomial function of x and its associated inequality, e.g., $0.5x^1 + 0.16435x^3 + 0.05402x^5 + \cdots \geq R$. To solve the constraints on an individual component, RaPiD uses Newton's method, due to its fast convergence rate to the solution/root. RaPiD calculates the lower bounds on x for finitely many schedulers among which the maximum value gives us the minimum requirement on component reliability.

- "Which component is most critical to system reliability among all system components?"

The answer can be addressed via ***sensitivity analysis***. This analysis is essential to improve the overall system reliability effectively with limited resource. For example, if a system is shown to be not reliable enough based on current components, it is desirable to prioritize the components such that reliability improvement of a higher priority component would result in better improvement on overall system reliability. Sensitivity analysis requires all component reliabilities to be known in advance and an indication on which one of those components needs to be evaluated, as shown in Figure 4.2 (c). The sensitivity Δ_i of component i with reliability R_i is defined as a partial derivation of system reliability R, i.e., $\Delta_i = \frac{Pr(\mathcal{M},s)}{\delta R_i}$. Here, $Pr(\mathcal{M}, s)$ are polynomials obtained via parametric reachability analysis. RaPiD has equipped with polynomial solvers to solve these differential equations. Noted that, similar to the reliability distribution, due to multiple schedulers in an MDP, results for sensitivity analysis ranges from its maximum and minimum sensitivity values.

4.3 AMUPADH System

Dementia is a progressive, disabling, and chronic disease common in elderly people. Elders with dementia often have declining short-term memory and have difficulties in remembering necessary activities of daily living. However, they are able to live independently in assisted living facilities with little supervision. An ambient assisted living system for elderly dementia people's healthcare (AMUPADH) system is designed for this purpose by providing necessary assistance in the form of reminders.

AMUPADH is a project initiated in Singapore to design smart healthcare systems for monitoring and assisting the daily living of elderly people with dementia. This project started with three months' visits to PeaceHeaven[1] nursing home for collecting requirements. By observing the patients' daily life and interviewing nurses/doctors, two critical issues associated with dementia patients are raised which are sleeping behavior in bedroom and showering behavior in bathroom. With 21 months research and development focusing on providing assistance on these two scenarios, the system was finally deployed in the nursing home to be tested with real dementia patient users for 6 months.

Preliminary data collected in the trial shows that the importance of system reliability is un-

[1]Located at 9 Upper Changi Road North, Singapore, 507706. Tel: +65-65465678.

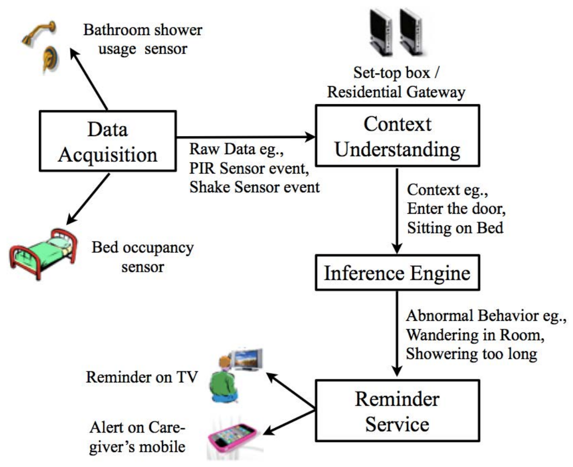

Bathroom shower usage sensor
Set-top box / Residential Gateway
Data Acquisition
Context Understanding
Raw Data eg., PIR Sensor event, Shake Sensor event
Context eg., Enter the door, Sitting on Bed
Bed occupancy sensor
Inference Engine
Abnormal Behavior eg., Wandering in Room, Showering too long
Reminder on TV
Reminder Service
Alert on Care-giver's mobile

Figure 4.4: An overview of AMUPADH system design

derestimated. Unreliability caused by sensor/device failures, network issues and unforeseen human errors draws considerable attentions of the stakeholders.

4.3.1 System Components

The design of the system is shown in Figure 4.4. It mainly consists of three sub-systems, *Data Acquisition* component containing various sensors, the *Context Processing* and *Inference Engine* components based on first order logic rules and *Reminder System* for rendering suitable reminder services to the patients.

- **Data Acquisition** In the system, multiple sensors are deployed to acquire information from the home environment. For example, if someone turns on the shower tap, the shake sensor on shower pipe will be triggered and change its status to *Unstationary*. A signal is generated and then sent to the central system via a Zigbee network. AMUPADH adopts a multi-modal sensor[2] design for user monitoring. This is due to users'

[2]Multi-modal sensor also known as sensor fusion is the combining of sensory data from disparate sources such that the resulting information is more accurate than using the sources individually.

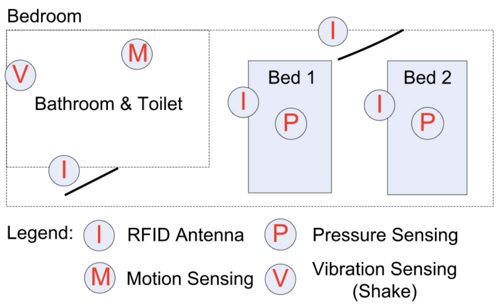


Figure 4.5: Sensor deployment in a PeaceHeaven nursing home

privacy concerns, video cameras are refused in bedrooms.

The PeaceHeaven nursing home has 13 separate Resident Living Areas (RLAs), each designed as an individual home-like environment. Selected rooms for AMUPADH system deployment are equipped with two/three beds with a shower facility. Three rooms are selected for deployment; and each room is shared by 2 or 3 people. Figure 4.5 shows an exemplar sensor deployment for a twin shared room. The pressure sensor under a bed mattress is used for detecting sitting/lying behavior, while the RFID readers are for detecting the identity of the person near the location. In the bath room, a motion sensor detects human presence in the room, while the vibration sensors are attached to water pipes and the soap dispenser for detecting their usages.

- **Context Processing and Reasoning** Upon receiving a sensor signal, the central system translates it into low-level context *sensor events* i.e., a signal *unstationery* from shake sensor on shower pipe is translated to “Shower Tap On". Different low-level contexts are provided from different sensors. They are aggregated in the inference engine for reasoning and generating high-level contexts, *activities*. This task is performed by evaluating predefined activity recognition rules based on prior knowledge of user behavior. A typical rule is like: if shower tap is on and lasts for 30 minutes, at the meantime a PIR sensor detects movements of someone in the washroom; an abnormal behavior, *showering for too long* is recognized, then a message will be sent to the server indicating some patient is in the abnormal state of showering for too long. The messages are sent out via a shared bus within the central system. Note that, AMUPADH aims for a multi-user sharing environment which is a challenging topic in the activity recognition area. In fact, it is not only important to know which activity is being carried on but also who is doing this activity. This adds complexity to define the activity recognition rules. In the case that if the patient's identity is missing in a rule, the activity could be recognized for a different person, causing a subsequent reminder to be prompted to a wrong patient. Our previous work [89, 83] discussed this issue in details. In this work, when defining the reliability of the inference engine, we take this factor into account.

- **Reminding System** The reminding system listens to the messages sent from the inference engine and decides which reminder service to render. For example, upon receiving the message *Activity.error.ShowerTooLong.patientA*, the system will invoke the service of playing a preloaded sound reminder on bluetooth speaker located in the shower room correspondingly. In this case, the message is transferred via bluetooth technology. In general, different message transmitting technologies are used for different rendering devices. For instance, for reminders on mobile phones, messages are transmitted through 3G network, while for iPad case, a small home wide Wi-Fi network is used.

4.3.2 Six Reminding Scenarios

In AMUPADH system, there are six reminding scenarios targeting at providing assistance for six abnormal behavior of elderly patients with dementia.

- **Using Wrong Bed (UWB)** Since a room in the PeaceHeaven nursing home is shared by 2-3 patients, some of them, especially the new residences, tend to lie on a bed without recognizing whether it is his/her own bed. This behavior is detected by the bed pressure sensor and RFID reader. The reminder will be prompted by a Bluetooth speaker beside bed or an iPad on the wall asking the patient to leave the bed.

- **Sitting on Bed for Too Long (SBTL)** Some of the agitated patients often have sleeping problems. They are easily bothered and irritated by the environment. A typical symptom is that the patient will get up at midnight and sit on the bed for very long time until assisted by nurses/caregivers. The abnormal scenario is captured by bed pressure sensor and a timer in the mini-server. A reminder will be prompted using similar devices as UWB scenario whispering the person to sleep or send an alert to nurse's PC console/ mobile phone.
- **Shower No Soap (SNS)** Due to memory loss, dementia patients constantly forget the normal steps of performing an daily activity. In the taking shower activity, the patient could forget what to do next right after the shower tap is turned on. It is reported by the nurses that some of the patients finish the shower very fast without applying soap. Concerned about the personal hygiene, patients presenting this behavior need help. Vibration sensors on the shower pipe and soap dispenser are used to capture the activity. A reminder instructing the person to use soap will be prompted by a Bluetooth speaker.
- **Showering for Too Long (STL)** Similar to the SNS scenario, some patients will stand under the shower head for a long time. This is a critical issue that exposing in the water for a long time could cause the patient to faint. If not helped immediately, it can even cause death. Similar sensors and devices are used as SNS scenario to help the patient stop showering.
- **Tap Not Off (TNO)** It is often the case that dementia patients forget to turn off the tap after showering. In order to save water and energy, this scenario is detected by a

RFID reader, a motion sensor and two vibration sensors. A reminder is prompted on proper device according to patient location asking someone to turn off the tap.

- **Wandering in Washroom (WiW)** Caused by memory loss, it is possible for the patient to forget at any step during showering. Thus, a wandering behavior is also typical and patients need assistance in such cases. The wandering behavior is monitored by a RFID reader and a motion sensor in the washroom. A leave-washroom reminder will be prompted to the person.

In fact, taking shower turns out to be the most concerned issue of nursing elderly patients with dementia. In PeaceHeaven, the nurses need to monitor the showering activity of every patient. Considering the ratio of nurses to patients is 1:15, it creates a heavy burden to nurses. To alleviate the problem, a two-level reminding solution is provided in AMUPADH. When the system recognizes an abnormal behavior, it will prompt a reminder to the patient. If the problem remains, an alert will be sent to the nurse's mobile phone or PC console to raise her attention.

4.4 Modeling AMUPADH System

AAL systems are user driven such that the system behavior contains non-determinism due to the unpredictable user behavior. Thus, MDP is chosen as the modeling formalism in this work. Compared to DTMC, MDP allows us to capture both probabilistic and non-deterministic behavior. A central issue in the AMUPADH system modeling is that when to use non-deterministic choices over probabilistic choices. As we mentioned in Section 3.4.2 in Chapter 3, *probabilistic choices can be viewed as informed non-deterministic choices.* That is, we use a non-deterministic choice when we have no definitive information on how the choice is resolved. Although the construction of reliability models have already been introduced in

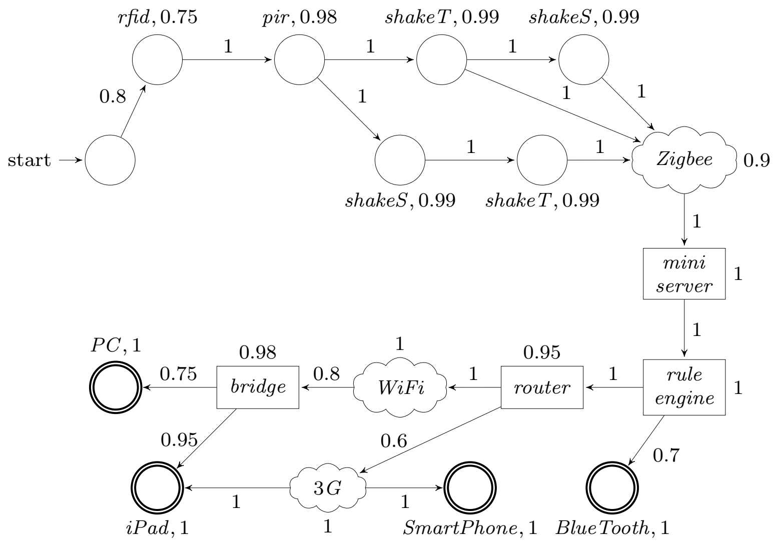


Figure 4.6: Bathroom scenario: Tap Not Off (TNO)

Section 3.4.2, we demonstrate the details on how to model an AAL system based on those contraction rules.

In practice, it turns out to be unrealistic to model all the scenarios using one MDP model considering the complexity and readability. Thus, we split the model into six models according to different scenarios by duplicating the same components. In the following, we shall explain the modeling processes. Scenario TNO as shown in Figure 4.6 is taken as an example for its richness of involved components. In order to build the reliability model, there are three major elements to extract from AAL systems, i.e., the nodes, the transitions and the reliability values.

4.4.0.1 Nodes

Typically, in an AAL system, the sources of unreliability could be failure of sensors and network devices, error in softwares and connection loss/transmission failure in networks.

Thus, in an MDP model of an AAL System, nodes are the abstraction of sensors, software components and network devices. To decide which device/component is necessary to be modeled, we need to analyze the activity recognition rules. In TNO case, four sensors are used for recognizing this behavior as introduced in Section 4.3.2. Besides, there are multiple choices of playing this reminder, e.g., playing on an iPad where reminder command is received via Wi-Fi network or on a Smart Phone through 3G network. Thus, the four sensors, iPad, smart phone, Wi-Fi network and 3G network need to be included in the model as nodes. Similarly, the Zigbee network, mini server and the rule engine need to be modeled as well.

In Figure 4.6, circle nodes denote sensors, square nodes denote hardware devices and cloud shape nodes denote networks. Double circled nodes are accepting nodes representing a reminder is successfully delivered. The different shapes of nodes are used to show the different types of components. In the MDP model, they are treated the same.

4.4.0.2 Transitions

In AAL systems, there are usually two types of relations between nodes, happen-before and message-forwarding relations. Happen-before relation usually exists among sensors saying that some sensor is triggered earlier than the others. It is able to be derived from analyzing the temporal relations between sensors according to their spatial distribution. For example, in Figure 4.5 in Page 60, the RFID reader near to the bedroom door is triggered earlier than the other sensors assuming the system starts with all users outside. Thus, in the MDP model, it should be placed in front of the rest of sensor nodes.

However, sometimes, the happen-before relation is not deterministic. For instance, for the model shown in Figure 4.6, there is no specific triggering orders between shake sensors on the tap and soap dispenser. Thus, we need to enumerate all the possible orderings. Besides, there is one rule deciding this abnormal behavior based on shower-pipe vibration sensor only. Thus, there is a transition link from *ShakeT* to *Zigbee* making the ordering asymmetric. Our experience suggests that it is better to enumerate all the possible transition orders in the initial model, especially when there are multiple rules defined for recognizing the same scenario.

As for message-forwarding relations, they are extracted from the system design. For example, in the TNO model, the messages are sent to the mini server via Zigbee network. Thus, a Zigbee node is placed between the sensors and mini server. The transitions between nodes denote the direction of message transmission. Similar methods are applied for the rest of the transitions.

4.4.0.3 Reliability and Transition Probability

The final step is to label the nodes and transitions with probability values. Nodes are labelled with reliability values of the corresponding devices. For transitions, there are different cases. At the initial node, the outgoing transitions usually representing the user behavior. In the TNO case shown in Figure 4.6, there is 20% of time, the user will throw the RFID tags away (result drawn from an experiment conducted by the engineers). Thus, initially, there are only 0.8 probability leading to the next node. Additionally, the happen-before relations are usually non-deterministic choices with no specific probabilities due to randomness of user activities, thus by default, we assign the value 1. As for forwarding relations, due to the signal strength, transitions to/from network nodes have different reliability values. Transitions from Wi-Fi node to bridge node has the reliability of 0.8 since the bridge is placed on the wall outside the bedroom. The nurse PC in common area is further away from the bedroom, thus the transition from bridge to PC is as low as 0.75.

In our case study, these reliability values are provided by system engineers. Fortunately,

Reliability	**UWB**	**SBTL**	**SNS**	**STL**	**TNO**	**WiW**
Number of Schedulers in MDP	32	24	32	16	64	16
Max Reliability	0.3744	0.4190	0.3670	0.3707	0.3707	0.3707
Min Reliability	0.2956	0.2463	0.2897	0.2927	0.2897	0.2927
Calculation Time	<1 ms					

Table 4.1: Results of reliability prediction

Reliability Requirement	**Nodes**	**UWB**	**SBTL**	**SNS**	**STL**	**TNO**	**WiW**
0.4	Network	0.854	0.904	0.913	0.911	0.911	0.911
	Sensor	0.886	0.938	0.941	0.923	0.923	0.923
0.5	Network	0.914	-	0.965	0.963	0.963	0.963
	Sensor	0.996	-	0.995	0.994	0.994	0.994
Time(s)		3.45	2.68	3.86	1.87	11.00	2.35

Table 4.2: Results of reliability distribution

AMUPADH system has been deployed in a real user environment for data collection. During the 6 months trial deployment and 3 consecutive months, 24 hours data collecting, the engineers are able to log every details of how the system works. By comparing to the ground truth (manually logged by nurses in the nursing home) and conducting statistical analysis, they are able to provide a good estimation of each component's reliability.

4.5 Reliability Analysis on AMUPADH

Based on the MDP models constructed in Section 4.4, we use the RaPiD tool for reliability analysis. All the experiments are carried out on a PC with 2.7 GHz Intel CPU, 8 GB memory and 64-bit Windows 7 operating system. In the following, we listed the settings and results of three groups of experiments respectively. Interested readers are referred to [1] for details.

4.5.1 Reliability Prediction

As shown in Table 4.1, the reliability of six scenarios ranges from 25% to 40% with different scheduler which is quite low considering using the system at home with no human supervision.

One general observation from this experiment is that the system uses the RFID sensors in many places for identity tracking. However, the RFID sensors have the lowest reliability among all the sensors. In fact, due to budget issues, these RFID readers used in the system have a half meter detecting radius which are much cheaper but have a lower accuracy than others with a larger radius. Besides, the dementia patients tend to remove their RFID tags from time to time causing the failure of identity tracking. It is also an important lesson learned that AAL systems cannot rely on patients to provide the critical information. Thus, we suggest the designers to replace the RFID reader to the one with a larger detecting range or the one does not require a tag.

Besides, the six reminding scenarios have similar reliability except for SBTL case. By a careful examination, we discover that the rule defined for SBTL has an error. Because the engineer failed to put the user's identity information into the rule's condition, this reminder will be sent to the wrong user in 50% chance. This shows that reliability analysis is also useful in identifying system bugs.

4.5.2 Reliability Distribution Analysis

Further, we explored how to distribute reliability on certain components so as to reach an overall reliability requirement. Two groups of nodes are tested which are sensor nodes and network related nodes. By fixing reliability of the network related nodes, we calculated the distribution on sensor nodes and vice versa. We consider a uniform distribution (where all the nodes have the equal weight) among sensors since they have similar reliability.

As shown in Table 4.2, it requires each network related node to have a reliability of 0.913 in order for all the scenarios to achieve a reliability of 0.4. However, it is impossible when the requirement raises to 0.5. The reason also points to a failure rule in SBTL scenario. Moreover, it becomes unrealistic that if we expect the system reliability to reach 0.5 based on the current design, it requires highly accurate and stable sensors which are of much higher cost. For example, a short range RFID reader may cost a few hundred US dollars, but for a higher range, the price raises to a few thousand US dollars. Considering the AAL systems are to be deployed in normal homes, the cost becomes unaffordable for normal families. Thus, this group of experiment results requests AMUPADH designers to rethink about the system design rather than simply replace sensors.

Besides, it is still intuitive to ask the question that which node or group of nodes affects the system reliability more than the others? If improvements are made on such node(s), it will be more efficient. Thus, we seek the answer from sensitivity analysis.

4.5.3 Sensitivity Analysis Experiments

There are multiple schedulers in each MDP model as shown in Table 4.2. Due to page limits, we present one typical scheduler in this experiments. The UWB scenario refers to Section 4.3.2 is modeled in Figure 4.7. The path connected by thick black links are the target scheduler. It is a typical case which relies on two RFID sensors and multiple other sensors. The iPad case is chosen since playing reminders on iPad is the most common way in practice.

Two nodes and a bundle of nodes are chosen for the experiment which are RFID reader node, Zigbee network node, and bundle of nodes related to Wi-Fi network (If network reliability is improved, the reliability of message transmission paths i.e., bridge and transition to bridge, will also be improved.). Figure 4.8 (a) shows the reliability distribution on these nodes.

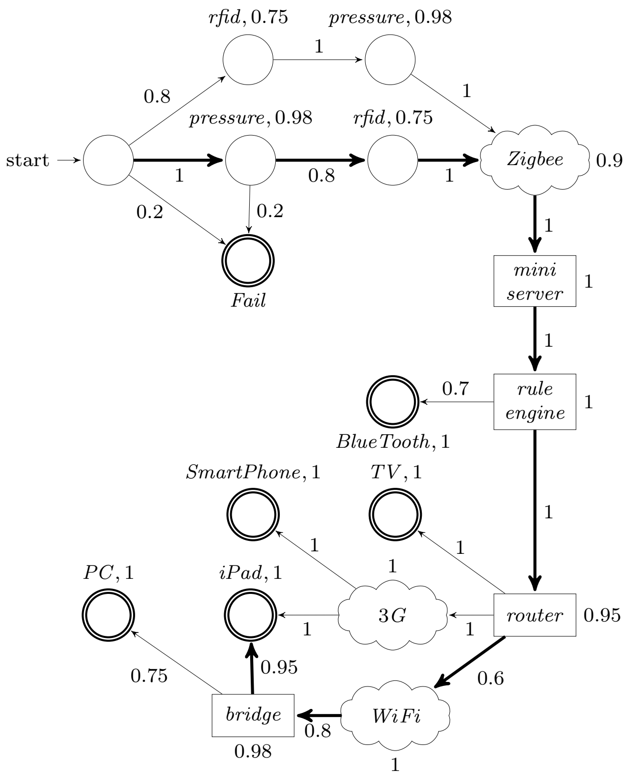
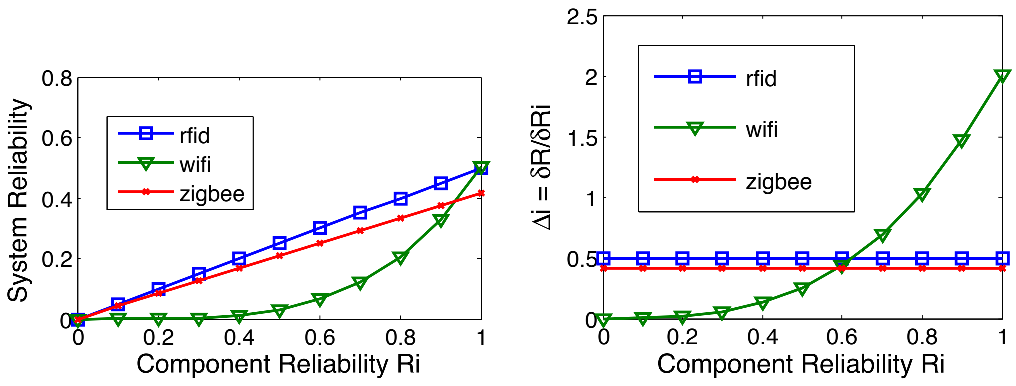


Figure 4.7: Bedroom scenario: Using Wrong Bed (UWB)

As we can see, improvement on RFID reader node and Wi-Fi bundle can achieve a higher reliability than Zigbee node.

Recall that the sensitivity measures how quickly the system reliability changes when one of its component's reliability changes. Therefore, the sensitivity analysis is conducted with respect to a particular component. The reliability of this component varies and those of the others are kept constant. In Figure 4.8 (b), two horizontal lines indicates that system reliability increases constantly (i.e., in a linear rate) with respect to the component reliability. It further indicates that when the reliability of these nodes are greater then 0.7, increasing reliability of nodes in Wi-Fi bundle can achieve better improvement than other nodes. In practice, increasing the reliability of a network might be cheaper than purchasing a sensor



(a) Reliability Distribution (b) Sensitivity Result

Figure 4.8: Reliabiltiy Analysis on Nodes for UWB

with higher reliability, e.g., placing more bridges along the path.

4.5.4 Discussions

Although we have only carried out detailed analysis in RaPiD on one AAL system, AMUPADH, we contend that the approach considered here is widely applicable to many other similar systems for the following reasons.

Modeling Applicability Layered architecture and multi-sensor platform are widely adopted in AAL systems like AMUPADH system because of the low cost and high extensibility, e.g., plug and play [6]. Thus, the modeling techniques introduced in this case study are easily adaptable to other similar systems by extracting necessary information from the system design and codes about scenarios, transition relations among related sensors and actuators and the internal reasoning mechanism. Our approach requires the knowledge and experience on modeling system in MDP models, which makes it not easy for engineers to use without necessary background.

Usefulness The above experiments show that our approach can give a good estimation on the overall system reliability. Upon a reliability requirement on overall system, we can provide suggestions of the least requirements on certain nodes. Additionally, our approach provides useful guidance on improving the system effectively, such that relatively more effort and fund can be spent on those critical components in budget concerned systems. Thus, our approach is able to solve practical problems and give useful suggestions for the improvement of the system design.

4.6 Related Work

The tools like SMERFS [39], SoRel [73], CASRE [90, 91], and RAT [85] can be used to estimate software reliability using the failure data to drive the software reliability growth models (SRGM) [81, 72]. Due to the need for learning SRGM, all those tools can only be used very late in the software life-cycle. Moreover, they treat software as a black box without utilizing any architecture information.

In performing white box based software reliability prediction using software architecture or usages scenarios, though some techniques have been proposed in [76, 103, 51, 55, 106, 128], there are only two tools available. One is SREPT [110], the other is a simple GUI toolkit that has been developed for the purpose of education demonstration [131]. All those approaches are based on deterministic reliability models, which have not considered the non-deterministic factors. Our toolkit RaPiD are designed for non-deterministic systems based on Markov decision processes.

Although RaPiD uses probabilistic model checking techniques, unlike the general probabilistic model checkers including PRISM [78] , MRMC [74] and LiQuor [27], RaPiD is tailored to software reliability analysis. In particular, it provides fully automated solutions to reliability prediction and reliability distribution problems as well as sensitivity analysis.

For the applications, there are also some works on analyzing reliability of complex systems in the literature. Reliability analysis by modeling system architecture as discrete time Markov chain (DTMC) is first proposed by Cheung [26] in 1980. It has been applied in various case studies, e.g., Gokhale et al. [52] analyzed a system called SHARPE by constructing a DTMC and found out the relation between system reliability and fault density per subsystem. Goseva et al. [53] performed reliability predicting and sensitivity analysis on a system of the European Space Agency. Wang et al. [129] analyzed a stock market system by constructing DTMC and predicted the system reliability. However, to the best of our knowledge, there is no reliability analysis has been conducted on any AAL system which involves not only system reactions but also non-deterministic human behavior. In such a complex system, probability distribution of transitions among system components are hard to obtain. To overcome this challenges, we choose Markov decision process as the reliability model over discrete time Markov chain. Furthermore, most of the works are focusing on predicting reliability of current systems while we contribute more on finding the best solutions to improve system reliability via reliability distribution and sensitivity analysis.

4.7 Summary

In this chapter, we have introduced our reliability analysis toolkit called RaPiD, and then demonstrate the usage and usefulness of RaPiD on a ambient assisted living room system by studying on a real smart healthcare system [18]. The reliability models are manually constructed from the design and implementation of the systems. Three important reliability analysis activities are performed on this system, including reliability prediction, reliability distribution and sensitivity analysis. The experiments show that the overall system reliability is hardly able to reach 50%. It is also suggested that to improve the reliability of Wi-Fi network will be more efficient to improve the system reliability.

Chapter 5

Improved Reachability Analysis based on SCC Reduction

One fundamental task in quantitative analysis of probabilistic system is to decide the probability of reaching a set of target states in the system. We refer to this as the *reachability analysis* problem. It is a crucial step in probabilistic model checking as well as model-based reliability analysis. Thus, improving the efficiency of reachability analysis can be considerably beneficial. In this chapter, we introduce graph-based reduction techniques to improve the reachability analysis for two typical Markov models, i.e., Markov decision process (MDP) and discrete time Markov chain (DTMC).

5.1 Introduction

Reachability analysis in two Markov models MDPs are extensively used to model a system with both non-determinism and probabilistic behavior. A DTMC can be considered as a special form of an MDP with unique reachability probability since it has only one probability distribution at each state. Given an MDP and a set of target states, a variable can be created for each state to present the probability of that state reaching the target states. There are two main methods to calculate or approximate the values of these variables [16]. One method encodes the probabilistic reachability problem into a linear optimization problem where each probability distribution is encoded into an inequality. Thus, the goal is to maximize or minimize the sum of the variables. It should be noted that the state-of-the-art linear solvers are limited to small cases with up to thousands of variables. However, a practical Markov model is often resulted from parallel composition of several MDPs/DTMCs, which would have an even larger number of states.

The other method is based on value iteration by finding a better approximation iteratively until the result satisfies a certain stopping criterion, and performs generally better in system with a large number of states [16]. The approximation of the variable of a state needs to be updated whenever any of its successive states are changed. When there are loops in an MDP, this approach tends to require many iterations before converging to a value, and thus lead to *slow convergence*.

A motivating example Fig. 5.1 (a) shows an example of a simple MDP with loops among states s_1, s_2 and s_3. Suppose the task is to calculate the probability of reaching state s_4 from state s_0. If the approximation in s_2 is updated during the k^{th} iteration, the approximation in s_1 will be updated during the $(k+1)^{th}$ iteration as s_2 is successive to s_1. The update of s_1 will trigger s_3 to update its value subsequently, which requires s_2 to be updated again. This iteration can only be stopped by enforcing a stopping criterion, thus one major issue associated with such an approach is that the difference between the approximated and 'actual' probabilities remains unknown even after the iteration is stopped [44]. On the other hand, in an acyclic MDP in Figure 5.1 (b), each state will be visited only a few rounds for backward calculation without iterations. In this case, the exact maximum and minimum

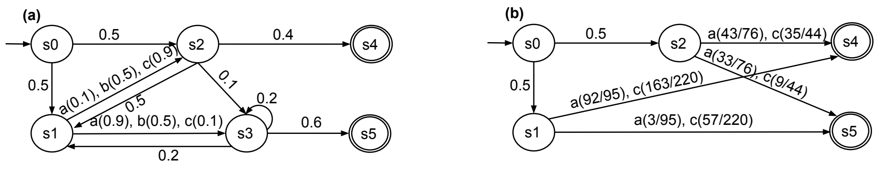


Figure 5.1: Running examples of (a) an MDP and (b) an acyclic MDP

probabilities can be calculated without the necessity of approximation. DTMC can be taken as a special MDP that has exact one probability distribution at each state. Hence, the same issues due to the existence of loops also exist in DTMCs. Therefore, in this work, we are motivated to improve reachability analysis by removing loops in both Markov models.

SCCs elimination Some foundation has been established by recent works on the elimination of strongly connected components (SCCs) in DTMC [8, 5]. To remove the loops, SCCs are first identified, and the transition probabilities from every input to output states of each SCC are calculated. The loops can then be removed by connecting the inputs to the outputs with the computed probability transitions (i.e., *abstraction* of SCC). After all the SCCs are abstracted, the whole model becomes acyclic. With such an acyclic set of states, value iteration can be used to calculate the probability from initial states to the target states.

Our work is inspired by the above mentioned SCC reduction methods. We propose approaches based on divide-and-conquer algorithms that divide an SCC into arbitrary parts and resolve the loops in each small part. Instead of simple extensions, our divide-and-conquer approaches are tailored to the particular structure i.e., DTMC or MDP, respectively, and integrated with other techniques. More details are introduced as follows.

Our SCCs reduction algorithm in DTMC For DTMC, we divide each SCC having a large number of states to several smaller partitions. For each partition, abstract transitions from its input to output are calculated via solving linear equations. Here we use Gauss-

Jordan elimination [7]. Further, the states in each partition which are not input states will be removed, and thus the states in the SCC can be reduced. Afterwards, the new SCC is ready for next iteration of divide and conquer. This procedure for each SCC will be done iteratively until any of the following three criteria is satisfied. First, there is no more loop in the reduced SCC. Then this part will be left alone since it is already acyclic. Second, the number of remaining states in reduced SCC is small enough to be solved via a linear solver. Third, the last iteration does not reduce any states. In the second and third scenarios, the final SCC will be solved via linear equation again, and final abstract transitions will be generated. After all loops in SCCs are resolved, the whole DTMC becomes acyclic, and value iteration is used to calculate the probability from initial states to targets. Since the abstract transitions from each partition's input states to output states are determined by the partition itself and independent to other partitions, multi-cores or distributed computers can be straightforwardly used here to solve each partition simultaneously, which makes the verification faster.

Our SCCs reduction algorithm in MDP In an MDP, however, eliminating loops is particularly challenging due to the existence of multiple probability distributions. The number of memoryless schedulers of an MDP increases exponentially with the number of the states that have multiple probability distributions. During the abstraction of a group of states, a probability distribution must be calculated under different memoryless scheduler in the group. As a result, the total number of probability distributions can increase exponentially after abstraction. Therefore, directly applying the existing approaches [8, 5, 117] to MDPs is often infeasible.

To overcome this challenge, we propose another divide-and-conquer algorithm to remove loops in MDP. For each SCC, we first construct *blocks*, i.e., each state in the SCC forms a block. By solving sets of linear equations, new probability distributions can be calculated from each block to replace the loops without varying the overall reachability probabilities. With the new equivalent probability distributions, the new block will be free of loops, and have the same reachability probabilities with the original model. We repeatedly merge a few blocks into one block, and eliminate loops in this new block by performing the above abstraction until only one block is left in the SCC. After the reduction for all the SCCs, the remaining acyclic MDP can be solved efficiently via the standard value iteration approach. After this reduction, the maximum and minimum reachability probabilities of the reduced MDP remain unchanged as compared with those of the original MDP. As introduced earlier, reducing states in SCCs of an MDP may result in exponentially many probability distributions, and our algorithm is thus designed to eliminate redundant or infeasible probability distributions on-the-fly to achieve better performance. The underlying observation is that, a probability distribution will not affect the maximum or the minimum reachability probability, if it is not a vertex of the convex hull of a set of probability distributions.

Organization The rest of this chapter is structured as follows. The relevant background information is recalled in Section 5.2; Our SCC reduction approach for DTMC is introduced in Section 5.3 and for MDP is in Section 5.4; the evaluations for both approaches are reported in Section 5.5; and related work is surveyed in Section 5.6.

5.2 Preliminaries

In this section, we recall some relevant background knowledge.

5.2.1 Some Graph Definitions on Markov Models

Discrete time Markov chains (DTMCs) and Markov decision processes (MDPs) are the most popular Markov models. DTMCs model a purely probabilistic system, while Markov decision processes (MDPs) can model systems exhibiting both probabilistic and non-deterministic behavior [16]. The two kinds of Markov models and their associated reachability computations are detailed in Chapter 2. In the following part, we formalize the definitions of input states and output states of a group of states, and strongly connected components in a Markov model. We will only introduce those definitions for general MDPs, which can be easily adapted to DTMCs, since DTMCs can be treated as special MDPs as mentioned previously.

Input and Output States Similar to [8, 5, 117], in an MDP $\mathcal{M} = (S, \mathit{init}, \mathit{Act}, \mathit{Pr})$, we define input and output states of a group of states $\mathcal{K} \subseteq \mathcal{S}$ as follows.

$$\begin{aligned} \mathit{Inp}(\mathcal{K}) &= \{s' \in \mathcal{K} \mid \exists\, s \in S \backslash \mathcal{K}, \exists\, a \in \mathit{Act} \cdot \mathit{Pr}(s,a)(s') > 0\}, \\ \mathit{Out}(\mathcal{K}) &= \{s' \in S \backslash \mathcal{K} \mid \exists\, s \in \mathcal{K}, \exists\, a \in \mathit{Act} \cdot \mathit{Pr}(s,a)(s') > 0\}. \end{aligned}$$

Here, the set of input states of $\mathcal{K}$, $\mathit{Inp}(\mathcal{K})$, contains the states in $\mathcal{K}$ that have incoming transitions from states outside $\mathcal{K}$; and the set of output states of $\mathcal{K}$, $\mathit{Out}(\mathcal{K})$, contains all the states outside $\mathcal{K}$ that have direct incoming transitions from states in $\mathcal{K}$. In addition, without loss of generality, if a group contains the initial state *init*, we include *init* to its input states (with an imaginary transition leading to *init* from outside).

Strongly Connected Components A set of states $C \subseteq S$ is called *strongly connected* in $\mathcal{M}$ iff $\forall\, s, s' \in C$, there exists a finite path $\pi = \langle s_0, s_1, \cdots, s_n \rangle$ satisfying $s_0 = s \wedge s_n = s' \wedge \forall\, i \in [0, n], s_i \in C$. *Strongly connected components* (SCCs) are the maximal sets of the strongly connected states. All SCCs can be automatically identified by Tarjan's approach [126], with a complexity of $\mathcal{O}(n + l)$, where n and l are the numbers of states and transitions, respectively.

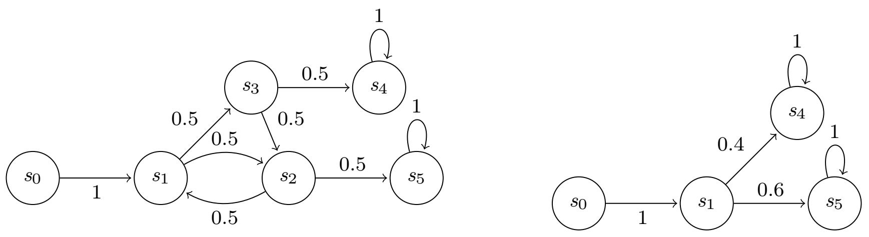
In Figure 5.1 (a) in Page 77, $\{s_0\}$, $\{s_4\}$, $\{s_5\}$ and $\{s_1, s_2, s_3\}$ are the SCCs in the model. We define SCCs as trivial if they do not have any outgoing transitions (e.g., $\{s_4\}$, $\{s_5\}$) or are not involved in loops (e.g., $\{s_0\}$, an SCC of one single state without any loop). As a result, $\{s_1, s_2, s_3\}$ is the only nontrivial SCC in Figure 5.1 (a). An MDP is considered *acyclic* if it contains only trivial SCCs. An example of an acyclic MDP is shown in Figure 5.1 (b). Note that an acyclic MDP may still have absorbing states, but it does not affect the computation of reachability probabilities.

In addition, we define an *adjacent group* (AG) $D \subseteq S$ such that $\exists\, s \in D, \forall\, s' \in D \wedge s' \neq s$, there is a finite path $\pi = \langle s_0, s_1, \cdots, s_n \rangle$ satisfying $s_0 = s \wedge s_n = s' \wedge \forall\, i \in [0, n], s_i \in D$, and s is called *root* state in D. AGs are more complex, for example, in Figure 5.1 (a) $\{s_0, s_1, s_2\}$ and $\{s_1, s_2, s_4\}$ are AGs and there are other possible combinations. Note that a set of states like $\{s_0, s_1, s_4\}$ is not a valid AG because there is no such a root state. Strongly connected subgraphs are AGs but the reverse is not always true, e.g., $\{s_0, s_1, s_3\}$ is an AG but not a *connected* subgraph.

5.2.2 States Abstraction and Gauss-Jordan Elimination

Given a set $\mathcal{K}$, if a state is not an input state, we call it as an inner state. We can eliminate all the inner states of $\mathcal{K}$ by calculating the direct transition probabilities from $Inp(\mathcal{K})$ to $Out(\mathcal{K})$. This process is called *abstraction*. It eliminates all loops in $\mathcal{K}$, and meanwhile, *preserves* the maximum and minimum reachability probabilities from inputs to the outputs of $\mathcal{K}$. There are known algorithms in [5, 117] to perform the abstraction. However, they are only applicable to DTMCs. For an MDP $\mathcal{M}$, the abstraction are performed on its induced DTMC, $\mathcal{M}_\sigma$, with its every memoryless schedulers σ. In the following, we introduce the details on how to achieve the abstraction on a given DTMC.

Given a DTMC $(S, init, Pr)$ and a group of states $\mathcal{K} \subseteq S$, $\mathcal{K}$ can be abstracted by calculating the transition probability from $Inp(\mathcal{K})$ to $Out(\mathcal{K})$. According to the proof in [5], the abstraction of any arbitrary set of states is independent from others, and the abstract



(a) Before Abstraction (b) After Abstraction

Figure 5.2: States abstraction via Gauss-Jordan Elimination in a small DTMC

transitions **preserve** the probability of reaching target states G.

One example of the abstraction is in Figure 5.2. Figure 5.2 (a) is the original DTMC, which has one SCC $\mathcal{K} = \{s_1, s_2, s_3\}$. $\mathit{Inp}(\mathcal{K}) = \{s_1\}$ and $\mathit{Out}(\mathcal{K}) = \{s_4, s_5\}$. In order to abstract $\mathcal{K}$[1], the probability from $\mathit{Inp}(\mathcal{K})$ to each state $s_{out} \in \mathit{Out}(\mathcal{K})$ should be calculated. Theoretically, the calculation from an SCC's inputs to outputs can be solved via linear equations or value iteration approaches[2]. However, for value iteration approach, since there could be several output states in $\mathit{Out}(\mathcal{K})$, we have to separately calculate the probability from input states to each output state. If there are many output states, this method could be inefficient. In addition, the existence of loops still causes slow convergence issue. Furthermore, using value iteration, there will be some errors because of the user-defined precision, but there is no way to know the error bounds. Therefore, we use a specific linear equation solving technique: Gauss-Jordan elimination [7] to do the abstraction.

Gauss-Jordan elimination is an algorithm for getting matrices in reduced row echelon form that placing zeros above and below each pivot [7]. Here, we briefly introduce how it works in our setting.

Assume there are m states in a set of states $\mathcal{K}$, and $|\mathit{Out}(\mathcal{K})| = n$. Then two matrices A

[1]Here we take an SCC as an example. Actually this abstraction can be applied to arbitrary set of states, according to [5].

[2]Different from our previous discussion which focuses the calculation from the initial state to targets, here we discuss the probability from input states to every output state of an SCC.

and B, containing linear equations information of all transitions in $\mathcal{K}$, are first introduced as follows.

$$A(i,j) = \begin{cases} 1, & \text{if } i = j; \\ -Pr(i,j), & \text{otherwise.} \end{cases} \qquad\qquad B(i,k) = -Pr(i,k).$$

Here, A is an $m \times m$ square matrix. $A(i,j)$ is a negative value of probability of transition from i^{th} state to j^{th} state in $\mathcal{K}$ if $i \neq j$. The diagonal elements of A are filled by 1. This records the transition relationship within $\mathcal{K}$. B is an $m \times n$ matrix to record the transition relationship from $\mathcal{K}$ to $Out(\mathcal{K})$. k represents the k^{th} state in $Out(\mathcal{K})$.

Next, augmenting the square matrix A with matrix B, we will have $[A \mid B]$. Gauss-Jordan elimination on $[A \mid B]$ will then produces $[I \mid C]$. Here, I is the identity matrix with 1s on the main diagonal and 0s elsewhere. The new transition probability e.g., $Pr'(i,k)$, stores the transition probability from i^{th} state in $\mathcal{K}$ and k^{th} state in $Out(\mathcal{K})$, which is actually $-C(i,k)$. Now take Figure 5.2 (a) as an example. Its $[A \mid B]$ and resulting $[I \mid C]$ are listed as follows.

$$[A \mid B] = \left[\begin{array}{ccc|cc} 1 & -0.5 & -0,5 & 0 & 0 \\ -0.5 & 1 & 0 & 0 & -0.5 \\ 0 & -0.5 & 1 & -0.5 & 0 \end{array}\right]; \quad [I \mid C] = \left[\begin{array}{ccc|cc} 1 & 0 & 0 & -0.4 & -0.6 \\ 0 & 1 & 0 & -0.2 & -0.8 \\ 0 & 0 & 1 & -0.6 & -0.4 \end{array}\right]$$

Here the transitions from all the states in $\mathcal{K}$ to $Out(\mathcal{K})$ are obtained. Note that those states which are not in $Inp(\mathcal{K})$ will be removed. Therefore we are just interested in the new transitions from $Inp(\mathcal{K})$ to $Out(\mathcal{K})$, which are

$$Pr'(s_1, s_4) = 0.4; \quad Pr'(s_1, s_5) = 0.6;$$

Accordingly, we can obtain a new probability distribution as $\{0.4 \mapsto s_4, 0.6 \mapsto s_5\}$ at state

s_1 in the abstracted DTMC, which is shown in Figure 5.2 (b). Given a group of states $\mathcal{K}$ in an DTMC, this abstraction procedure is defined as a method $Abs(\mathcal{K})$.

Note that in practice, most transition matrices in probabilistic model checking have a very sparse structure that contains a large number of zeros. A compressed-row representation [120] can be further adopted as the data structure to present matrices in Gauss-Jordan elimination.

5.3 SCC Reduction on Discrete Time Markov Chains

As illustrated in the introduction, for a large DTMC with complicated loop structure, both linear equations and value iteration method are ineffective, even unworkable. In this section, we propose a divide-and-conquer approach which tackles the above-mentioned problem. Our main idea is similar to work [8, 5], which transfers the original DTMC to an acyclic one by abstracting SCCs recursively so as to reduce the number of state and loops.

Intuitively, our approach divides large SCCs into smaller partitions, each of which will be solved via Gauss-Jordan elimination independently. Through this approach, loops will be eliminated. Afterwards, value iteration method is used to decide the final probability of reaching targets. In the following, we introduce our algorithm in details.

5.3.1 Overall Algorithm

Given a DTMC $\mathcal{M} = (S, init, Pr)$ and target states $G \subseteq S$, the probability of reaching G, denoted as $\mathcal{P}(s_{init} \models \Diamond G)$, can be solved by Algorithm 1. Note that B is an input parameter, which indicates SCCs having more than B states should be divided. $Abs(K)$ is defined in Section 5.2.2. *ValueIteration*$(\mathcal{M}, G)$ indicates calculating the probability of reaching G via value iteration. The procedure of the algorithm is explained in the following.

Algorithm 1: SCC Reduction in a DTMC via Divide-and-Conquer

```
input : A DTMC $\mathcal{M} = (S, init, Pr)$, target states $G \subseteq S$ and a Bound $B$
output: $\mathcal{P}(s_{init} \models \Diamond G)$
1  Let $\mathcal{C}$ be the set of all nontrivial SCCs in $\mathcal{M}$;
2  while $|\mathcal{C}| > 0$ do
3      Let $\mathcal{D} \in \mathcal{C}$;
4      if $|\mathcal{D}| \leq B \vee Out(\mathcal{D}) \leq 1$ then
5          $Abs(\mathcal{D})$ and $\mathcal{C} \leftarrow \mathcal{C} \backslash \mathcal{D}$
6      else
7          Divide $\mathcal{D}$ into a set of AGs denoted as $\mathcal{A}$;
8          for each $\mathcal{E} \in \mathcal{A}$ do $Abs(\mathcal{E})$;
9          Let $\mathcal{D}'$ be the set of remaining states in $\mathcal{D}$;
10         if $|\mathcal{D}'| \leq B \vee |\mathcal{D}'| = |\mathcal{D}|$ then
11             $Abs(\mathcal{D}')$ and $\mathcal{C} \leftarrow \mathcal{C} \backslash \mathcal{D}$
12         else
13             Let $\mathcal{C}_{\mathcal{D}'}$ be the set of all nontrivial SCCs in $\mathcal{D}'$;
14             $\mathcal{C} \leftarrow (\mathcal{C} \backslash \mathcal{D}) \cup \mathcal{C}_{\mathcal{D}'}$;
15 return $ValueIteration(\mathcal{M}, G)$;
```

- The first step is to find all SCCs $\mathcal{C}$ in $\mathcal{M}$ by Tarjan's approach [126], and their input and output states are recorded as well. This is captured by line 1.

- For each SCC $\mathcal{D} \in \mathcal{C}$, we will first check whether $|\mathcal{D}|$ exceeds B or whether $|Out(\mathcal{D})| > 1$. If not, $Abs(\mathcal{D})$ will be executed directly. States in $\mathcal{D}$ but not in $Inp(\mathcal{D})$ will be removed. Afterwards $\mathcal{D}$ will be removed from $\mathcal{C}$, as shown in lines 4-5. The reason why we directly abstract cases $|Out(\mathcal{D})| \leq 1$ is as follows.

  - If $|Out(\mathcal{D})| = 0$, $\mathcal{D}$ has no outgoing transitions, then no matter whether $\mathcal{D}$ has target states or not, we do not need to solve $\mathcal{D}$. If $\mathcal{D} \cap G = \phi$, it is obvious that all states in $\mathcal{D}$ has probability 0 to reach G; otherwise, it is trivial to show that all states in $\mathcal{D}$ has probability 1 to reach G.

  - If $|Out(\mathcal{D})| = 1$, assume s_{out} is the output state. All paths entering $\mathcal{D}$ will leave it eventually. Therefore, for every $s_i \in Inp(\mathcal{D})$, the probability of paths entering

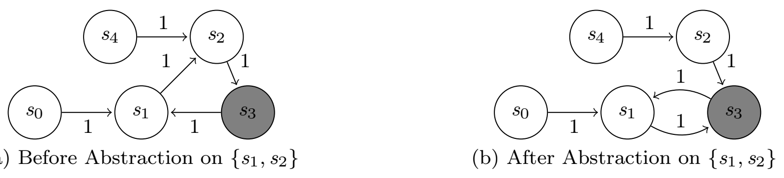

s_4
s_2
s_0
s_1
s_3
1
(a) Before Abstraction on $\{s_1, s_2\}$
(b) After Abstraction on $\{s_1, s_2\}$

Figure 5.3: Destruction of SCC during abstraction taken on $\{s_1, s_2\}$

$\mathcal{D}$ via s_i, staying in $\mathcal{D}$ and exiting $\mathcal{D}$ to s_{out} should be 1. So $\mathcal{D}$ can be abstracted directly.

- Lines 7-14 describe the case when $\mathcal{D}$ needs to be divided, i.e., when the SCC has more than B states. First we divide $\mathcal{D}$ into several groups based on some heuristics, each of which has a reasonably small number of state, i.e., less than B. Therefore, for each group $\mathcal{E}$ we use $Abs(\mathcal{E})$ to get the abstraction. Here we choose AG as the structure of each partition, because the existence of the root state, say s_r, may remove the most states after abstraction. In the extreme case where $Inp(\mathcal{E}) = \{s_r\}$, all states in $\mathcal{E}$ except s_r can be removed.

- By removing the states which are not input states of any $\mathcal{E}$, the number of states in $\mathcal{D}$ is often (not always) reduced. Line 10 checks two situations. 1) the size of $\mathcal{D}'$ is smaller than or equal to B, and 2) there is no reduction for $\mathcal{D}$ in this iteration. If 1) is true, then there is no need to divide $\mathcal{D}'$ again, and $Abs(\mathcal{D}')$ is executed directly. If 2) is true, i.e., no state is reduced after divide and conquer, the main reason should be that each state in $\mathcal{D}$ has a lot of pre-states. Therefore every state in one group is an input state and cannot be removed. In this case, $\mathcal{D}'$ should also be abstracted. Afterwards, $\mathcal{D}$ is removed from $\mathcal{C}$. If 1) and 2) are both false, lines 13-14 will be executed.

- Because of the abstraction, $\mathcal{D}$ may not be an SCC now. An example is shown in Figure 5.3. On the left hand side, $\mathcal{D} = \{s_1, s_2, s_3\}$; if we group s_1 and s_2 together, then s_3 is this group's output. It is easy to get the abstract transitions between them, as shown in right hand side. Because both s_1 and s_2 are input states, no state is removed. However, it is obvious that $\mathcal{D}' = \{s_1, s_2, s_3\}$ is not an SCC anymore. Tarjan's algorithm is used again to find new SCCs in the $\mathcal{D}'$, captured by lines 13-14. New SCCs will be added to $\mathcal{C}$ for another iteration.

- When the iteration terminates, there is only trivial SCCs in $\mathcal{M}$ now; in other words, $\mathcal{M}$ is acyclic. Value iteration approach can be used to calculate the probability from the initial state to targets efficiently, and this is captured by line 15.

As we mentioned in Section 5.2.2, the iterative abstraction will not affect the final result of the probability calculation. The following theorem establishes that the algorithm is always terminating.

Theorem 5.3.1 *Given a finite state DTMC $\mathcal{M}$, Algorithm 1 always terminates.*

Proof We assume $\hat{S} = \Sigma_{\mathcal{D}\in\mathcal{C}} |\mathcal{D}|$, in other words, $\hat{S}$ is the total number of states in $\mathcal{C}$. Then the theorem can be proved by showing (1) $\hat{S}$ is finite at the beginning, and (2) $\hat{S}$ monotonically decreases after each iteration.

(1) is obviously true because $\mathcal{M}$ has finite number of states, and $\hat{S} \leq |S|$ where S is the set of states of $\mathcal{M}$.

Given an SCC $\mathcal{D} \in \mathcal{C}$, if it satisfies the condition in line 4, then $\mathcal{D}$ will be removed from $\mathcal{C}$, thus $\hat{S}$ is reduced. Otherwise, from line 6, there are two possible outputs. (i) $\exists \mathcal{E} \in \mathcal{A}$, $Abs(\mathcal{E})$ reduces its number of states, or (ii) $\forall \mathcal{E} \in \mathcal{A}$, $Abs(\mathcal{E})$ does not reduce its number of states. If (i) is true, then $\hat{S}$ is also reduced. If (ii) is true, then $|\mathcal{D}'| = |\mathcal{D}|$. According to line 8, $\mathcal{D}$ will be abstracted directly and be removed from $\mathcal{C}$. Thus $\hat{S}$ is still reduced. Therefore (2) is true, and the theorem holds. □

5.3.2 Dividing Strategies

Although the divide-and-conquer approach is correct and terminating, its efficiency is highly dependent on how an SCC is divided. Assume $\mathcal{A}$ is the set of partitions after dividing an SCC, then a suitable partition, say $\mathcal{E} \in \mathcal{A}$, should satisfy the following conditions.

1. $\mathcal{E}$ should not have too many states, since each partition is abstracted using Gauss-Jordan elimination which is limited to a relatively small number of states;
2. $\mathcal{E}$ should not have too few states as well, otherwise there will be too many partitions to be solved, and the states reduction for $\mathcal{E}$ is inefficient;
3. The smaller $|Out(\mathcal{E})|$ is, the better reduction is achieved. Too many output states will make the input states of $\mathcal{E}$ have too many abstract transitions, which makes the remaining structure complicated, and affects the efficiency of the following abstraction.

As a result, the remaining issue is that given an SCC $\mathcal{D}$, is there any *optimal* strategy to divide it into *suitable* AGs? In practice, the structure of $\mathcal{D}$ could be arbitrary. This increases the difficulty of finding a general strategy for all cases.

The simplest division method is to try to set each AG to have the same number of states. Assume each AG should have N states. Then starting from one input state of $\mathcal{D}$, depth first search (DFS) or breadth first search (BFS) can be used to group every N states together. Afterwards, each AG can be abstracted, and the remaining states are combined together to do the next iteration. The advantage of this strategy is that the number of states in each partition is easily controlled. It can be very efficient in cases where the states in $\mathcal{D}$ has few transitions. However, this method cannot control the number of output states of each partition, and a predefined N may not be suitable for $\mathcal{D}$'s structure.

Therefore, another improved strategy is used to automatically decide the number of states in each AG. Instead of picking a constant N in the beginning, we set a lower bound B_L

and an upper bound B_U for each partition. Thus the number of states in each partition should be between B_L and B_U. At first, B_L states will be grouped into $\mathcal{E}$, and $|Out(\mathcal{E})|$ is recorded. Afterwards, some states in $Out(\mathcal{E})$ are added into $\mathcal{E}$, and $|Out(\mathcal{E})|$ is updated. If $|Out(\mathcal{E})|$ keeps unchanged or even becomes smaller after the update, we will try to add more states into $\mathcal{E}$ again. If $|Out(\mathcal{E})|$ is increased but the increase is not significant, a few states will be added into $\mathcal{E}$ but the number should be small. Otherwise $\mathcal{E}$ is confirmed and ready for $Abs(\mathcal{E})$. Note the number of states in $\mathcal{E}$ should be always below B_U. This strategy guarantees

1. the number of states in $\mathcal{E}$ is under control. B_L and B_U guarantee that the size of $\mathcal{E}$ should not be too large or too small.
2. the outputs of $\mathcal{E}$ are also manageable. This guarantees the states structure after abstraction is not too complicated, and is suitable for next iteration.

Parameters B, B_L and B_U can be adjusted according to the specific DTMC to improve the efficiency.

5.3.3 Parallel Computation

Previous work such as [80, 28] depends on the topological order between different SCCs. Therefore, parallel computation is not so easy to use in their setting. On the contrary, our algorithm eliminates loops via abstracting every SCC one by one, without considering their order. The independence between different SCCs can be proved following the proof in [5]. What is more, even each AG in one SCC is also independent from others, and the proof actually follows the same idea of SCC's independence. Thus, parallelization is suitable in our setting in order to solve different AGs simultaneously.

In details, after finding all SCCs, they are stored with their input and output states. For each SCC, a spare thread can be used to solve it. Therefore, lines 2-14 in Algorithm 1 can be solved via parallel computation. In addition, whenever an AG is grouped, another spare thread, if there is any, can be used to abstract it. Thus line 8 in Algorithm 1 can also be handled in parallel.

5.4 SCC Reductions on Markov Decision Processes

As both approaches based on solving linear programming and value iteration have their own limitations, we propose a new approach to abstract away the loops in each strongly connected component (SCC) of an MDP based on a divide-and-conquer algorithm, and then apply value iteration to the resulting acyclic MDP. Without loops, the calculation of reachability probabilities will be faster, and also will be more accurate than the pure value iteration case with an unspecified amount of errors.

Reducing SCCs in MDP while preserving the results of reachability analysis is highly non-trivial, and may lead to extra schedulers and an exponential increase in the number of PDs if not handled properly. In this work, the proposed divide-and-conquer algorithm works on blocks; hence effectively avoids the generation of extra PDs. Moreover, we can further reduce the redundant PDs based on the convex property.

In the following, we will use a running example to illustrate the main idea of the divide-and-conquer approach, and then present the overall algorithm and detailed methodologies on performing state abstraction in an MDP, followed by its optimization on the reductions of probability distributions.

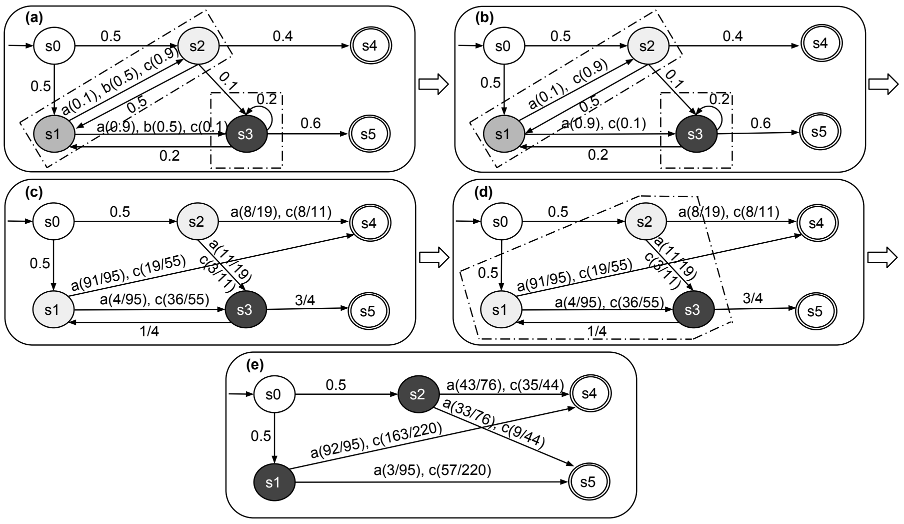


Figure 5.4: A running example of transforming the MDP in Figure 5.1 (a) to the acyclic MDP in Figure 5.1 (b)

5.4.1 A Running Example

To reduce an SCC, our reduction approach starts from adding each state in the SCC into a new block. It then divides these blocks into groups. For each group, it eliminates loops within the group and merges its components into a new block. We call this process abstraction. This step repeats until the whole SCC becomes one block, which is guaranteed to be free of loops. In this part, we demonstrate our main idea with a running example that transfers the MDP in Figure 5.1 (a) to the acyclic MDP in Figure 5.1 (b). The execution of each step is demonstrated in Figure 5.4.

First, the states $\{s_1, s_2, s_3\}$ are identified as the only nontrivial SCC in the MDP, and there are three blocks, i.e., $\{s_1\}, \{s_2\}, \{s_3\}$, labeled using different grayscale in Figure 5.4(a). Let Λ be the set of all current blocks in the SCC, i.e., $\Lambda = \{\{s_1\}, \{s_2\}, \{s_3\}\}$. We then divide Λ into two groups, as enclosed by dashed lines, such that the blocks $\{s_1\}$ and $\{s_2\}$ form one group, and $\{s_3\}$ alone forms the other.

Subsequently, abstraction is performed on both groups. The main idea of the abstraction is to eliminate loops in the group by connecting the inputs and outputs using equivalent non-redundant probability distributions. In the first step, we need to remove the redundant probability distributions (PDs) in each block of the group. Recall that each PD can form a linear constraint according to Eq. (2.1). According to the PDs in Figure 5.4 (a), it can be proved that the constraint from PD b of state s_1 is redundant as it can be represented by a linear combination of the constraints from PDs a and c. As a result, PD b can be removed. The updated MDP is shown in Figure 5.4 (b).

The second step of abstraction is to calculate the equivalent PDs. In the present case, block $\{s_1\}$ has two actions and block $\{s_2\}$ has only one action, thus there are two $(2 \cdot 1)$ schedulers in total. We define σ_1 as the scheduler selecting PD a at block $\{s_1\}$, based on which a set of linear equations can be formed as

$$V(s_1) = 0.1\,V(s_2) + 0.9\,V(s_3); \qquad V(s_2) = 0.5\,V(s_1) + 0.1\,V(s_3) + 0.4\,V(s_4) \tag{5.1}$$

Similar definition applies to scheduler σ_2 for PD c, we have

$$V(s_1) = 0.9\,V(s_2) + 0.1\,V(s_3); \qquad V(s_2) = 0.5\,V(s_1) + 0.1\,V(s_3) + 0.4\,V(s_4) \tag{5.2}$$

To eliminate the transitions between s_1 and s_2, we need to first select a particular scheduler, and then perform Gauss Jordan elimination. Under the selection of scheduler σ_1, we can have the following new transitions based on Eq. (5.1),

$$V(s_1) = \frac{4}{95}V(s_3) + \frac{91}{95}V(s_4); \qquad V(s_2) = \frac{11}{19}V(s_3) + \frac{8}{19}V(s_4) \tag{5.3}$$

Similarly, with the selection of σ_2, we have the following based on Eq. (5.2),

$$V(s_1) = \frac{36}{55}V(s_3) + \frac{19}{55}V(s_4); \qquad V(s_2) = \frac{3}{11}V(s_3) + \frac{8}{11}V(s_4) \tag{5.4}$$

As a result, the updated PDs can be established based on Eq. (5.3) and Eq. (5.4). As illustrated in Figure 5.4 (c), a new block can then be formed by grouping states s_1 and s_2, and states s_3 and s_4 continue to serve as outputs. Each state (s_1 or s_2) in the new block now has two PDs (i.e. a and c), which appears to create a larger number ($2 \cdot 2 = 4$) of schedulers. However, it should be noted that the newly generated PDs in s_2 are derived based on the choice of scheduler in s_1 and thus **not** independent. For example, Eq. (5.3) and (5.4) are derived based on Eq. (5.1) and (5.2), respectively. That means a scheduler selects action a in s_1 and action c in s_2 is equivalent to a non-memoryless scheduler in the original MPD (with both selections of a and c at s_1). Therefore, two of the four schedulers in the new block are equivalently non-memoryless and thus redundant for obtaining the maximum and minimum reachability probabilities (please refer to Section 2.2). Effectively, the number of schedulers to be handled in the new blocks remains as two. To easily allocate these schedulers, we denote the PDs for s_1 and s_2 obtained from the same set of equations by the same index or the same action name. Thus, given a block, a scheduler only selects an index or an action, which means the PD with that index or action will be selected at each state. Similarly, we can obtain the abstraction on the other block $\{s_3\}$. The resulting MDP as shown in Figure 5.4 (c) has only two blocks ($\Lambda = \{\{s_1, s_2\}, \{s_3\}\}$) in the SCC, both of which are free of loops and redundant PDs.

To finally achieve a single block, another round of grouping and abstraction needs to be performed. There are now two blocks, and we combine them into one group as shown in Figure 5.4 (d). As explained above, during the calculation of maximum and minimum reachability probabilities, block $\{s_1, s_2\}$ can be described using two schedulers, and the other block $\{s_3\}$ has only one scheduler. Therefore, the total number of schedulers within the group is two (i.e., $2 \cdot 1$). Let σ_3 be a scheduler selecting the PD of action a, i.e., $\sigma_3(\{s_1, s_2\}) = a$, and σ_4 be the other scheduler selecting the PD with action c, i.e., $\sigma_4(\{s_1, s_2\}) = c$. A set of linear equations can be formed similarly as Eq. (5.1) and (5.2), and the solutions connect

Algorithm 2: SCC Reduction in an MDP via Divide-and-Conquer

```
input  : An MDP $\mathcal{M} = (S, s_{init}, Act, Pr)$, target states $G \subseteq S$
output: $\mathcal{P}(\mathcal{M} \models \Diamond G)$
1  $\mathcal{M}' := \mathcal{M}$
2  $\mathcal{C}$ := the set of all nontrivial SCCs in $\mathcal{M}'$;
3  for each $\mathcal{D} \in \mathcal{C}$ do
4      $\Lambda := \varnothing$;      //to record a set of partitions
5      $\forall s \in \mathcal{D}, \Lambda := \Lambda \cup \{\{s\}\}$;   //each state is a partition initially
6      repeat
7          Divide $\Lambda$ into a set of groups of partitions denoted as $\mathcal{A}$;
8          $\Lambda' = \varnothing$;
9          for each $\mathcal{J} \in \mathcal{A}$ do
10             $\mathcal{E}' = AbstractionMDP(\mathcal{J})$;      //$\mathcal{J}$ is a set of partitions
11             $\Lambda' = \Lambda' \cup \mathcal{E}'$;
12         $\Lambda = \Lambda'$;
13     until $|\Lambda| == 1$;
14 return $ValueIteration(\mathcal{M}, G)$ ;
```

the input states s_1 and s_2 directly to the output states s_4 and s_5. With such a new block, the inner state s_3 can be removed from the MDP. Up to this point, there is only one block left and our reduction finishes. The final acyclic MDP is shown in Figure 5.4 (e).

5.4.2 Overall Algorithm

The overall algorithm for SCCs reduction is presented in Algorithm 2. It is based on a divide-and-conquer approach that works on *blocks* of an MDP. Given a set of states S, a block $\mathcal{E}$ is a subset of S such that $\bigcup_i \mathcal{E}_i = S$; and $\forall \mathcal{E}_i, \mathcal{E}_j, \mathcal{E}_i \neq \mathcal{E}_j, \mathcal{E}_i \cap \mathcal{E}_j = \varnothing$. Given an MDP $\mathcal{M} = (S, S_{init}, Act, Pr)$ and target states $G \subset S$, Algorithm 2 removes all loops in $\mathcal{M}$ (i.e., producing a new acyclic MDP $\mathcal{M}'$) and computes reachability probabilities in $\mathcal{M}'$ by value iteration. We remove loops according to the following steps.

- Line 2 finds all SCCs by Tarjan's approach [126], and adds all nontrivial SCCs to $\mathcal{C}$. Lines 3–13 present the divide-and-conquer procedure for each SCC in $\mathcal{C}$. Let Λ be a set of blocks of an SCC. Initially, each state of SCC forms a block in Λ, as shown in lines 4–5.

- Lines 6–13 perform the divide-and-conquer in the blocks of Λ until there is only one block left. Within each round, line 7 first divides all the blocks Λ into several groups, denoted by $\mathcal{A}$. Here, the groups are formed dynamically that each has relatively small number of output states. Each element $\mathcal{J}$ in $\mathcal{A}$ is a group of blocks. There is always a group containing more than one blocks unless there is only one block in $\mathcal{A}$. Next, lines 9–11 remove loops and the inner states in each $\mathcal{J}$ through *AbstractionMDP*() method, which takes a group of blocks as the input and returns a new acyclic block that can represent the previous group. As a result, after each round, the number of blocks decreases and loops inside each block are eliminated. Details for the abstraction process will be presented in Section 5.4.3.
- After the iteration terminates, the resulting MDP becomes acyclic. The value iteration method, detailed in Section 2.2, can then be applied to calculate the probability from the initial state to the target states efficiently.

As we can see, in order to support the divide-and-conquer algorithm for MDPs, the overall algorithm incorporates methods like abstraction and PD reduction. In the following parts, we will introduce details of these two methods.

5.4.3 States Abstraction in an MDP

Given a set of blocks, denoted by $\mathcal{J}$, the *abstraction* process removes the inner states in each block, and merges all blocks into a new block, denoted by $\mathcal{E}'$. The detailed algorithm of abstraction is presented in Algorithm 3. It takes $\mathcal{J}$ as the input and returns a new acyclic block $\mathcal{E}'$. The procedure works as follows.

Algorithm 3: States Abstraction in MDP

```
input  : A set of partitions of states $\mathcal{J}$ in an MDP
output : A new partition $\mathcal{E}'$
      //step 1: remove redundant PDs in each partition
1  for each $\mathcal{E} \in \mathcal{J}$ do
         //$\mathcal{I}$ is to record whether a PD is non-redundant
2      Let $\mathcal{I}$ be a set of Boolean variables initialized with false;
3      for each $s \in \mathcal{E}$ do
4          Indices := indices non-redundant PDs of $s$;
5          for each index $\in$ Indices do $\mathcal{I}'[index]$ =: true;
6          $\mathcal{I} = \mathcal{I}'$;
7      for each $s \in \mathcal{E}$ do Update PDs according to $\mathcal{I}$;
      //step 2: calculate new PDs from inputs to outputs
8  $\mathcal{K} = \bigcup_{\mathcal{E} \in \mathcal{J}} \mathcal{E}$;
9  $\forall s \in Inp(\mathcal{K}) \cdot \mathbf{U}'_s := \varnothing$;
10 $\Sigma$ := all the schedulers in $\mathcal{J}$ based on partitions;
11 for each $\sigma \in \Sigma$ do
12     calculate PDs from $Inp(\mathcal{K})$ to $Out(\mathcal{K})$ according to $\sigma$;
13     Let $u_s$ be the calculated PD of a input state $s$;
14     $\forall s \in Inp(\mathcal{K}) \cdot \mathbf{U}'_s := \mathbf{U}'_s \cup \{u_s\}$;
      //step 3: form a new partition
15 $\mathcal{E}' = Inp(\mathcal{K})$ ;
16 $\forall s \in \mathcal{E}'$, replace PDs of $s$ by $\mathbf{U}'_s$;      //re-connect $Inp(\mathcal{E}')$ to $Out(\mathcal{E}')$
17 return $\mathcal{E}'$;
```

- The first step, as shown in lines 1–7, is to reduce redundant PDs in each block. As demonstrated in Section 5.4.1, within a block, the PDs of the same index are originated from the same scheduler in the original model. Thus, they are not independent and can only be removed if they are all redundant. The detailed operations are as follows. For each block, we use a Boolean set $\mathcal{I}$ to record whether a PD is redundant. Initially, line 2 sets all elements in $\mathcal{I}$ to *false*. For each state of the block, line 4 gets all indices of the non-redundant PDs, and line 5 sets the respective elements in $\mathcal{I}$ to *true*. Here, the non-redundant PDs can be identified by finding the vertices of the convex hull, detailed in Section 5.4.4. After the **for** loop in lines 3 - 6, a *false* in $\mathcal{I}$ means the corresponding PD in each state is redundant. As a result, line 7 removes the respective PDs at the indices for all states.

- Line 8 combines states in all blocks of $\mathcal{J}$ into a group $\mathcal{K}$. The second step is to calculate new PDs from $Inp(\mathcal{K})$ to $Out(\mathcal{K})$ for all schedulers. Line 9 creates an empty set for each state in $Inp(\mathcal{K})$, which is used to store new PDs. Line 10 finds all the schedulers in $\mathcal{J}$ and assigns them to Σ. As reviewed in Section 5.2.1, for any given state, a scheduler is used to select a PD, and the total number of schedulers is exponential to the number of states. As mentioned, within a block, the PDs with the same index are not independent, we thereby create a scheduler in such a way that it can only select PDs with the same index at all states in the block. This can avoid the generation of extra schedulers by including all the combinations of PDs. Lines 11 –14 calculate the new equivalent PDs by calculating the transition probabilities, from $Inp(\mathcal{K})$ to $Out(\mathcal{K})$. For each scheduler σ, we calculate the probabilities from any input to output states in the group DTMC states $\mathcal{K}^{\sigma}$ according to Gauss-Jordan method detailed Section 5.2.2. Line 14 adds the new PDs to each state.

- Since the sets of PDs from $Inp(\mathcal{K})$ to $Out(\mathcal{K})$ have been obtained, the inner states of K are then redundant for the calculation of reachability probabilities. As a result, line 15 creates a new block $\mathcal{E}'$ by adding only the inputs states of $\mathcal{K}$, and updates the PDs of each state in $\mathcal{E}'$ by $\mathbf{U}'_s$. The new block $\mathcal{E}'$ is free of loops.

5.4.4 Reduction of Probability Distributions based on Convex Hull

Within a set of probability distributions (PDs), if a PD can be represented by a convex combination of the other PDs, we call it a *redundant* PD. As demonstrated, PD b in Figure 5.1 (a) is redundant as it can be represented by a combination of 50% of PD a and 50% of PD b. It can be proved that the redundant PDs are irrelevant to the maximum and minimum reachability probabilities [21].

There are two scenarios when redundant PDs might be introduced. One is during system modeling, for instance, the PDs could originate from a set of working profiles (modeling complex system environment) and some of working profiles are indeed redundant for calculating the maximum or minimum probability. The other is during the removal of the inner states within a group of states $\mathcal{K}$. In this case, the equivalent PDs are created to connect inputs to outputs of $\mathcal{K}$, the number of those is equal to the total number of schedulers in $\mathcal{K}$. As a result, there could be redundant PDs, especially when obtained PDs of a state have only a few successive states. In fact, the number of PDs of a state can be minimized and replaced by a *unique* and *minimal* set of PDs. If we consider PDs as a set of points in a Euclidean space and each successive state in a PD provides a dimension in the Euclidean space, finding the set of non-redundant PDs is equivalent to the problem of identifying all the vertices of the convex hull of all the PDs. This has been already proved in [21]. In the following, we have a brief review on the convex hull property.

The *convex hull* of a set Q of points, denoted by $CH(Q)$, is the smallest convex polygon or polytope in the Euclidean plane or Euclidean space that contains Q [35]. Mathematically, the convex hull of a finite point set, e.g., $Q = \{\mathbf{q}_1, \cdots, \mathbf{q}_n\}$, is a set of all *convex combinations* of each point $\mathbf{q}_i$ assigned with a coefficient r_i, in such a way that the coefficients are all non-negative with a summation of one; i.e., $CH(Q) = \{\sum_{i=1}^{n} r_i \cdot \mathbf{q}_i \mid (\forall\, i : r_i \geqslant 0) \wedge \sum_{i=1}^{n} r_i = 1\}$. We denote the set of *vertices of a convex hull* as $V_{CH}(Q)$. Each $\mathbf{q}_i \in V_{CH}(Q)$ is also in Q, but it is not in the convex hull of the other points (i.e., $\mathbf{q}_i \notin CH(Q \setminus \{\mathbf{q}_i\})$). In other word, the points $V_{CH}(Q)$ are the essential points that generate all the other points in $CH(Q)$ via a convex combination. Given a set of n points (Q) in d-dimension, the algorithms to determine the vertices of the convex hull are also known as the redundancy removal for a point set Q in $\mathbb{R}^d$. This problem can be reduced to solving $\mathcal{O}(n)$ linear programming problems with many polynomial time algorithms available [21].

To further accelerate the calculation, we adopt an approximation algorithm proposed by

Bentley et al. [17], who use the convex hull of some *subset* of given points as an approximation to the convex hull of all the points. Here, a user-defined parameter β controls degree of approximation. For instance, in xy-plane, we first divide the area between the minimum and maximum (i.e., extreme) values in x-dimension into 'strips', with a width of β. We then select the points with the extreme values in y-dimension within each strip, and the points with x-dimension extreme. Last, we construct the convex hull based on these selected points (in the worst case, there are only $2(1/\beta + 2)$ points). Here, β specifies the relative approximation error; i.e., any point outside the approximate hull is within β distance of the 'true' hull, as proved in [17]. Hence, a larger β implies a faster calculation but a coarser approximation. In terms of reachability analysis, the schedulers, after approximation, are only a subset of original ones. Ignoring some of the PDs means the maximum or minimum reachability probability will be a safe approximation; i.e., the maximum probability is smaller than the 'true' maximum, and the minimum probability is larger than the 'true' minimum.

5.4.5 Termination and Correctness

In this section, we discuss the termination and the correctness of our approach.

Theorem 5.4.1 *Given a finite states MDP, Algorithm 2 always terminates.*

Proof: Given a finite number of states, the **for** loop in Algorithm 2 always terminates as the number of SCCs is finite. The theorem can then be proved by showing (1) the **repeat** loop can terminate and (2) *AbstractionMDP*() can also terminate.

For (1), the proof for the one state SCC is trivial. For an SCC having more than one states, there are at least one group in $\mathcal{A}$ that has more than one block, which can be merged into one new block through *AbstractionMDP*(). The total number of blocks is guaranteed to decrease after each round of the **repeat** loop. Thus the termination condition $\mid \Lambda \mid == 1$ can always be fulfilled. For (2), the abstraction, as in Algorithm 3, always terminates because all **for** loops work on a finite set of elements. As both conditions are fulfilled, the theorem holds. □

Theorem 5.4.2 *Given a finite states MDP, Algorithm 2 always produces an acyclic MDP.*

Proof: To prove the theorem, it is equal to show that Algorithm 2 can remove all loops in each SCC. As proved above, Algorithm 2 always transfers each SCC into one block, the theorem can be proved by showing that the abstraction process always returns a loop-free block. Assuming a set of blocks $\mathcal{J}$ are the input, Algorithm 3 always creates a new block by recalculating the probability distributions from $Inp(\mathcal{J})$ to $Out(\mathcal{J})$. As $Inp(\mathcal{J}) \cap Out(\mathcal{J}) = \varnothing$, the new block is guaranteed to be acyclic. Therefore, the theorem holds. □

As Algorithm 2 always terminates with an acyclic MDP, our approach can always provide an accurate result. Recall that loops in each SCC of the MDP are resolved by solving sets of equations, which is based on an accurate method. Further, we could trade off a certain level of accuracy for better performance with approximate convex hull.

5.5 Implementation and Evaluation

We implement the algorithms (for both DTMCs and MDPs) in our model checking framework PAT [121]. As the only difference between the ordinary and our proposed value iteration methods is the algorithm of reachability analysis, it is fair to check the effectiveness of the new method through direct comparison of their performance. Hereafter, we refer the implementations with and without our approach as PAT(w) and PAT(w/o), respectively. For the value iteration method, we use the default stopping criterion in PAT, i.e., the maximum ratio of difference is $1E$-6. The testbed is an Intel(R) Xeon(R) CPU at 2.67 GHz with 12

GB RAM. All related materials, including the tools, models, and evaluation results, are available at [2].

5.5.1 Evaluations in Discrete Time Markov Chains

In this section, we report the experiments evaluations of Algorithm 1 in deterministic systems that are modeled in discrete time Markov chains (DTMCs). In these experiments, we use the **improved** dividing strategy, and B, B_L, B_U are set to be 300, 100, 150 respectively. In other words, an SCC with more than 300 states should be divided; each group has states between 100 and 150. These parameters are manually selected based on our experimental experience, i.e., generally these parameters have better performance compared with others. The testbed is a server running Windows Server 2008 64 Bit with Intel Xeon 4-Core CPU$\times$2 and 32 GB memory.

We apply our approach to several more meaningful systems and demonstrate that our approach can still improve the efficiency significantly.

In multi-agent systems, dispersion games [56] represent an important scenario, i.e., dispersion games are the generalization of anti-coordination games to an arbitrary number of players and actions. Two strategies are designed for dispersion games: *basic simple strategy* (BSS) and *extend simple strategy* (ESS). BSS assumes the number of players and the number of actions are the same, while ESS does not have this assumption. In each round of the game, every player chooses one action following specific probabilistic distribution, which is updated roundly according to the output of last round. There is a desired outcome in this game called Maximal Dispersion Outcome (MDO), and one property is to calculate the probability that MDO can be achieved.

Another case used in our experiments is coin flipping protocol for polynomial randomized consensus [9] (CS). This case focuses on modeling and verifying the shared coin protocol of

System	States	Prob	PAT (w)			PAT (w/o)		
			Time	BMR	Memory	Time	BMR	Memory
BSS (4)	4196	1	1.3	92.3%	39	0.2	50%	35
BSS (5)	49572	1	3.5	94.3%	297	4.4	11.4%	142
BSS (6)	605890	1	41.4	72.7%	1297	105.3	6.7%	417
BSS (7)	7462639	1	1671	30.1%	6350	2073.1	4.1%	5039
ESS (6, 4)	32662	1	1.4	92.8%	16.3	2.7	14.8%	5.6
ESS (6, 5)	162945	1	6.7	91.1%	48.5	11.4	16.7%	13.9
ESS (7, 5)	463460	1	27.9	84.9%	310	75.8	7.1%	292
ESS (8, 5)	1114480	1	70.5	74.7%	619	278.5	6.1%	643
ESS (8, 6)	6476524	1	438.0	68.5%	4209	1168.1	7.5%	3904
CS (4, 3)	4966	0.023	0.8	87.5%	45	2.4	8.3%	35
CS (6, 3)	34529	0.023	15.7	81.5%	214	124.1	0.9%	108
CS (6, 4)	45281	0.015	24.8	86.7%	324	243.8	0.6%	81
CS (6, 5)	56033	0.012	38.6	91.2%	312	432.1	0.4%	104
CS (7, 4)	99265	0.014	102.3	87.6%	1062	983.1	0.4%	97
CS (7, 5)	122785	0.011	161.7	92.1%	1145	1384.8	0.3%	97
CS (7, 6)	146305	0.01	245.5	94.9%	1404	2409.5	0.2%	156
CS (8, 4)	200083	0.013	585.1	93.4%	1974	-	-	-

Table 5.1: Experiments: benchmark systems

the randomized consensus algorithm. CS is used as a benchmark system in the state-of-the-art probabilistic model checker PRISM [78]. Here we use a safety property in the system as our target.

The experiments based on these three models are listed in Table 5.1. Here, time and memory are recored in the units of second and megabyte, respectively. in $BSS(N)$ indicates there are N players (also N actions) in the game; $ESS(N, K)$ means there are N players and K actions; $CS(N, K)$ indicates there are N processes and K is a constant used in the model. Here we are interested in the ratio of model building (BM) time to the total time, which is denoted as BMR in the table. In $PAT(w)$, BM means the time for building **acyclic** DTMC, i.e., the overall time consumed by eliminating loops in DTMC; in $PAT(w/o)$, it indicates the time for building the whole system. In both PAT versions, value iteration is used to get the final result after building the model. '-' indicates the verification takes more than 1 hour thus the result is not taken into consideration. From the table, we have several observations.

1. For some small examples such as *BSS*(4), our new approach is slower. This is due to the overhead taken by the SCC searching algorithm, and value iteration approach is efficient when loops are small.

2. As the examples become larger, the verification speed is increased by our proposed approach. This improvement is obvious especially in large-scale systems such as *ESS*(8, 5), *ESS*(8, 6) and *CS*(8, 4).

3. *CS* consumes more resource than *BSS* and *ESS* when they have similar size of state space, such as *CS*(7, 6) and *ESS*(6, 5). The reason is that *CS* has more complicated SCCs, and both our new approach and traditional value iteration method have to use more time and memory to solve it. As a result, the SCCs' structure affects the verification efficiency to a large extent.

4. According to *BMR*, we can see that in the previous version of PAT, building the model costs small portion of the overall verification time compared with the value iteration procedure. The average value of *BMR* is less than 10%, which means slow convergence indeed exists in systems having large SCCs. *CS* has very small *BMR* and this is consistent with the fact that *CS* has complicated SCCs. In the new approach, time is mainly used by abstractions, as average *BMR* is more than 80%. It indicates that the efficiency of the divide-and-conquer strategy is critical in the whole verification now, and optimal dividing strategy is worthy to explore.

On the other hand, we want to share some limitations of our approach according to the experimental information. The efficiency of this approach is dependent on whether large SCCs exist in the system. During our experiment, the new approach performs slower than value iteration method in several cases. The main two reasons include 1) there is no loops in the system, thus the SCC searching algorithm makes the whole verification slow; 2) the system just has small SCCs while the whole state space is large, thus the gain of the

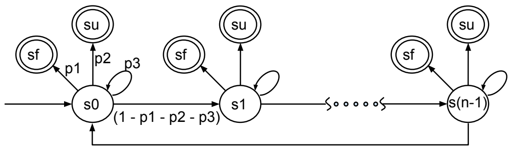


Figure 5.5: A reliability model, the states s_u and s_f are copied for a clear demonstration

abstraction is limited.

5.5.2 Evaluations in Markov Decision Processes

In this section, we report the experiments evaluations of Algorithm 2 in non-deterministic systems that are modeled in Markov decision processes (MDPs). For the new approach in MDPs, we set the maximum number of blocks in a group to 3, and the parameter for convex hull approximation to 0.001. We perform an analysis on case studies on two MDP-based systems: one is software reliability assessment model and the other is tennis tournament prediction model. Both systems have many probability transitions and loops, thus may encounter slow convergence issue especially when the systems become large. Thus, we evaluate how our new approach can benefit those cases.

5.5.2.1 Case Study on Software Reliability Assessment

Reliability and fault tolerance are central concerns to many software systems. The reliability problem can be transferred into a reachability problem in an MDP [58, 88]. In this case study, we model a system that undergoes n tasks and then standbys at the initial state. Each task is exposed to a certain probability of failure or self-recovering situation, before successfully transferring to the next task or service. A highly abstracted reliability model is shown in Figure 5.5, which consists of $n+2$ states, i.e., $\{s_f, s_u, s_0, s_1, \cdots, s_{n-1}\}$, representing different system status. The failure state s_f is the state that the system fails, and the success state s_u

Table 5.2: Comparison between PAT with and without SCC reduction for reliability model

Parameters			PAT (w/o)		PAT (w)	
m	n	#Trans.	Pmax	Time(s)	Pmax	Time(s)
4	40	0.6K	0.499985	0.03	0.500000	0.01
	400	6K	0.499999	0.22	0.500000	0.13
	20K	320K	0.499999	547.52	0.500000	55.97
	40K	640K	0.499999	1389.55	0.500000	314.73
10	40	2K	0.499985	0.04	0.500000	0.11
	400	16K	0.499999	0.41	0.500000	0.20
	20K	800K	0.499999	894.34	0.500000	111.62
	40K	1600K	0.499999	2168.04	0.500000	597.44

# States $\approx n$

is the state that the system finishes a requirement successfully. Each state s_i transits to s_f with a probability of p_1; to s_u with a probability of p_2; to itself with a probability of p_3; and otherwise, to the next state $s_{(i+1)\%n}$. Multiple sets of values for $\{p_1, p_2, p_3\}$ are considered. We then perform reachability analysis, e.g., computing the maximum probability of reaching state s_u, under different scale by varying the parameters n and m, where n controls the number of states and m is the number of probability distributions of each state.

The experiments are summarized in Table 5.2. The number of states being generated is approximately equal to n; *Trans.* represents the total number of transitions in the model; P_{max} represents the maximum reachability probability; and *Time* represents the total time spent on the verification. We have the following observations.

- The overall verification time of the new approach (PAT(w)) is much less than that of the previous approach (PAT(w/o)). Three factors here can affect the rate of value iteration in this model: 1) the self-loops at each state s_i; 2) the large SCC formed by $\{s_0, s_1, \cdots, s_{n-1}\}$; and 3) the various probability distributions in the model. Our approach reduces loops prior to value iteration, as detailed in Section 5.4. With PAT(w), the resulting acyclic MDP consists of only three states, s_0 (the only input of the SCC), and s_f and s_u (the outputs of the SCC). Thus, time spent on value iteration can almost be negligible (less than 0.001s). In addition, due to the PD reductions based on the convex hull, our reduction approach can work under many probability distributions without much overhead, as evidenced by the cases with $m = 10$.

- The result obtained from the new approach is closer to the true value. Through manual analysis, we know that 0.5 is the accurate result. In fact, our reduction approach removes loops by solving a set of linear equations, which yields accurate results. As mentioned above, the resulting model is an acyclic MDP of only three states, on which value iteration stops naturally without using any stopping criterion. On the other hand, the ordinary value iteration approach keeps iterating over loops until a stopping criterion is met, thus the result is an approximation.

The experiment above considers only one SCC in the reliability model. However, often, a system may have a large number of SCCs in its reliability model. Our preliminary result shows that, with the increase of SCCs, the total time increases exponentially for the ordinary value iteration approach, while remains at a low level with our approach [2]. This is because our approach resolves each SCC independently while the ordinary approach has to iterate over all SCCs until converging to a stable result.

5.5.2.2 Case Study on Tennis Tournament Prediction

A tennis match is won when a player wins the majority of prescribed sets. At a score of 6 - 6 of a set, an additional 'tiebreaker' game is played to determine the winner of the set. In this case study, we model a 7 point tiebreaker. Our model encodes the outcomes of individual player's actions (e.g., serve and baseline) according to the past scoring profiles available at `http://www.tennisabstract.com`, and predicts the winning probability for one player against the other. In particular, we predict the game between two tennis giants Federer

Table 5.3: Comparison between PAT with and without SCC reduction for tennis prediction model

#	Pro.	#States	#Trans.	PAT (w/o) Pmin	PAT (w/o) Pmax	PAT (w/o) B (s)	PAT (w/o) V (s)	PAT (w) Pmin	PAT (w) Pmax	PAT (w) B (s)	PAT (w) V (s)
1	a	15K	26K	0.4585	0.5077	0.16	0.01	0.4585	0.5077	0.22	0.00
	b	15K	26K	0.4923	0.5415	0.14	0.01	0.4923	0.5415	0.24	0.00
	c	17K	30K	0.4678	0.4786	0.19	13.44	0.4678	0.4786	0.58	0.33
	d	17K	30K	0.5214	0.5322	0.16	13.34	0.5214	0.5322	0.50	0.32
3	a	62K	108K	0.7877	0.8075	0.66	64.72	0.7877	0.8075	1.55	2.94
	b	62K	108K	0.8116	0.8303	0.64	65.54	0.8116	0.8303	1.48	2.96
	c	71K	123K	0.4576	0.4649	0.74	133.89	0.4576	0.4649	1.95	9.32
	d	71K	123K	0.5351	0.5424	0.72	133.03	0.5351	0.5424	1.98	8.45
5	a	141K	278K	0.9194	0.9271	1.42	266.26	0.9194	0.9271	3.66	23.25
	b	141K	245K	0.9332	0.9401	1.43	265.80	0.9332	0.9401	3.65	23.35
	c	160K	279K	0.4486	0.4554	1.58	434.29	0.4486	0.4554	4.37	41.65
	d	160K	278K	0.5446	0.5514	1.53	428.62	0.5446	0.5514	4.32	36.93

and Nadal. A play wins the set if he wins one tiebreaker, or best of 3 (or 5) tiebreakers. Thus, we analyze all the three situations. For each situation, we calculate four probabilities: (a) Federer scores the first point in any tiebreaker; (b) Nadal scores the first point in any tiebreaker; (c) Federer wins the set; and (d) Nadal wins the set.

The verification results are shown in Table 5.3. # represents the numbers of tiebreakers; *Pro.* represents the properties to be verified; *#States* and *#Trans.* represent the total numbers of states and transitions in the system, respectively; P_{min}/P_{max} records the minimum/maximum reachability probability; and B and V record the time costs on building the MDP model (for PAT(w), it includes the additional time spent on SCC reduction) and on value iteration, respectively. Notice that the summation of these two time costs is the total time spent on the verification. We have the following observations.

Comparing the time costs in B and V columns, for the ordinary approach, though the time for building an MDP model is very short, the verification time increases quickly when the size of system becomes large. On the other hand, with slightly longer time spent on model building, our new approach reduces the value iteration time significantly. This is because the new approach removes all SCCs prior to value iteration and the probability computation is thereby accelerated. In this case study, both approaches generate the same results up to four decimal points.

5.6 Related Work

In recent years, some approaches [80, 28, 8, 5, 117] have been proposed to improve probability reachability calculation. The key idea is to reduce iterations on the state space. [80, 28] improve value iteration in MDPs by backward iterating over each SCC in topological order, i.e., an SCC will not be visited until the reachability probabilities of all its successive SCCs converge. However, since it requires iterating over each SCC, this approach only alleviates the slow convergence problem to a certain degree without completely solving the problem. Compared to their work, our reduction on each SCC is independent to others, so that multi-cores or distributed computers can be directly applied, which can make the verification even faster.

The approaches [8, 5] are on SCCs elimination by connecting inputs to outputs of an SCC with equivalent probability transitions in DTMCs. These works have been successfully applied to the probabilistic counterexample generation. The algorithms proposed in [5] and our proposed approach [117] can both reduce large SCCs. [5] iteratively searches for and solves the smallest loops within an SCC. But our SCC reduction in DTMC [117] uses a divide-and-conquer algorithm that iteratively divides an SCC into even smaller parts and resolves loops in each part. On the other hand, eliminating loops in an MDP is particularly challenging due to the existence of many probability distributions. To the best of our knowledge, there has been no previous work on SCC reductions for MDP. Instead of a simple extension of our divide-and-conquer for DTMC in [117], our divide-and-conquer algorithm for

MDP [59] is carefully designed to avoid generation of extra schedulers. To further accelerate the elimination of loops, we actively detect and remove redundant probability distributions of each state based on the convex hull property.

5.7 Summary

In this chapter, we have proposed divide-and-conquer algorithms to speed up reachability analysis in both DTMCs and MDPs. Because SCCs are one of main reasons that the probability calculation is slow, we focus on abstracting SCCs via calculating the transition probability from their inputs to outputs. For DTMCs, we divide every SCC, whose states exceed some specific bound, into several partitions having reasonable number of states, and can be solved efficiently via Gauss-Jordan elimination. To further cope with the non-determinism in MDPs, our divide-and-conquer algorithm is then designed to work on blocks. Initially, each state in an SCC is considered as a block. The blocks are repeatedly merged together until there is only one left. During the abstraction, loops within a block are replaced by equivalent probability distributions between inputs and outputs. The convex hull property is applied to further reduce the redundant probability distributions. We have implemented this algorithm in a model checker PAT. The evaluation results on some benchmark systems and two practical case studies show that our method can improve reachability analysis.

Chapter 6

Improved Reliability Assessment for Distributed Systems via Abstraction and Refinement

Our reliability analysis for non-deterministic systems is based on the techniques from probabilistic model checking, to deal with situations involving both probabilistic behavior (e.g., reliabilities of system components) and non-determinism. However, the application of probabilistic model checking is currently limited due to the issue of state space explosion, which makes reliability assessment of distributed system particularly difficult and even impossible. In this chapter, we improve the probabilistic model checking through a method of abstraction and refinement, which controls the communications among different system components and actively reduces the size of each component.

6.1 Introduction

Assessing the reliability of a distributed system can be highly non-trivial due to non-determinism compared to that of a sequential system, which is often deterministic. In distributed systems, the order of executions among various system components is highly unpredictable and dependent on the operating environment. Therefore, the precise probability distribution of the execution orders is hard to obtain if not impossible, and it is more suitable to model those non-deterministically. However, non-determinism invalidates existing reliability assessment approaches based on Markov chain models [26, 70, 55, 50, 43], which fundamentally assume that there is only one probability distribution of event occurrences at any system state. It is thus necessary to develop a method for assessing the reliability of non-deterministic systems.

A potential candidate is probabilistic model checking [16, 32] based on Markov decision processes, which is designed to deal with both probabilistic behavior and non-determinism. However, its application is limited to small scale distributed systems as it works by exhaustively exploring the global state space, which is a product of the state spaces of all components and often huge. Therefore, we are motivated to develop a scalable approach to assess the reliability of distributed systems (e.g., web services, wireless sensor network) that often consist of many components (e.g., clients, sensors).

In this work, we assume that a distributed system consists of a set of system components, each of which has its local state space and interfaces for communications. Our key concept is to shrink the global state space by controlling the communications among different system components and actively reducing the sizes of their local state spaces. More specifically, we start with an abstract system by turning a subset of communication events into local (i.e., non-communication) events, which can be effectively removed afterwards by re-calculating the local probability distributions for the rest of the communication events. We then perform probabilistic model checking to calculate the reliabilities (here referred to as 'approximations'). If the resulting approximations are not precise enough, refinement steps can be performed by incrementally enlarging the set of communication events, which eventually yield an actual result based on the complete model. We prove the soundness of our approach by showing that probabilistic model checking on the reduced system always results in safe approximations, i.e., the minimum approximation is no larger than the actual minimum and the maximum approximation is no less than the actual maximum. Our empirical study shows that the approximations are often very close to the actual values, which allow us to deduce conclusive results based on a smaller state space depending on the level of reliability requirement. The underlying principle is that *the additional behavior contained in the reduced system is often irrelevant or negligible for reliability calculation.*

In summary, we contribute to the following technical aspects. First, we propose an abstraction technique to significantly reduce the state space for reliability assessment by hiding not only local events but also communication events. As a result, we can assess the reliability of distributed systems, which are normally too large for probabilistic model checking. Second, we prove that results obtained from the reduced state space always produce safe approximations, and we empirically show that it is often satisfactory to explore a reduced state space with a given reliability requirement. Third, we develop a framework so that an overly coarse abstraction can be refined incrementally. To choose a better refinement strategy, we develop different heuristics for adding back the communication events, through analyzing the verification result on the reduced state space. The empirical study shows that the application of heuristics often accelerates the refinement process. Last but not the least, we implement the proposed technique in the toolkit RaPiD [58] to support the reliability assessment of distributed systems, and show that our method improves its performance significantly on multiple systems.

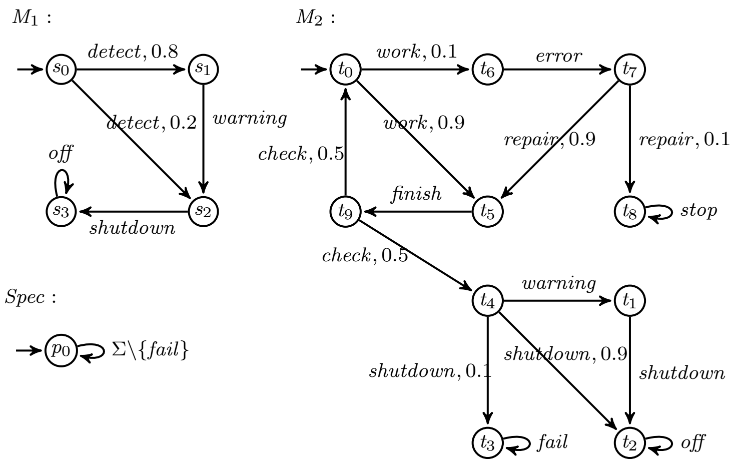


Figure 6.1: Two Markov models and an LTS specification

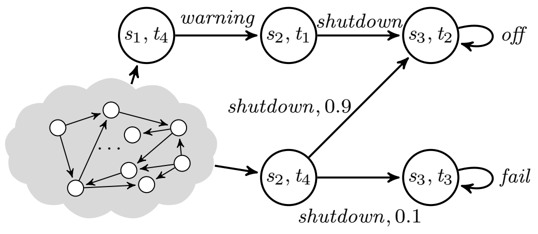


Figure 6.2: The state space of the product of M_1 and M_2

Organization The remainder of the chapter is organized as follows. Section 6.2 presents a motivating example, which illustrates the main steps of our approach. Section 6.3 reviews relative background. Section 6.4 introduces our approach in details. Section 6.5 reports the experiments and evaluations. Section 6.6 surveys related work.

6.2 Motivating Example

A simple model of a device controlling system, which is a variant of [79], is shown in Figure 6.1. A device is modeled as a Markov decision process (MDP) M_2. Its shutdown process

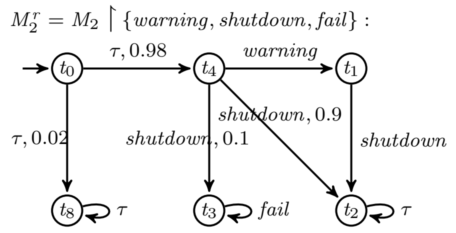


Figure 6.3: A reduced model by hiding local and *off* events

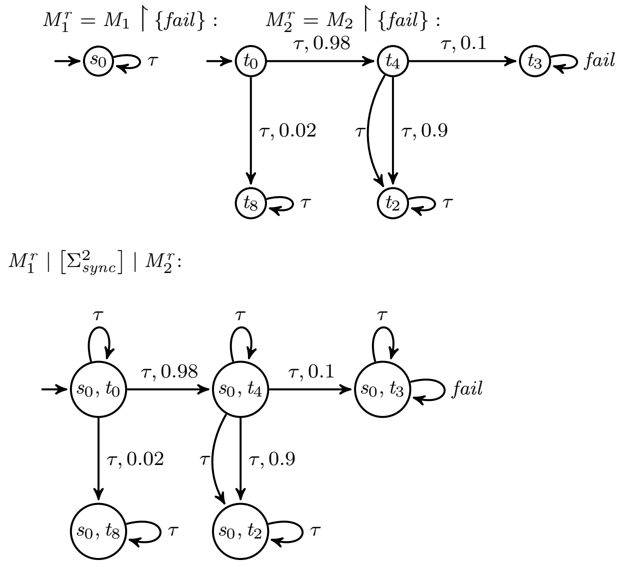


Figure 6.4: Reduced models with only *fail* event visible

is coordinated by a controller, modeled as a Markov chain (MC) M_1. Each probability distribution is labeled with an event name. If a probability distribution connects to only one successor, we drop the probability value of 1 to keep the graphs concise. There is one probability distribution at state t_0 in M_2. The transition from t_0 to t_6 (labeled with $work, 0.1$) means that while the device is working on a task, it has a probability of 0.1 going to state t_6, which subsequently leads to an *error* event, and activates a repair mechanism at state t_7. With a probability of 0.9, the error can be fixed and the device goes to state t_5 and subsequently finishes the task with a *finish* event. After finishing a task, the device checks if there are new tasks. With a probability of 0.5, it will work on a new task and then go back to state t_0; otherwise, the device will go to state t_4 (i.e., prepare for shutdown) if there is no more new task. The controller and the device communicate through synchronizing common events, i.e., *warning*, *shutdown* and *off*, as highlighted in bold. Via a communication, the advancement to the next state only happens when this communication event occurs simultaneously in both the controller and the device. Intuitively, M_1 first receives a *detect* signal, after which it sends a *warning* message and coordinates the shutdown behavior of the device by sending a *shutdown* command. However, with a probability of 0.2, it fails to issue a *warning* message. If M_2 receives *warning*, it shuts down correctly; otherwise, it only shuts down correctly with a probability of 0.9. Notice that events *warning* and *shutdown* are modeled non-deterministically at state t_4, as the exact probability of each event occurrence depends on practical control environment.

A labeled transition system, *Spec* in Figure 6.1, specifies a system such that the *fail* event never occurs, i.e., a system that always shuts down properly. Here Σ is an abbreviation of the set of all events in M_1 and M_2, and the transition labeled with $\Sigma \backslash \{fail\}$ denotes a group of transitions that are labeled with any event in Σ except *fail* event. The question is how reliable the system is for accomplishing a shutdown properly; or equivalently what the minimum probability is for the system to satisfy the *Spec*.

The standard approach works as follows. First, we compute the synchronous product of M_1 and M_2. Notice that all common events *warning*, *shutdown* and *off* are to be synchronized. They are referred to as communication events, and the rests are referred to as local events. The result is an MC partially shown in Figure 6.2. There are in total 24 states and 47 transitions. Due to the space limit, we only show the part that contains communication events. Next, we apply probabilistic model checking to the computation of the probability that the product satisfies *Spec*, which is reduced to the problem of computing the probability of not reaching state (s_3, t_3) in the product [123]. Using standard techniques like value iteration or solving a linear equation system, we obtain a reliability of 0.9804. Besides the state space explosion problem, it should be noticed that the calculation based on value iteration is only an approximation and the accuracy is user-defined via setting conditions on when the iteration can be stopped. Although directly solving linear equations can provide the most accurate results, but it only works with small models [16].

An improved approach being proposed in this work is based on two major observations: (1) some of the communication events are not essential in computing this probability, and (2) in general, we can always identify lower and upper bounds of the reliability even if we choose to ignore some of the communications among the components. In the following, we show these through two cases.

In the first case, we show that ignoring some of the communication events allows us to work with a smaller state space without changing the result. In fact, local events do not affect the communication among components, and communication event *off* is perhaps not that relevant to overall reliability intuitively, as both M_1 and M_2 remain at the same state after it occurs. Thus, we hide all local and *off* events by replacing them with an invisible event, designated by τ event. M_1 and M_2 are reduced to M_1^τ, M_2^τ, respectively. M_1^τ is exactly the same as M_1 except that the self-loop transition labeled with *off* event previously at state s_3 is hidden and labeled with τ event instead. To further explain the reduction process, M_2^τ is shown in Figure 6.3. After hiding, the states t_0, t_5, t_6, t_7, and t_9 are only connected by τ events and thus can be collapsed into one state t_0 connecting to states t_4 and t_8 with an equivalent probability distribution, which can be obtained via solving a simple linear program [117, 5]. The states t_0, t_1, t_2, t_3, t_4 and t_8 are kept in M_2^τ as they cannot be further reduced. As a result, the number of states of the parallel composition is reduced from 24 to 13. With probabilistic model checking, the minimum probability of the reduced model is 0.9804, which is the same as that of the original model.

In the second case, we show that a safe approximation of the minimum reliability can be obtained even though the ‘wrong’ events are ignored. We consider an extreme case by ignoring all local and communication events except *fail* event as *fail* is the only event related to *Spec*. The reduced models M_1^r and M_2^r are shown in Figure 6.4. As all the events in M_1 are hidden, the whole model is reduced to one state with a self-looping τ transition M_1^r. Similar to the first case, the states t_5, t_6, t_7, and t_9 are removed in M_2^r. The transitions between the states t_4 and t_2, which are previously linked by events *shutdown* and *warn*, are now reduced into two direct τ transitions. The parallel composition of the two reduced models is shown in the third model of Figure 6.4. Notice that the number of states of the parallel composition is reduced from 24 to 5. Furthermore, the product has no loops with multiple states so that probabilistic model checking with value iteration is likely to converge fast. Based on this highly abstracted model, the minimum probability is calculated as 0.9020, which is smaller than the actual value 0.9804, and thus a safe approximation. If the question is whether the system has a reliability of at least 0.9 shutting down successfully, we can conclude positively with this result. However, it should be noted that according to the actual requirement of the reliability assessment, refinement can always be performed to yield results with a higher order of accuracy.

This example provides insights on the effectiveness of our approach in speeding up the reliability assessment as well as potential challenges. In the following, we present details of our approach including the heuristics on choosing the ‘right’ events to ignore.

6.3 Preliminaries

In this work, we assume that a distributed system with failure behavior can be modeled as a network of Markov decision processes (MDPs), and the specification on the required reliable system is modeled as a linear transition system (LTS). As the basic definition on MDPs are introduced in Chapter 2, we only provide the definitions on LTS and some details on how to model a distributed system using a set of MDPs, and then describe the procedures of applying standard probabilistic model checking on reliability assessment with a given specification in this section.

6.3.1 Some More on Model Formalisms

Labeled Transition System We start with defining a labeled transition system (LTS). Let Σ be a set of event names; τ denote an internal event that is invisible from external; and Σ_τ denote $\Sigma \cup \{\tau\}$. An LTS is a tuple $\mathcal{L} = (S, \mathit{init}, \mathit{Act}, T)$ where S is a set of states; $\mathit{init} \in S$ is the initial state; $\mathit{Act} \subseteq \Sigma_\tau$ is a set of events (or called an alphabet); and $T \subseteq S \times \mathit{Act} \times S$ is a labeled transition relation. A simple example is the *Spec* shown in Figure 6.1.

Let $s, s' \in S$ and $a \in \Sigma_\tau$, a transition between two states s, s', is denoted as $(s, a, s') \in T$ or written as $s \overset{a}{\to} s'$ for simplicity. In this case, we say a is enabled at s. We write $u \rightsquigarrow v$ to denote that v is reachable from u through τ transitions, i.e., there exists a finite sequence of states $\langle s_0, s_1, \cdots, s_n \rangle$ such that $s_i \overset{\tau}{\to} s_{i+1}$ for all $i \in [0, n-1]$ and $u = s_0$ and $v = s_n$. We write $u \overset{a}{\rightsquigarrow} v$ if $u \rightsquigarrow u'$ and $u' \overset{a}{\to} v'$ and $v' \rightsquigarrow v$. This means that the two states are connected via a series of τ transitions and one a transition. A path of $\mathcal{L}$ is a sequence of alternating states/events $\pi = \langle s_0, a_0, s_1, a_1 \cdots \rangle$ such that $s_0 = \mathit{init}$ and $s_i \overset{a_i}{\to} s_{i+1}$ for all $i \geq 0$. The set of all paths of $\mathcal{L}$ is written as $\mathit{paths}(\mathcal{L})$. Given a path π, we can obtain the corresponding trace, written as $\mathit{trace}(\pi)$, by omitting states and τ events. The traces of $\mathcal{L}$ are denoted as $\mathit{traces}(\mathcal{L}) = \{\mathit{trace}(\pi) \,|\, \pi \in \mathit{paths}(\mathcal{L})\}$. An LTS is deterministic iff for all $s \in S$ and $e \in \Sigma_\tau$, if $s \overset{e}{\to} u$ and $s \overset{e}{\to} v$, then $u = v$. Otherwise, it is non-deterministic. A non-deterministic LTS can be translated into a trace-equivalent deterministic LTS using the standard power set construction [113, 123].

Relationships Between LTS, MDP and DTMC Informally, an LTS can be turned into an MDP by incorporating probability distributions, e.g., to model system failures probabilistically. A discrete time Markov chain (DTMC) can be viewed as a special MDP with every state having exactly one probability distribution, and thus is deterministic. An MDP $\mathcal{M}$ can be viewed as a group of DTMCs, each of which is obtained with a different *scheduler*. The scheduler, denoted as σ, selects an event (and the corresponding probability distribution) at each state so that the result is a DTMC, denoted as $\sigma(\mathcal{M})$. A memoryless scheduler is a function $\sigma : S \to Act$ that always chooses the same event given the same state. It has been shown that memoryless schedulers are sufficient for our present task [16]. More details on MDP, DTMC and the scheduler are referred to Chapter 2.

Given a DTMC $\sigma(\mathcal{M})$, a path is a sequence $\langle s_1, a_1, s_2, a_2, \cdots \rangle$ such that $a_i = \sigma(s_i)$ is the event chosen by the scheduler σ and $Pr(s_i, a_i)(s_{i+1}) > 0$. For each path, we can calculate its probability as $\Pi_i Pr(s_i, a_i)(s_{i+1})$. Given the path, we can obtain a trace $\langle a_1, a_2, \cdots \rangle$ by omitting the states and τ events. The probability of a given trace tr, written as $Pr(tr, \sigma(\mathcal{M}))$, is defined as the accumulated probability of all the paths in $\sigma(\mathcal{M})$ that exhibit tr.

Synchronization among a set of MDPs A distributed system, with failure behavior, can often be modeled as a network of MDPs. Given a system composed of multiple MDPs, events to be synchronized are called communication events that are the common events among the MDPs. Within a set of events, visible events are the ones that can be observed from outside, and the rests are called local or internal events. A communication event can be synchronized if and only if it is a visible event and is enabled in all MDPs. In the following, we define the parallel composition of two MDPs over a set of visible events, which can be readily extended to multiple MDPs.

Let $\mathcal{M}_i = (S_i, init_i, Act_i, Pr_i)$ where $i \in \{1, 2\}$ be two MDPs and Σ_v be a set of visible events. The synchronization composition $\mathcal{M}_1$ and $\mathcal{M}_2$ over Σ_v, written as $\mathcal{M}_1 \,|[\Sigma_v]|\, \mathcal{M}_2$, is an MDP $\mathcal{M} = (S_1 \times S_2, (init_1, init_2), Act_1 \cup Act_2, Pr)$, where Pr is the probability transition relation satisfying the following conditions:

- if $s_1 \xrightarrow{e} \mu$ in Pr_1 and $e \notin \Sigma_v$, then $(s_1, s_2) \xrightarrow{e} \mu'$ in Pr for all $s_2 \in S_2$ such that $\mu'((s_1', s_2)) = \mu(s_1')$ for all $s_1' \in S_1$;
- if $s_2 \xrightarrow{e} \mu$ in Pr_2 and $e \notin \Sigma_v$, then $(s_1, s_2) \xrightarrow{e} \mu'$ in Pr for all $s_1 \in S_1$ such that $\mu'((s_1, s_2')) = \mu(s_2')$ for all $s_2' \in S_2$;
- if $s_1 \xrightarrow{e} \mu_1$ in Pr_1, $s_2 \xrightarrow{e} \mu_2$ in Pr_2 and $e \in \Sigma_v$, then $(s_1, s_2) \xrightarrow{e} \mu'$ in Pr such that $\mu'((s_1', s_2')) = \mu_1(s_1') \cdot \mu_2(s_2')$ for all $s_1' \in S_1$ and $s_2' \in S_2$.

Examples of the composition is shown in Figure 6.2 and Figure 6.4. We remark that, though only synchronous communication is allowed in the above definition, asynchronous communication, which is typical for distributed systems, can be easily constructed by modeling the communication media explicitly. For instance, if $\mathcal{M}_1$ and $\mathcal{M}_2$ communicate through radio, which is common for sensors, we can model the lossy channel using an MDP that essentially receives messages and later on either forgets about the messages or forwards them. The entire system is then a composition of the three MDPs.

6.3.2 Reliability Assessment with a Given Specification

In this work, we use an LTS as a specification of correct system behavior and calculate reliability as the probability of a system model (which is a network of MDPs) satisfying the specification. In the simplest case, the specification is an LTS that prevents the *fail* event from happening and allows other events repeatedly to happen at any time, i.e., the *Spec* shown in Figure 6.1. In general, having an LTS specification (which models the behavior of a perfectly reliable system according to different service requirements) allows more flexibility than always having the same LTS, *Spec*, as the specification (which specifies that the system should not *fail*). For instance, the specification can model a system that fails once but successfully activates a backup service, or a system that works correctly for important system functionalities whereas fails and recovers only during less important missions, etc.

Furthermore, different systems may have different focuses with respect to the reliability. For instance, reliability of a sensor in a wireless sensor network may be defined as the probability of detecting certain phenomena, whereas the reliability of a web server is associated with the probability that it reacts to web page requests without errors.

The task of reliability assessment is thus to calculate the exact probability of a system satisfying the specification. That is, reliability is the probability of the traces of the system model being a subset of those of the specification. Formally, let $\mathcal{M}$ be the system model with failure behavior and $\mathcal{S}$ be the specification. Reliability is the accumulated probability of all traces of $\mathcal{M}$ that are also traces of $\mathcal{S}$. Let σ be a scheduler of $\mathcal{M}$, the reliability with σ, written as $\mathcal{R}(\sigma(\mathcal{M}), \mathcal{S})$, is defined as:

$$\mathcal{R}(\sigma(\mathcal{M}), \mathcal{S}) = \Sigma\{Pr(tr, \sigma(\mathcal{M})) \mid tr \in traces(\mathcal{S})\}.$$

Notice that with different schedulers, the probability is often different and there are potentially infinitely many schedulers. Therefore the measurement of interest is the minimum and maximum probabilities, which are defined as:

$$\begin{aligned} \mathcal{R}^{min}(\mathcal{M}, \mathcal{S}) &= \inf_{\sigma} \mathcal{R}(\sigma(\mathcal{M}), \mathcal{S}); \\ \mathcal{R}^{max}(\mathcal{M}, \mathcal{S}) &= \sup_{\sigma} \mathcal{R}(\sigma(\mathcal{M}), \mathcal{S}). \end{aligned}$$

Note that the supremum ranges over all, potentially infinitely many schedulers. Using the approach proposed in [123], the range of reliability can be calculated using the following steps. First, we construct a deterministic LTS, denoted as $d(\mathcal{S})$, which is equivalent to $\mathcal{S}$ using the standard power set construction [113]. Next, we compute the synchronous product of $\mathcal{M}$ and $d(\mathcal{S})$, which is defined as follows. Let $\mathcal{M} = (S_{\mathcal{M}}, init_{\mathcal{M}}, Act_{\mathcal{M}}, Pr_{\mathcal{M}})$ and $d(\mathcal{S}) = (S_{\mathcal{S}}, init_{\mathcal{S}}, Act_{\mathcal{S}}, T_{\mathcal{S}})$. The product, written as $\mathcal{M} \times d(\mathcal{S})$, is an MDP $(S_{\mathcal{M}} \times S_{\mathcal{S}}, (init_{\mathcal{M}}, init_{\mathcal{S}}), Act_{\mathcal{M}} \cup Act_{\mathcal{S}}, Pr)$, where Pr is defined as follows:

- if $s_m \xrightarrow{e} \mu$ in $Pr_{\mathcal{M}}$ and $e \notin Act_{\mathcal{S}}$, then $(s_m, s_s) \xrightarrow{e} \mu'$ in Pr for all $s_s \in S_{\mathcal{S}}$ such that $\mu'((s'_m, s_s)) = \mu(s'_m)$ for all $s'_m \in S_{\mathcal{M}}$;

- if $s_m \xrightarrow{e} \mu$ in $Pr_{\mathcal{M}}$, $e \in Act_{\mathcal{S}}$ and $s_s \xrightarrow{e} s'_s$ in $d(\mathcal{S})$, then $(s_m, s_s) \xrightarrow{e} \mu'$ in Pr for all $s_s \in S_{\mathcal{S}}$ such that $\mu'((s'_m, s'_s)) = \mu(s'_m)$ for all $s'_m \in S_{\mathcal{M}}$.

Here, $Act_{\mathcal{S}}$ is the set of specification events. Given a state $(s_m, s_s) \in S_{\mathcal{M}} \times S_{\mathcal{S}}$, if there exists $e \in Act_{\mathcal{S}}$ such that $s_m \xrightarrow{e} \mu$ in $\mathcal{M}$ and e is not enabled at s_s, any trace of $\mathcal{M}$ reaching s_m extended with e will not be a trace of $\mathcal{S}$, i.e., a trace example of $\mathcal{M}$ not satisfying $\mathcal{S}$. Thus, we call such states *witness states*. The reliability assessment problem is thus reduced to the problem of finding the probability of reaching any witness state. For instance, if the maximum probability of reaching the witness states is q, then $\mathcal{R}^{min}(\mathcal{M}, \mathcal{S})$ is $1 - q$. There are known methods for probabilistic reachability analysis like value iteration or solving linear programs [16]. The details for reachability analysis is shown in Chapter 2. In general, value iteration often yields better performance than other methods in practice [78, 122]. However, slow convergence is often an issue there due to large loops in the model and furthermore when value iteration is stopped, it is hard to know how far the approximation is from the actual result [24, 44].

6.4 Our Approach

There are two major challenges in applying probabilistic model checking to reliability assessment of distributed systems. One is state space explosion, as the global state space is the product of the state spaces of all distributed components. The other is that the result is often an approximation without knowing the discrepancy from the actual result due to the limitation of value iteration. In this section, we present our approach, which aims to tackle the former and help the latter challenges by reducing states and loops. Our approach assumes

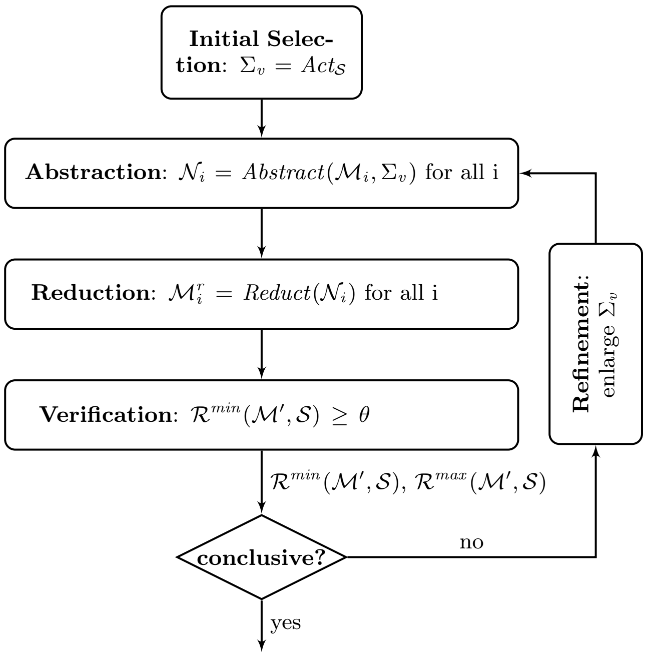

Initial Selection: $\Sigma_v = Act_{\mathcal{S}}$
Abstraction: $\mathcal{N}_i = Abstract(\mathcal{M}_i, \Sigma_v)$ for all i
Reduction: $\mathcal{M}_i^r = Reduct(\mathcal{N}_i)$ for all i
Verification: $\mathcal{R}^{min}(\mathcal{M}', \mathcal{S}) \geq \theta$
$\mathcal{R}^{min}(\mathcal{M}', \mathcal{S})$, $\mathcal{R}^{max}(\mathcal{M}', \mathcal{S})$
conclusive?
no
yes
Refinement: enlarge Σ_v

Figure 6.5: Overall workflow

that MDP is deadlock-free[1]. We present our approach by adopting a deadlock-free MDP $\mathcal{M} = (S_{\mathcal{M}}, init_{\mathcal{M}}, Act_{\mathcal{M}}, Pr_{\mathcal{M}})$ as the system model and an LTS $\mathcal{S} = (S_{\mathcal{S}}, init_{\mathcal{S}}, Act_{\mathcal{S}}, T_{\mathcal{S}})$ as the specification.

6.4.1 Overview

The problem of reliability assessment for distributed systems is often stated as follows. Given a distributed system with multiple components, it is to decide whether the overall reliability is no less than some threshold θ. Thus, without loss of generality, we assume that we need to check whether $\mathcal{R}^{min}(\mathcal{M}, \mathcal{S}) \geq \theta$ in the following. The workflow of our approach is shown in Figure 6.5. Recall that Σ_v denotes the set of visible events and $Act_{\mathcal{S}}$ denotes the set of specification events. Any event not in Σ_v can be turned into a τ event. During **initial**

[1]As mentioned, this is a standard assumption. Nonetheless, if $\mathcal{M}$ is the parallel composition of multiple components, even if the components are deadlock-free, there is no guarantee that the composition is. On the other hand, there are methods that allow us to construct deadlock-free systems from given components [54].

selection, we always initialize Σ_v to $Act_{\mathcal{S}}$. Because hiding any event in the specification would invalidate the verification results. The other four main steps are detailed as follows.

- In the **abstraction** step, we hide any event that is in $Act_{\mathcal{M}}$ but not in Σ_v. Specifically, for each component $\mathcal{M}_i$ in $\mathcal{M}$, if an event a is to be hidden, we label all transitions with τ instead of a in $\mathcal{M}_i$. As a result, if two MDPs communicate by event a, then the communication is lost. This alters the behavior of the system. Nonetheless, as we show later, this allows us to work with a reduced global state space after the reduction while being able to produce useful results.
- In the **reduction** step, we minimize each component $\mathcal{M}_i$ by removing transitions labeled with τ in a way such that the verification result is not affected. We remark that, as the global state space is the product of the local state spaces, minimizing the local spaces would often lead to a significant reduction in the global state space.
- In the **verification** step, we apply probabilistic model checking to check whether $\mathcal{R}^{min}(\mathcal{M}', \mathcal{S}) \geq \theta$ is true, where $\mathcal{M}'$ is the parallel composition based on all components after abstraction and reduction. In fact, $\mathcal{R}^{min}(\mathcal{M}', \mathcal{S})$ is always less than or equal to $\mathcal{R}^{min}(\mathcal{M}, \mathcal{S})$ and $\mathcal{R}^{max}(\mathcal{M}', \mathcal{S})$ is always larger than or equal to $\mathcal{R}^{max}(\mathcal{M}, \mathcal{S})$, as we will prove in Theorem 6.4.1.
- After the verification step, we decide whether the result is conclusive. If $\mathcal{R}^{min}(\mathcal{M}', \mathcal{S}) \geq \theta$ is shown to be true, we conclude that $\mathcal{R}^{min}(\mathcal{M}, \mathcal{S}) \geq \theta$ is true and we are done with the assessment. If $\mathcal{R}^{max}(\mathcal{M}', \mathcal{S}) \leq \theta$, then we conclude that $\mathcal{R}^{min}(\mathcal{M}, \mathcal{S}) \geq \theta$ is false. Otherwise, the result is inconclusive, and we need to refine the abstraction so that more precise and perhaps conclusive results will be obtained.
- In the **refinement** step, we reduce the set of events to be hidden by restoring some communication events. That is, we enlarge Σ_v so that the resultant system model $\mathcal{M}'$ is refined towards the original model $\mathcal{M}$. Afterwards, we repeat from the abstraction

step. If all communication events are in Σ_v, we present with the verification result of the last iteration as the final conclusion.

Our approach always terminates. The worst case is that, after a few rounds of abstraction/refinement, it terminates when all communication events are made visible. In our experiments, we show that our approach often produces relatively accurate results and concludes in early rounds. Even in the worst case, there can still be a reduction as we always hide the non-communication events. In the following, we present details of the key steps.

6.4.2 Abstraction and Reduction

In this part, we show how to systematically hide events in a system component and then build a reduced MDP that preserves the verification result on the overall system.

Given a component modeled as an MDP, denoted as $\mathcal{M}_i$ and a set of visible events Σ_v, the abstraction is done straightforwardly by turning transition labels not in Σ_v into τ. Let $\mathcal{N}_i$ denote the MDP of the component *after abstraction*. In the reduction step, we build a new MDP $\mathcal{M}_i^r$ such that probabilistic analysis results based on $\mathcal{N}_i$ are preserved in $\mathcal{M}_i^r$ (which could be different from that based on the original system component model obviously). The basic idea of the reduction is to remove τ transitions and group states which are connected by τ transitions (since they are indistinguishable from an external point of view).

Let $\mathcal{N}_i = (S, init, \Sigma_v \cup \{\tau\}, Pr)$. We formally define $\mathcal{M}_i^r$ as an MDP $(S', init,\ \Sigma_v, Pr')$ satisfying the following two conditions. First, $S' = \{init\} \cup \{s_i \in S \mid \exists\, s_j \in S, e \in \Sigma_v \cdot Pr(s_i, e)(s_j) > 0\}$ such that it contains the initial state of $\mathcal{N}_i$ and all states in S which have at least one outgoing transition labeled with a visible event. Intuitively, all other states in S are not in S' as all of their outgoing transitions are labeled with τ, and thus can be collapsed into some state in S'. Second, Pr' satisfies the following condition: if $s_i \in S'$, for all states

$s_j \in S'$ such that there exists a scheduler σ that $s_i \overset{e}{\rightsquigarrow} s_j$ in $\sigma(\mathcal{N}_i)$ where $e \in \Sigma_v$, then $Pr'(s_i, e)(s_j) = Pr(reach(s_i, s_j), \sigma(\mathcal{N}_i))$. Here $Pr(reach(s_i, s_j), \sigma(\mathcal{N}_i))$ is the probability of reaching state s_j from state s_i in a DTMC $\sigma(\mathcal{N}_i)$. Calculating the probability of reaching a certain state in DTMC is a known problem with efficient solutions [16].

In the following, we discuss the complexity of the reduction defined above, especially in constructing Pr'. If $\mathcal{N}_i$ is a DTMC rather than an MDP, there is only a single scheduler to consider and the above construction is relatively inexpensive, as experimentally evidenced in [8, 5, 117]. If $\mathcal{N}_i$ is an MDP[2], in constructing Pr', e.g., in identifying $Pr'(s_i, e)(s_j)$, we must explore all memoryless schedulers which result in DTMCs containing s_i and s_j. In the worst case, the number of such schedulers is exponential to the number of non-deterministic choices. This implies that, although there are fewer states in $\mathcal{M}_i^r$, the number of schedulers may remain the same.

The above defines the maximum reduction that we could achieve by eliminating all states whose outgoing transitions are labeled with τ. It is in general expensive to obtain the maximum reduction due to the large number of schedulers given a complicated MDP [37]. In order to obtain a reasonable reduction without paying the full price, our implementation for an MDP reduction focuses on two aspects. First, we identify all strongly connected components (SCCs) in $\mathcal{N}_i$ which only contain τ transitions and collapse all the states in each SCC into one representative. Recall that slow convergence in the value iteration method is often due to loops and a loop containing only τ transitions in $\mathcal{N}_i$ will result in many loops in the global space. Second, we remove states with all non-probabilistic outgoing transitions labeled with τ which are not part of any loop, and direct all their incoming transitions to the respective successors.

We remark that, a component-based reduction not only eases the state space explosion (since

[2] $\mathcal{M}_i^r$ is a probabilistic automaton rather than an MDP. In this work, the difference is irrelevant for simplicity in presentation.

the number of states in each component may be reduced) but is also helpful in solving the slow convergence problem in probabilistic model checking. The reduction cost is relatively small compared to the potential saving due to the facts that (1) the size of a local state space corresponding to a distributed component is relatively small, thus requires a relatively low reduction cost; and (2) any reduction on a local state space can produce exponential benefits when performing calculations in the global state space as many states and loops may have been eliminated. This is evidenced by the empirical evaluation in Section 6.5.

6.4.3 Verification

After abstracting and reducing each component, we apply probabilistic model checking to the analysis on whether $\mathcal{R}^{min}(\mathcal{M}', \mathcal{S}) \geq \theta$ is true or not, where $\mathcal{M}'$ is the synchronization composition of all the system components after abstraction and reduction. In the following, we discuss the effectiveness and the soundness of the abstraction and reduction by establishing that the verification result obtained based on $\mathcal{M}'$ is safe.

Let A be the alphabet of the specification $\mathcal{S}$. Let G and L represent the sets of communication events and local events in $\mathcal{M}$, respectively. For each component, in the worst case, there are $2^{|A \cup G \cup L|}$ states where $\mid A \cup G \cup L \mid$ is the size of the union of A, G and L. Because for each event, there could be two states: either it is enabled or not. Assume there are X components running concurrently. In the worst case, there are $2^{|A \cup G \cup L| \times X}$ states. Given the set of visible events is Σ_v, the worst case number of states in the abstract model is $2^{|\Sigma_v| \times X}$, thus $2^{|(A \cup G \cup L - \Sigma_v|) \times X}$ states are reduced. In the extreme case, $\Sigma_v = A$ (i.e., all events not in A are hidden) and the worst case state space is $2^{|A| \times X}$, which is exponentially smaller than the original state space. We remark that the above state space calculation is only an estimate based on the assumption that states can be distinguished by their outgoing transitions. In the setting of an MDP, because the same event may be associated with different probability distributions, the resultant worst case state space could be even larger and so could the reduction. Nevertheless, the above analysis can provide some insights on how the proposed method may significantly improve reliability assessment for distributed systems.

Theorem 6.4.1 *Let $\mathcal{M}$ be a deadlock-free MDP and $\mathcal{M}'$ be the abstract reduced MDP as described in Section 6.4.2. $\mathcal{R}^{min}(\mathcal{M}', \mathcal{S}) \leq \mathcal{R}^{min}(\mathcal{M}, \mathcal{S}) \leq \mathcal{R}^{max}(\mathcal{M}, \mathcal{S}) \leq \mathcal{R}^{max}(\mathcal{M}', \mathcal{S})$.*

Proof It is trivial to see that $\mathcal{R}^{min}(\mathcal{M}, \mathcal{S}) \leq \mathcal{R}^{max}(\mathcal{M}, \mathcal{S})$ and thus in the following, we show $\mathcal{R}^{min}(\mathcal{M}', \mathcal{S}) \leq \mathcal{R}^{min}(\mathcal{M}, \mathcal{S})$ and $\mathcal{R}^{max}(\mathcal{M}, \mathcal{S}) \leq \mathcal{R}^{max}(\mathcal{M}', \mathcal{S})$. Furthermore, extending the proof that a DTMC after reduction is equivalent to the original DTMC in probability measurement [117, 5], we can show that parallel composition of all components $\mathcal{N}_i$ (i.e., the component after abstraction) and that of all components $\mathcal{M}_i^r$ are equivalent. That is, assuming $\mathcal{N}$ is the parallel composition of $\mathcal{N}_i$, $\mathcal{R}^{min}(\mathcal{N}, \mathcal{S}) = \mathcal{R}^{min}(\mathcal{M}', \mathcal{S})$ and $\mathcal{R}^{max}(\mathcal{N}, \mathcal{S}) = \mathcal{R}^{max}(\mathcal{M}', \mathcal{S})$. Thus, we are left with proving that the abstraction step is safe, i.e., $\mathcal{R}^{min}(\mathcal{N}, \mathcal{S}) \leq \mathcal{R}^{min}(\mathcal{M}, \mathcal{S})$ and $\mathcal{R}^{max}(\mathcal{M}, \mathcal{S}) \leq \mathcal{R}^{max}(\mathcal{N}, \mathcal{S})$.

For simplicity, $\mathcal{M}$ is assumed to be the parallel composition of two components $\mathcal{M}_1$ and $\mathcal{M}_2$. It should be straightforward to extend the proof to multiple components. Let σ be an arbitrary memoryless scheduler for $\mathcal{M}$ and

$$\pi = \langle (s_0, t_0), a_0, (s_1, t_1), a_1, \cdots, a_{n-1}, (s_n, t_n) \rangle$$

be a path of $\sigma(\mathcal{M})$ where (s_i, t_i) is a state of $\mathcal{M}$. By an induction on the length of π, we show that there is always a scheduler σ' for $\mathcal{N}$ and a path $\pi' = \langle (s_0', t_0'), a_0', (s_1', t_1'), a_1', \cdots, a_{n-1}', (s_n', t_n') \rangle$ of $\sigma'(\mathcal{N})$ such that the probability of the two paths are the same and π and π' share the same sequence of events in $Act_{\mathcal{S}}$ and $s_n = s_n'$ and $t_n = t_n'$. The base case is when π is $\langle (s_0, t_0) \rangle$, which is trivially true. The induction hypothesis is that the above is true when π has n states. By assumption (that $\mathcal{M}$ is deadlock-free), there must be an event a_n such that $((s_n, t_n), a_n, \mu_{n+1})$ is a transition in $Pr_{\mathcal{M}}$. Here, a_n is assumed to be the choice of σ at (s_n, t_n). We discuss different cases in the following.

- If $a_n \in \Sigma_v$, by definition, a_n must be enabled at (s'_n, t'_n) and it leads to the same probability distribution. Therefore, we set $\sigma'((s'_n, t'_n))$ to be a_n and the induction holds.

- If $a_n \notin \Sigma_v$ and a_n is a local event, there must be a τ transition enabled at (s'_n, t'_n) and it leads to the same probability distribution. Therefore, we set $\sigma'((s'_n, t'_n))$ to be a_n and the induction holds.

- If $a_n \notin \Sigma_v$ and a_n is a communication event, there must be a transition labeled with a_n enabled at s_n leading to μ^1_{n+1} and a transition labeled with a_n enabled at t_n leading to μ^2_{n+1}. By definition of parallel composition, let the resultant product probability distribution to be μ_{n+1}. There must be a τ transition enabled at s'_n leading to the same probability distribution μ^1_{n+1} and there must be a τ transition enabled at t'_n leading to the same distribution μ^2_{n+1}. We set σ' such that $\sigma'(s'_n, t'_n)$ is the τ transition at s'_n leading to μ^1_{n+1}. Next, for all states (s'_{n+1}, t'_n) such that $\mu^1_{n+1}(s'_{n+1}) > 0$, we set $\sigma'(s'_{n+1}, t'_n)$ to be the τ transition at t'_n leading to μ^2_{n+1}. It can be shown that the following two paths: where $\frown$ denotes sequence concatenation,

$$\pi \frown \langle a_n, (s_{n+1}, t_{n+1})\rangle;$$
$$\pi' \frown \langle \tau, (s'_{n+1}, t'_n), \tau, (s'_{n+1}, t'_{n+1})\rangle,$$

  have the same probability in $\sigma(\mathcal{M})$ and $\sigma'(\mathcal{N})$; and have the same sequence of events in $Act_{\mathcal{S}}$ (since a_n is not in Σ_v and therefore $Act_{\mathcal{S}}$); and have $s_{n+1} = s'_{n+1}$ and $t_{n+1} = t'_{n+1}$.

Thus, the scheduler space is enlarged after abstraction such that every scheduler in $\mathcal{M}$ will be still in $\mathcal{M}'$. Therefore, we conclude the induction holds. □

Based on the theorem, if $\mathcal{R}^{min}(\mathcal{M}', \mathcal{S}) \geq \theta$ is shown to be true, we conclude that $\mathcal{R}^{min}(\mathcal{M}, \mathcal{S}) \geq \theta$ is true and we are done with the assessment. If $\mathcal{R}^{max}(\mathcal{M}', \mathcal{S}) \leq \theta$, we conclude that

$\mathcal{R}^{min}(\mathcal{M}, \mathcal{S}) \geq \theta$ is false. On the other hand, the usefulness of verification results based on $\mathcal{M}'$ depends on how close the results are to the verification results based on $\mathcal{M}$. In Section 6.5, we empirically show that they are indeed close and thus useful.

6.4.4 Refinement

In the refinement step, the set Σ_v is enlarged by adding communication events. The order of adding the events and how the events are added (e.g., one at a time or multiple at a time) are critical as they have an impact on the effectiveness of the reduction as well as the number of refinements required. The problem is that there are exponentially many ways of adding the events, e.g., $K!$ options for K events to be added to Σ_v one by one, and finding the optimal way is highly non-trivial. Ideally, we should add the events such that the size of the resultant system $\mathcal{M}'$ is small whereas the verification results are accurate. This is difficult as there is no analytical solution to estimate the size of $\mathcal{M}'$ or the verification result without conducting the verification. In fact, The computation of such a minimal alphabet is shown to be NP-hard. Instead, we propose two heuristics to automatically guide the refinement, which we demonstrate empirically to be effective.

With the first heuristic (hereafter $\mathcal{H}_1$), we always give higher priority to an event that is to be synchronously engaged by more components. Intuitively, if an event is to be synchronized by multiple components, adding it into Σ_v would not introduce many new states since the event is likely to be disabled most of the time. The purpose is to keep $\mathcal{M}'$ small.

With the second heuristic (hereafter $\mathcal{H}_2$), we evaluate and select the best group of events at each refinement iteration. In the initial step, we divide all candidate events into groups and each group contains the same events in symmetrical components. In each refinement step, we evaluate the effectiveness of the candidate groups and the events in the most effective group are then added to Σ_v. We quantify the effectiveness of every group, denoted by p, as follows,

$$p = w \cdot \underbrace{\frac{\mid \mathcal{R}(\mathcal{M}', \mathcal{S}) - \mathcal{R}(\mathcal{M}'_c, \mathcal{S}) \mid}{\mathcal{R}(\mathcal{M}', \mathcal{S})}}_{p_a} + \underbrace{\mathrm{sgn}(N - N_c)\,\mathrm{lg}(\frac{\mid N - N_c \mid}{N})}_{p_s}$$

Here, $\mathcal{M}'$ and $\mathcal{M}'_c$ are the abstract reduced model before and after including the event into Σ_v; N and N_c are the number of states in $\mathcal{M}'$ and $\mathcal{M}'_c$; sgn() is the sign function. The first part denoted by p_a measures the improvement in the accuracy of a newly added event; and the second part denoted by p_s measures the changes in the size of state space in orders of magnitude. w is a weighting factor designed by users to control the degree of preference over the two aspects. It is not hard to observe that p measures the effectiveness of the newly added group reasonably well, because p increases when the verification accuracy is improved and the number of states is reduced; and vice versa.

Compared with $\mathcal{H}_1$, $\mathcal{H}_2$ tends to provide a better performance more often because it is based on a more precise calculation. The price to pay is that $\mathcal{H}_2$ requires more effort on calculating the effectiveness factor p for each event group in every refinement step. $\mathcal{H}_1$ requires less as it only counts the number of system components that synchronize each event in the initial step.

Lastly, we argue that there are at most $L + 1$ refinement steps where L is the total number of communication events, and therefore our approach always terminates.

6.5 Experiments and Evaluations

The proposed method has been implemented in RaPiD (Reli*a*bility Pred*i*ction and Distribution) [58] to support the reliability assessment of distributed systems, with 6.5K lines of C# code. RaPiD is a self-contained toolkit for reliability assessment and publicly available at [3]. It provides a user-friendly interface to draw MDP models as well as fully automated methods for reliability analysis. All models and evaluation results are available online at [3].

6.5.1 Case Studies

We evaluate RaPiD using three systems, two benchmarks taken from the literature and one healthcare system developed jointly by our group and a hospital in Singapore [89].

The first system is a client-server system (CSS) [46, 41], which is a typical distributed protocol that partitions workloads between a service provider (called a server) and service requesters (called clients). The server and clients often communicate over a network. Each client sends reservation requests for a common resource, waits for the server to grant, uses the resource, and then cancels the reservation. As the number of clients increases, the interactions between the clients and the server become more complicated. Performing reliability analysis of such a system requires extensive expertise and time. We build a CSS system model with one server and k clients, where k is an integer of at least 2. The resource is to be shared by the clients in a mutually exclusive way. Each client initially has a probability of 0.1 getting failure. In addition, we consider a variant that is slightly more complicated and denoted as CSSr. Besides the behavior in CSS, each client can be successfully repaired with a probability of 0.9. In both cases, the overall reliability to be estimated is the probability that any client can successfully access the resource and meanwhile no multiple clients are accessing the resource simultaneously. The specification used in this case study has multiple events like *cancel* and *granted* (instead of a single *failure*) which models a system where mutual exclusion is guaranteed perfectly.

The second system is an automatic gas station system (GSS) [65], consisting of one operator, n pumps, one queue for each pump and m customers. The operator handles payments and schedules the use of pumps. Each customer first goes to the operator and prepays a certain amount. Then, the customer will be randomly allocated to a pump. There is a queue of waiting customers at each pump. A pump must be activated before serving customers. Once filling finishes, the pump signals the operator with the amount of gas provided to the customer. Next, the operator calculates the balance and then gives the change back to the customer. We consider two kinds of failure behavior of each pump, i.e., in the beginning, there is a probability of 0.01 that a pump cannot be started successfully, and during the service, there is also a probability of 0.01 that the pump fails to deliver gas. Whenever a failure occurs, there is a corresponding maintenance that has a probability of 0.99 to rectify the pump system. We calculate the overall system reliability, i.e., the probability of providing at least 100 times pumping services to the customers without any failure. The size of the system depends on the number of pumps in the station (n), and the number of customers simultaneously asking for the service (m).

The third system is a smart healthcare system (SHS). It provides assistance to mild dementia patients who have difficulties in memorizing things and taking care of themselves. The system has been deployed in a nursing home over half a year for a trial [89]. This is a typical ambient intelligent system that is sensitive and responsive to the presence of patients with the aid of many sensors and reminders. To assist patients, the system has multiple sensors in the patients' room to monitor the patients' behavior, e.g., entering the room, lying down on a bed, leaving the bed, etc. The sensor signals are then interpreted by an inference engine. Once detecting any inappropriate behavior, the inference engine activates a reminder system, which sends a message to the patient's smartphone and/or screens in the room, or an alert to nurses for serious issues, via a Bluetooth speaker, TV, or iPad. The system reliability depends on the reliabilities of its sensors and networks. In this case study, we model failure behavior of each sensor and network according to the data collected from the project engineers.

In the following, we use one of them for illustration, i.e., the system must correctly send an alert when a patient is lying on the wrong bed. The reliability metric is to measure the minimum probability that the alert is sent without any error.

The SHS system has many requirements, which form a complicated LTS specification. Interested readers are referred to [89, 88] for more details.

6.5.2 Evaluation Results

The underlying assumptions of our approach are as follows: (1) by reducing each system component, we could significantly reduce the state space hence the reliability assessment time, and (2) ignoring some of the communication events has limited impact on the assessment results. In the following, we evaluate the assumptions and the efficiency of our reduction techniques by answering three research questions.

RQ 1: How effective is our technique in terms of reducing the size of the state space and assessment time?

To answer this question, we compared the number of states in a system and the total time cost for assessment (including time spent on abstraction, reduction and verification) with/without use of reduction technique in RaPiD (referred to as RaPiD and RaPiDr, respectively). For cross referencing, we also compared RaPiDr with PRISM v4.0.3 [78] on an Intel(R) Xeon(R) CPU at 2.67 GHz with 12 GB RAM. We created several models with different sizes by varying the number of system components. Specifically, for GSS, we constructed 18 cases by adjusting the numbers of available pumps (m =1 to 3) and customers (n = 1 to 6); for CSS and CSSr, we constructed 9 cases by increasing the number of clients gradually (k = 2 to 10); and for SHS, we followed the actual system design, and did not increase its size by adding extra components.

Table 6.1: Results of comparison between RaPiD and RaPiDr under different levels of abstraction

		RaPiD w/o reductions		RaPiD with reductions (RaPiDr)							
				minimum abstraction				maximum abstraction			
Case(*para.*)	(*para.*)	#States	Time (s)	#States *Ratio**	Time *Ratio**	$\mathcal{R}_{min}$	$\mathcal{R}_{max}$	#States *Ratio**	Time *Ratio**	$\mathcal{R}'_{min}$	$\mathcal{R}'_{max}$
GSS(n, m)	(1, 1)	1,428	0.21	1.17	0.98	0.98985	1.00000	3.52	0.96	0.98985	1.00000
	(1, 4)	99,760	38.18	1.42	9.18	0.98985	1.00000	98.87	351.14	0.98985	1.00000
	(1, 6)	728,100	290.68	1.39	7.80	0.98985	1.00000	516.02	1892.11	0.98985	1.00000
	(2, 1)	14,218	2.28	1.40	3.62	0.98975	1.00000	1.93	2.76	0.98965	1.00000
	(2, 4)	3,060,585	1202.35	1.41	4.39	0.98965	1.00000	81.76	546.69	0.98965	1.00000
	(2, 6)	OM	OM	OM	OM	OM	OM	∞	∞	0.98965	1.00000
	(3, 1)	100,000	23.89	1.85	9.44	0.98965	1.00000	1.83	9.96	0.98945	1.00000
	(3, 4)	OM	OM	OM	OM	OM	OM	∞	∞	0.98945	1.00000
CSS(k)	(2)	32	0.02	1.00	0.33	0.81000	0.99000	1.45	0.34	0.81000	0.99000
	(4)	682	0.02	1.00	0.21	0.65610	0.99990	2.07	0.14	0.65610	0.99990
	(8)	158,866	9.26	1.00	0.90	0.43047	1.00000	3.39	2.86	0.43047	1.00000
	(10)	2,052,094	187.12	1.00	1.13	0.34868	1.00000	4.07	5.29	0.34868	1.00000
CSSr(k)	(2)	45	0.02	1.41	0.04	0.97814	0.99988	2.05	0.04	0.97814	0.99988
	(4)	1,515	0.06	2.22	0.53	0.95676	1.00000	4.59	0.30	0.95676	1.00000
	(8)	1,031,735	77.58	6.49	7.69	0.91540	1.00000	22.03	24.68	0.91540	1.00000
	(10)	OM	OM	∞	∞	0.89539	1.00000	∞	∞	0.89539	1.00000
SHS		1,296,000	207.33	3.47	4.98	0.63274	1.00000	13.51	27.03	0	1.00000

Ratio* = *RaPiD*/*RaPiDr*

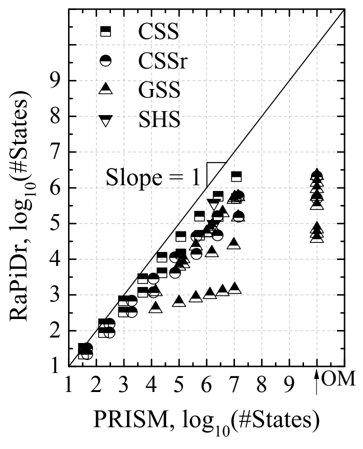
The results based on RaPiD and RaPiDr are compared in Table 6.1. The degree of abstraction can affect reduction strength, we therefore evaluate two extreme cases in RaPiDr: one is the minimum abstraction that does not hide any event; the other is the maximum abstraction that hides all events except those related to the reliability specification. The default algorithm in RaPiDr is to hide all events except specification events in the initial round and refine from there if the result is not satisfactory. Thus, we can often achieve a reduction closer to the maximum one and the minimum abstraction marks the 'worst' case of our method. The (*para.*) column contains the parameters for different systems, i.e., the numbers of pumps and customers for GSS and the number of clients for CSS and CSSr. States number and time cost for RaPiD without reductions are shown by exact values. To compare the effect of the reduction, the results for RaPiDr are shown in terms of ratios in the table, i.e., the ratios of the results from RaPiD to that of RaPiDr. A higher ratio indicates a higher degree of reduction. In the case that RaPiD runs out of memory (OM) and RaPiDr does not, then the ratio is presented as infinity (∞).

We have three main observations from Table 6.1. First, there is a general trend that the efficiency of reduction in state space and time increases as the system becomes larger. Moreover, there are some cases, e.g., GSS(2, 6), GSS(3, 4) and CSSr(10), that RaPiD runs out of memory and RaPiDr can still handle with relatively small number of states generated. Second, we observe that the time is reduced considerably in RaPiDr. This is because many states and redundant loops are removed in RaPiDr, and the convergence rate based on the value iteration is improved. The reduction in the total time implies that the time cost for the reduction is much less than its savings. Third, different levels of abstractions can save the state space and time to different extents because the more events are hidden, the greater reduction can be achieved. We also find that without hiding any events, our reduction method can still reduce the state spaces for GSS, CSSr and SHS, this is because our method can also remove internal τ transitions.

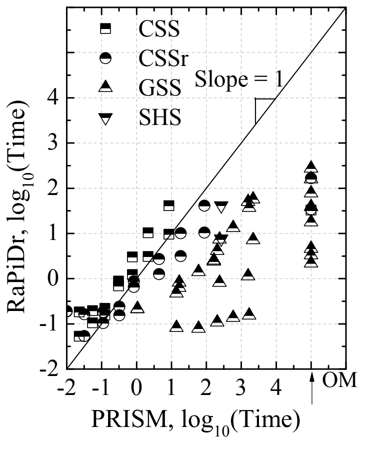
In addition, we notice that RaPiDr provides significantly better reductions on CSSr compared to the ones on CSS. Because CSS has few internal events, the state space is not reduced via the minimum abstraction, thus more time is spent. In contrast, the failure-repair loops in individual components of CSSr result in a dramatically large global space in RaPiD. However, they are effectively removed in RaPiDr even via the minimum abstraction. Therefore, the number of states and the time are reduced significantly. We remark that, by hiding communication events, we can still achieve a reduction ratio of more than 5 in some CSS cases. Moreover, it should be noticed that accuracy of abstraction is not affected by the difference between $\mathcal{R}_{min}$ and $\mathcal{R}_{max}$, as demonstrated by CSS cases.

The comparisons on PRISM and RaPiDr are presented in Figure 6.6. Exactly the same models are taken as inputs to PRISM and RaPiDr. Similarly, we show the best/worst possible reduction in RaPiDr. In Figure 6.6 (a), x- and y-axis of the plot are the number of states generated by PRISM and RaPiDr, respectively. Thus, the region below the diagonal line (i.e., slope = 1) indicates fewer states are generated by RaPiDr than by PRISM. The lower the point is, the higher degree of reduction RaPiDr provides. Note that the number of states is plotted in a logarithmic scale to focus on the comparison in terms of the order of magnitude. 'OM' stands for 'out of memory' and is only indicated qualitatively in the plot. We observe that RaPiDr is much more scalable than PRISM as all the scatters are within the lower-half region, and it can reduce the number of states by several orders of magnitude. There are several cases that PRISM runs out of memory whereas RaPiDr can still keep the number of states within 10^7, evidenced by the points in the vertical 'OM' line.

Similarly, Figure 6.6 (b) is a plot of the total time cost. We can observe that PRISM outperforms RaPiDr when the time taken is less than 10 seconds, although time factor is less critical for small systems. However, the benefit of adopting RaPiDr increases tremendously as the system grows larger. As shown in the plots, the scattered points tend to reside below and shift further away from the diagonal line for systems with the higher number of states



(a)



(b)

Figure 6.6: Comparisons between RaPiDr and PRISM on (a) states and (b) time (unit: second) in logarithmic scale

and longer verification time.

In summary, via comparing RaPiDr with RaPiD and PRISM, we find that our reduction method is more scalable and can reduce the total time significantly in many cases, especially for moderate and large systems. As RaPiDr reduces states and transitions in individual components, the overall states are reduced exponentially. Together with the removal of loops, reliability assessment can be speeded up significantly by RaPiDr.

RQ 2: How accurate is the assessment using our technique?

This question is essentially related to the validation of our assumption that ignoring certain events does not change the assessment result significantly. First, we compare assessment results from RaPiDr based on the maximum/minimum abstraction. The maximum abstraction hides all the communication events except specification events, thus it just provides approximations; and the minimum abstraction does not hide any communication events, thus it provides as accurate results as RaPiD does. The results are presented in Table 6.1, where $\mathcal{R}_{min}, \mathcal{R}_{max}$ are the actual results and $\mathcal{R}'_{min}$, $\mathcal{R}'_{max}$ are the approximations based on the maximum abstraction. The reliability results are reported to an accuracy up to five decimal places. It should be noted that the accuracy of prediction (i.e. difference between the actual result and approximation) is dominated by the choice of visible events. Further increasing the number of decimal places during computations can improve the precision but has been found to introduce insignificant effect on the accuracy. As shown in Table 6.1, $\mathcal{R}'_{min}$ is always less than or equal to $\mathcal{R}_{min}$, and $\mathcal{R}'_{max}$ is always larger than or equal to $\mathcal{R}_{max}$, which complies with our proof in Section 6.4.3. Moreover, we can achieve an approximation of the maximum reliability that is equal to the actual maximum for all these cases.

Comparing the minimum reliabilities ($\mathcal{R}_{min}$ and $\mathcal{R}'_{min}$), we can observe that: (1) for CSS and CSSr, results are not affected by the abstraction, and thus we can achieve the maximum reduction. (2) for GSS system, the results are accurate except for some cases when m is less than n. This is because the details for the customers and queues have been hidden after abstraction. For example of GSS(3, 1), the discrepancy between the approximation (0.98945) and the actual result (0.98965) is 0.2%. If the reliability requirement is to ensure a reliability above 0.989, the result is conclusive and the verification can stop with the maximum reduction; otherwise, the refinement process will be carried out. (3) for the SHS system, the minimum reliability calculated is 0. This means there is no useful information obtained in the first round of abstraction and reduction, and some more events should be added back in the subsequent refinement step. In the following, we experimentally show that that actual results can be obtained after several rounds of refinements.

We refine the abstract model by adding communication events back to improve the approximations. To have a quantitative evaluation, we keep track of the number of states and the assessment results during iterations of abstraction/refinement, and discuss the effect of the two refinement heuristics. In order to obtain a complete picture on how the state space and the assessment result change through refinements, instead of verifying against a given

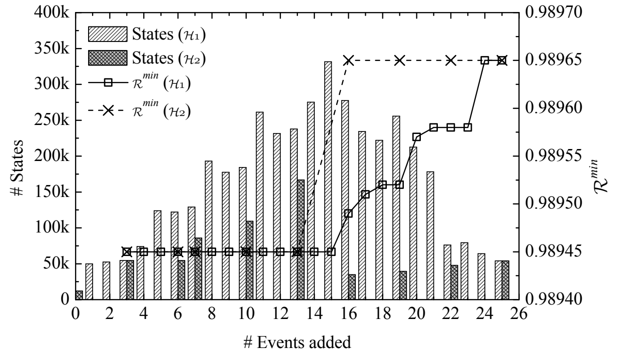


Figure 6.7: Refinement analysis results for GSS(3, 1)

reliability requirement in which case our approach may terminate early, we continue until all of the communication events have been added into Σ_v. In the following part, we take GSS(3, 1) for an example.

There are 25 communication events for GSS(3, 1), apart from the events in the specification. We compare the above-mentioned two heuristics in each refinement step, and the results are shown in Figure 6.7. The numbers of states are described in two bar charts and values can be read off via the y-axis on the left; and the resulting minimum probabilities are described in two sets of marked scatters and their values can be read off via the y-axis on the right. According to $\mathcal{H}_1$, 25 events are ranked according to the number of components synchronizing on the event. During each refinement, Σ_v is enlarged by adding one new event. Therefore, there are 25 bars and 25 points for $\mathcal{H}_1$. The event selection completes within one step and the time cost is negligible. For $\mathcal{H}_2$, we set the weighting factor w to 100 and divide the events into 10 groups according to the symmetry. We add the events by groups into Σ_v so there are 10 bars and points for $\mathcal{H}_2$ in Figure 6.7. The total time spent on selecting events based on $\mathcal{H}_2$ is 610 seconds.

We have the following observations based on Figure 6.7. First, the calculated minimum reliabilities converge to the actual result, regardless of the heuristics. Second, by comparing two heuristics, we can find $\mathcal{H}_2$ outperforms $\mathcal{H}_1$. The number of states generated by $\mathcal{H}_2$ is much less than that by $\mathcal{H}_1$; and the resulting probability converges to the actual result (0.98965) much faster for $\mathcal{H}_2$. At a particular point based on $\mathcal{H}_2$, when 16 events are added, the approximation reaches the actual result while the state space remains relatively small. This information can be used to guide reliability assessment for even larger GSS systems, i.e., the similar set of events can always be selected first. We remark that although $\mathcal{H}_2$ can provide relatively better performance than $\mathcal{H}_1$, it requires extra knowledge in dividing events into groups and more efforts in evaluating the effectiveness of each candidate group at each refinement iteration. Last, although the size of the state space may increase after refinement, it remains manageable along the way, i.e., much less than that generated by RaPiD (without the reductions) and PRISM.

Similar refinements have been conducted on SHS. As a result, we have identified a group of 24 (out of 33) events based on which the approximated minimum reliability converges to the actual result. The number of states is $73,054$ and verification time is 9.4s.

In summary, starting from hiding all events not mentioned in the specification, we can obtain safe approximations on the verification result. Refinement can incrementally help to improve the accuracy. Comparing the two heuristics, the results show that $\mathcal{H}_2$ often guide the refinement better than $\mathcal{H}_1$ while the time cost of using $\mathcal{H}_1$ is much lower.

RQ 3: How efficient is our method given a reliability requirement?

The efficiency of our approach is directly related to the reliability requirement. Intuitively, our approach terminates quicker given a less restrictive requirement. The question is then how sensitive our approach is regarding different reliability requirements. Table 6.2 shows the reliability assessment against different levels of reliability requirements, i.e., 0.40, 0.90, and 0.99. If the system's minimum reliability is above or equal to a requirement, RaPiDr is expected to report 'valid'; and otherwise 'invalid'. Besides the assessment results, we also

Table 6.2: Reliability assessment against reliability requirements

requirement		GSS(3, 1)	CSS(8)	CSSr(8)	SHS
	result	valid	valid	valid	valid
$\mathcal{R}^{min} \geq 0.40$	time (s)	10.06	3.18	3.22	3,039
	#iteration	4	1	1	34
	result	valid	invalid	valid	invalid
$\mathcal{R}^{min} \geq 0.90$	time (s)	12.14	100.00	3.00	3,037
	#iteration	4	19	1	34
	result	invalid	invalid	invalid	invalid
$\mathcal{R}^{min} \geq 0.99$	time (s)	649.48	98.99	95.58	3,040
	#iteration	26	19	19	34
$\mathcal{R}^{min}$	result	0.98965	0.43047	0.91540	0.63274
$\mathcal{R}^{max}$	result	1.00000	1.00000	1.0000	1.00000

record the time and the number of abstraction/refinement iterations needed. If 'invalid' is reported, RaPiDr still reports the actual assessment results, i.e., $\mathcal{R}^{min}$ and $\mathcal{R}^{max}$, shown in the last two rows of the table. As shown, if the reliability requirement is 0.40, all results are 'valid' and RaPiDr terminates quickly with only a few iterations. If the requirement is $\mathcal{R}^{min} \geq 0.90$, results from CSS(8) and SHS become 'invalid' (as expected) and more time and more iterations are needed. If the requirement is $\mathcal{R}^{min} \geq 0.99$, all results become 'invalid'. We remark that, the refinement here is based on $\mathcal{H}_1$, which is fully automatic without any domain information. If there is some domain information to guide the refinement, fewer iterations can be expected.

We conclude that RaPiDr can terminate early when the requirement is low or when the system is relatively reliable. For a less reliable system and relatively high requirement, RaPiDr may indeed take more iterations and time. In the worst case, it will run all the model versions between the maximum abstraction and minimum abstraction, thus take more time than that required to verify the minimum abstraction. This worst case situation is similar to other approaches on alleviating the state explosion problem [42, 40, 79, 75, 68]. In practice, distributed systems can easily have many components such that the state space is too large and reliability assessment based on the concrete model is impossible. Instead, RaPiDr can always produce a safe approximation based on abstract models; and with more time and computing resources, RaPiDr can produce increasingly more accurate approximations.

6.6 Related Work

In the field of reliability assessment, some works have considered each component as a single node and composed all the nodes in one Markov chain according to the system architectures or service usage scenarios [26, 70, 55, 50, 43, 20, 34, 49]. In particular, [128] has deduced close form reliability expressions for different architecture style of local components. This can only partially support non-determinism, as only limited number of architecture styles are discussed and the underlying reliability models are still DTMCs. In the field of multi-agent systems, some works have modeled the distributed components as Markov chains [36] or MDPs [127, 133] and produced results by verifying over LTL specifications. In our work, we model each component as an MDP and perform reliability assessment based on a set of MDPs. We believe this is more realistic as non-determinism is unavoidably part of distributed systems' behavior. This allows users to focus more on the behavior of individual component and avoids constructing the complete Markov model that involves considerable efforts or even is infeasible when there are many distributed components in the system. We further apply abstraction and refinement on the communication events to reduce state spaces.

There are two main approaches on alleviating state space explosion via the compositional verification, which are orthogonal to our approach. One is counterexample-guided abstraction refinement (CEGAR) [30, 31]. Recently, it has been extended to probabilistic systems based on predicate abstraction and predicate refinement according to spurious probabilistic paths/counterexamples analysis [68]. [68] only calculates upper bounds on maximum probabilities. Instead of working on predicates, our abstraction and refinement work on communication events among components. These provide both lower and upper bounds and avoid enumerating possibly large or even infinite set of paths.

The other approach is assume-guarantee verification [104, 67, 48], with its extensions in probabilistic systems [42, 40, 79, 75]. One of the main challenges for assume-guarantee reasoning is to automatically generate small assumptions. To overcome this challenge, [47, 105, 22] proposed alphabet refinement in a learning framework, however, that is only applicable for non-probabilistic systems. In [42, 40], the authors proposed an assumption generation approach based on automata learning techniques without termination being guaranteed. In [75], the authors proposed a complete and fully automatic solution by iteratively abstracting and refining the inferred assumptions via counterexamples analysis [62, 8, 5]. But finding a counterexample in probabilistic systems itself can be quite involved. Moreover, a practical investigation on assume-guarantee analysis in [33] has shown that the decomposition is critical and can affect the performance and scalability significantly. In many assume-guarantee reasoning works, there is no guidance on how the system shall be decomposed to achieve good performance. In contrast, our method works on MDPs and reduces every component according to a subset of the communication events, thus there is no need to find assumptions or suitable decomposition or analyze on counterexamples. In fact, as our approach can produce a safe approximation on each component, it can be used prior to the assume guarantee approaches for even larger systems that neither approach can handle alone; i.e., the individual component is first reduced via the attraction and reduction steps and the assume guarantee approaches are then applied to perform verification on the resulting components.

6.7 Summary

In this work, we proposed a scalable approach based on improved probabilistic model checking. An abstraction technique is adopted to effectively reduce the local state space of each component and thus the global state space by hiding local events and part of communication events. We proved that the results always produce safe approximations. To further refine the results, we have developed a framework to incrementally add the communication events back based on our proposed heuristics. Our empirical studies showed that our method could reduce the state space by several orders of magnitude and speed up reliability assessment.

Chapter 7

Conclusion

In this chapter, we briefly summarize the contributions of this Ph.D. thesis and discuss possible topics for the future research work.

7.1 Summary

The present study has systematically investigated the reliability analysis for non-deterministic systems based on Markov decision processes.

First, to perform reliability analysis on non-deterministic systems, we have proposed a framework that combines hypothesis testing and probabilistic model checking. This proposed method applies hypothesis testing to deterministic system components and uses the probabilistic model checking techniques to resolve non-determinism. Based on this proposed framework, we have designed and developed a toolkit RaPiD to support automatic software reliability analysis including reliability prediction, reliability distribution and sensitivity analysis. To demonstrate its usefulness, we have performed case studies on a stock trading system and a hospital therapy control system.

Second, to have some more further practical evaluation on our proposed framework, we have conducted reliability analysis on an ambient assisted living system called AMUPADH. We have constructed the reliability models from the design and implementation of the systems. Using our reliability analysis toolkit RaPiD, we have accomplished three groups of experiments to answer the questions of “What is the overall system reliability with known reliability value of each nodes?", “To reach a certain overall system reliability, how reliable should the sensors/networks be?" and “Which part among a sensor or network devices affects the overall reliability most significantly?". Experiments show that the overall system reliability can hardly reach 50%. From the analysis, we have identified that the overal system reliability can be improved most efficiently by improving the Wi-Fi network. That information is shown to be helpful to AMUPADH designers in improving the system reliability.

Probability reachability analysis serves as a fundamental step in our reliability analysis. To improve the efficiency of our approach, we have proposed divide-and-conquer algorithms to improve reachability analysis in both discrete time Markov chains (DTMCs) and Markov decision processes (MDPs), respectively. Because strongly connect components (SCCs) are one of the main reasons for the slow probability computation, the proposed approaches focus on abstracting SCCs via calculating the transition probabilities from their inputs to outputs. For DTMCs, we have repeatedly divided every SCC to several smaller parts and resolved loops in each until there is no loop in the states of the original SCC. To further cope with the non-determinism in MDPs, our divide-and-conquer algorithm has then been designed to work on blocks so as to maintain the number of probability distributions in a manageable level. To further reduce the redundant probability distributions, we have applied reductions based on the convex hull property. The efficiency could be enhanced significantly based on our approaches, as evidenced by a series of benchmark systems and two practical case studies.

Last but not the least, to improve the scalability of reliability analysis, we have proposed an approach to improve current probabilistic model checking via abstracting and refining the communications among distributed components. An abstraction technique is introduced to effectively reduce the local state space of each component and thus the global state space by hiding local events and part of communication events. We have proved that the results always produce safe approximations. To further refine the results, we have developed a framework to add back the communication events incrementally based on our proposed heuristics. Our empirical studies showed that our method could reduce the state space by several orders of magnitude and accelerate the reliability assessment.

7.2 Future Works

In this section, we outline the possible extensions of our work presented in this thesis.

- In Chapter 3, we have presented a framework that combines testing and model checking. For reliability analysis, we only focus on the special property where the global property is exact the same as local properties, i.e., reachability property. It cannot support the quantitative measurement like "the probability of warning messages failing to send before failure occurs is at least 0.99". This is related to the decomposition of global property into local properties, which is complicated in general. So we are motivated to study conditions under which such decomposition is sound (i.e., if the components satisfy the local properties, then the global property is guaranteed) or complete or both. A possible approach is to start with coarse sound decomposition and refine the decomposition through a method similar to counterexample guided abstraction refinement [30, 68].
- Moreover, our current work is based on the assumption of Markovian transfer among components. Although it is a widely used assumption to estimate most software systems, this assumption will not hold for some software applications, where the execution

history determines the next component to be executed. Therefore, another interesting line of future work will be on exploring a higher order state space representation for non-Markovian applications, where the level of history retained in the model needs to be determined in practice [50]. Moreover, our current reliability model only works on the discrete time Markov model. In this model, the probability of failure is considered as the reliability factor. However, in some cases, it may be more accurate to consider failure rate (i.e., the failure density that is a function of time), and continuous time Markov models (e.g., CTMC and CTMDP) are more suitable. We are motivated to extend current (parametric) probability model checking techniques to CTMC and CTMDP, so as to support various reliability analysis with consideration of the failure rate.

- In Chapters 3 and 4, we have demonstrated that our reliability analysis framework is useful in analyzing highly dynamic systems, e.g., a stock trading system, a therapy control system and an ambient assisted living room system. Obtaining the corresponding reliability models is always a prerequisite for the reliability analysis. In this thesis, the input reliability model of our reliability analysis toolkit RaPiD is an MDP. This MPD is manually obtained from either system architecture or usage scenarios. In the future, we shall extend this toolkit to obtain MDP or its coarser MDP skeleton automatically from a given high-level document. There are already some works on automatically transferring from high-level system model (e.g., architecture model or UML model) into some semantic models like labeling transition system [118, 87]. Inspiring by [98, 96] that integrating probability into CSP and Event-B, we shall incorporate the reliability information into the high-level model, based on which we shall then develop an automatic way to generate MDPs for reliability analysis.
- In Chapter 5, the algorithms based on divide-and-conquer techniques can eliminate SCCs so as to accelerate reachability analysis in DTMCs and MDPs. There are two

future directions. Current method can only work for non-parametric reachability analysis; therefore, it can only improve the efficiency for reliability prediction. However, for reliability distribution and sensitivity analysis, there are some transition probabilities that are unknown and hereby marked as parameters in an MDP. In fact, our current techniques can help to reduce an MDP into an acyclic one, based on which the parameter reachability analysis can be much easier. Therefore, one direction is on applying current techniques to enhance parametric probability reachability analysis [61, 95, 60]. The other is on improving the current techniques. Our algorithms require the prior knowledge of the dividing parameters that are currently manually set based on users' experience. Different systems may require the different set of parameters for the best performance. Therefore, one potential topic is to find more efficient dividing strategies.

- In Chapter 6, the reliability specification is expressed based on labeling transition system. One possible line of future research is to apply our abstract and refinement methods to assessing a certain reliability that is specified in temporal logic, e.g., probabilistic computation tree logic [63] or linear temporal logic [107]. On the other hand, the efficiency of our abstraction and refinement heavily depends on the selections of the alphabet to hide. The selection strategy is currently based on the two proposed heuristics. The future work is to develop more effective ways in selecting the most critical subset of communication events that can reduce the state space most while producing relatively accurate results.

- In addition, the thesis focuses on reliability analysis for non-deterministic systems. The proposed framework and the underlying techniques presented in Chapter 3, 5 and 6 based on Markov decision processes can also be applied to the analysis of other quantitative properties (e.g., availability [70], performability [52, 64, 116, 14, 15], security [94, 93, 13]) in more application domains (e.g., web service composition[125, 25, 23], wireless sensor network [139, 138, 137]).

Bibliography

[1] `http://www.comp.nus.edu.sg/~yanliu/reliability.html`. 4.5

[2] `http://www.comp.nus.edu.sg/~pat/rel/mdpcut`. 5.5, 5.5.2.1

[3] `http://www.comp.nus.edu.sg/~pat/rel/distributed`. 6.5

[4] RaPiD. `http://www.comp.nus.edu.sg/~pat/rapid`. 1, 3.5, 4.2, A.1, A.1

[5] E. Ábrahám, N. Jansen, R. Wimmer, J.-P. Katoen, and B. Becker. DTMC model checking by SCC reduction. In *International Conference on Quantitative Evaluation of SysTems (QEST)*, pages 37–46, 2010. 5.1, 5.1, 5.2.1, 5.2.2, 1, 5.3, 5.3.3, 5.6, 6.2, 6.4.2, 6.4.3, 6.6

[6] H. Aloulou, M. Mokhtari, T. Tiberghien, J. Biswas, and L. J. H. Kenneth. A semantic plug & play based framework for ambient assisted living. In *International Conference On Smart Homes and Health Telematics (ICOST)*, pages 165–172, 2012. 4.5.4

[7] S. C. Althoen and R. McLaughlin. Gauss - Jordan reduction: a brief history. In *The American Mathematical Monthly*, volume 94(2), pages 130–142, 1987. 5.1, 5.2.2, 5.2.2

[8] M. E. Andrés, P. R. D'Argenio, and P. V. Rossum. Significant diagnostic counterexamples in probabilistic model checking. In *Hardware and Software: Verification and Testing (HVT)*, pages 129–148, 2008. 5.1, 5.1, 5.2.1, 5.3, 5.6, 6.4.2, 6.6

[9] J. Aspnes and M. Herlihy. Fast Randomized Consensus Using Shared Memory. *Journal of Algorithms*, 15(1):441–460, 1990. 5.5.1

[10] M. Avriel. *Nonlinear Programming: Analysis and Methods*. Dover Publishing, 2003. 3.4.4

[11] A.Wald. Sequential tests of statistical hypotheses. *Annal of mathematical statistics 16*, 2:117–186, 1945. 3, 3.2

[12] A.Wald. *Sequential Analysis*. Wiley, 1947. 3.2, 3.2

[13] G. Bai, J. Hao, J. Wu, Y. Liu, Z. Liang, and A. Martin. Trustfound: Towards a formal foundation for model checking trusted computing platforms. In *Formal Methods (FM)*, pages 110–126. Springer, 2014. 7.2

[14] C. Baier, E. Hahn, B. Haverkort, H. Hermanns, and J. Katoen. Model checking for performability. *Mathematical Structures in Computer Science*, 23(4):751–795, 2013. 7.2

[15] C. Baier, B. R. Haverkort, H. Hermanns, and J.-P. Katoen. Model checking meets performance evaluation. *ACM SIGMETRICS Performance Evaluation Review*, 32(4):10–15, Mar. 2005. 2, 7.2

[16] C. Baier and J. Katoen. *Principles of Model Checking*. The MIT Press, 2008. 2, 2.1, 2.1.2, 2.1.2, 2.2, 3.1, 3.4.1, 3.4.4, 4.2.2, 5.1, 5.2.1, 6.1, 6.2, 6.3.1, 6.3.2, 6.4.2

[17] J. L. Bentley, F. P. Preparata, and M. G. Faust. Approximation algorithms for convex hulls. *Communications of the ACM*, 25(1):64–68, 1982. 5.4.4

[18] J. Biswas, M. Mokhtari, J. S. Dong, and P. Yap. Mild dementia care at home-integrating activity monitoring, user interface plasticity and scenario verification. In *International Conference On Smart Homes and Health Telematics (ICOST)*, pages 160–170, 2010. 4, 4.7

[19] J. Bogdoll, L. M. F. Fioriti, A. Hartmanns, and H. Hermanns. Partial order methods for statistical model checking and simulation. In *Formal Techniques for Distributed Systems (FMOODS/FORTE)*, pages 59–74. Springer, 2011. 3.6

[20] R. Calinescu, C. Ghezzi, M. Kwiatkowska, and R. Mirandola. Self-adaptive software needs quantitative verification at runtime. *Communications of the ACM*, 55(9):69–77, Sept. 2012. 6.6

[21] S. Cattani and R. Segala. Decision algorithms for probabilistic bisimulation*. In *Concurrency Theory (CONCUR)*, pages 371–386. Springer, 2002. 5.4.4

[22] S. Chaki and O. Strichman. Optimized l*-based assume-guarantee reasoning. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 276–291. Springer, 2007. 6.6

[23] P. Chan and M. Lyu. Dynamic web service composition: A new approach in building reliable web service. In *International Conference on Advanced Information Networking and Applications (AINA)*, pages 20–25, March 2008. 7.2

[24] K. Chatterjee and T. A. Henzinger. Value iteration. In *25 Years of Model Checking*, pages 107–138. Springer, 2008. 6.3.2

[25] M. Chen, T. Tan, J. Sun, Y. Liu, J. Pang, and X. Li. Verification of functional and non-functional requirements of web service composition. In *International Conference on Formal Engineering Methods (ICFEM)*, volume 8144 of *Lecture Notes in Computer Science*, pages 313–328. Springer, 2013. 7.2

[26] R. C. Cheung. A user-oriented software reliability model. *IEEE Transactions on Software Engineering*, SE-6(2):118–125, 1980. 1.1.1, 2, 3.1, 3.4.1, 3.6, 4.2.1, 4.6, 6.1, 6.6

[27] F. Ciesinski and C. Baier. Liquor: A tool for qualitative and quantitative linear time analysis of reactive systems. In *International Conference on Quantitative Evaluation of Systems (QEST)*, volume 6, pages 131–132, 2006. 4.6

[28] F. Ciesinski, C. Baier, M. Grosser, and J. Klein. Reduction techniques for model checking Markov decision processes. In *International Conference on Quantitative Evaluation of SysTems (QEST)*, pages 45–54. IEEE, 2008. 5.3.3, 5.6

[29] E. Clarke, A. Donzé, and A. Legay. Statistical model checking of mixed-analog circuits with an application to a third order δ- σ modulator. In *Hardware and Software: Verification and Testing (HVT)*, pages 149–163. Springer, 2009. 3.6

[30] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *International Conference on Computer Aided Verification (CAV)*, pages 154–169. Springer, 2000. 6.6, 7.2

[31] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement for symbolic model checking. *Journal of the ACM*, 50(5):752–794, 2003. 6.6

[32] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking.* The MIT Press, 1999. 3.1, 6.1

[33] J. M. Cobleigh, G. S. Avrunin, and L. A. Clarke. Breaking up is hard to do: an investigation of decomposition for assume-guarantee reasoning. In *International Symposium on Software Testing and Analysis (ISSTA)*, pages 97–108. ACM, 2006. 6.6

[34] V. Cortellessa and V. Grassi. Reliability modeling and analysis of service-oriented architectures. In *Test and Analysis of Web Services*, pages 339–362. Springer, 2007. 6.6

[35] M. De Berg, M. Van Kreveld, M. Overmars, and O. C. Schwarzkopf. *Computational geometry*. Springer, 2000. 5.4.4

[36] M. I. Dekhtyar, A. J. Dikovsky, and M. K. Valiev. Temporal verification of probabilistic multi-agent systems. In *Pillars of computer science*, pages 256–265. Springer, 2008. 6.6

[37] C. Eisentraut, H. Hermanns, J. Schuster, A. Turrini, and L. Zhang. The quest for minimal quotients for probabilistic automata. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 16–31. Springer, 2013. 6.4.2

[38] I. Epifani, C. Ghezzi, R. Mirandola, and G. Tamburrelli. Model evolution by run-time parameter adaptation. In *International Conference on Software Engineering (ICSE)*, pages 111–121, 2009. 3.6

[39] W. H. Farr and O. D. Smith. Statistical modeling and estimation of reliability functions for software (smerfs) users guide. *Naval Surface Warfare Center*, 1993. 4.6

[40] L. Feng, T. Han, M. Kwiatkowska, and D. Parker. Learning-based compositional verification for synchronous probabilistic systems. In *International Symposium on Automated Technology for Verification and Analysis (ATVA)*, pages 511–521. Springer, 2011. 6.5.2, 6.6

[41] L. Feng, M. Kwiatkowska, and D. Parker. Compositional verification of probabilistic systems using learning. In *International Conference on Quantitative Evaluation of SysTems (QEST)*, pages 133–142. IEEE CS Press, 2010. 6.5.1

[42] L. Feng, M. Kwiatkowska, and D. Parker. Automated learning of probabilistic assumptions for compositional reasoning. In *International Conference on Fundamental Approaches to Software Engineering (FASE)*, pages 2–17. Springer, 2011. 6.5.2, 6.6

[43] A. Filieri, C. Ghezzi, and G. Tamburrelli. Run-time efficient probabilistic model checking. In *International Conference on Software Engineering (ICSE)*, pages 341–350. ACM, 2011. 3.6, 6.1, 6.6

[44] V. Forejt, M. Kwiatkowska, G. Norman, and D. Parker. Automated verification techniques for probabilistic systems. In *Formal Methods for Eternal Networked Software Systems (FMENSS)*, pages 53–113. Springer, 2011. 5.1, 6.3.2

[45] J. Geldenhuys, M. B. Dwyer, and W. Visser. Probabilistic symbolic execution. In *International Symposium on Software Testing and Analysis (ISSTA)*, pages 166–176. ACM, 2012. 3.6

[46] M. Gheorghiu, D. Giannakopoulou, and C. S. Păsăreanu. Refining interface alphabets for compositional verification. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 292–307. Springer, 2007. 6.5.1

[47] M. Gheorghiu, C. S. Păsăreanu, and D. Giannakopoulou. Automated assume-guarantee reasoning by abstraction refinement. In *International Conference on Computer Aided Verification (CAV)*, pages 135–148. Springer, 2008. 6.6

[48] D. Giannakopoulou, C. S. Păsăreanu, and J. M. Cobleigh. Assume-guarantee verification of source code with design-level assumptions. In *International Conference on Software Engineering (ICSE)*, pages 211–220. IEEE Computer Society, 2004. 6.6

[49] S. Gilmore, L. Gönczy, N. Koch, P. Mayer, M. Tribastone, and D. Varró. Non-functional properties in the model-driven development of service-oriented systems. *Software and System Modeling*, 10(3):287–311, 2011. 6.6

[50] S. Gokhale. Architecture-based software reliability analysis: Overview and limitations. *IEEE Transactions on Dependable and Secure Computing*, 4(1):32–40, 2007. 2, 3.4.1, 3.6, 6.1, 6.6, 7.2

[51] S. S. Gokhale and K. S. Trivedi. Analytical models for architecture-based software reliability prediction: A unification framework. *IEEE Transactions on Reliability*, 55(4):578–590, 2006. 2, 4.6

[52] S. S. Gokhale, W. E. Wong, J. R. Horgan, and K. S. Trivedi. An analytical approach to architecture-based software performance and reliability prediction. *Performance Evaluation*, 58(4):391–412, 2004. 4.6, 7.2

[53] K. Goseva-Popstojanova, A. P. Mathur, and K. S. Trivedi. Comparison of architecture-based software reliability models. In *International Symposium on Software Reliability Engineering (ISSRE)*, pages 22–33, 2001. 4.6

[54] G. Gössler and J. Sifakis. Component-based construction of deadlock-free systems: Extended abstract. In *Foundations Software Technology and Theoretical Computer Science (FSTTCS)*, volume 2914 of *Lecture Notes in Computer Science*, pages 420–433. Springer, 2003. 1

[55] K. Goševa-Popstojanova and K. S. Trivedi. Architecture-based approach to reliability assessment of software systems. *Performance Evaluation*, 45(2-3):179–204, 2001. 2, 3.4.1, 3.6, 4.6, 6.1, 6.6

[56] T. Grenager, R. Powers, and Y. Shoham. Dispersion Games: General Definitions and Some Specific Learning Results. In *Association for the Advancement of Artificial Intelligence (AAAI)*, pages 398–403, 2002. 5.5.1

[57] L. Gui. Reliability analysis of non-deterministic systems. In *Formal Methods (FM) Doctor Symposium*, TRA5/14. NUS, 2014. 1.4

[58] L. Gui, J. Sun, Y. Liu, Y. J. Si, J. S. Dong, and X. Y. Wang. Combining model checking and testing with an application to reliability prediction and distribution. In *International Symposium on Software Testing and Analysis (ISSTA)*, pages 101–111. ACM, 2013. 1.4, 5.5.2.1, 6.1, 6.5

[59] L. Gui, J. Sun, S. Song, Y. Liu, and J. S. Dong. SCC-based improved reachability analysis for markov decision processes. In *International Conference on Formal Engineering Methods (ICFEM)*. Springer, 2014. 1.4, 5.6

[60] E. M. Hahn, T. Han, and L. Zhang. Synthesis for pctl in parametric markov decision processes. In *NASA Formal Methods (NFM)*, pages 146–161. Springer, 2011. 7.2

[61] E. M. Hahn, H. Hermanns, B. Wachter, and L. Zhang. Param: A model checker for parametric markov models. In *International Conference on Computer Aided Verification (CAV)*, pages 660–664. Springer, 2010. 7.2

[62] T. Han and J.-P. Katoen. Counterexamples in probabilistic model checking. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 4424 of *LNCS*, pages 72–86. Springer Berlin Heidelberg, 2007. 6.6

[63] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. In *Formal Aspects of Computing 6(5)*, pages 512–535, 1994. 3.3, 7.2

[64] B. R. Haverkort. *Performance of Computer Communication Systems: A Model-Based Approach.* John Wiley & Sons, Inc., New York, NY, USA, 1998. 2, 7.2

[65] D. Heimbold and D. Luckham. Debugging ada tasking programs. *EEE Software*, 2(2):47–57, 1985. 6.5.1

[66] D. Henriques, J. G. Martins, P. Zuliani, A. Platzer, and E. M. Clarke. Statistical model checking for Markov decision processes. In *International Conference on Quantitative Evaluation of SysTems (QEST)*, pages 84–93. IEEE, 2012. 3.6

[67] T. A. Henzinger, S. Qadeer, and S. K. Rajamani. You assume, we guarantee: Methodology and case studies. In *International Conference on Computer Aided Verification (CAV)*, pages 440–451. Springer, 1998. 6.6

[68] H. Hermanns, B. Wachter, and L. Zhang. Probabilistic cegar. In *International Conference on Computer Aided Verification (CAV)*, pages 162–175. Springer, 2008. 6.5.2, 6.6, 7.2

[69] C. Y. Huang and M. R. Lyu. Optimal testing resource allocation, and sensitivity analysis in software development. *IEEE Transactions on Reliability*, 54(4):592–603, 2005. 3.6

[70] A. Immonen and E. Niemel. Survey of reliability and availability prediction methods from the viewpoint of software architecture. *Software and Systems Modeling*, 7(1):49–65, 2008. 1.1.1, 2, 3.1, 3.4, 3.4.1, 3.6, 6.1, 6.6, 7.2

[71] Z. Jelinski and P. Moranda. Software reliability research. *Freiberger, W.(ed.): Statistical Computer Performance Evaluation*, pages 465 – 484, 1972. 3.1

[72] A. M. Johnson, Jr. and M. Malek. Survey of software tools for evaluating reliability, availability, and serviceability. *ACM Computing Surveys*, 20(4):227–269, Dec. 1988. 4.6

[73] K. Kanoun, M. Kaaniche, J.-C. Laprie, and S. Metge. Sorel: A tool for reliability growth analysis and prediction from statistical failure data. In *International Symposium on Fault-Tolerant Computing (FTCS)*, pages 654–659. IEEE, 1993. 4.6

[74] J.-P. Katoen, I. S. Zapreev, E. M. Hahn, H. Hermanns, and D. N. Jansen. The Ins and Outs of The Probabilistic Model Checker MRMC. In *International Conference on Quantitative Evaluation of SysTems (QEST)*, pages 167–176. IEEE Computer Society, 2009. `www.mrmc-tool.org`. 4.6

[75] A. Komuravelli, C. S. Păsăreanu, and E. M. Clarke. Assume-guarantee abstraction refinement for probabilistic systems. In *International Conference on Computer Aided Verification (CAV)*, pages 310–326. Springer, 2012. 6.5.2, 6.6

[76] S. Krishnamurthy and A. P. Mathur. On the estimation of reliability of a software system using reliabilities of its components. In *International Symposium on Software Reliability Engineering (ISSRE)*, pages 146–155. IEEE, 1997. 4.6

[77] P. Kubat. Assessing reliability of modular software. *Operations Research Letters*, 8(1):35–41, 1989. 1.1.1, 3.1

[78] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *International Conference on Computer Aided Verification (CAV)*, pages 585–591, 2011. 3.5.4, 4.6, 5.5.1, 6.3.2, 6.5.2

[79] M. Kwiatkowska, G. Norman, D. Parker, and H. Qu. Assume-guarantee verification for probabilistic systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 23–37. Springer, 2010. 6.2, 6.5.2, 6.6

[80] M. Kwiatkowska, D. Parker, and H. Qu. Incremental quantitative verification for Markov decision processes. In *International Conference on Dependable Systems and Networks (DSN)*, pages 359–370. IEEE, 2011. 5.3.3, 5.6

[81] J. C. Laprie and K. Kanoun. *Handbook of software Reliability Engineering*, chapter Software Reliability and System Reliability, pages 27–69. McGraw-Hill, New York, NY, 1996. 1.1.1, 3.1, 4.6

[82] R. Lassaigne and S. Peyronnet. Approximate planning and verification for large markov decision processes. In *Annual ACM Symposium on Applied Computing (SAC)*, pages 1314–1319. ACM, 2012. 3.6

[83] V. Lee, Y. Liu, X. Zhang, C. Phua, K. Sim, J. Zhu, J. Biswas, J. Dong, and M. Mokhtari. Acarp: Auto correct activity recognition rules using process analysis toolkit (PAT). In *International Conference On Smart Homes and Health Telematics*

(ICOST), volume 7251 of *Lecture Notes in Computer Science*, pages 182–189. Springer, 2012. 4.3.1

[84] A. Legay, B. Delahaye, and S. Bensalem. Statistical model checking: An overview. In *Runtime Verification (RV)*, pages 122–135, 2010. 3.6

[85] P. Liggesmeyer and T. Ackermann. Applying reliability engineering: empirical results, lessons learned, and further improvements. In *International Symposium on Software Reliability Engineering (ISSRE), Fast Abstracts and Industrial Practices*, pages 263–271, Germany, 1998. 4.6

[86] B. Littlewood and J. L. Verrall. A bayesian reliability growth model for computer science. *Journal of the Royal Statistical Society, Ser. A (Applied Statistics)*, pages 332 – 346, 1973. 3.1

[87] S. Liu, Y. Liu, É. André, C. Choppy, J. Sun, B. Wadhwa, and J. S. Dong. A formal semantics for complete uml state machines with communications. In *International Conference on Integrated Formal Methods (IFM)*, pages 331–346, 2013. 7.2

[88] Y. Liu, L. Gui, and Y. Liu. Mdp-based reliability analysis of an ambient assisted living system. In *International Symposium on Formal Methods (FM) Industry Track*, pages 688–702, Singapore, May 2014. 1.4, 5.5.2.1, 6.5.1

[89] Y. Liu, X. Zhang, J. S. Dong, Y. Liu, J. Sun, J. Biswas, and M. Mokhtari. Formal analysis of pervasive computing systems. In *International Conference on Engineering of Complex Computer Systems (ICECCS)*, pages 169–178, 2012. 4.3.1, 6.5.1

[90] M. R. Lyu and A. Nikora. Casre: a computer-aided software reliability estimation tool. In *International Workshop on Computer-Aided Software Engineering (CASE)*, pages 264–275, Montreal, Canada, 1992. IEEE. 4.6

[91] M. R. Lyu, A. P. Nikora, and W. H. Farr. A systematic and comprehensive tool for software reliability modeling and measurement. In *International Symposium on Fault-Tolerant Computing (FTCS)*, pages 648–653. IEEE, 1993. 4.6

[92] M. R. Lyu, S. Rangarajan, and A. P. A. van Moorsel. Optimal allocation of test resources for software reliability growth modeling in software development. *IEEE Transactions on Reliability*, 51(2):183–192, 2001. 3.6

[93] B. B. Madan, K. Goševa-Popstojanova, K. Vaidyanathan, and K. S. Trivedi. Modeling and quantification of security attributes of software systems. In *International Conference on Dependable Systems and Networks (DSN)*, pages 505–514. IEEE, 2002. 7.2

[94] B. B. Madan, K. Goševa-Popstojanova, K. Vaidyanathan, and K. S. Trivedi. A method for modeling and quantifying the security attributes of intrusion tolerant systems. *Performance Evaluation*, 56(1):167–186, 2004. 7.2

[95] I. Meedeniya and L. Grunske. An efficient method for architecture-based reliability evaluation for evolving systems with changing parameters. In *International Symposium on Software Reliability Engineering (ISSRE)*, pages 229–238. IEEE, 2010. 3.6, 7.2

[96] C. Morgan, T. S. Hoang, and J. Abrial. The Challenge of Probabilistic *Event B* - Extended Abstract. In *Formal Specification and Development in Z and B (ZB)*, volume 3455 of *LNCS*, pages 162–171. Springer, 2005. 7.2

[97] C. Morgan and A. McIver. pgcl: Formal reasoning for random algorithms. *South African Computer Journal*, pages 14–27, 1999. 3.6

[98] C. Morgan, A. McIver, K. Seidel, and J. W. Sanders. Refinement-oriented probability for csp. *Formal Aspects in Computing*, 8(6):617–647, 1996. 7.2

[99] J. D. Musa. Operational profiles in software-reliability engineering. *IEEE Transactions on Software Engineering*, 10(2):14–32, 1993. 3.4

[100] J. D. Musa and K. Okumoto. A logarithmic poisson execution time model for software reliability measurement. *Malaiya, Y. K.; Srimani, P. K. (ed.): Software Reliability Models - Theoretical Developments, Evaluation & Applications*, pages 23 – 31, 1990. 3.1

[101] J. Nehmer, M. Becker, A. Karshmer, and R. Lamm. Living assistance systems: an ambient intelligence approach. In *International Conference on Software Engineering (ICSE)*, pages 43–50, 2006. 4

[102] A. Padovitz, S. W. Loke, and A. B. Zaslavsky. On uncertainty in context-aware computing: Appealing to high-level and same-level context for low-level context verification. In *International Workshop on Ubiquitous Computing (IWUC)*, pages 62–72, 2004. 4.1

[103] D. L. Parnas. The influence of software structure on reliability. In *ACM SIGPLAN Notices*, volume 10, pages 358–362. ACM, 1975. 4.6

[104] C. S. Păsăreanu, M. B. Dwyer, and M. Huth. Assume-guarantee model checking of software: A comparative case study. In *Theoretical and Practical Aspects of SPIN Model Checking*, pages 168–183. Springer, 1999. 6.6

[105] C. S. Păsăreanu, D. Giannakopoulou, M. Bobaru, J. Cobleigh, and H. Barringer. Learning to divide and conquer: applying the L* algorithm to automate assume-guarantee reasoning. *Formal Methods in System Design*, 32(3):175–205, 2008. 6.6

[106] R. Pietrantuono, S. Russo, and K. S. Trivedi. Software reliability and testing time allocation: An architecture-based approach. *IEEE Transactions on Software Engineering*, 36:323–337, 2010. 3.6, 4.6

[107] A. Pnueli. The temporal logic of programs. In *The IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 46–57, 1977. 3.3, 7.2

[108] M. L. Puterman. Markov decision processes. *Handbooks in operations research and management science*, 2:331–434, 1990. 3.4.1

[109] M. L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*, volume 414. John Wiley & Sons, 2009. 2

[110] S. Ramani, S. S. Gokhale, and K. S. Trivedi. Srept: software reliability estimation and prediction tool. *Performance evaluation*, 39(1):37–60, 2000. 4.6

[111] A. Ranganathan, J. Al-Muhtadi, and R. H. Campbell. Reasoning about uncertain contexts in pervasive computing environments. *IEEE Pervasive Computing*, 3(2):62–70, Apr. 2004. 4.1

[112] G. Rodrigues, D. Rosenblum, and S. Uchitel. Using scenarios to predict the reliability of concurrent component-based software systems. In *Fundamental Approaches to Software Engineering (FASE)*, pages 111–126. Springer, 2005. 3.6

[113] A. W. Roscoe. Model-checking CSP. *A classical mind: essays in honour of CAR Hoare*, pages 353–378, 1994. 6.3.1, 6.3.2

[114] H. Sandoh. Reliability demonstration testing for software. *IEEE Transactions on Reliability*, 40(1):117–119, 1991. 3.1

[115] K. Sharma, R. Garg, C. K. Nagpal, and R. K. Garg. Selection of optimal software reliability growth models using a distance based approach. *IEEE Transactions on Reliability*, 59(2):266–276, 2010. 3.2

[116] V. Sharma and K. Trivedi. Quantifying software performance, reliability and security: An architecture-based approach. *Journal of Systems and Software*, 80(4):493–509, 2007. 7.2

[117] S. Song, L. Gui, J. Sun, Y. Liu, and J. S. Dong. Improved reachability analysis in DTMC via divide and conquer. In *International Conference on Integrated Formal Methods*, pages 162–176, 2013. 5.1, 5.2.1, 5.2.2, 5.6, 6.2, 6.4.2, 6.4.3

[118] S. Song, J. Zhang, Y. Liu, M. Auguston, J. Sun, J. Dong, and T. Chen. Formalizing and verifying stochastic system architectures using monterey phoenix. *Software and Systems Modeling*, pages 1–19, 2014. 7.2

[119] W. J. Stewart. *Introduction to the numerical solution of Markov chains.* Princeton University Press, 1994. 2.2.2

[120] J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis.* Berlin, New York: Springer-Verlag, 2002. 5.2.2

[121] J. Sun, Y. Liu, J. S. Dong, and J. Pang. PAT: Towards flexible verification under fairness. In *International Conference on Computer Aided Verification (CAV)*, pages 709–714. Springer Berlin Heidelberg, 2009. 5.5

[122] J. Sun, Y. Liu, J. S. Dong, and J. Pang. PAT: Towards flexible verification under fairness. In *International Conference on Computer Aided Verification (CAV)*, volume 5643 of *LNCS*, pages 709–714. Springer, 2009. 6.3.2

[123] J. Sun, S. Z. Song, and Y. Liu. Model checking hierarchical probabilistic systems. In *International Conference on Formal Engineering Methods (ICFEM)*, pages 388–403, 2010. 1.4, 6.2, 6.3.1, 6.3.2

[124] O. Tal, C. McCollin, and T. Bendell. Reliability demonstration for safety-critical systems. *IEEE Transactions on Reliability*, 50(2):194–203, 2001. 3.2

[125] T. H. Tan, M. Chen, É. André, J. Sun, Y. Liu, and J. S. Dong. Automated runtime recovery for qos-based service composition. In *international conference on world wide*

web (WWW), pages 563–574. International World Wide Web Conferences Steering Committee, 2014. 7.2

[126] R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160, 1972. 5.2.1, 15, 14

[127] M. Valiev and M. Dekhtyar. Complexity of verification of nondeterministic probabilistic multiagent systems. *Automatic Control and Computer Sciences*, 45(7):390–396, 2011. 6.6

[128] W.-L. Wang, D. Pan, and M.-H. Chen. Architecture-based software reliability modeling. *Journal of Systems and Software*, 79(1):132–146, 2006. 4.6, 6.6

[129] W.-L. Wang, D. Pan, and M.-H. Chen. Architecture-based software reliability modeling. *J. Syst. Softw.*, 79(1), 2006. 4.6

[130] M. Weiser. The computer for the 21st century. *Scientific American*, 265(3):94–104, 1991. 4

[131] W. Wen-Li and D. Scannell. An architecture-based software reliability modeling tool and its support for teaching. In *ASEE/IEEE Frontiers in Education Conference*, pages T4C–T4C, 2005. 4.6

[132] D. M. Woit. *Estimating software reliability with hypothesis testing.* Citeseer, 1993. 1.1.1, 3.1, 3.1, 3.2, 3.6

[133] T. Wongpiromsarn, A. Ulusoy, C. Belta, E. Frazzoli, and D. Rus. Incremental synthesis of control policies for heterogeneous multi-agent systems with linear temporal logic specifications. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5011–5018. IEEE, 2013. 6.6

[134] S. M. Yacoub, B. Cukic, and H. H. Ammar. Scenario-based reliability analysis of component-based software. In *International Symposium on Software Reliability Engineering (ISSRE)*, pages 22–31. IEEE, 1999. 3.6

[135] H. Younes. *Verification and Planning for Stochastic Processes with Asynchronous Events*. PhD thesis, Carnegie Mellon, 2005. 3.2

[136] H. Younes and R. G. Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In *International Conference on Computer Aided Verification (CAV)*, pages 223–235. Springer, 2002. 3.6

[137] M. Zheng, D. Sanán, J. Sun, Y. Liu, J. S. Dong, and Y. Gu. State space reduction for sensor networks using two-level partial order reduction. In *Verification, Model Checking, and Abstract Interpretation (VMCAI)*, pages 515–535, 2013. 7.2

[138] M. Zheng, J. Sun, Y. Liu, J. S. Dong, and Y. Gu. Towards a model checker for nesc and wireless sensor networks. In S. Qin and Z. Qiu, editors, *International Conference on Formal Engineering Methods (ICFEM)*, volume 6991 of *Lecture Notes in Computer Science*, pages 372–387. Springer, 2011. 7.2

[139] M. Zheng, J. Sun, D. Sanán, Y. Liu, J. S. Dong, and Y. Gu. Towards bug-free implementation for wireless sensor networks. In *9th International Conference on Embedded Networked Sensor Systems (SenSys 2011)*, pages 407–408. ACM, 2011. 7.2

Appendix A

RaPiD User Guide

A.1 Basic Features

Download and run RaPiD.exe from `http://www.comp.nus.edu.sg/~pat/rapid`. Noted that if the current PC has no MATLAB installed, there is a need to install MCRInstaller.exe (available at [4]) before using RaPiD, to view graphical plots. Reliability analysis activities including reliability prediction, distribution and sensitivity analysis can be carried out as follows.

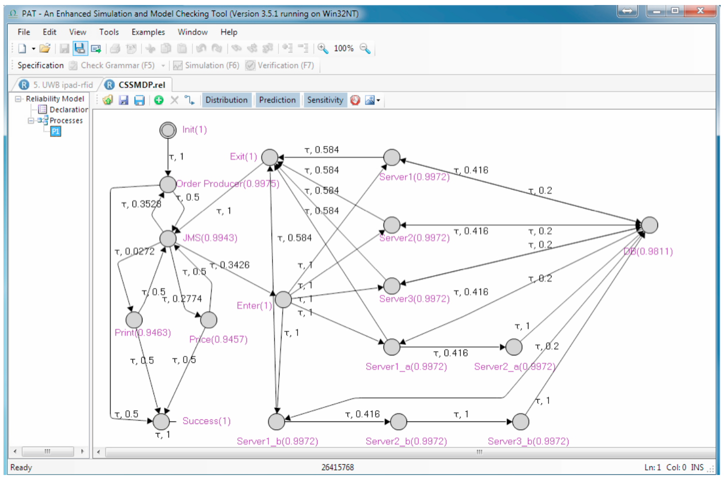
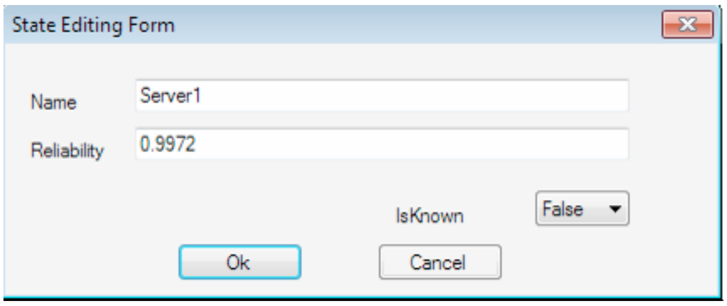


Figure A.1: Reliability model in RaPiD editor

With RaPiD editor, the first task is to construct a reliability model. A call cross system (CCS) model is shown in Figure A.1 as an running example below. All the examples are in the Example folder, which is in the same directory with RaPiD.exe file, which can be downloaded at [4].

Double click a node or an edge to edit the details for a state or a transition, as shown in Figure A.2 and A.3, respectively.

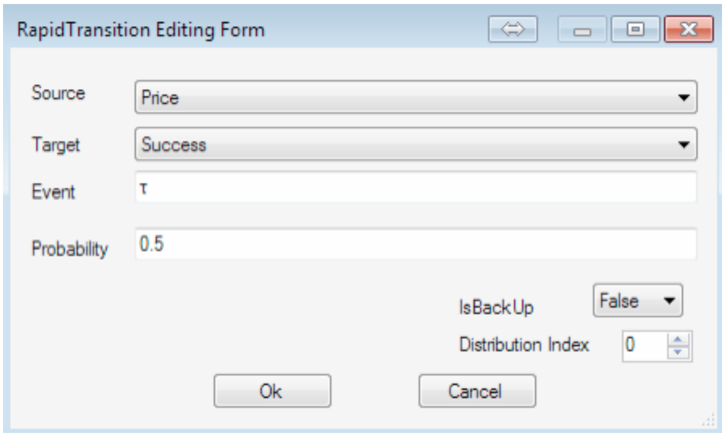

State Editing Form

Name: Server1

Reliability: 0.9972

IsKnown: False

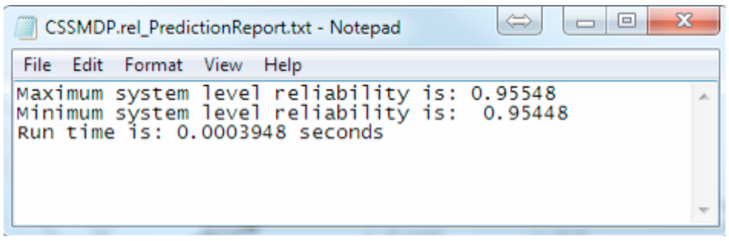
Figure A.2: State editing form



Source: Price
Target: Success
Event: τ
Probability: 0.5
IsBackUp: False
Distribution Index: 0

Figure A.3: Transition editing form

By clicking Prediction button, RaPiD then calculates the minimum and maximum reliabilities and displays the results using the default text editor, as shown in Figure A.4.


CSSMDP.rel_PredictionReport.txt - Notepad
File Edit Format View Help

```
Maximum system level reliability is: 0.95548
Minimum system level reliability is:  0.95448
Run time is: 0.0003948 seconds
```

Figure A.4: Reliability prediction result presented in a text viewer

For reliability distribution, right-click process button, select Process Details in the drop-down menu as shown in Figure A.5, and then write the overall reliability requirement for the system as shown in Figure A.6.

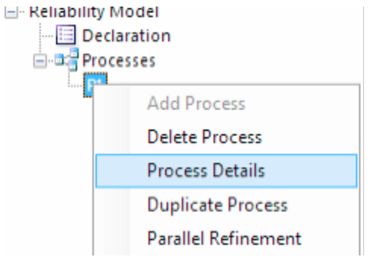
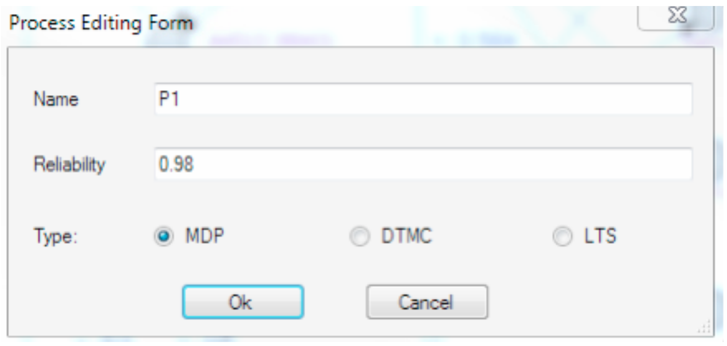


Figure A.5: A drop-down menu at a process

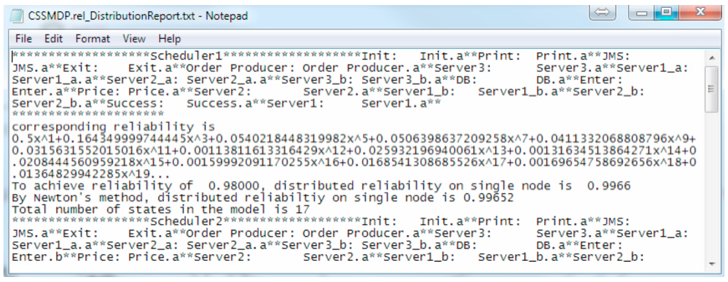


Process Editing Form

Name	P1		
Reliability	0.98		
Type:	[● MDP]	○ DTMC	○ LTS

Figure A.6: Overall reliability requirement editing form used for reliability distribution

By clicking Distribution button, RaPiD outputs text report, as shown in Figure A.7 which presents the details on the schedulers and distributed reliability requirements. In addition, RaPiD outputs a Matlab figure, which is a plot of the system reliability over component reliability, as shown in Figure A.8. Clicking legend button to view legend. Clicking zoom in/out button to adjust the presentation of different level of details of the figure.



CSSMDP.rel_DistributionReport.txt - Notepad

```
*******************Scheduler1*******************Init:   Init.a**Print:  Print.a**JMS:
JMS.a**Exit:    Exit.a**Order Producer: Order Producer.a**Server3:      Server3.a**Server1_a:
Server1_a.a**Server2_a: Server2_a.a**Server3_b: Server3_b.a**DB:        DB.a**Enter:
Enter.a**Price: Price.a**Server2:       Server2.a**Server1_b:   Server1_b.a**Server2_b:
Server2_b.a**Success:   Success.a**Server1:     Server1.a**
*********************
corresponding reliability is
0.5x^1+0.164349999744445x^3+0.0540218448319982x^5+0.0506398637209258x^7+0.0411332068808796x^9+
0.0315631552015016x^11+0.0011381161331642 9x^12+0.025932196940061x^13+0.0013163451386427 1x^14+0
.0208444560959218x^15+0.00159992091170255x^16+0.0168541308685526x^17+0.00169654758692656x^18+0
.0136482994 2285x^19...
To achieve reliability of  0.98000, distributed reliability on single node is  0.9966
By Newton's method, distributed reliabiltiy on single node is 0.99652
Total number of states in the model is 17
*******************Scheduler2*******************Init:   Init.a**Print:  Print.a**JMS:
JMS.a**Exit:    Exit.a**Order Producer: Order Producer.a**Server3:      Server3.a**Server1_a:
Server1_a.a**Server2_a: Server2_a.a**Server3_b: Server3_b.a**DB:        DB.a**Enter:
Enter.b**Price: Price.a**Server2:       Server2.a**Server1_b:   Server1_b.a**Server2_b:
```

Figure A.7: Reliability distribution result in a text viewer

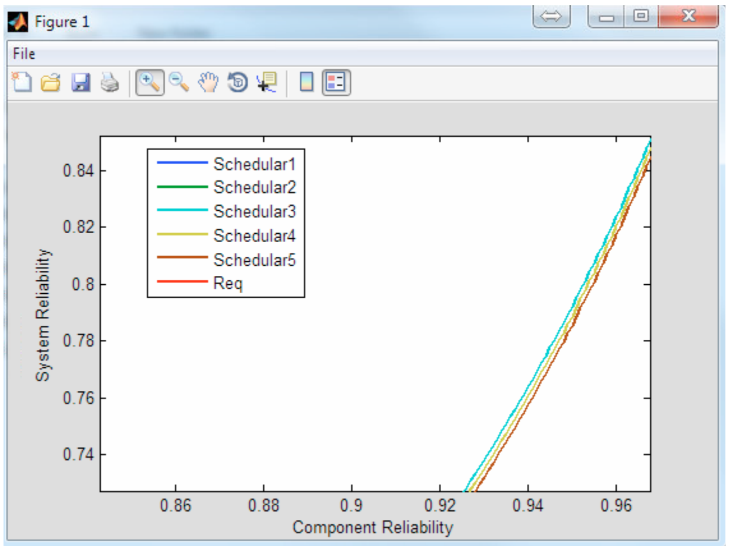
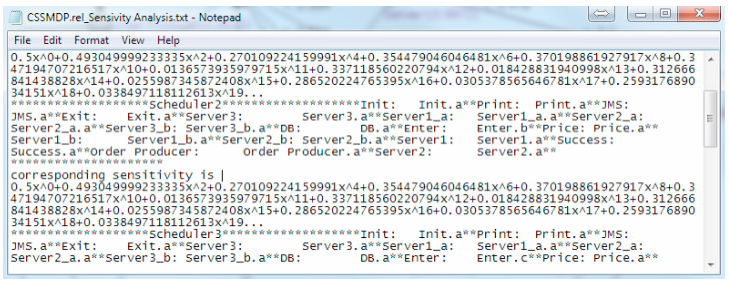


Figure A.8: Reliability distribution result in a Matlab figure

For sensitivity analysis, it first requires to specify a component/state on which the sensitivity analysis is carried out. Right-click that node and select Sensitivity Analysis in the drop-down menu. Similarly, a plot and a text report on sensitivity analysis are generated, as shown in Figure A.9 and A.10.



```
0.5x^0+0.493049999233335x^2+0.270109224159991x^4+0.354479046046481x^6+0.370198861927917x^8+0.3
47194707216517x^10+0.013657393597971 5x^11+0.337118560220794x^12+0.018428831940998x^13+0.312666
841438828x^14+0.025598734587240 8x^15+0.286520224765395x^16+0.030537856564678 1x^17+0.2593176890
34151x^18+0.033849711811261 3x^19...
********************Scheduler2********************Init:   Init.a**Print:  Print.a**JMS:
JMS.a**Exit:    Exit.a**Server3:        Server3.a**Server1_a:   Server1_a.a**Server2_a:
Server2_a.a**Server3_b: Server3_b.a**DB:        DB.a**Enter:    Enter.b**Price: Price.a**
Server1_b:      Server1_b.a**Server2_b: Server2_b.a**Server1:   Server1.a**Success:
Success.a**Order Producer:      Order Producer.a**Server2:      Server2.a**
********************
corresponding sensitivity is |
0.5x^0+0.493049999233335x^2+0.270109224159991x^4+0.354479046046481x^6+0.370198861927917x^8+0.3
47194707216517x^10+0.013657393597971 5x^11+0.337118560220794x^12+0.018428831940998x^13+0.312666
841438828x^14+0.025598734587240 8x^15+0.286520224765395x^16+0.030537856564678 1x^17+0.2593176890
34151x^18+0.033849711811261 3x^19...
********************Scheduler3********************Init:   Init.a**Print:  Print.a**JMS:
JMS.a**Exit:    Exit.a**Server3:        Server3.a**Server1_a:   Server1_a.a**Server2_a:
Server2_a.a**Server3_b: Server3_b.a**DB:        DB.a**Enter:    Enter.c**Price: Price.a**
```

Figure A.9: Sensitivity analysis result in a text viewer

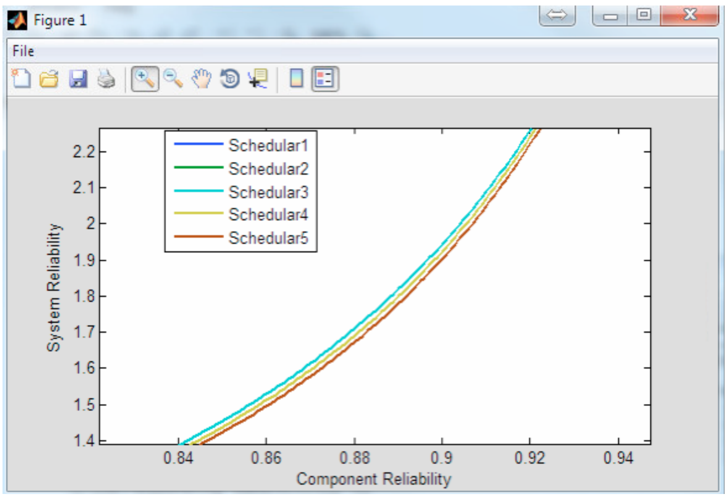


Figure A.10: Sensitivity analysis result in a Matlab figure

A.2 Advanced Features

In this section, some advanced features on reliability assessment for distributed system with a control on state space are presented. For distributed system, instead of a single process, RaPiD models each system in an MDP and the overall system is the parallel composition of all those MDPs. A simple model of a distributed controller device system in Figure A.11 is shown as a running example in this section.

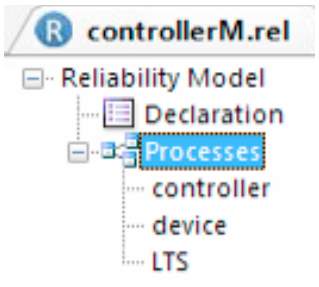


Figure A.11: A set of processes

The reliability is the probability of the system model satisfying the specification that is modeled in labeled transition system. This reliability assessment can be initiated by right clicking on the Processes and selecting Parallel Refinement option, as shown in Figure A.12.

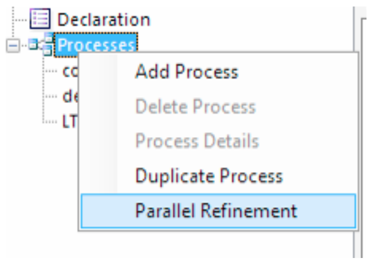
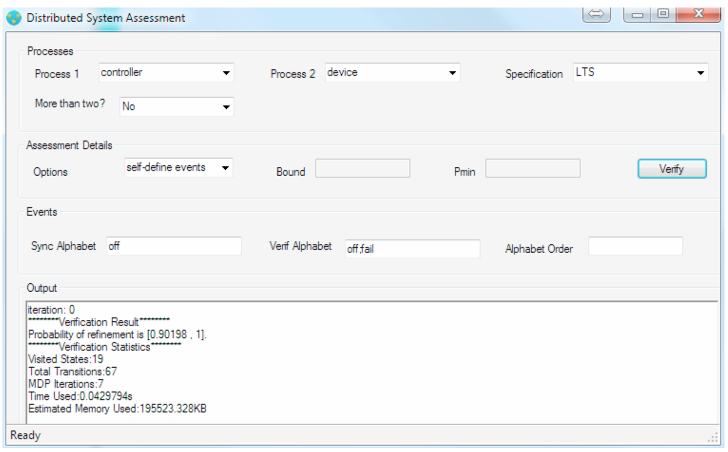


Figure A.12: Reliability assessment based on refinement for a parallel composition of a set of processes


Distributed System Assessment

Processes

Process 1: controller | Process 2: device | Specification: LTS

More than two?: No

Assessment Details

Options: self-define events | Bound: | Pmin: | Verify

Events

Sync Alphabet: off | Verif Alphabet: off,fail | Alphabet Order:

Output

```
iteration: 0
*********Verification Result*********
Probability of refinement is [0.90198 , 1].
*********Verification Statistics*********
Visited States:19
Total Transitions:67
MDP Iterations:7
Time Used:0.0429794s
Estimated Memory Used:195523.328KB
```

Ready

Figure A.13: Reliability assessment form for distributed system via abstraction and refinement on communication alphabet

In the form, as shown in Figure A.12, the first panel is the place to specify the distributed systems and the specification. The second panel is the place to specify assessment details, with three options available. Events panel is used to specify the synchronization alphabet, verification alphabet, as well as the alphabet order that is used to guide refinement process. Result is displayed in Output panel.