STAMINA: STOCHASTIC APPROXIMATE MODEL-CHECKER FOR INFINITE-STATE ANALYSIS

by

Thakur Neupane

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Computer Engineering

Approved:

Zhen Zhang, Ph.D. Major Professor	Koushik Chakraborty, Ph.D. Committee Member
Chris Winstead, Ph.D. Committee Member	Richard S. Inouye, Ph.D. Vice Provost for Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2019

ABSTRACT

STAMINA: STochastic Approximate Model-checker for INfinite-state Analysis

by

Thakur Neupane, Master of Science

Utah State University, 2019

Major Professor: Zhen Zhang, Ph.D.
Department: Electrical and Computer Engineering

Continuous Time Markov Chains (CTMCs) are one of the most prominent probabilistic models that are used in performance and dependability analysis of various control and communication systems, network protocols, queuing and reliability problems, and biological-systems. Probabilistic model checking has demonstrated significant potential in analyzing the probabilistic behaviors of complex concurrent systems in various application domains. Usually model checking involves exhaustive exploration of the system's state space, which limits their scalability because of the infamous state explosion problem.

This thesis presents a new infinite state CTMC model checker, STAMINA, with efficient and scalable model truncation for probabilistic verification. STAMINA uses a novel state space approximation method to truncate large and possibly infinite-state CTMC models to finite-state representations that are amenable to existing probabilistic model checkers. It also uses a new property-based state exploration approach that reduces the size of state space further without losing the analysis accuracy. Demonstration of our prototype tool on several benchmark models shows promising results in terms of analysis efficiency and accuracy, compared with a state-of-the-art infinite-state CTMC model checker.

(55 pages)

PUBLIC ABSTRACT

STAMINA: STochastic Approximate Model-checker for INfinite-state Analysis

Thakur Neupane

Reliable operation of every day use computing system, from simple coffee machines to complex flight controller system in an aircraft, is necessary to save time, money, and in some cases lives. System testing can check for the presence of unwanted execution but cannot guarantee the absence of such. Probabilistic model checking techniques have demonstrated significant potential in verifying performance and reliability of various systems whose execution are defined with likelihood. However, its inability to scale limits its applicability in practice.

This thesis presents a new model checker, STAMINA, with efficient and scalable model truncation for probabilistic verification. STAMINA uses a novel model reduction technique generating a finite state representations of large systems that are amenable to existing probabilistic model checking techniques. The proposed method is evaluated on several benchmark examples. Comparisons with another state-of-art tool demonstrates both accuracy and efficiency of the presented method.

ACKNOWLEDGMENTS

The work presented in this thesis would not have been possible without the support of many people who supported me in many ways over the past two years. I take this opportunity to thank everyone who helped me make this thesis a possibility.

First and foremost, I would like to express my deepest gratitude to my advisor Dr. Zhen Zhang for his invaluable contribution in this work. He always believed in me and provided me with every bit of guidance and expertise throughout my graduate program. I would also like to thank him for introducing me to three wonderful people: Dr. Chris J. Myers, Dr. Curtis K. Madsen and Dr. Hao Zheng, who would later play inalienable roles in shaping this research. I'm deeply indebted to Chris for his innovative idea and ingenious suggestions. I'm extremely grateful to Curtis and Hao for their invaluable insights into the implementation of idea. Besides my advisor, I would like to thank the rest of the committee members - Dr. Koushik Chakraborty and Dr. Chris Winstead, for their valuable insights and comments on this research.

I am grateful to the wonderful faculty and staff of the ECE department. I'd like to acknowledge the assistance of Tricia Brandenburg, Kathy Phippen and Diane Buist for handling all the administrative requirements of my degrees allowing me to focus on my research and study.

If it were not for my family, I would not be the person I am today. I can't thank them enough for their unwavering love and support. I am forever indebted to my grandparents, uncles and aunts for their blessings and support. Being the eldest among my siblings, my brothers and sisters always looked up at me as their role model; however, my success was only possible with their love and support.

Most of all, I cannot begin to express my gratitude to my best friend since childhood, love of my life - Sangita for all of her sacrifice, love and support.

Thakur Neupane

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACRONYMS

AP	atomic proposition
BFS	breadth-first search
CEGAR	counterexample-guided abstraction refinement
CME	chemical master equation
CSL	continuous stochastic logic
CTMC	continuous-time Markov chain
DTMC	discrete-time Markov chain
FSP	finite-state projection
JQN	Jackson queuing network
MC	Markov chain
MCM	method of conditional moments
MDP	Markov decision process
MTBDD	multi-terminal binary decision diagram
PCTL	probabilistic computation tree logic
POR	partial order reduction
PRISM	probabilistic symbolic model checker
SG	state graph
STAMINA	stochastic approximate model-checker for infinite-state analysis
STAR	stochastic analysis of biochemical reaction networks

NOTATION

$\mathcal{C}$	CTMC model
$\mathbf{S}$	set of states
$\mathbf{R}$	transition rate matrix
$\mathbf{s}_0$	initial state
$\mathbf{E}(\mathbf{s})$	exit rate for state $\mathbf{s}$
$\mathcal{D}$	DTMC model
$\mathbf{P}$	transition probability matrix
$\mathcal{G}$	state graph
δ	set of state transitions
$\mathbf{s} \xrightarrow{r} \mathbf{s}'$	transition from state $\mathbf{s}$ to $\mathbf{s}'$ with rate r
$\hat{\kappa}(\mathbf{s})$	reachability-value of state $\mathbf{s}$
$\varkappa$	reachability threshold used to truncate the CTMC model
$\mathcal{C}\rfloor_{\varkappa}$	CTMC model truncated with parameter $\varkappa$
$\mathbf{s}_{abs}$	abstract state
Φ	CSL state formula
φ	CSL path formula
$\mathsf{S}_{\sim p}$	steady-state CSL operator
$\mathsf{P}_{\sim p}$	transient CSL operator
$\mathcal{U}$	CSL until operator
$\mathcal{X}$	CSL next operator
I	time interval $[t_1, t_2]$
${}^{i}\mathbf{S}$	set of states with depth i
${}^{i}\zeta$	sum of reachability values of states ${}^{i}\mathbf{S}$
ϵ	analysis precision
$\mathbb{R}_{\geq 0}$	set of non-negative real number
$\mathbb{Z}_{\geqslant 0}$	set of non-negative integer

CHAPTER 1

INTRODUCTION

Now more than ever, our daily life is becoming heavily dependent on computerized systems. Such systems range from simple coffee machines to complex airplanes. Reliable operation of these computer systems is necessary to save time, money, and in some cases lives. Incorrect operation of a coffee machine or a smart phone causes inconvenience for a small amount of time, but failure in the software controlling critical systems like flight controllers and medical equipments can have catastrophic consequences. Three cancer patients died because of the race condition in the control software of Therac-25 radiation machine for treatment of cancer patients. In another incidence, Ariane 5 rocket lost its inertial reference system and exploded mid-air costing more than 500 millions US dollars, because of invalid data conversion from 64-bit floating point to 16-bit integer causing overflow in hardware. To prevent such dramatic consequences, we have to make sure that every possible execution scenario of such systems do not lead to failure.

System validation is the process of asserting that a computerized system complies with its intended functions. System validation techniques can be broadly categorized into two categories: testing and simulation, and formal verification.

Testing is a basic system validation process in which a real implementation of the system or its prototype is evaluated for correctness against system specification. However, considering the complexity of a typical modern day system with tens to thousands of components interacting in complex manner, testing those systems with limited input combinations may check the presence of bugs but not the absence of them. Testing, therefore, cannot guarantee correctness of critical systems. *Simulation*, on the other hand, can be used to investigate the behavior of the system without actually developing it. Simulation, being similar to testing, is not suitable to exhaustively find subtle system errors.

Formal verification is a process of mathematically proving or disproving the correctness

of the system against some formal specification. Given a system model and some specification that the system must satisfy, formal verification provides provable guarantees that the system satisfies the specification. If not, it often generates evidence in the form of a sequence of events that lead to the failure of system against that particular specification. For example, consider a mutual exclusion protocol which describes the behavior of two or more processes sharing access to a common resource. Two most important specifications to this system are "at most one process can be in the critical section at any time" and "if multiple processes are trying to enter the critical section, one of them will eventually do so". With the help of formal verification, we can guarantee that any correct mutual exclusion protocol satisfies these two requirements. However, in practice, formal verification sometimes requires human guidance to prove the correctness of a given system.

Model checking is an automated formal verification technique that systematically checks whether a finite-state model of a system satisfies a formal specification. Because of the fact that model checking is fully automated, it has drawn a lot of interest in academia and industry. In a typical model checking process, user often constructs the model of the system under consideration using some formal model description language, and formalizes the property to be checked with some property specification language. The model checker then explicitly explores all the possible states of the system model and checks if the property is satisfied by the model. If the property is violated, it produces the counter-example, which is a sequence of states/transitions that, starting from the initial state, leads to a failure state of the property. *Probabilistic model checking* refers to a range of techniques for calculating the probability of the occurrence of certain events during the execution of probabilistic system. Probabilistic model checkers can answer, for example, "how likely the main processor in embedded control system fails to cause the shutdown within 5 years?" or "the probability that an airbag in a car deploys within 0.01s".

Many real-life hardware and software systems exhibit probabilistic behavior. Markov Chains (MCs) are commonly used to model probabilistic systems; and *Markov decision processes* (MDPs) are used to describe the nondeterminism of concurrent probabilistic systems.

Continuous Time Markov Chains (CTMCs) are extensively used in analyzing the performance and reliability of control and communication systems, network protocols, queuing systems, and biological systems. CTMCs are used in order to quantify the rate of failures in reliable systems (e.g. embedded control system [1]). Quality of service (QoS) parameters like performance and availability of queuing networks (e.g. tandem queuing network [2], Jackson queuing network [3]) are expressed in terms of probability. Recent efforts to understand the behavior of biological processes (e.g. genetic toggle switch [4]) further supports the need of probabilistic modeling as those systems are inherently probabilistic.

Model checking generally requires explicitly enumerating all the reachable states of a model. But real-world computing systems are often complex and large, and the size of their state space grows exponentially with respect to their number of processes and variables. Therefore it can be computationally intractable to exhaustively enumerate the entire state space—a problem typically known as *state explosion*; and probabilistic model checking is no exception to that. Numerous state representation, reduction, and approximation methods have been proposed to combat the state explosion problem.

Researchers have proposed techniques like symbolic model checking [5,6], bisimulation minimization [7–9], probabilistic abstraction-refinement [10–12], symmetry reduction [13,14] and partial order reduction [15] mainly to analyze discrete-time finite-state probabilistic systems. However, these techniques do not scale well and are not directly applicable for systems with infinite states. In order to use techniques developed for finite-state models to analyze infinite-state models, it is required to manually truncate those during modeling. This truncation introduces uncertainty that cannot be quantified during verification. Automatic truncation based approach [16] has been presented to analyze certain finite and infinite-state continuous-time probabilistic systems. However, this approach can again show exponential state growth with respect to the exploration depth.

1.1 Contributions

This thesis presents a new infinite-state CTMC model checker, STAMINA: STochastic Approximate Model-checker for INfinite-state Analysis, with a novel model truncation tech-

nique that handles the state explosion problem. We also propose a novel property-based state exploration approach that helps to reduce the size of state space further without losing the analysis accuracy. Additionally, to account for the error introduced by the model truncation, we propose an error estimation method based on an abstract state.

The main contributions of this research include:

- State truncation algorithms: Our state exploration method maintains a probability estimate, reachability-value, of each path being explored in the state space, and when the currently explored path probability drops below a specified threshold, it halts exploration of this path. All transitions exiting this state are redirected to an abstract state to estimate the truncation error.

- Property-based state exploration method: We have also developed a new property-based state exploration technique, which identifies the path prefixes that are known to satisfy or dissatisfy specific path formulas; and shortens them by making the last state of each prefix absorbing during state exploration.

- The development of a prototype tool: State truncation algorithms and property-based exploration method are implemented as a prototype tool. Markov chain analysis on the approximate state space constructed is performed using PRISM's explicit-state CTMC model checker engine.

- The evaluation of these methods by their application to case studies from a variety of fields, including genetic-toggle switch, grid world robot navigation system and models of queuing networks.

This work has been integrated into the PRISM model checker [17]. STAMINA can be downloaded freely from https://github.com/formal-verification-research/stamina.

1.2 Thesis Outline

The remaining thesis is organized as follows. Chapter 2 describes related work. Chapter 3 provides the background information including the CTMC models and their truncations, probabilistic model checking procedure of CTMC, and the PRISM probabilistic model checker. In Chapter 4, a methodology for approximating and refining the state space is presented. It describes the truncation process to construct the sufficient state space that is enough to analyze CTMC models upto a certain precision. Chapter 5 applies the presented methods to analyze several CTMC models from different application areas. It also compares the efficiency and accuracy of STAMINA with a state-of-the-art infinite-state CTMC model checker. Finally, Chapter 6 concludes this thesis and presents possible future research directions.

CHAPTER 2

LITERATURE REVIEW

State space explosion has drawn the attention of many researchers in the model checking arena. Over the time, numerous state reduction and approximation algorithms have been proposed to alleviate this problem. In this chapter we review these major reduction techniques which enable the analysis of large and infinite-state probabilistic systems. Techniques like symbolic model checking [5, 6], bisimulation minimization [7–9], probabilistic abstraction-refinement [10–12], symmetry reduction [13, 14] and partial order reduction [15] have been mainly extended to discrete-time, finite-state probabilistic systems. To the best of our knowledge, only a few tools can analyze infinite-state probabilistic models, namely, INFAMY: An Infinite-State Markov Model Checker [16], and STAR: STochastic Analysis of biochemical Reaction networks [18].

2.1 Verification of Finite-State Probabilistic Systems

Symbolic model checking algorithms quickly construct a very compact representation of extremely large probabilistic models. Parker implemented *multi-terminal binary decision diagrams* (MTBDDs) based symbolic model checking for probabilistic systems [6]. MTBDDs based model checking is best applied to the MDP models, typically of randomized, distributed algorithms as they have few distinct probability values. MTBDDs have successfully analyzed the shared coin protocol from [19], an MDP model, with more than 7.5 billions states [6]. However, MTBDDs are often inefficient for models with lots of different and distinct probability/rate values because of the inefficient representation of the solution vectors. Generally, CTMC models whose state transition rate is a function of state values contain many distinct rate values. As a result, symbolic model checker can run out of memory while verifying a typical CTMC model with as few as 73000 states [6].

Bisimulation minimization is the process of merging states that are bisimulation equivalent to minimize the resulting state space. Probabilistic bisimilarity is an equivalence relation for probabilistic labeled transition systems introduced in [20]. In this equivalence relation, equivalent states have the same label and the same probability to make a transition into any given equivalence class. In probabilistic setting, bisimulation minimization can reduce the state space up to a logarithmic scale [7]. However, bisimulation minimization requires the exploration of the full state space [21–23]. It might not be always possible to explore the entire state space, if the state space is very large or infinite.

Probabilistic abstraction-refinement is a two step process. Abstraction is a process of coarsely merging multiple states to achieve better reduction, while ensuring a simulation relation between the abstract and concrete Markov models. During abstraction, explicit transition probability on concrete model is replaced by an interval with the maximal and minimal probabilities for taking the equivalent transition in abstract model. Refinement is a process of adding more details to the abstract model by partitioning an abstract state into two or more states. Hermanns *et al.* extended *counterexample-guided abstraction refinement* (CEGAR) in probabilistic setting [12]. CEGAR algorithm constructs an abstract model from the given concrete model and model checks the abstract model. If the abstract model satisfies the property, concrete model also satisfies it. If the abstract model fails the property, it is not known whether its concrete counterpart fails the property, because the abstract model introduces additional behaviors. The resulting counterexample is then used to refine the abstract model. Then the verification repeats until sufficient refinement obtained. Similarly, a theoretical framework to reduce CTMCs using three-valued abstraction is presented in [10]. However, the accuracy of the abstraction is affected by the partitioning of the state space. Moreover, refinement of abstraction to improve abstraction process in case of inconclusive results is not discussed.

Symmetry reduction presented in [13] exploits the component symmetry present in a model. Symmetry reduction can reduce the state space from M^N to $N!$ where N is the number of symmetric components and M is the number of states in each component.

This method also first builds the full model then reduces to quotient model. Furthermore, detecting the symmetric components in a given model can be expensive to compute [13].

Partial Order Reduction (POR) is a technique to reduce the size of the state space by exploring only a subset of all possible interleaving of concurrently executed transitions. POR is extensively studied for model reduction in non probabilistic context [24–29]. Groesser and Baier extended and applied the POR technique for MDPs in [15]. To our best knowledge, POR has not been applied to reduce the CTMC models.

2.2 Verification of Infinite-State Probabilistic Systems

All the works discussed in Section 2.1 deal with the verification of finite-state probabilistic systems. Those techniques, however, cannot directly be applied to infinite-state probabilistic systems.

Lapin *et al.* presented STAR tool [18] primarily to analyze biochemical reaction networks. It approximates solutions to the *chemical master equation* (CME) using the *method of conditional moments* (MCM) [30] that combines moment-based and state-based representations of probability distributions. This hybrid approach represents species with low concentrations using a discrete stochastic description and numerically integrates a small master equation using the fourth order Runge-Kutta method over a small time interval [31]; and solves a system of conditional moment equations for higher concentration species, conditioned on the low concentration species. This method has been optimized to drop unlikely states and add likely states on-the-fly. STAR relies on a well-structured underlying Markov process with small sensitivity on the transient distribution. Also, it mainly reports state reachability probabilities, instead of checking a given probabilistic property.

A similar approach to this thesis work is the method presented in INFAMY [16]. INFAMY is a truncation-based approach which explores the model's state space up to a certain finite depth k. The k-truncation of a CTMC defined by the truncation depth k is a CTMC model where all the states with depth, defined as minimal the length of any finite path starting from the initial state and ending in current state, larger than k is abstracted to a single state. INFAMY provides dynamic-uniformization-based error estima-

tion method [32], including *finite state projection (FSP)*, *uniform*, and *layered* to maintain a small error probability. The error probability computed during the model checking introduced by the truncation depends on the depth of state exploration. In order to maintain insignificant error probability, higher exploration depth is required which in turn causes the exponential growth of the truncated state space.

STAMINA does not use the same depth to truncate all the paths in state space as INFAMY does, but rather terminates each individual path based on its likelihood to contribute to the analysis of the model. It does so by exploring only those states whose path probability estimate is greater than a threshold. STAMINA also employs a property-based search to further maintain the manageable state space growth.

CHAPTER 3

PRELIMINARIES

In this chapter, we present the relevant background knowledge for this thesis. Section 3.1 and 3.2 present the formal definitions of CTMC and its finite truncation, and property specification language, namely, *continuous stochastic logic* (CSL), to describe the property that can be verified for CTMC models. We briefly describe the CSL model checking procedure in Section 3.3. Finally, Section 3.4 presents introduction to the PRISM probabilistic model checker and the syntax of the PRISM modeling language.

3.1 Continuous-time Markov Chains and Finite Truncation

The high-level modeling formalism used in this thesis is the CTMC model, which is defined below.

Definition 1. *Let AP be a fixed set of finite set of atomic propositions then a (labeled) CTMC can be defined as a tuple* $\mathcal{C} = \langle \mathbf{S}, \mathbf{R}, \mathbf{s}_0, \mathbf{L} \rangle$ *where:*

- $\mathbf{S}$ *is a non-empty set of states;*
- $\mathbf{R} : \mathbf{S} \times \mathbf{S} \to \mathbb{R}_{\geq 0}$ *is the transition rate matrix;*
- $\mathbf{s}_0 \in \mathbf{S}$ *is the initial state;*
- $\mathbf{L} : \mathbf{S} \to 2^{AP}$ *is a labeling function that assigns each state* $\mathbf{s} \in \mathbf{S}$ *the set* $\mathbf{L}(\mathbf{s})$ *of atomic propositions that are valid in the state.*

Each element $\mathbf{R}(\mathbf{s}, \mathbf{s}')$ of transition rate matrix $\mathbf{R}$ gives the rate of transition happening between states $\mathbf{s}$ and $\mathbf{s}'$, denoted by $\mathbf{s} \xrightarrow{\mathbf{R}(\mathbf{s},\mathbf{s}')} \mathbf{s}'$. Typically, in a CTMC model, each state has more than one enabled transitions. A transition is said to be enabled if $\mathbf{R}(\mathbf{s}, \mathbf{s}') > 0$; and the probability of executing this transition within t time units is determined by the transition rate $\mathbf{R}(\mathbf{s}, \mathbf{s}')$, expressed as $1 - e^{-\mathbf{R}(\mathbf{s},\mathbf{s}')t}$. The CTMC resides in a state $\mathbf{s}$ before

taking any enabled transition and the delay before such transition occurring from this state is determined by the *exit rate* of that state defined by:

$$\mathbf{E}(\mathbf{s}) \stackrel{\text{def}}{=} \sum_{\mathbf{s}' \in \mathbf{S}} \mathbf{R}(\mathbf{s}, \mathbf{s}')$$

The actual probability of a transition $\mathbf{s} \xrightarrow{\mathbf{R}(\mathbf{s},\mathbf{s}')} \mathbf{s}'$ eventually happening, irrespective of time, can be defined with the following embedded *discrete time Markov chains* (DTMC).

Definition 2. *The embedded DTMC for a CTMC* $\mathcal{C} = \langle \mathbf{S}, \mathbf{R}, \mathbf{s}_0, \mathbf{L} \rangle$ *is defined as a tuple* $\mathcal{D} = \langle \mathbf{S}, \mathbf{P}, \mathbf{s}_0, \mathbf{L} \rangle$ *where:*

$$\mathbf{P}(\mathbf{s}, \mathbf{s}') = \begin{cases} \mathbf{R}(\mathbf{s}, \mathbf{s}')/\mathbf{E}(\mathbf{s}) & if\ \mathbf{E}(\mathbf{s}) \neq 0 \\ 1 & if\ \mathbf{E}(\mathbf{s}) = 0\ and\ \mathbf{s} = \mathbf{s}' \\ 0 & otherwise \end{cases}$$

If there is an enabled transition from a state $\mathbf{s}$ to $\mathbf{s}'$, then $\mathbf{s}$ is a direct predecessor of $\mathbf{s}'$, and $\mathbf{s}'$ is a direct successor of $\mathbf{s}$. The set of all direct successors, generally refered as successors, for state $\mathbf{s}$ can be defined as:

$$\texttt{Succ}(\mathbf{s}) \stackrel{\text{def}}{=} \{\mathbf{s}' \in \mathbf{S} \mid \mathbf{R}(\mathbf{s}, \mathbf{s}') > 0\}$$

Similarly, predecessor state set $\texttt{Pre}(\mathbf{s})$ can be defined as:

$$\texttt{Pre}(\mathbf{s}) \stackrel{\text{def}}{=} \{\mathbf{s}' \in \mathbf{S} \mid \mathbf{R}(\mathbf{s}', \mathbf{s}) > 0\}$$

We define a reachability-value function $\hat{\kappa} : \mathbf{S} \to \mathbb{R}_{\geq 0}$ for a CTMC model $\mathcal{C}$ as follows:

$$\hat{\kappa}(\mathbf{s}) \stackrel{\text{def}}{=} \sum_{\mathbf{s}' \in \texttt{Pre}(\mathbf{s})} \left(\hat{\kappa}(\mathbf{s}') \cdot \frac{\mathbf{R}(\mathbf{s}', \mathbf{s})}{\mathbf{E}(\mathbf{s}')} \right)$$

Reachability-value of a state $\mathbf{s} \in \mathbf{S}$ estimates the probability of reaching that state, indicating whether the state search should terminate from that state onwards.

Definition 3. *Given a truncation parameter $\varkappa \in [0,1]$, referred as reachability threshold, we define a $\varkappa$-truncation of a CTMC model $\mathcal{C} = \langle \mathbf{S}, \mathbf{R}, \mathbf{s}_0, \mathbf{L} \rangle$ as a tuple $\mathcal{C}\rfloor_\varkappa = \langle \mathbf{S}\rfloor_\varkappa, \mathbf{R}\rfloor_\varkappa, \mathbf{s}_0, \mathbf{L}\rfloor_\varkappa \rangle$ where:*

- $\mathbf{S}\rfloor_\varkappa \subseteq \mathbf{S}$ *is a non-empty subset of states which contains all the states whose reachability-value is greater than $\varkappa$ and an abstract state $\mathbf{s}_{abs}$ which abstracts all the states $\mathbf{S}\backslash\mathbf{S}\rfloor_\varkappa$;*
- $\mathbf{R}\rfloor_\varkappa : \mathbf{S}\rfloor_\varkappa \times \mathbf{S}\rfloor_\varkappa \to \mathbb{R}_{\geq 0}$ *is the transition rate matrix for the truncated state space;*
- $\mathbf{s}_0 \in \mathbf{S}\rfloor_\varkappa$ *is the initial state;*
- $\mathbf{L}\rfloor_\varkappa$ *is a labeling function for the truncated state space.*

Finite truncation of state space leads to probability leakage (i.e., cumulative probabilities of reaching states not included in the explored state space) during the CTMC analysis. To account for probability loss, an abstract state $\mathbf{s}_{abs}$ is introduced to abstract all the states in $\mathbf{S}\backslash\mathbf{S}\rfloor_\varkappa$. The transition rate matrix is restricted to the truncated state space $\mathbf{S}\rfloor_\varkappa$ given by following expression:

$$\mathbf{R}\rfloor_\varkappa(\mathbf{s},\mathbf{s}') = \begin{cases} \mathbf{R}(\mathbf{s},\mathbf{s}') & if\ \mathbf{s},\ \mathbf{s}' \in \mathbf{S}\rfloor_\varkappa \backslash \mathbf{s}_{abs} \\ \sum_{\mathbf{s}'' \in \mathtt{Succ}(\mathbf{s})} \mathbf{R}(\mathbf{s},\mathbf{s}'') & if\ \mathbf{s} \in \mathbf{S}\rfloor_\varkappa \backslash \mathbf{s}_{abs},\ \mathbf{s}' = \mathbf{s}_{abs} \\ 1 & if\ \mathbf{s} = \mathbf{s}_{abs},\ \mathbf{s}' = \mathbf{s}_{abs} \\ 0 & otherwise \end{cases}$$

The $\varkappa$-truncation of a CTMC is illustrated in Figure 3.1. If $\varkappa = 0.08$, the truncated state set $\mathbf{S}\rfloor_\varkappa$ only contains white-colored states and $\mathbf{s}_{abs}$.

The CTMC model can also be viewed as a State Graph (SG) as defined below:

Definition 4. *A SG is a tuple $\mathcal{G} = \langle \mathbf{S}^{\mathcal{G}}, \delta, \mathbf{s}_0 \rangle$ where*

- $\mathbf{S}^{\mathcal{G}}$ *is a non-empty set of states*
- $\delta \subseteq \mathbf{S}^{\mathcal{G}} \times \mathbf{R} \times \mathbf{S}^{\mathcal{G}}$ *is the set of state transitions;*

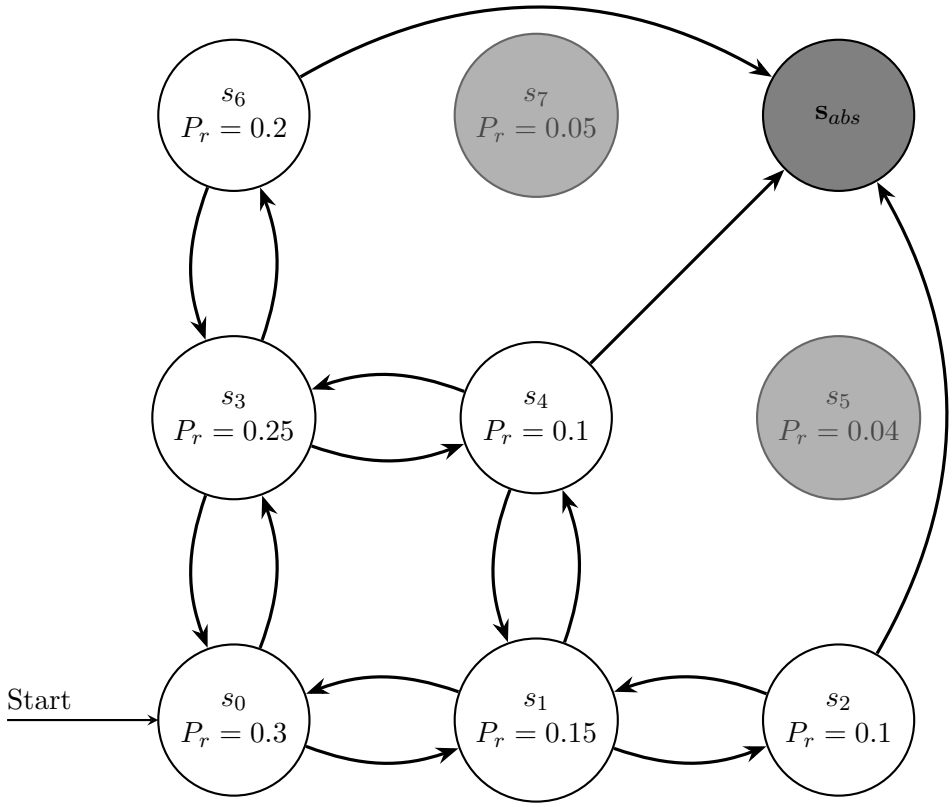


Fig. 3.1: $\varkappa$-truncation CTMC model.

– $\mathbf{s}_0 \in \mathbf{S}^{\mathcal{G}}$ *is the initial state.*

Note that $|\mathcal{G}|$ *represents the state count of* $\mathcal{G}$*.*

For this thesis, the term SG and truncated CTMC can be used interchangeably. SG is preferred to make it more consistent with the graph search terminologies. A $\varkappa$-truncated CTMC $\mathcal{C}\rfloor_{\varkappa}$ can be converted to SG as follows:

$$SG(\mathcal{C}\rfloor_{\varkappa}) \stackrel{\text{def}}{=} \mathcal{G} = \langle \mathbf{S}^{\mathcal{G}}, \delta, \mathbf{s}_0 \rangle$$

where,

$$\mathbf{S}^{\mathcal{G}} = \mathbf{S}\rfloor_{\varkappa}, \delta = \{(\mathbf{s}, \mathbf{R}\rfloor_{\varkappa}(\mathbf{s}, \mathbf{s}'), \mathbf{s}') \mid \mathbf{s}, \mathbf{s}' \in \mathbf{S}^{\mathcal{G}} \wedge \mathbf{R}\rfloor_{\varkappa}(\mathbf{s}, \mathbf{s}') > 0\}$$

3.2 Continuous Stochastic Logic

Specifications for CTMC model are written using the logic *continuous stochastic logic* (CSL) introduced by Aziz *et al.* in [33] and later refined by Baier *et al.* in [34].

A CSL property consists of state formulas and path formulas defined using the following grammar:

Definition 5. *Let an atomic proposition a, probability bound $p \in [0,\ 1]$ and $\sim \in \{<,\ >,\ \leq,\ \geq\}$, CSL state formulas are defined as:*

$$\Phi \ ::= \ true \mid a \mid \neg\Phi \mid \Phi \ \wedge \ \Phi \mid \mathsf{P}_{\sim p}(\varphi) \mid \mathsf{S}_{\sim p}(\Phi)$$

and for I an interval of $\mathbb{R}_{\geq 0}$, CSL path formulas are defined by:

$$\varphi \ ::= \ \mathcal{X}\ \Phi \mid \Phi\, \mathcal{U}^{I}\ \Phi$$

Although formally not defined, operators P and S can return the actual probability computed if they are the outermost operators and defined in in the form $\mathsf{P}_{=?}(\varphi)$ and $\mathsf{S}_{=?}(\Phi)$.

CSL state formulas are evaluated over states of a CTMC and can be *true* or $false$. Path formulas in CSL always occur inside the P operator and are *true* or $false$ along a path of the CTMC. The transient probability operator $\mathsf{P}_{\sim p}(\varphi)$ asserts that the probability measure of all paths starting from some initial state and satisfying φ meets the probability expression $\sim p$. Similarly, steady-state probability operator $\mathsf{S}_{\sim p}(\Phi)$ assures that the long-run probability of being in states Φ satisfies the bound $\sim p$.

The temporal property $\Phi\, \mathcal{U}^{I}\ \Psi$ asserts that Ψ will be satisfied at some time instant in the interval I and that at all preceding time instants Φ holds. For $t_1, t_2 \in \mathbb{R}_{\geq 0}$ and $t_1 \leq t_2$, interval I can be one of $[0, t_1]$, $[t_1, t_2]$ and $[t_1, +\infty)$. $\mathcal{X}\Phi$ asserts that there exists a state $\mathbf{s}'$, which can be reached by executing a single transition from some initial state, such that $\mathbf{s}'$ satisfies Φ. Note that the formula $\mathcal{X}\Phi$ does not involve the time interval I.

Below are some examples of CSL formulas:

- $\mathsf{P}_{=?}\ [\ \neg shutdown\, \mathcal{U}^{\leqslant T}\ failure]$ describes "what is the probability that the $failure$ in system causes shutdown within T time units?".

- $\mathsf{S}_{\geq 0.9}$ [*idle*] asserts that in long-run the system stays idle with at least 90% probability.

- $\mathsf{P}_{=?}$ [($\mathsf{P}_{\geqslant 0.9}$ [*true* $\mathcal{U}^{\leqslant 1}$ $speed_{avg}$]) $\mathcal{U}^{\leqslant 10}$ $speed_{max}$] denotes the probability that the system reaches the maximum speed within 10 time units while periodically maintaining the average speed with a least probability of 0.9.

3.3 Model Checking CTMCs Over CSL

CSL model checking process, proposed in [33, 34], first discretizes the CTMC into an embedded DTMC, from which many properties of the corresponding CTMC can be deduced, for example, checking state reachability properties regardless of how long it takes, and the expected time objectives. For checking state reachability within some time bound, the transition rate matrix $\mathbf{R}$ is converted to *infinitesimal generator matrix* $\mathbf{Q}$ whose diagonal entries are the negated exit rate. However, the positive and negative entries in $\mathbf{Q}$ cause numerical instability during transient analysis because of truncation of the infinite summation. To avoid such numerical instability, the matrix $\mathbf{Q}$ is normalized with respect to the fastest exit rate, known as the *uniformization rate* (q), to create *uniformized* DTMC. This process of normalization is called *uniformization* [35, 36]. The uniformized DTMC provides numerically stable representation of transition rate matrix and preserves the state resident time so that its transient behavior is equal (up to some accuracy) to the corresponding CTMC.

In this work, we consider time-bounded until CSL property i.e., $\mathsf{P}_{\sim p}(\Phi\ \mathcal{U}^I\ \Psi)$ or $\mathsf{P}_{=?}(\Phi\ \mathcal{U}^I\ \Psi)$. In following section we discuss the model checking procedure for such properties.

3.3.1 Model Checking CSL Time-Bounded Until

All non-nested CSL path formulas φ (except those containing the "next" operator) derive from $\Phi\,\mathcal{U}^I\ \Psi$. The path formula $\Phi\,\mathcal{U}^I\ \Psi$ holds if Ψ is satisfied at some time instant in the interval I and Φ holds at all preceding time instants. The analysis of time-bounded

until $\mathsf{P}_{\sim p}(\Phi\ \mathcal{U}^I\ \Psi)$ over a CTMC can be reduced to transient analysis on a transformed CTMC [34].

Consider a CTMC $\mathcal{C} = \langle \mathbf{S}, \mathbf{R}, \mathbf{s}_0, \mathbf{L} \rangle$ and a time-bounded CSL property $\mathsf{P}_{\sim p}(\Phi\ \mathcal{U}^I\ \Psi)$. For simplicity, we consider a time interval $I = [0, t]$. A modified CTMC $\mathcal{C}[\phi]$ is obtained by making all states satisfying ϕ absorbing. *Absorbing state* is created by replacing all the outgoing transitions from a state with a transition into the same state. The path formula $\varphi = \Phi\ \mathcal{U}^I\ \Psi$, is satisfied if a Ψ state is reached within time t via some state that satisfies Φ. As proposed in [34], $\mathcal{C}$ can be transformed to $\mathcal{C}[\Psi]$ without affecting the satisfiability of φ because the satisfiability can be determined without further expanding this path beyond the Ψ-state. Similarly, $(\neg\Phi \wedge \neg\Psi)$-states can also be made absorbing since φ will never be satisfied once $(\neg\Phi \wedge \neg\Psi)$-state is reached regardless the following states along this path. Therefore, in modified CTMC $\mathcal{C}[\Psi][\neg\Phi \wedge \neg\Psi]$, it is not possible to exit, once entered, any state satisfying either Ψ or $(\neg\Phi \wedge \neg\Psi)$. The probability of $\Phi\ \mathcal{U}^I\ \Psi$ satisfying in $\mathcal{C}$ now becomes the probability of being in a state that satisfies Ψ at time t in modified CTMC $\mathcal{C}[\Psi][\neg\Phi \wedge \neg\Psi]$, which is equivalent to $\mathcal{C}[\neg\Phi \vee \Psi]$ [34]. Using the fact that model checking $\mathcal{C}$ is equivalent in checking $\mathcal{C}[\neg\Phi \vee \Psi]$, one can apply pre-processing steps on $\mathcal{C}$ to terminate paths that satisfy $(\neg\Phi \vee \Psi)$, which narrows down the state search as described in Section 4.1.3.

3.4 The PRISM Model Checker

PRISM is a probabilistic model checking tool that supports model checking of CSL over CTMC and of *probabilistic computation tree logic* (PCTL) over DTMC and MDP. The tool takes a model description written in the PRISM language and a property specified using CSL or PCTL. PRISM then constructs the appropriate model, either CTMC, DTMC or MDP, and explores the set of reachable states of the model. PRISM has four engines that implements the numerical computation for model checking: MTBDD, sparse, hybrid and explicit. First three engines constructs the model symbolically and employ different numerical methods for model checking. The explicit engine, on the other hand, entirely uses explicit-state data structure for model construction and verification.

3.4.1 PRISM Language

In this section we briefly describe the PRISM language. Full details about PRISM and PRISM language can be found in [17]. A model is described using two main components: modules and variables. A model is a composition of a number of modules. Each module contains a number of finite integer variables. A valuation of variable local to a module represents the state of that module. A global state of the model is a combination of valuations of all the local variables.

The behavior of each module is described using a finite number of commands. A command is in the following format:

$$[]\ g \rightarrow \lambda_1 : update_1 + \lambda_2 : update_2 + \cdots + \lambda_n : update_n$$

The guard g is a Boolean expression over all the variables. A transition is enabled in any state if the expression g is evaluated to *true* in that state and updates the variables described by $update_i$. λ_i is either rate expression for CTMCs or probability values for DTMCs and MDPs for that transition.

```
ctmc
const double lambda = 1.0;
const double p = 0.2;
module random_walk
    m : [0..100] init 0;
    [] (m = 0) → p * lambda : (m' = m + 1);
    [] (m > 0&m < 100) → p * lambda : (m' = m + 1) + (1 − p) * lambda : (m' = m − 1);
    [] (m = 100) → (1 − p) * lambda : (m' = m − 1);
endmodule
```

Fig. 3.2: Simple PRISM model

Figure 3.2 shows a simple CTMC model with 101 states. First line declares the model type. Following the model type are two constant declarations of type double: *lambda* whose value is 1.0 and p with value 0.2. This model is composed of a single module named random_walk. It has only one variable m with range 0 to 100, and is initialized to 0. There

are 3 commands in the form [] $g \rightarrow u$ where g is the guard and u is combination of one or more updates. In PRISM $m\prime$ represents the updated value of variable m. Therefore, command "[] $(m = 0) \rightarrow p * lambda : (m\prime = m + 1)$;" reads as "If variable m has value equal to 0, m will be incremented by 1 with transition rate $p * lambda$". In case of second command, there are two possible updates. In such scenario, the rate of each transition being executed is given by the valuation at the current state of their corresponding rate expression. A CSL property for this CTMC model is "what is the probability that within 10 seconds the system reaches the state with m greater than 10". The PRISM formula for this property is $P =?$ [$true\ U <= 10\ m > 10$].

CHAPTER 4

STATE SPACE APPROXIMATION AND ANALYSIS

In this chapter, we discuss the implementation of STAMINA and the procedure to construct the approximate, finite state space for a CTMC model. Section 4.1 presents the architecture of the STAMINA. Implementation of algorithms that approximates, refines and analyzes the given CTMC model is presented in Section 4.1.1 and 4.1.2. Section 4.1.3 describes the idea behind property-based state space exploration. Finally, the last section describes the termination proof of the implemented algorithms.

4.1 Architecture of STAMINA

The state approximation methods proposed in this thesis are implemented as a tool, STAMINA. Probabilistic model checking is usually performed in two phases: model construction and numerical model checking. Model construction is the process of generating the state space of the system from high level model description. After constructing the appropriate model from the model description, model checking applies numerical methods to compute the actual probabilities and verifies the specification. STAMINA implements several algorithms to construct a finite state space of a CTMC model (currently, state approximation only applies to CTMC models) from the given finite or infinite state model description. For CTMC analysis, STAMINA relies on PRISM's explicit-state CSL model checker.

The architecture of STAMINA is presented in Figure 4.1.

- **State Space Approximation**: It constructs the approximate state space, $\mathcal{C}\rfloor_{\varkappa}^{r+1}$, by refining the state space constructed in previous iteration r using Algorithms 2 and 3.
- **Model Checking Framework**: It performs the CTMC analysis on the truncated CTMC $\mathcal{C}\rfloor_{\varkappa}$ using the PRISM's CSL model checker. PRISM returns the minimum and maximum probability, $[l, u]$, that the property holds. If the verification result is

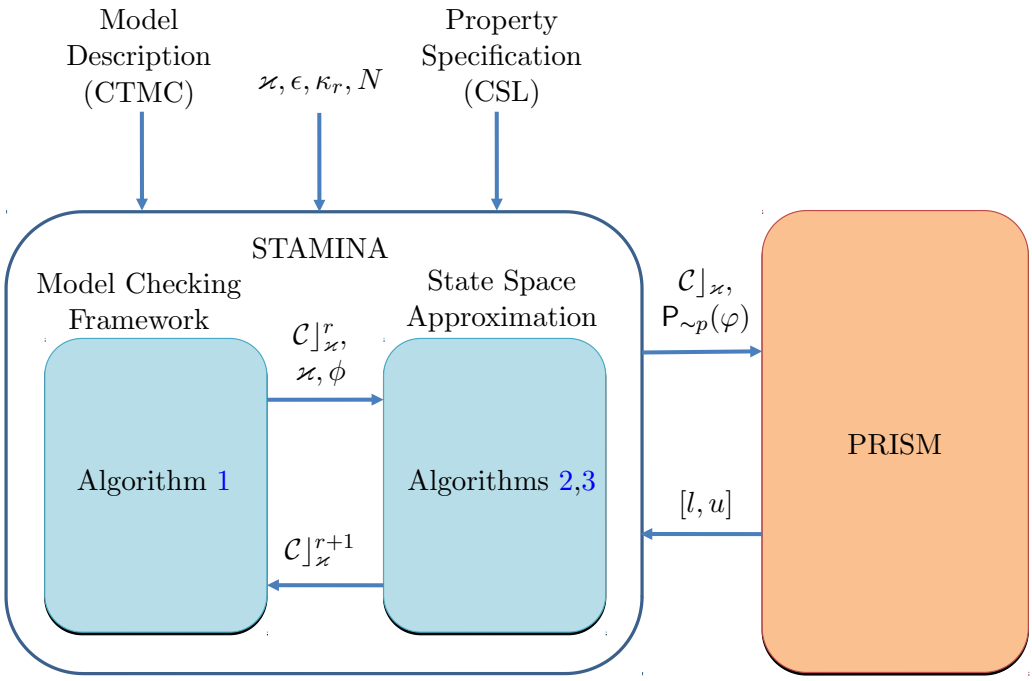


Fig. 4.1: Architecture of STAMINA

definitive i.e., either $p \notin [l, u]$ or $|u - l| < \epsilon$, the process terminates. Otherwise it triggers another iteration of state space approximation. The iterative process repeats until the termination condition is satisfied.

The details about each modules are presented in following sections.

4.1.1 Model Checking Framework

Algorithm 1 describes the iterative model checking framework for a given CTMC model $\mathcal{C} = \langle \mathbf{S}, \mathbf{R}, \mathbf{s}_0, \mathbf{L} \rangle$. The reachability-value function $\hat{\kappa}$ estimates the probability of reaching a state $\mathbf{s} \in \mathbf{S}$. For state exploration purpose we use reachability-value and state probability interchangeably. It should be noted that this reachability-value/state probability value for each state is only used during model construction and is omitted for the CTMC analysis. We define another function $\hat{\gamma} : \mathbf{S} \to \mathbb{R}_{\geq 0}$, which computes the estimate reachability-value

used during following *breadth-first search* (BFS) iteration in Algorithm 2. To start the state exploration process, the truncated CTMC model $\mathcal{C}\rfloor_{\varkappa}$ is initialized such that it contains only the initial state $\mathbf{s}_0$ in the state set without any transition relation. The single initial state $\mathbf{s}_0$ is assigned a value 1 to the reachability-value which will be used by subsequent state exploration method.

A truncated CTMC model $\mathcal{C}\rfloor_{\varkappa}$ with finite states is generated for a parameter $\varkappa$ (kappa) from the original infinite-state CTMC model $\mathcal{C}$ using Algorithm 2. The model $\mathcal{C}\rfloor_{\varkappa}$ is then model checked against the CSL property $\mathsf{P}_{\sim p}(\varphi)$, which returns the lower- and upper-bounds, l and u, respectively, of the probability that the property holds. The probability accumulated in the abstract state $\mathbf{s}_{abs}$ is $(u-l)$. For a defined value of p, if $p \in (l,u)$, it is not known whether the CSL property $\mathsf{P}_{\sim p}(\varphi)$ holds. On the other hand, if exact probability is of interest, having a large $(u-l) > \epsilon$ may not generate meaningful verification result.

Algorithm 1: Probabilistic model checking

Input: An CTMC model $\mathcal{C} = \langle \mathbf{S}, \mathbf{R}, \mathbf{s}_0, \mathbf{L} \rangle$, CSL property $Prop = \mathsf{P}_{\sim p}(\varphi)$.

1 $\hat{\kappa}(\mathbf{s}_0) := 1$;
2 $\mathcal{C}\rfloor_{\varkappa} = \langle \mathbf{S}\rfloor_{\varkappa}, \mathbf{R}\rfloor_{\varkappa}, \mathbf{s}_0, \mathbf{L}\rfloor_{\varkappa} \rangle$, where $\mathbf{S}\rfloor_{\varkappa} = \{\mathbf{s}_0\}$, $\mathbf{R}\rfloor_{\varkappa} = \emptyset$;
3 $\mathcal{C}\rfloor_{\varkappa} \leftarrow$ Construct property agnostic finite truncation of CTMC model $\mathcal{C}$ using Algorithm 2 ($\mathcal{C}\rfloor_{\varkappa}$, *null*).
4 $l, u \leftarrow$ Model check $\mathcal{C}\rfloor_{\varkappa}$ against CSL property $\mathsf{P}_{\sim p}(\varphi)$.
5 **if** $p \notin [l,u] \ \vee \ |u-l| < \epsilon$ **then**
6 | `Exit`
7 $r := 0$;
8 **repeat**
9 | $\varkappa := \varkappa / \kappa_r$;
10 | $\phi \leftarrow (\Phi \, \mathcal{U} \ \Psi)$ if φ is non-nested until formula; else *null*.
11 | $\mathcal{C}\rfloor_{\varkappa} \leftarrow$ Refine $\mathcal{C}\rfloor_{\varkappa}$ using Algorithm 2 ($\mathcal{C}\rfloor_{\varkappa}$, ϕ).
12 | $l, u \leftarrow$ Model check $\mathcal{C}\rfloor_{\varkappa}$ against CSL property $\mathsf{P}_{\sim p}(\varphi)$.
13 | $r := r + 1$;
14 **until** $p \notin [l,u] \ \vee \ |u-l| < \epsilon \ \vee \ r > N$;

If the verification result obtained form property-agnostic step is inconclusive, the finite truncated model $\mathcal{C}\rfloor_{\varkappa}$ is iteratively refined (lines 9 to 14 of Algorithm 1). Algorithm 2 can also use the CSL property to intelligently expand the state space when the path formula

φ belongs to non-nested until CSL-class (line 10 of Algorithm 1). The property-based exploration is described in detail in Section 4.1.3. Note that $\varkappa$ also drops by the reduction factor κ_r (line 9) to enable states that were previously ignored due to a low probability estimate to be included in the current state expansion. The refined CTMC model $\mathcal{C}\rfloor_{\varkappa}$ is then model checked to obtain a new probability bound $[l, u]$. This process repeats until one of the following conditions holds: (1) the target probability p falls outside the probability bound $[l, u]$, (2) the bound is sufficiently small (less than ϵ), or (3) a maximal number of iterations N has been reached (line 14).

4.1.2 State Space Approximation

Algorithm 2 constructs the approximated state space for a reachability threshold $\varkappa$ using finite number of BFS iterations. Given a truncated CTMC model $\mathcal{C}\rfloor_{\varkappa} = \langle \mathbf{S}\rfloor_{\varkappa}, \mathbf{R}\rfloor_{\varkappa}, \mathbf{s}_0, \mathbf{L}\rfloor_{\varkappa} \rangle$, the state exploration process starts by adding the initial state $\mathbf{s}_0$ to the exploration `queue`. For all the states scheduled for exploration, successor states for each state $\mathbf{s}$ is generated by executing all the enabled transitions form state $\mathbf{s}$. A successor state $\mathbf{s}'$ for state $\mathbf{s}$ is generated by firing a transition $\mathbf{s} \xrightarrow{\mathbf{R}(\mathbf{s},\mathbf{s}')} \mathbf{s}'$ (line 9). If $\mathbf{s}'$ is a new state (line 10), it is included in the state set $\mathbf{S}^k$ (line 16) only if state $\mathbf{s}$ is not absorbing (line 12) and its current reachability-value $\hat{\kappa}(\mathbf{s})$ is at least $\varkappa$ (line 13). A state is considered absorbing only during the property-based exploration i.e., $\phi \neq null$, if it satisfies $\mathbf{s} \models (\neg \Phi \vee \Psi)$, otherwise all states are considered non-absorbing (line 12). Property based exploration is described in Section 4.1.3.

The new state-transition relation $(\mathbf{s}, \mathbf{R}(\mathbf{s}, \mathbf{s}'), \mathbf{s}')$ is added to current state-transition relation δ^k (line 14) and the estimate reachability-value $\hat{\gamma}(\mathbf{s}')$ is computed for state $\mathbf{s}'$ in line 15. The estimate reachability-value $\hat{\gamma}$ is not used to update the reachability-value for other states in current iteration k, and only becomes available at the end of the current iteration, at which point it is assigned to the current value of $\hat{\kappa}$ (line 27). The reachability-value $\hat{\gamma}(\mathbf{s}')$ has contributions from all of its predecessor states. For each predecessor state $\mathbf{s}''$ of $\mathbf{s}'$, its contribution to $\hat{\gamma}(\mathbf{s}')$ is the product of its current reachability-value $\hat{\kappa}(\mathbf{s}'')$ and the probability of transitioning from $\mathbf{s}''$ to $\mathbf{s}'$, defined as the ratio of transition rate $\mathbf{R}(\mathbf{s}'', \mathbf{s}')$,

to the exit rate $\mathbf{E}(\mathbf{s}'')$ evaluated at state $\mathbf{s}''$ (line 15). Intuitively, $\hat{\gamma}(\mathbf{s}')$ path probability form all of its predecessor states that have been explored till iteration k. Finally, state $\mathbf{s}'$ is scheduled for exploration if it has not been visited (line 17). If a non-absorbing state has reachability-value $\hat{\kappa}(\mathbf{s})$ less than $\varkappa$, it becomes (partially) terminal state if there exists a transition $\mathbf{s} \xrightarrow{\mathbf{R}(\mathbf{s},\mathbf{s}'')>0} \mathbf{s}''$ such that $\mathbf{s}'' \notin \mathbf{S}^k$.

For the case where $\mathbf{s}'$ exists in the set $\mathbf{S}^k$ (line 20 to 25), the transition relation δ^k and the estimate reachability-value $\hat{\gamma}$ are updated and state $\mathbf{s}'$ is scheduled for exploration if it is not in the `visited` set (line 23). The reachability-value $\hat{\gamma}(\mathbf{s})$ is updated since there may exist a path $\mathbf{s}'' \rightarrow \mathbf{s}'$ such that $\mathbf{s}' \in \mathbf{S}^k$ (line 22). The reachability-value update is performed every time a new incoming path is added to a state. It is crucial to have frequent updates since a new incoming path can add its contribution to the state, potentially bringing the reachability-value above $\varkappa$, which in turn changes a terminal state to be non-terminal. This update, therefore, guarantees to explore a state with many incoming paths whose accumulative reachability-values are significant, although each individual one might be low compared to $\varkappa$. After exploring all the scheduled states, current BFS iteration terminates by adding all the terminal states, from the set $\mathbf{S}^k$ to the exploration queue (line 29).

The subsequent state graphs are then constructed using the same algorithm. Both state graphs $\mathcal{G}^{k-1}$ and $\mathcal{G}^k$ are constructed based on the same CTMC model $\mathcal{C}$. The difference is that $\mathcal{G}^k$ updates $\hat{\kappa}$ values for some explored states in $\mathcal{G}^{k-1}$, which may expand $\mathcal{G}^{k-1}$ to include new states in $\mathcal{G}^k$. This process of expansion and refinement is repeated until the size of the approximate state graph stabilizes (line 31), at which point an abstract state is added to this state graph by Algorithm 3. Algorithm 2 terminates by returning the new truncated CTMC model $\mathcal{C}\rfloor_{\varkappa}$.

When the truncated CTMC model $\mathcal{C}\rfloor_{\varkappa}$ is analyzed, it introduces some error in the probability value of the property under verification, because of leakage the probability (i.e., cumulative probabilities of reaching states not included in the explored state space) during the CTMC analysis. To account for probability loss, an abstract state $\mathbf{s}_{abs}$ is created as the sole successor state for all terminal states on each truncated path, and is added by

Algorithm 2: State space approximation/refinement using breadth-first search

Input: Truncated CTMC model $\mathcal{C}\rfloor_{\varkappa} = \langle \mathbf{S}\rfloor_{\varkappa}, \mathbf{R}\rfloor_{\varkappa}, \mathbf{s}_0, \mathbf{L}\rfloor_{\varkappa}\rangle$, CSL path ϕ.
Output: Truncated CTMC model $\mathcal{C}'\rfloor_{\varkappa} = \langle \mathbf{S}'\rfloor_{\varkappa}, \mathbf{R}'\rfloor_{\varkappa}, \mathbf{s}_0, \mathbf{L}'\rfloor_{\varkappa}\rangle$.

$k := 0$;
$\mathcal{G}^k = SG(\mathcal{C}\rfloor_{\varkappa})$;
$Enqueue(\texttt{queue}, \mathbf{s}_0)$;
$\texttt{visited} := \{\mathbf{s}_0\}$;
repeat
  $k := k + 1$;
  while *queue* $\neq \emptyset$ **do**
    $\mathbf{s} := Dequeue(\texttt{queue})$;
    forall $\mathbf{s}' \in \mathcal{S}ucc(\mathbf{s})$ **do**
      if $\mathbf{s}' \notin \mathbf{S}^k$ **then**
        $\hat{\kappa}(\mathbf{s}') := 0$;
        if $\phi = null \vee \mathbf{s} \models \neg(\neg\Phi \vee \Psi)$ **then**
          if $\hat{\kappa}(\mathbf{s}) \geq \varkappa$ **then**
            $\delta^k := \delta^k \cup \{(\mathbf{s}, \mathbf{R}(\mathbf{s}, \mathbf{s}'), \mathbf{s}')\}$;
            $\hat{\gamma}(\mathbf{s}') := \sum_{\mathbf{s}'' \in \texttt{Pre}(\mathbf{s}')} \left(\hat{\kappa}(\mathbf{s}'') \cdot \frac{\mathbf{R}(\mathbf{s}'', \mathbf{s}')}{\mathbf{E}(\mathbf{s}'')}\right)$;
            $\mathbf{S}^k := \mathbf{S}^k \cup \{\mathbf{s}'\}$;
            if $\mathbf{s}' \notin$ *visited* **then**
              $Enqueue(\texttt{queue}, \mathbf{s}')$;
              $\texttt{visited} := \texttt{visited} \cup \{\mathbf{s}'\}$;
        else
          $\delta^k := \delta^k \cup \{(\mathbf{s}, \mathbf{R}(\mathbf{s}, \mathbf{s}'), \mathbf{s}')\}$;
          $\hat{\gamma}(\mathbf{s}') := \sum_{\mathbf{s}'' \in \texttt{Pre}(\mathbf{s}')} \left(\hat{\kappa}(\mathbf{s}'') \cdot \frac{\mathbf{R}(\mathbf{s}'', \mathbf{s}')}{\mathbf{E}(\mathbf{s}'')}\right)$;
          if $\mathbf{s}' \notin$ *visited* **then**
            $Enqueue(\texttt{queue}, \mathbf{s}')$;
            $\texttt{visited} := \texttt{visited} \cup \{\mathbf{s}'\}$;
  forall $\mathbf{s} \in \mathbf{S}^k$ **do**
    $\hat{\kappa}(\mathbf{s}) := \hat{\gamma}(\mathbf{s})$;
    if $|\mathcal{S}ucc(\mathbf{s} \in \mathbf{S}\rfloor_{\varkappa})| < |\mathcal{S}ucc(\mathbf{s} \in \mathbf{S})|$ **then**
      $Enqueue(\texttt{queue}, \mathbf{s})$;
      $\texttt{visited} := \texttt{visited} \cup \{\mathbf{s}\}$;
until $|\mathcal{G}^k| = |\mathcal{G}^{k-1}|$;
Update $\mathcal{G}^k$ by adding an an extra abstract state $\mathbf{s}_{abs}$ using Algorithm 3.

Algorithm 3 to the state space generated by Algorithm 2. For all states in the global state set, if transition $\mathbf{s} \xrightarrow{\mathbf{R}(\mathbf{s},\mathbf{s}')} \mathbf{s}'$ is not in the transition relation δ^k, state transition relation $(\mathbf{s}, \mathbf{R}(\mathbf{s}, \mathbf{s}_{abs}), \mathbf{s}_{abs})$ is added to δ^k, where $\mathbf{R}(\mathbf{s}, \mathbf{s}_{abs})$ is computed using Definition 3. It is obvious that all unexplored transitions from such a terminal state $\mathbf{s}$ lead to the abstract

Algorithm 3: Abstract state update from approximated global state graph.

Input: An approximated global state graph $\mathcal{G}$.
Output: Updated state graph $\mathcal{G}$ with an abstract state $\mathbf{s}_{abs}$.

```
1 S^G := S^G ∪ {s_abs};
2 forall s ∈ S^G do
3 |   forall s' ∈ Succ(s) do
4 |   |   if (s, R(s, s'), s') ∉ δ then
5 |   |   |   δ := δ ∪ {(s, R(s, s_abs), s_abs)};
```

state.

4.1.3 Property Based State Space Exploration

Model checking of non-nested time-bounded until formula, $\Phi\ \mathcal{U}^I\ \Psi$, on CTMC $\mathcal{C}$ is reduced to transient analysis on a transformed CTMC $\mathcal{C}[\neg\Phi \vee \Psi]$ by making $(\neg\Phi \vee \Psi)$-states absorbing (refer to Section 3.3). Since the states reachable from $(\neg\Phi \vee \Psi)$-states in $\mathcal{C}$ becomes unreachable in $\mathcal{C}[\neg\Phi \vee \Psi]$, it is sufficient to explore only those states that satisfy $\neg(\neg\Phi \vee \Psi) \equiv (\Phi \wedge \neg\Psi)$. Our property-guided state space expansion method therefore identifies those states satisfying $(\Phi \wedge \neg\Psi)$ and schedules them for exploration (line 12 in Algorithm 2).

4.2 Proof of the Termination Condition

The presented algorithms in Section 4.1.2 are guaranteed to terminate under certain conditions. This section provides a description of the termination conditions for each algorithm, and presents a proof for termination.

To facilitate the following proof, we first define finite paths of a state graph and depth for breadth-first search. A finite path ρ of a state graph is a sequence $\mathbf{s}_0 \xrightarrow{\mathbf{R}(\mathbf{s}_0,\mathbf{s}_1)} \mathbf{s}_1 \xrightarrow{\mathbf{R}(\mathbf{s}_1,\mathbf{s}_2)} \ldots \mathbf{s}_{n-1} \xrightarrow{\mathbf{R}(\mathbf{s}_{n-1},\mathbf{s}_n)} \mathbf{s}_n$ such that for every $0 \leqslant i < n$, $(\mathbf{s}_i, \mathbf{R}(\mathbf{s}_i, \mathbf{s}_{i+1}), \mathbf{s}_{i+1}) \in \delta$ holds for some $\mathbf{R}(\mathbf{s}_i, \mathbf{s}_{i+1}) > 0$. State $\mathbf{s}_n$ is reachable in $\mathcal{G}$ if $\mathbf{s}_n$ is reachable from the initial state through a finite path included in $\mathcal{G}$. We denote the set of all states with depth $\imath$ as ${}^{\imath}\mathbf{S}$. At depth 0, ${}^{0}\mathbf{S} = \{\mathbf{s}_0\}$. We define one BFS-step from depth $\imath \geqslant 0$ as the process of exploring all immediate successors of the states in ${}^{\imath}\mathbf{S}$ to generate new state set ${}^{\imath+1}\mathbf{S}$.

Therefore, the depth for a state is determined when it is explored for the first time. Note that ${}^{0}\mathbf{S} \cap {}^{1}\mathbf{S} \cdots {}^{\imath-1}\mathbf{S} \cap {}^{\imath}\mathbf{S} = \emptyset$.

Termination condition for Algorithm 2 requires that, as both the depth $\imath$ and iteration k increase, the sum of reachability-values for all states of ${}^{\imath}\mathbf{S}^{k}$ decreases, with possibly finitely many iterations where this sum remains constant. This is formulated by Theorem 1 below.

Theorem 1 (Termination of Algorithm 2)**.** *Algorithm 2 terminates after a finite number of iterations with a given $\varkappa$, where $0 < \varkappa << 1$, if the state graph $\mathcal{G}^{\jmath+1}$ satisfies the following condition: for each depth $\jmath > 0$, there must exist depth $0 \leqslant \imath \leqslant \jmath$ such that $\mathbf{s}_d \xrightarrow{\mathbf{R}(\mathbf{s}_d,\mathbf{s}_{d+1})} \mathbf{s}_{d+1} \xrightarrow{\mathbf{R}(\mathbf{s}_{d+1},\mathbf{s}_{d+2})} \cdots \mathbf{s}_{d+(m-1)} \xrightarrow{\mathbf{R}(\mathbf{s}_{d+(m-1)},\mathbf{s}_{d+m})} \mathbf{s}_{d+m}$ is a finite path in $\mathcal{G}^{\jmath+1}$, and $\mathbf{s}_d \in {}^{\imath}\mathbf{S}^{\jmath+1}$, $\mathbf{s}_{d+(m-1)} \in {}^{\jmath}\mathbf{S}^{\jmath+1}$, $\mathbf{s}_{d+m} \in {}^{0}\mathbf{S}^{\jmath+1} \cup {}^{1}\mathbf{S}^{\jmath+1} \cup \cdots {}^{\jmath-1}\mathbf{S}^{\jmath+1} \cup {}^{\jmath}\mathbf{S}^{\jmath+1}$, and $m \in \mathbb{Z}_{\geqslant 0}$.*

Proof. Initially, $\mathbf{S}^{0} = \{\mathbf{s}_0\}$ and $\hat{\kappa}(\mathbf{s}_0) = 1$. At iteration $k = 1$, during the construction of $\mathcal{G}^{1}$, each state at depth 1, ${}^{1}\mathbf{s} \in {}^{1}\mathbf{S}^{1}$, is discovered for the first time when $\mathbf{s}_0$ is explored. Therefore, the current reachability-value $\hat{\kappa}({}^{1}\mathbf{s})$ is assigned a 0 (line 11 of Algorithm 2), but its estimate reachability-value $\hat{\gamma}({}^{1}\mathbf{s})$ gets updated by $\hat{\kappa}(\mathbf{s}_0)$, so that $0 < \hat{\gamma}({}^{1}\mathbf{s}) \leqslant 1$. Each new state ${}^{2}\mathbf{s} \in {}^{2}\mathbf{S}^{1}$ generated from ${}^{1}\mathbf{S}^{1}$, is ignored, since $\hat{\kappa}({}^{1}\mathbf{s}) = 0$, which is less than $\varkappa$, and ${}^{2}\mathbf{s} \notin \mathbf{S}^{1}$ (line 13 to 17 of Algorithm 2).

Then at iteration $k = 2$, the sum of reachability-values is ${}^{1}\zeta^{2} = \sum_{\mathbf{s}\in {}^{1}\mathbf{S}^{2}} \hat{\kappa}(\mathbf{s})$, where each $\hat{\kappa}(\mathbf{s})$ is a fraction of ${}^{0}\zeta^{1}$, and ${}^{0}\zeta^{1} = \hat{\kappa}(\mathbf{s}_0) = 1$. Therefore, ${}^{1}\zeta^{2}$ is solely contributed from ${}^{0}\zeta^{1}$. If a self-loop transition $\{\mathbf{s}_0, \mathbf{R}(\mathbf{s}_0, \mathbf{s}_0), \mathbf{s}_0\}$ exists, then ${}^{0}\zeta^{1} > {}^{1}\zeta^{2}$; otherwise ${}^{0}\zeta^{1} = {}^{1}\zeta^{2}$. Therefore, ${}^{0}\zeta^{1} \geqslant {}^{1}\zeta^{2}$. Similar to the previous iteration, the updated $\hat{\gamma}({}^{2}\mathbf{s})$ will be used in the next iteration.

In general, state set ${}^{\imath}\mathbf{S}$ at depth $\imath$ is first obtained in iteration $\imath$ by collecting all the new states, i.e., states whose depth has not been determined, which are expanded from states in ${}^{\imath-1}\mathbf{S}$. The sum of all reachability-values for states in ${}^{\imath}\mathbf{S}$ is calculated at iteration $\imath + 1$ by either line 15 or 22 of Algorithm 2. To differentiate the reachability-value function $\hat{\kappa}$ in different iterations, we denote $\hat{\kappa}^{\imath}(\mathbf{s})$ as the reachability-value for state $\mathbf{s}$ at iteration $\imath$. The sum of all reachability-values at iteration $\imath + 1$ is computed as follows:

$$
\begin{aligned}
{}^{\imath}\zeta^{\imath+1} &= \sum_{\mathbf{s}' \in {}^{\imath}\mathbf{S}^{\imath+1}} \hat{\kappa}^{\imath+1}(\mathbf{s}') \\
&= \sum_{\mathbf{s}' \in {}^{\imath}\mathbf{S}^{\imath+1}} \sum_{\mathbf{s}'' \in \mathtt{Pre}(\mathbf{s}')} \left(\hat{\kappa}(\mathbf{s}'') \cdot \frac{\mathbf{R}(\mathbf{s}'', \mathbf{s}')}{\mathbf{E}(\mathbf{s}'')} \right)
\end{aligned}
$$

If $\bigcup_{\mathbf{s}' \in {}^{\imath}\mathbf{S}^{\imath+1}} \mathtt{Pre}(\mathbf{s}')$ is equal to all transition firings of every state in ${}^{\imath-1}\mathbf{S}^{\imath}$, then reachability-values for all the states at depth $\imath - 1$ are passed to depth $\imath$, and hence ${}^{\imath}\zeta^{\imath+1} = {}^{\imath-1}\zeta^{\imath}$. On the other hand, if there exists one or more transition firings from ${}^{\imath-1}\mathbf{S}^{\imath}$ to depth other than $\imath$, then ${}^{\imath}\zeta^{\imath+1} < {}^{\imath-1}\zeta^{\imath}$. Moreover, if certain states from ${}^{\imath-1}\mathbf{S}^{\imath}$ are made absorbing by the property-based exploration, it further decreases the accumulated reachability-value for all the states passed from depth $\imath - 1$ to depth $\imath$. Therefore, ${}^{\imath-1}\zeta^{\imath} \geqslant {}^{\imath}\zeta^{\imath+1}$.

We can, therefore, establish the following conclusion:

$$
1 = {}^{0}\zeta^{1} \geqslant {}^{1}\zeta^{2} \geqslant \cdots {}^{\imath-1}\zeta^{\imath} \geqslant {}^{\imath}\zeta^{\imath+1} \cdots {}^{\jmath-1}\zeta^{\jmath} \geqslant {}^{\jmath}\zeta^{\jmath+1}
$$

From the termination condition stated in Theorem 1, the slowest termination scenario, i.e., the maximal number of iterations required to terminate Algorithm 2, is the following:

$$
1 = {}^{0}\zeta^{1} = {}^{1}\zeta^{2} = \cdots = {}^{\imath}\zeta^{\imath+1} = \cdots = {}^{\jmath-1}\zeta^{\jmath} > {}^{\jmath}\zeta^{\jmath+1}.
$$

The inequality ${}^{\jmath-1}\zeta^{\jmath} > {}^{\jmath}\zeta^{\jmath+1}$ holds only if at least one state in ${}^{\jmath-1}\mathbf{S}^{\jmath}$ executes a transition leading to a state in ${}^{0}\mathbf{S}^{\jmath+1} \cup {}^{1}\mathbf{S}^{\jmath+1} \cup \cdots {}^{\jmath-1}\mathbf{S}^{\jmath+1}$, but not in ${}^{\jmath}\mathbf{S}^{\jmath+1}$. State $\mathbf{s}_{d+m}$ in Theorem 1 is such a state. Additionally, the termination condition requires that at least ${}^{\imath}\zeta^{\imath+1} = \cdots = {}^{\jmath-1}\zeta^{\jmath} > {}^{\jmath}\zeta^{\jmath+1}$ holds for *every* depth $\jmath$. This requirement guarantees that the sum of reachability-values keeps decreasing, with possibly many (or zero) iterations where this sum remains unchanged. Therefore, after a finite number ξ of iterations, ${}^{\xi-1}\zeta^{\xi} < \varkappa$. Since ${}^{\xi-1}\zeta^{\xi}$ is the sum of all individual reachability-values, in the next iteration $(\xi + 1)$, reachability-value $\hat{\kappa}({}^{\xi}\mathbf{s})$ is less than $\varkappa$ for all states in ${}^{\xi}\mathbf{S}^{\xi+1}$, and they become terminal

states. Hence, $|\mathcal{G}^{\xi}| = |\mathcal{G}^{\xi+1}|$. □

CHAPTER 5

RESULTS

This chapter presents case studies that are used to evaluate the effectiveness of the proposed method. In Section 5.1, we present the analysis of several benchmark models from different application domains. Section 5.2 provides the comparison of our tool STAMINA with the state-of-art infinite-state probabilistic model checking tool INFAMY [16].

All the experiments presented are performed on a 3.2 GHz AMD Debian Linux PC with six cores and 64 GB of RAM. Starting value of reachability threshold $\varkappa$ varies for different case studies. The reduction factor κ_r is kept constant to 1000, and the maximal number of iterations N is set to 10. The analysis precision ϵ set is 10^{-3}. Currently, the property-based state exploration only supports non-nested bounded-until transient properties and is set to default on. For other types of properties, it reverts back to the property-agnostic state expansion with reduced $\varkappa$.

5.1 Case Studies

This section presents verification results on the following case studies to illustrate the accuracy and efficiency of STAMINA: a genetic toggle switch from [4], a grid world robot navigation, a cyclic server polling system, and a tandem queuing network from the PRISM benchmark suite [37] and a Jackson queuing network from INFAMY case studies [38].

For all tables in this section, column $\varkappa$ reports the reachability threshold used to terminate state generation in STAMINA. The state space size is listed in column $|\mathcal{G}|$. Column $Time(C/A)$ reports the state space construction (C) and analysis (A) time in seconds. Since this approach requires two separate CTMC analyses to compute the lower and upper bound on the probability for the given property, the analysis time (A) is the sum of two time analysis time. Columns P_{min} and P_{max} list the lower and upper probability bounds for the property under verification.

5.1.1 Genetic Toggle Switch

Behaviors of synthetic biological systems are governed by a set of chemical reactions acting on a set of chemical species (molecules). These processes typically involve low molecule counts making the circuit extremely noisy [39]. It is, therefore, necessary to evaluate a genetic circuit's behaviors using stochastic analyses.

The analogous electronic representation of the genetic toggle switch circuit presented in [4] is shown in Figure 5.1. This toggle switch circuit has two inputs and two outputs, aTc and IPTG, and LacI and TetR respectively. Unlike a digital circuit, the logic levels of a genetic circuit are represented by the number of chemical species (molecules). The ON state of this toggle switch is characterized by high TetR (TetR $>$ 40) and low LacI (LacI $<$ 20) molecular count. Similarly, the OFF state is represented by TetR dropping below 20 molecules and LacI rising above 40 molecules. Two important properties for a toggle switch circuit are the response time and the failure rate.

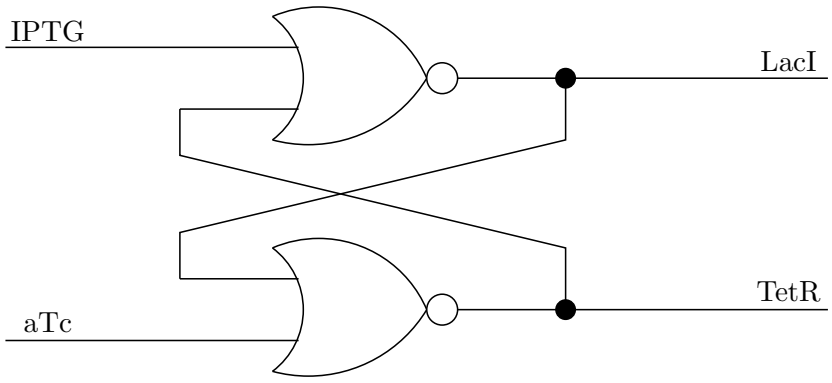


Fig. 5.1: A digital circuit representation of the genetic toggle switch.

In order to measure genetic toggle switch's response time (i.e., the time it takes to switch from the OFF state to the ON state), it is initialized to OFF state with LacI at 60 and TetR at 0 molecules. Number of IPTG molecules is set to 100 representing the circuit has just received the set input to switch to the ON state. Input value of 100 molecules is chosen to ensure that the circuit should switch to the ON state, but any moderately large value of input could be used. The CSL property, $\mathsf{P}_{=?}\ [\ true\ \mathcal{U}^{\leqslant T}\ (\text{TetR} > 40 \wedge \text{LacI} < 20)]$,

describes the probability of the circuit switching to the ON state within time T.

Figure 5.2 shows the minimum and maximum probability of switching from OFF state to ON state within first T seconds after IPTG has been applied for two different values of $\varkappa$, 10^{-3} and 10^{-6}. T varies from 0 to 2100 (an approximation of the cell cycle in *E. coli* [40]). The probability of switching is significantly inaccurate w.r.t. the precision $\epsilon = 10^{-3}$ for the initial value of $\varkappa$ as shown in Figure 5.2a. The $\varkappa$ is then reduced to 10^{-6} and state generation switches to the property-guided refinement mode, where a new state space is generated by refining the previous state graph guided by the property and the model is analyzed again. In this case the difference between maximum and minimum probability of switching is decreased significantly shown in Figure 5.2b.

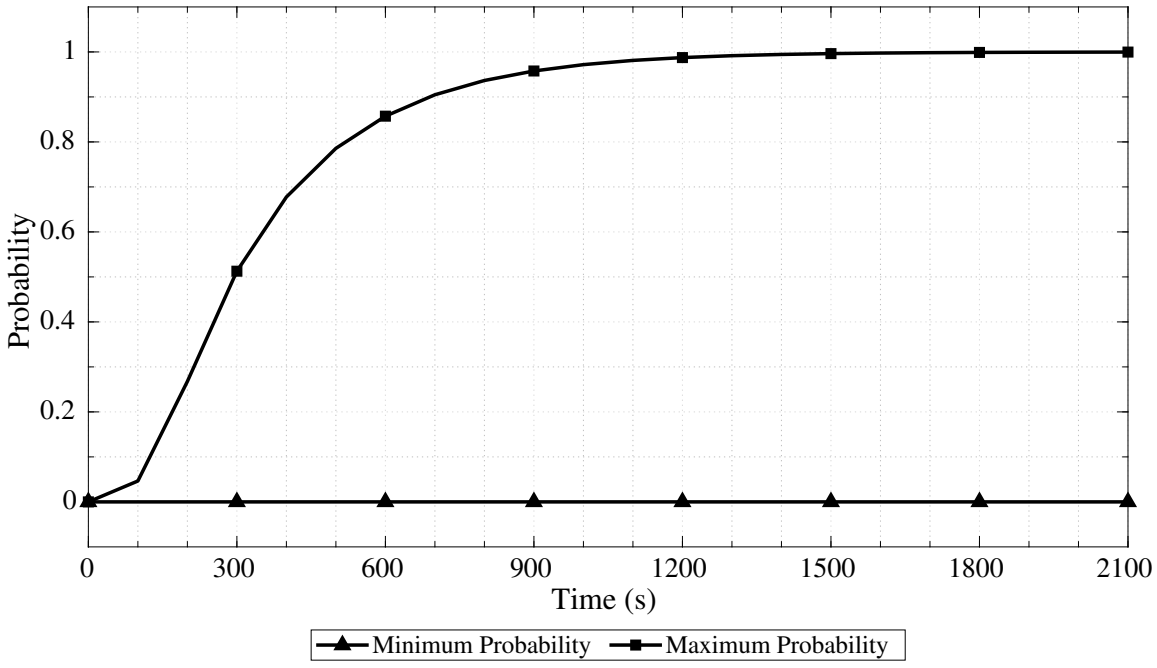
The second set of experiments involves computing the probability that the circuit changes state from OFF to ON erroneously within T seconds when aTc and IPTG are set to 0. This behavior occurs if production of LacI erroneously and significantly inhibits TetR's production to let TetR degrade away and consequently switch state. The toggle switch is initialized to OFF state with LacI at 60 and TetR at 0 molecules as in the previous experiment. The same CSL property is verified.

Figure 5.3 shows the probability of the circuit changing state erroneously within T seconds for two different values of $\varkappa$, 10^{-3} and 10^{-6}. Similar to response rate experiment, larger $\varkappa$-value produced imprecise probability bounds as shown in Figure 5.3a. After reducing the $\varkappa$-value to 10^{-6}, Figure 5.3b shows that the probability of circuit changing state erroneously within one cell cycle is less than 10%. Generally, smaller value of $\varkappa$ generates larger state space, but producing more precise verification results.

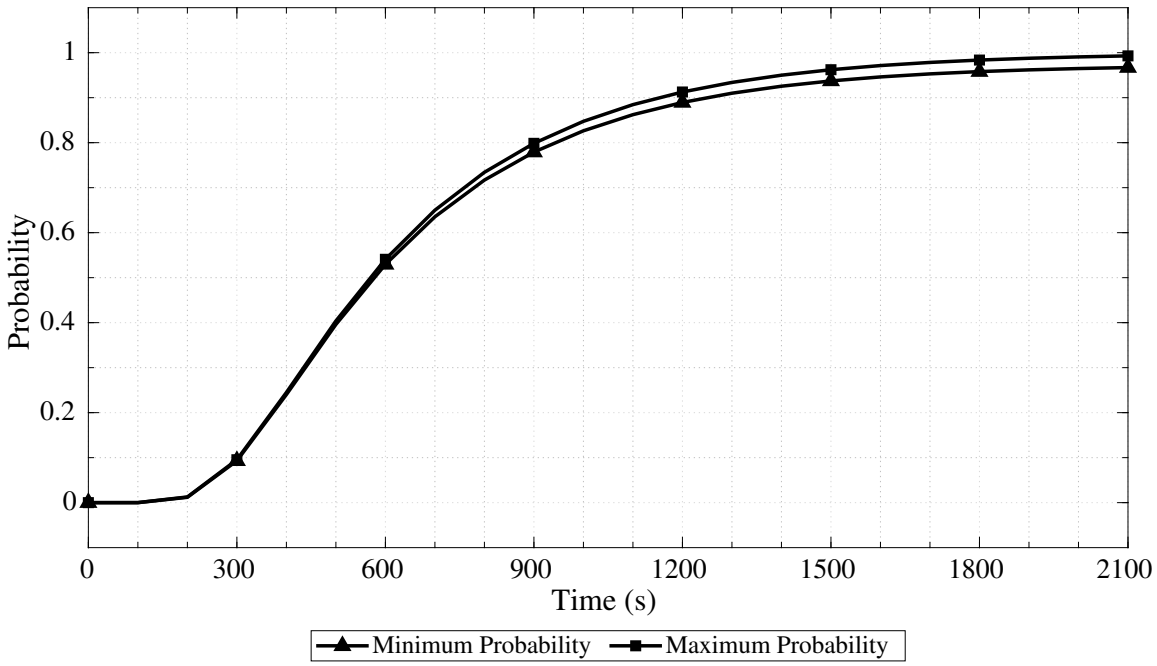
5.1.2 Grid World Robot Navigation

This case study considers a robot moving in a n-by-n grid world and a janitor moving in a larger grid Kn-by-Kn, where K is a constant scaling factor that can be used to significantly scale up the system's state space as shown in Figure 5.4.

The robot starts from the bottom left corner to reach the top right corner. The janitor moves around the larger grid randomly. Robot can only move to a grid if that

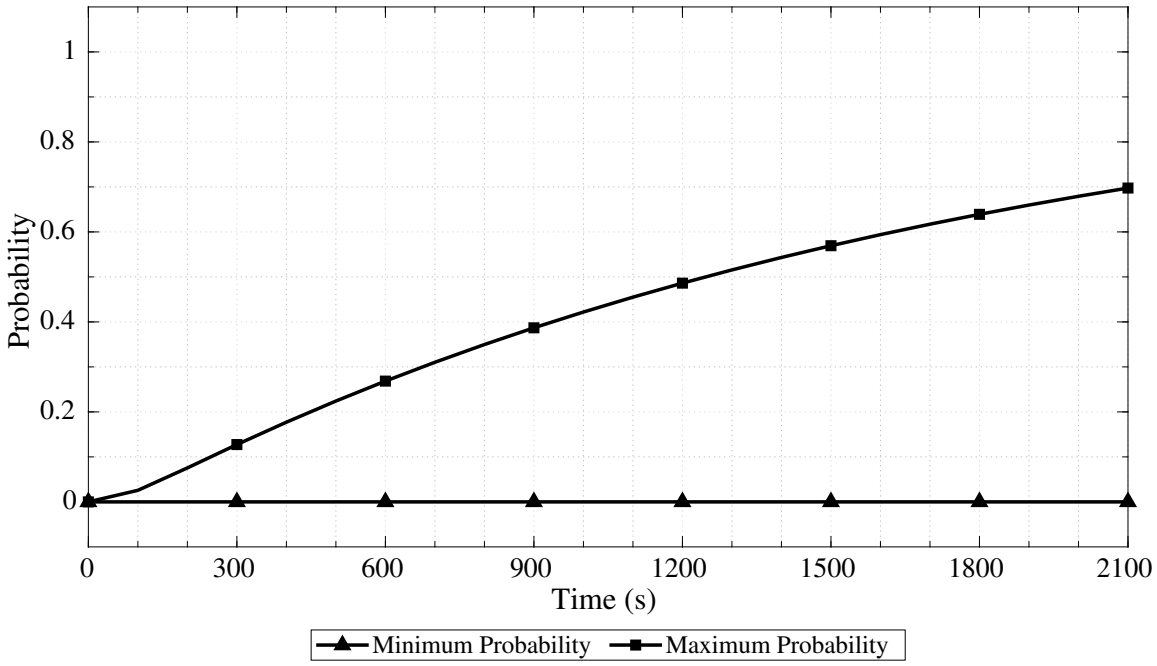


(a)

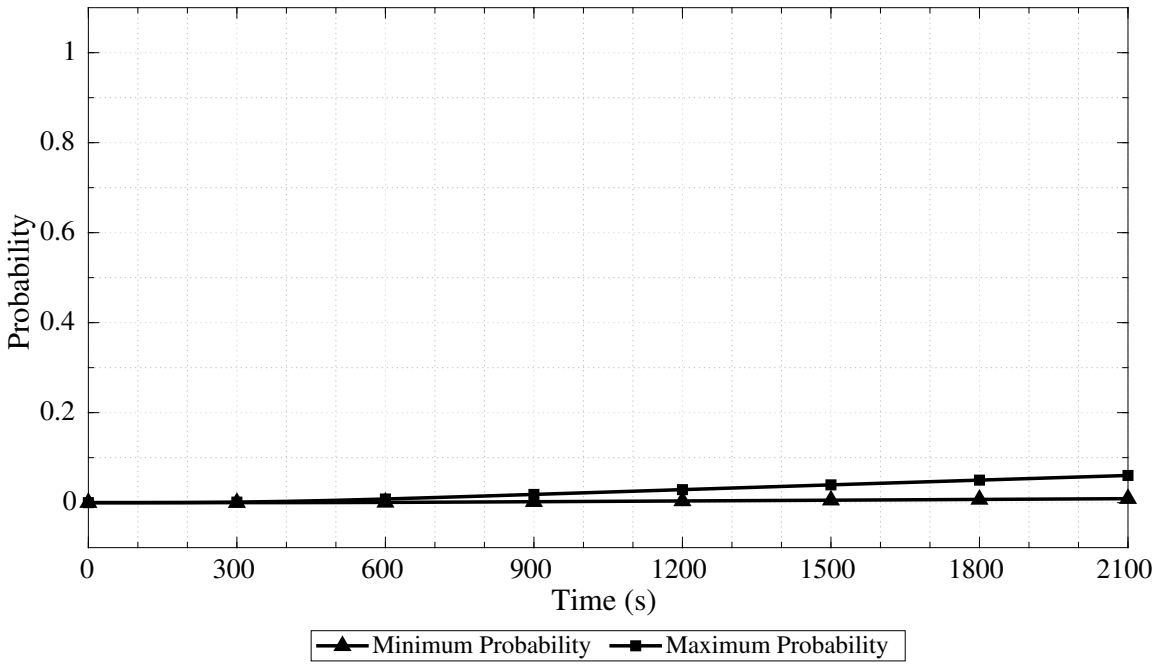


(b)

Fig. 5.2: Effect of $\varkappa$ on verification precision. (a) Time course plot showing the probability of the genetic toggle switch changing its state from OFF to ON when $\varkappa = 10^{-3}$. (b) Time course plot showing the probability of switching when $\varkappa = 10^{-6}$.



(a)



(b)

Fig. 5.3: Effect of $\varkappa$ on verification precision. (a) Time course plot showing the probability of the genetic toggle switch changing its state erroneously when $\varkappa = 10^{-3}$. (b) Time course plot showing the probability of erroneous switching when $\varkappa = 10^{-6}$.

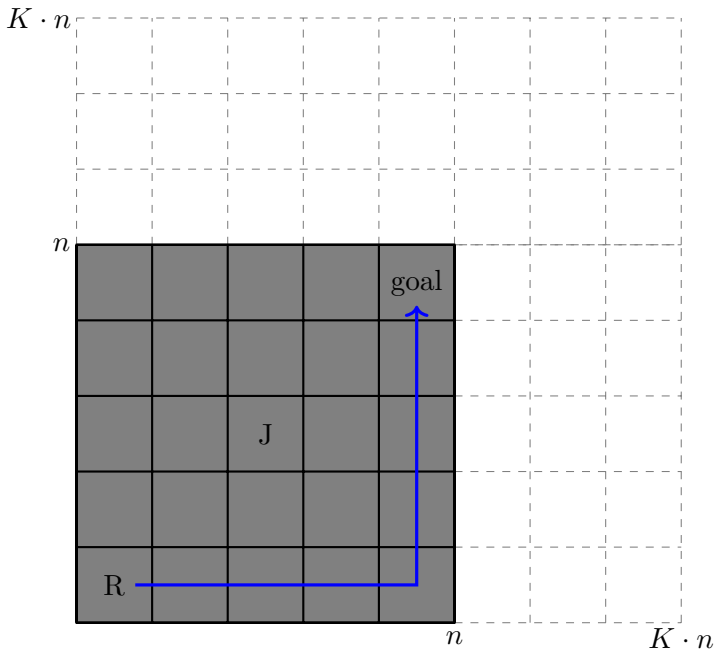


Fig. 5.4: Grid world robot navigation. Robot (R) moves from bottom left corner to top right corner along the direction shown in shaded region (grid size n-by-n). Janitor (J) can move to any position in entire grid of size Kn-by-Kn.

place is not occupied by the janitor. The robot also randomly communicates with a base station. The property of interest is $\mathsf{P}_{=?}$ [($\mathsf{P}_{\geqslant 0.5}$ [*true* $\mathcal{U}^{\leqslant 7}$ *communicate*]) $\mathcal{U}^{\leqslant 100}$ *goal*], the probability that the robot reaches the top right corner within 100 time units while periodically communicating with the base station.

Table 5.1 provides verification results for $K = 1024, 64$ and $n = 64, 32$. For smaller grid size i.e, 32-by-32, the robot can reach the goal with a high probability of 97.56%. Whereas for larger values of $n = 64$ and $K = 64$, the robot is not able to reach the goal with considerable probability. Since the property is only dependent on the size of the grid that the robot is traveling, the verification result did not change even the value of K is changed for constant n.

5.1.3 Jackson Queuing Network

A Jackson Queuing Network (JQN) consists of N interconnected nodes (queues) with

Table 5.1: Model construction and verification results for grid world robot navigation system.

n/K	$\varkappa$	$\lvert\mathcal{G}\rvert$	$T(C/A)$	P_{min}	P_{max}
32/ 64	10^{-6}	20,059	0.92/3.25	0.000000	0.999999
	10^{-9}	107,107	3.22/20.45	0.000000	0.999813
	10^{-12}	311,117	8.56/57.67	0.582066	0.979065
	10^{-15}	527,842	12.58/86.13	0.973985	0.975636
	10^{-18}	695,839	15.50/111.80	0.975613	0.975617
32/ 1024	10^{-6}	20,059	0.92/3.25	0.000000	0.999999
	10^{-9}	107,107	3.31/19.85	0.000000	0.999813
	10^{-12}	311,117	8.65/58.40	0.582066	0.979065
	10^{-15}	527,842	12.59/74.71	0.973985	0.975636
	10^{-18}	695,839	15.49/102.14	0.975613	0.975617
64/ 64	10^{-6}	20,204	0.89/2.91	0.000000	0.999999
	10^{-9}	107,914	3.22/19.77	0.000000	0.999738
	10^{-12}	310,828	9.04/53.56	0.000000	0.937148
	10^{-15}	699,171	19.55/100.71	0.000000	0.310284
	10^{-18}	1,347,528	37.63/176.84	0.000000	0.008826
	10^{-21}	2,272,949	64.52/316.18	1.46E-4	1.68E-4
64/ 1024	10^{-6}	20,204	0.94/2.89	0.000000	0.999999
	10^{-9}	107,914	3.27/19.47	0.000000	0.999738
	10^{-12}	310,828	8.84/56.59	0.000000	0.937148
	10^{-15}	699,171	19.88/91.43	0.000000	0.310284
	10^{-18}	1,347,528	36.65/165.99	0.000000	0.008826
	10^{-21}	2,272,949	61.98/284.88	1.46E-4	1.68E-4

infinite queue capacity. Initially, all queues are considered empty. Each station is connected to a single server which distributes the arrived jobs to different stations. Customers arrive as a Poisson stream with intensity λ for N queues. A customer, upon completing service at a node i, either leaves the network or enters another node j. We consider the case with $N = 4, 5$ with constant $\lambda = 5$. The model is taken from [3,16]. We compute the probability that, within 10 time units, the first queue has more than 3 jobs and the second queue has more than 5 jobs, given by $\mathsf{P}_{=?}\,[\,true\,\mathcal{U}^{\leqslant 10}\,(jobs\_1 \geqslant 4\ \wedge\ jobs\_2 \geqslant 6)]$.

Table 5.2 summarizes the model checking statistics for this JQN model. Model exploration starts with $\varkappa = 10^{-9}$. For smaller value of $N = 4$, the final probability value is within precision after one property guided refinement. However, for $N = 5$, the model exploration continued till $\varkappa$ reached a very small value 10^{-15} to explore sufficient states to

Table 5.2: Model construction and verification results for Jackson queuing network.

N/λ	$\varkappa$	$\lvert\mathcal{G}\rvert$	$T(C/A)$	P_{min}	P_{max}
4/5	10^{-9}	36,820	3.36/6.59	0.792071	0.940620
	10^{-12}	200,665	18.28/44.78	0.865409	0.865567
5/5	10^{-9}	21,087	4.50/8.47	0.305300	0.993958
	10^{-12}	360,685	89.24/108.96	0.801530	0.850927
	10^{-15}	2,539,456	896.23/878.25	0.819651	0.819705

give sufficiently precise verification result.

5.1.4 Cyclic Server Polling System

This case study is based on a cyclic server attending N stations. We consider the probability that station one is polled within 10 time units, $\mathsf{P}_{=?}$ [$true\ \mathcal{U}^{\leqslant 10}\ polled$]. This property is checked for $N = 12, 16, 20$ and Table 5.3 summarizes the results. The probability of station one being polled within 10 seconds is 1.0 for all configurations. $\varkappa = 10^{-6}$ is sufficient to generate enough states to obtain accurate probability.

Table 5.3: Model construction and verification results for cyclic server polling system.

N	$\varkappa$	$\lvert\mathcal{G}\rvert$	$T(C/A)$	P_{min}	P_{max}
12	10^{-6}	18,959	2.87/21.18	1.0	1.0
16	10^{-6}	57,302	18.41/69.98	1.0	1.0
20	10^{-6}	112,805	30.00/76.60	1.0	1.0

5.1.5 Tandem Queuing Network

A tandem queuing network is the simplest interconnected queuing network of two queues with one server each [17]. Customers join the first queue and enter the second queue immediately after completing the service. We consider both queues with capacity c. Probability that the first queue becomes full in T time units, depicted by the CSL property $\mathsf{P}_{=?}$ [$true\ \mathcal{U}^{\leqslant T}\ queue1\_full$], is plotted in Figure 5.5 for queue capacity $c = 4095$. Time T is varied from 0 to 0.5 since values larger than 0.5 generates the probability 1.0. The initial

value of $\varkappa = 10^{-6}$ is sufficient to keep the error probability below the analysis precision for all time points which is shown by the overlapping minimum and maximum probability values in the figure.

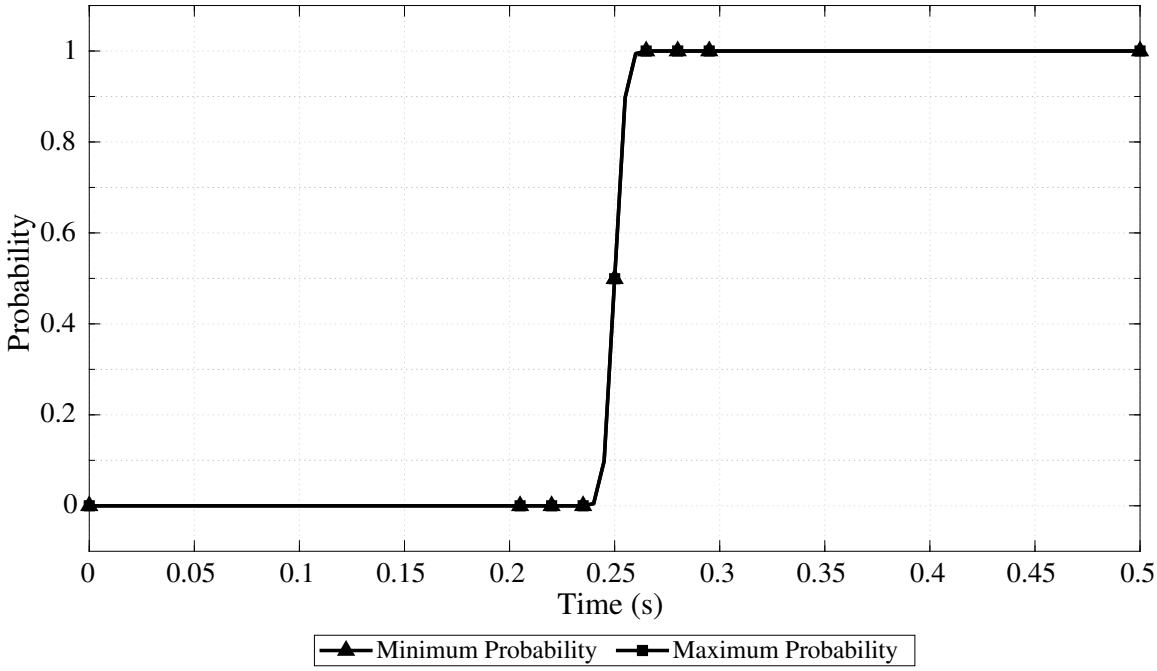


Fig. 5.5: Time course plot showing the probability of first queue becoming full for queue capacity $c = 4095$ and $\varkappa = 10^{-6}$. Time course plot for queue capacity $c = 2047$ shows similar behavior.

5.2 Comparison with INFAMY

This section compares state space and runtime between STAMINA and INFAMY. The state space size is listed in column $\mathcal{G}$ for both STAMINA and INFAMY. Column $Time(C/A)$ reports the construction and analysis time. For STAMINA, the total construction and analysis time is the cumulation of runtime for all $\varkappa$ values. We report the runtime with the fastest configuration for INFAMY. The improvement in state space size, ratio of state count generated by INFAMY to that of STAMINA (higher is better), and runtime, percentage improvement in runtime (higher is better), is listed in column $|\mathcal{G}|(X)$ and $T(\%)$.

The same CSL properties described in Section 5.1 are verified for grid world robot

Table 5.4: State space and runtime comparison between STAMINA and INFAMY.

Model	Params	STAMINA		INFAMY		Improvement	
		$\|\mathcal{G}\|$	$T(C/A)$	$\|\mathcal{G}\|$	$T(C/A)$	$\|\mathcal{G}\|(X)$	$T(\%)$
Robot (n/K)	32/64	$696K$	41/279	$1,591K$	492/18	2.3	37.3
	32/1024	$696K$	41/258	$1,591K$	501/18	2.3	42.4
	64/64	$2,273K$	135/669	$5,088K$	$1,625/53$	2.2	52.1
	64/1024	$2,273K$	132/621	$5,088K$	$1,625/53$	2.2	55.2
Jackson (N/λ)	4/5	$201K$	22/51	$635K$	109/5	3.2	36.1
	5/5	$2,539K$	990/996	$7,029K$	1668/108	2.8	-11.8
Polling (N)	12	$19K$	3/21	$74K$	1/2	3.9	-732.2
	16	$57K$	18/70	$1,573K$	5/54	27.6	-48.2
	20	$113K$	30/77	$31,457K$	151/1347	278.4	92.9
Tandem (c/T)	2047/0.25	$33K$	1/41	$2,392K$	3/38	72.5	-1.4
	4095/0.25	$66K$	1/141	$9,216K$	11/265	139.6	48.7

navigation system, Jackson queuing network, cyclic server polling system, and tandem queuing network using STAMINA and INFAMY. The probability values reported by both tool are within the same analysis precision $\epsilon = 10^{-3}$. The size of state space generated, and model construction and analysis time are compared in Table 5.4.

STAMINA, by selectively exploring the states whose reachability-value is higher than the given threshold, is able to reduce the state space by a factor of ~ 278 compared to the method deployed by INFAMY, which, on the other hand, explores all the states up-to certain depth. Smaller state space generated by STAMINA contributes to significantly smaller model construction time for all the examples. For polling server and tandem queuing network, the advantage of STAMINA in terms of runtime starts to manifest as the size of model (and hence state size) grows. INFAMY performed better in terms of analysis time when analyzing Jackson queuing network model with 5 stations despite exploring one-third of the states in smaller time. This can can be explained by the fact that our method relies on two separate CTMC analyses to compute the lower and upper bounds on the probability for each CSL property.

CHAPTER 6

CONCLUSIONS

6.1 Conclusion

We aimed to develop an infinite-state probabilistic model checker. When it comes to system with infinite state space, truncation is the only viable way to construct its model that are amenable to current model checking techniques. Manual model truncation during modeling phase however leads to analysis uncertainty. In a naive way, the model is explored to a finite depth and the rest of the state space is abstracted to a single state to account for the error introduced by truncation. In order to maintain the error introduced below a small precision, higher exploration depth is required, which in turn, causes the exponential growth of the truncated state space, limiting its scalability.

In this thesis we investigated a different method to truncate the infinite state space to a finite one. During exploration of a model, we maintained a parameter, reachability-value, for each of the state explored so far. Reachability-value of a state estimates how likely that state will contribute to the analysis of the model. Unlike the naive way, all the states with same exploration depth are not explored. We explore only those states whose reachability-value is higher than a specified threshold and terminate the path at current state if the reachability-value is less than the specified threshold. This avoids the exponential growth of the truncated state space.

The efficiency measures of interest are minimal time and space requirements of model checking. As demonstrated in the preceding chapters, we have successfully applied our method on case studies taken from various application domains and compared its performance and accuracy with the naive method implemented in INFAMY. We demonstrated that the size of state space is reduced for all the benchmarks by as much as 278.4 times compared to INFAMY. This reduction also contributes to significantly smaller model con-

struction time for all the benchmarks presented. The over all runtime is also improved for almost all case studies with large state space. The only case study for which STAMINA has longer runtime than INFAMY is the Jackson queuing network model with 5 stations. The dual CTMC analyses to find minimum and maximum probability explains the increased runtime for this system.

6.2 Future Work

Among numerous possible directions, we discuss few interesting ones to investigate. Currently, the methods presented in this thesis is only applicable for CTMC models. Semantically DTMCs are very similar to CTMCs. Instead of transition rate in CTMCs, DTMCs have the actual transition probability. We plan to extend our method to truncate the DTMCs as well. Adding support to another model-class increases the utility of STAMINA.

Another improvement would be to merge two CTMC analysis into one. Our method performs two separate CTMC analysis, one excluding and other including the abstract state, to compute minimum and maximum probability respectively. Instead, it can be tightly integrated to PRISM; and utilize intermediate information to compute both values only running one CTMC analysis. As observed in Section 5.2, this can significantly reduce the overall runtime making STAMINA more efficient.

Finally, the reduction of reachability threshold is done in constant rate dictated by reduction factor. Aggressive reduction of reachability threshold may explore unnecessary states and increase the model construction. On the other hand, slow decrease requires multiple iterations to compute the probability with in the given precision. We plan to investigate algorithms to determine the reduction factor on-the-fly based on the probability bound.

REFERENCES

[1] J. Muppala, G. Ciardo, and K. Trivedi, "Stochastic reward nets for reliability prediction," *Communications in Reliability, Maintainability and Serviceability*, vol. 1, no. 2, pp. 9–20, July 1994.

[2] H. Hermanns, J. Meyer-Kayser, and M. Siegle, "Multi terminal binary decision diagrams to represent and analyse continuous time Markov chains," in *Proc. 3rd International Workshop on Numerical Solution of Markov Chains (NSMC'99)*, B. Plateau, W. Stewart, and M. Silva, Eds. Prensas Universitarias de Zaragoza, 1999, pp. 188–207.

[3] J. R. Jackson, "Networks of waiting lines," *Oper. Res.*, vol. 5, no. 4, pp. 518–521, Aug. 1957. [Online]. Available: http://dx.doi.org/10.1287/opre.5.4.518

[4] C. Madsen, Z. Zhang, N. Roehner, C. Winstead, and C. Myers, "Stochastic model checking of genetic circuits," *J. Emerg. Technol. Comput. Syst.*, vol. 11, no. 3, pp. 23:1–23:21, Dec. 2014. [Online]. Available: http://doi.acm.org/10.1145/2644817

[5] C. Baier, E. M. Clarke, V. Hartonas-Garmhausen, M. Z. Kwiatkowska, and M. Ryan, "Symbolic model checking for probabilistic processes," in *Proceedings of the 24th International Colloquium on Automata, Languages and Programming*, ser. ICALP '97. London, UK, UK: Springer-Verlag, 1997, pp. 430–440. [Online]. Available: http://dl.acm.org/citation.cfm?id=646251.685846

[6] D. Parker, "Implementation of symbolic model checking for probabilistic systems," Ph.D. dissertation, University of Birmingham, 2002.

[7] J.-P. Katoen, T. Kemna, I. Zapreev, and D. N. Jansen, "Bisimulation minimisation mostly speeds up probabilistic model checking," in *Proceedings of the 13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, ser. TACAS'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 87–101. [Online]. Available: http://dl.acm.org/citation.cfm?id=1763507.1763519

[8] K. Fisler and M. Y. Vardi, "Bisimulation and model checking," in *Proceedings of the 10th IFIP WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, ser. CHARME '99. London, UK, UK: Springer-Verlag, 1999, pp. 338–341. [Online]. Available: http://dl.acm.org/citation.cfm?id=646704.702028

[9] K. Fisler and M. Y. Vardi, "Bisimulation minimization and symbolic model checking," *Form. Methods Syst. Des.*, vol. 21, no. 1, pp. 39–78, Jul. 2002. [Online]. Available: https://doi.org/10.1023/A:1016091902809

[10] J. P. Katoen, D. Klink, M. Leucker, and V. Wolf, "Three-valued abstraction for continuous-time markov chains," in *Proceedings of the 19th International Conference on Computer Aided Verification*, ser. CAV'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 311–324. [Online]. Available: http://dl.acm.org/citation.cfm?id=1770351.1770401

[11] H. Fecher, M. Leucker, and V. Wolf, "Don't know in probabilistic systems," in *Proceedings of the 13th International Conference on Model Checking Software*, ser. SPIN'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 71–88. [Online]. Available: http://dx.doi.org/10.1007/11691617_5

[12] H. Hermanns, B. Wachter, and L. Zhang, "Probabilistic cegar," in *Proceedings of the 20th International Conference on Computer Aided Verification*, ser. CAV '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 162–175. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-70545-1_16

[13] M. Kwiatkowska, G. Norman, and D. Parker, "Symmetry reduction for probabilistic model checking," in *Proceedings of the 18th International Conference on Computer Aided Verification*, ser. CAV'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 234–248. [Online]. Available: http://dx.doi.org/10.1007/11817963_23

[14] A. F. Donaldson and A. Miller, "Symmetry reduction for probabilistic model checking using generic representatives," in *Proceedings of the 4th International Conference on Automated Technology for Verification and Analysis*, ser. ATVA'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 9–23. [Online]. Available: http://dx.doi.org/10.1007/11901914_4

[15] M. Groesser and C. Baier, "Partial order reduction for markov decision processes: A survey," in *Proceedings of the 4th International Conference on Formal Methods for Components and Objects*, ser. FMCO'05. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 408–427. [Online]. Available: http://dx.doi.org/10.1007/11804192_19

[16] E. M. Hahn, H. Hermanns, B. Wachter, and L. Zhang, "Infamy: An infinite-state markov model checker," in *Proceedings of the 21st International Conference on Computer Aided Verification*, ser. CAV '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 641–647. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-02658-4_49

[17] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," in *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, ser. LNCS, G. Gopalakrishnan and S. Qadeer, Eds., vol. 6806. Springer, 2011, pp. 585–591.

[18] M. Lapin, L. Mikeev, and V. Wolf, "Shave: Stochastic hybrid analysis of markov population models," in *Proceedings of the 14th International Conference on Hybrid Systems: Computation and Control*, ser. HSCC '11. New York, NY, USA: ACM, 2011, pp. 311–312. [Online]. Available: http://doi.acm.org/10.1145/1967701.1967746

[19] J. Aspnes and M. Herlihy, "Fast randomized consensus using shared memory," *J. Algorithms*, vol. 11, no. 3, pp. 441–461, Sep. 1990. [Online]. Available: http://dx.doi.org/10.1016/0196-6774(90)90021-6

[20] K. G. Larsen and A. Skou, "Bisimulation through probabilistic testing," *Inf. Comput.*, vol. 94, no. 1, pp. 1–28, Sep. 1991. [Online]. Available: http://dx.doi.org/10.1016/0890-5401(91)90030-6

[21] C. Baier, B. Engelen, and M. Majster-Cederbaum, "Deciding bisimilarity and similarity for probabilistic processes," *J. Comput. Syst. Sci.*, vol. 60, no. 1, pp. 187–231, Feb. 2000. [Online]. Available: http://dx.doi.org/10.1006/jcss.1999.1683

[22] S. Cattani and R. Segala, "Decision algorithms for probabilistic bisimulation," in *Proceedings of the 13th International Conference on Concurrency Theory*, ser. CONCUR '02. Berlin, Heidelberg: Springer-Verlag, 2002, pp. 371–385. [Online]. Available: http://dl.acm.org/citation.cfm?id=646737.701950

[23] N. Kamaleson, "Model reduction techniques for probabilistic verification of markov chains," Ph.D. dissertation, University of Birmingham, 2018.

[24] P. Godefroid, "Using partial orders to improve automatic verification methods," in *Proceedings of the 2Nd International Workshop on Computer Aided Verification*, ser. CAV '90. Berlin, Heidelberg: Springer-Verlag, 1991, pp. 176–185. [Online]. Available: http://dl.acm.org/citation.cfm?id=647759.735044

[25] D. Peled, "Combining partial order reductions with on-the-fly model-checking," in *Computer Aided Verification*, D. L. Dill, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 377–390.

[26] R. Alur, R. K. Brayton, T. A. Henzinger, S. Qadeer, and S. K. Rajamani, "Partial-order reduction in symbolic state space exploration," in *Proceedings of the 9th International Conference on Computer Aided Verification*, ser. CAV '97. London, UK, UK: Springer-Verlag, 1997, pp. 340–351. [Online]. Available: http://dl.acm.org/citation.cfm?id=647766.733599

[27] R. Kurshan, V. Levin, M. Minea, D. Peled, and H. Yenigün, "Static partial order reduction," in *Tools and Algorithms for the Construction and Analysis of Systems*, B. Steffen, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 345–357.

[28] E. Clarke, O. Grumberg, M. Minea, and D. Peled, "State space reduction using partial order techniques," *International Journal on Software Tools for Technology Transfer*, vol. 2, no. 3, pp. 279–287, Nov 1999. [Online]. Available: https://doi.org/10.1007/s100090050035

[29] C. Flanagan and P. Godefroid, "Dynamic partial-order reduction for model checking software," in *Proceedings of the 32Nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ser. POPL '05. New York, NY, USA: ACM, 2005, pp. 110–121. [Online]. Available: http://doi.acm.org/10.1145/1040305.1040315

[30] J. Hasenauer, V. Wolf, A. Kazeroonian, and F. J. Theis, "Method of conditional moments (mcm) for the chemical master equation," *Journal of Mathematical Biology*, vol. 69, no. 3, pp. 687–735, Sep 2014. [Online]. Available: https://doi.org/10.1007/s00285-013-0711-5

[31] A. Andreychenko, L. Mikeev, D. Spieler, and V. Wolf, "Parameter identification for markov models of biochemical reactions," in *Computer Aided Verification*, G. Gopalakrishnan and S. Qadeer, Eds. Springer Berlin Heidelberg, 2011, pp. 83–98.

[32] L. Zhang, H. Hermanns, E. M. Hahn, and B. Wachter, "Time-bounded model checking of infinite-state continuous-time Markov chains," in *ACSD*. IEEE, 2008, pp. 98–107.

[33] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton, "Model-checking continuous-time markov chains," *ACM Trans. Comput. Logic*, vol. 1, no. 1, pp. 162–170, Jul. 2000. [Online]. Available: http://doi.acm.org/10.1145/343369.343402

[34] C. Baier, B. Haverkort, H. Hermanns, and J. . Katoen, "Model-checking algorithms for continuous-time markov chains," *IEEE Transactions on Software Engineering*, vol. 29, no. 6, pp. 524–541, June 2003.

[35] W. Grassmann, "Transient solutions in markovian queueing systems," *Computers & Operations Research*, vol. 4, no. 1, pp. 47 – 53, 1977. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0305054877900077

[36] D. Gross and D. R. Miller, "The randomization technique as a modeling tool and solution procedure for transient markov processes," *Oper. Res.*, vol. 32, no. 2, pp. 343–361, Apr. 1984. [Online]. Available: http://dx.doi.org/10.1287/opre.32.2.343

[37] M. Kwiatkowsa, G. Norman, and D. Parker, "The prism benchmark suite," in *Quantitative Evaluation of Systems, International Conference on(QEST)*, vol. 00, 09 2012, pp. 203–204. [Online]. Available: doi.ieeecomputersociety.org/10.1109/QEST.2012.14

[38] https://depend.cs.uni-saarland.de/tools/infamy/casestudies/.

[39] A. Eldar and M. B. Elowitz, "Functional roles for noise in genetic circuits," *Nature*, vol. 467, no. 7312, pp. 167–173, Sep. 2010. [Online]. Available: https://doi.org/10.1038/nature09326

[40] H. Zheng, P.-Y. Ho, M. Jiang, B. Tang, W. Liu, D. Li, X. Yu, N. E. Kleckner, A. Amir, and C. Liu, "Interrogating the escherichia coli cell cycle by cell dimension perturbations," *Proceedings of the National Academy of Sciences*, vol. 113, no. 52, pp. 15 000–15 005, 2016.