Towards secure and robust stateful deep learning systems with model-based analysis

Du, Xiaoning

2019

Du, X. (2019). Towards secure and robust stateful deep learning systems with model-based analysis. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/137015>

<https://doi.org/10.32657/10356/137015>

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License (CC BY-NC 4.0).

Downloaded on 26 Apr 2023 17:58:44 SGT

**Towards Secure and Robust Stateful Deep
Learning Systems with Model-Based Analysis**

**XIAONING DU
SCHOOL OF COMPUTER SCIENCE AND
ENGINEERING**

2019

Towards Secure and Robust Stateful Deep Learning Systems with Model-Based Analysis

XIAONING DU

School of Computer Science and Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirement for the degree of
Doctor of Philosophy

2019

Abstract

More and more we start to embrace the convenience and effectiveness of the rapidly advancing artificial intelligence (AI) technologies in our lives and different industries. Within this revolution, deep learning (DL), as one of the key innovation in AI, has made significant progress over the past decades. However, even the state-of-the-art DL systems are susceptible to minor adversarial perturbations, and suffer from quality, reliability and security problems, preventing the deployment of DL systems on safety- and security-critical applications. An early-stage assessment of DL systems is crucial in discovering defects and improving the overall product quality.

Mature analysis processes and techniques have been established for traditional software, but it is highly non-trivial to directly apply them to DL systems. These challenges have motivated researchers to investigate testing, verification and adversarial sample detection of feed-forward neural networks, but little has been done on the recurrent neural network (RNN)-based stateful DL systems. In this thesis, we initiate the first major effort on the white-box RNN analysis using model-based approach to focus on the security and robustness properties and demonstrate the usefulness with applications on test case production, attack generation, and adversarial sample detection. To further protect the DL systems, we propose an efficient monitoring algorithm which could be potentially used to shield DL systems against adversarial samples at runtime, based on the RNN behaviors reflected by the abstract models.

The first part of the thesis focuses on RNN model extraction and offline analysis on the security and robustness properties based on the models extracted.

First, we model an RNN as an abstract state transition system, namely Discrete-Time Markov Chain (DTMC), to characterize its internal behaviors. We design an algorithm to automatically extract DTMC from a given RNN. Based on the abstract model, we design two trace similarity metrics and five coverage criteria which enable the quantitative analysis of RNNs. We further propose two algorithms powered by the quantitative measures for adversarial sample detection and coverage-guided test generation. We evaluate the framework, *DeepStellar*, on four RNN-based systems covering image classification and automated speech recognition. The results demonstrate that the abstract model is useful in capturing the internal behaviors of RNNs, and confirm that (1) the similarity metrics could effectively capture the differences between samples even with very small perturbations and (2) the coverage criteria are useful in revealing erroneous behaviors.

Secondly, we model RNN as an input-aware probabilistic abstract state transition system, namely Markov Decision Process (MDP), and define robustness property over the MDP. Robustness depicts how vulnerable the RNN is under minor input perturbations, thus the input awareness is of great importance. We first profile and perturb training data to collect RNN internal state information. Based on this, we build a probabilistic model to compactly characterize the behaviors of the RNN, through abstraction. Furthermore, we propose a refinement algorithm to iteratively derive an abstract model that enables to accurately quantify the robustness of an RNN. We evaluate the effectiveness on an LSTM model and a GRU model trained separately with two popular natural language datasets. The results demonstrate that our refinement algorithm is more efficient in deriving an accurate model than the random strategy, and our approach enables quantitative robustness analysis, in rendering better efficiency, accuracy, and scalability than the state-of-the-art technique.

The second part of the thesis incorporates runtime monitoring as the last line of defense to detect whether the system is under attack in real-time. Defects in DL systems are difficult to fix, even with samples revealing such defects at

hand. Hence, runtime monitoring plays an important role here to guard the systems against the adversary. Based on the robustness analysis, we find that perturbations over the input elements where RNN presents weaker robustness can lead to attacks more easily. Alarms shall be raised if the RNN becomes unrobustness when dealing with samples. Here, we monitor how often the runtime RNN behavioral trace triggered by a sample visits unrobust states. Temporal logic is versatile to describe history-sensitive properties. Traditionally, the use of temporal logic is to capture the qualitative aspects of the monitored systems. Nevertheless, quantitative specification is required to describe security policies for the adversarial sample detection, as well as malware detection and intrusion detection in platforms like Android and autonomous vehicle, where frequency of events is a vital feature of the malicious behavior. There exist some extensions of temporal logic to allow quantitative policy specification but the cost of the corresponding monitoring algorithms cannot achieve a time and space complexity that is completely independent of the trace length. To guarantee the expressiveness and efficiency at the same time, we develop a logic specification language, MTL_{cnt} , and derive a complete and efficient runtime verification algorithm with trace-length independent performance. MTL_{cnt} allows specification on the number of times some sub-policies are satisfied in the specified past time interval. With trace-length independence (TLI), the monitoring cost does not increase as the execution trace grows, which is crucial for monitoring on devices with restricted computation and storage resources, such as smartphones and embedded systems. Theoretically, we show that a broad class of policies, but not all policies, specified with MTL_{cnt} can be monitored in a TLI way, and provide a concrete algorithm to do so. Experimentally, we implement a prototype and test our algorithm on detecting Android malware and anomaly traffic in autonomous vehicles.

This thesis makes the first attempt to perform model-based analysis for RNNs with both offline and online techniques, with which we hope to open a new research direction for the white-box analysis of stateful DL systems.

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.

. . . . . 2/6/2020 . . . . .

. . . . .

Date

. . . . .

Xiaoning Du

Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

... 2/6/2020 ...

.....

Date

.....

Yang Liu

Authorship Attribution

Statement

This thesis contains materials from 2 conference papers and 1 journal paper published with peer reviews.

Chapter 3 is published as

Xiaoning Du, Xiaofei Xie, Yi Li, Lei Ma, Yang Liu, and Jianjun Zhao, “Deep-Stellar: Model-based Quantitative Analysis of Stateful Deep Learning Systems,” *In Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2019)*, pp477-487, 2019. <https://dl.acm.org/citation.cfm?id=3338954>. DOI: 10.1145/3338906.3338954

The contributions of the co-authors are as follows:

- Prof Ma proposed the topic and revised the manuscript drafts.
- I co-designed the methodology with Dr Xie and Prof Li.
- I initialized the manuscript drafts and completed all the implementation and experiments.
- Profs Liu, Ma, and Zhao provided insightful comments and revised the drafts.

Chapters 5 is published as

Xiaoning Du, Yang Liu and Alwen Tiu “Trace-length Independent Runtime Monitoring of Quantitative Policies in LTL,” *In International Symposium on Formal Methods (FM 2015)*, pp231-247, 2015. DOI: 10.1007/978-3-319-19249-9_15

and extended to a journal as

Xiaoning Du, Alwen Tiu, Kun Cheng, and Yang Liu, “Trace-Length Independent Runtime Monitoring of Quantitative Policies” *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 2019. DOI: 10.1109/TDSC.2019.2919693

The contributions of the co-authors are as follows:

- Prof Tiu proposed the topic and examined all the proofs.
- I designed the methodology.
- I initialized the manuscript drafts and designed the experimental evaluations.
- Dr Cheng provided help on the autonomous vehicle experiments.
- Profs Liu and Tiu provided insightful comments and revised the drafts.

. . . . .2/6/2020 . . . . .

Date

. . . . .

Xiaoning Du

Acknowledgements

At the time of wrapping up this thesis, I would like to express my sincere gratitude to my Ph.D. supervisors Prof Liu Yang and Prof Alwen Tiu for their guidance, patience, encouragement, and support. Prof Tiu helped me quickly start my work on runtime verification and set up a solid foundation of the later works. Prof Liu allowed me to indulge myself with different directions and eventually find the path to analyze deep learning systems. Without their help, I will not be able to finish this thesis.

I feel so lucky to have Prof Ma Lei, Prof Li Yi and Dr Xie Xiaofei around when I started to work on the DL research. Their support, motivation, and help are crucial to the publications. I will not be able to survive the tough journey without their accompany in the daily discussion, peer programming, debugging and paper writing. My Ph.D. journey becomes so colorful because of their supervision, and now the friendship.

Second, I also want to thank SCSE cyber security lab mates for their friendship and help in my four years' Ph.D. study, especially Dr Wang Junjie, Dr Chen Bihuan, Dr Wang Haijun, Dr Meng Guozhu, Dr Zhou Yuan, Dr Chen Sen, Dr Fan Lingling, Dr Chen Hongxu, Mr Feng Ruitao, Mr Xu Zhengzi, Mr Tang Yun, Mr Li Yuekang, Mr Cheng Kun, and many others. A special thanks goes to Ms Li Yanling, who has been playing tennis with me every week to keep me fit and energetic for research. I also want to thank CSL lab technician Mr Tan SuanHai for the kind IT support.

Last but not least, I would like to express my gratitude to my family: my parents, my younger sister and brother, who offered their unconditional support to me, especially during the four year's Ph.D. study. I will not be standing here without their love and sacrifice.

Contents

Abstract	i
Statement of Originality	iv
Supervisor Declaration Statement	v
Authorship Attribution Statement	vi
Acknowledgements	viii
Contents	x
List of Figures	xi
List of Tables	xii
1 Introduction	1
1.1 Motivations and Challenges	2
1.2 Main Work	5
1.3 Contributions of the Thesis	7
1.4 List of Materials Related to the Thesis	8
1.5 Outline of the Thesis	10
2 Background and Preliminaries	11
2.1 Neural Networks	11
2.1.1 Convolutional Neural Networks	12
2.1.2 Recurrent Neural Networks	13
2.2 Analysis of Deep Learning Systems	14
2.2.1 Abstraction of RNN	14
2.2.2 Robustness Analysis	15
2.2.3 Testing of DNN	17
2.2.4 Adversarial Sample Detection	18
2.3 Trace-length Independent Runtime Monitoring	18
3 Model-based Quantitative Analysis of RNN	21
3.1 Overview	22
3.2 State Transition Modeling of RNN	24

3.2.1	RNN Internal States and State Transitions	24
3.2.2	Abstract State Transition Model	25
3.2.3	Trace Similarity Metrics	27
3.2.4	Representing Trained RNN as a Discrete-Time Markov Chain	28
3.3	Coverage Criteria for RNN	29
3.3.1	State-Level Coverage Criteria	29
3.3.2	Transition-Level Coverage Criteria	31
3.4	Applications	32
3.4.1	Adversarial Sample Detection for RNNs	32
3.4.2	Coverage-Guided Testing of RNNs	33
3.5	Evaluation	35
3.5.1	Experiment Settings	36
3.5.2	RQ1: Trace Similarity	39
3.5.3	RQ2: Coverage Criteria	40
3.5.4	RQ3: Adversarial Sample Detection	42
3.5.5	RQ4: Coverage-guided Testing	44
3.5.6	Threats to Validity	45
3.6	Chapter Summary	45
4	Model-based Robustness Analysis of RNN	47
4.1	Problem Formulation	50
4.1.1	Recurrent Neural Networks and Their Traces	50
4.1.2	Point-wise Robustness of an RNN	52
4.1.3	Overview	54
4.2	RNN Abstraction using Labeled MDP	55
4.2.1	Aggregated Pointwise Robustness	55
4.2.2	Abstraction as Labeled MDP	56
4.2.3	k -step Aggregated Pointwise Robustness	59
4.3	Model Refinement Through State Clustering	61
4.3.1	Input and Output Abstraction	61
4.3.2	Refinement of State Abstraction	62
4.4	Evaluation	67
4.5	Conclusion	72
5	Trace-length Independent Runtime Monitoring	73
5.1	Specification Language MTL_{cnt}	75
5.1.1	Syntax of MTL_{cnt}	75
5.1.2	Semantics of MTL_{cnt}	77
5.1.3	Language Comparison	80
5.2	Preliminaries	83
5.2.1	Trace-length Independent Monitoring of $ptMTL$	83
5.2.2	Isolated Variables	85
5.3	Trace-length Independent Monitoring of MTL_{cnt}	86
5.3.1	TLI-Monitorable Arithmetic Relations	87

5.3.2	TLI-Monitorable Multivariate Relations	92
5.3.3	Recursive Semantics of Non-metric Temporal Counting Quantifier	94
5.3.4	Recursive Semantics of Metric Temporal Counting Quantifier	100
5.4	Monitoring Algorithm of MTL_{cnt}	106
5.5	Case Studies	111
5.6	Evaluation	115
5.6.1	Application on Android	115
5.6.2	Application on Autonomous Vehicle	120
5.7	Related Work	122
5.8	Chapter Summary	124
6	Conclusion and Future Work	125
6.1	Summary of the Thesis	125
6.2	Future Work	127
A	Appendix	128
A.1	Proof of Proposition 5.18	128
A.2	Proof of Theorem 5.21	129
A.3	Proof of Lemma 5.23	131
A.4	Proof of Lemma 5.24	133
A.5	Proof of Lemma 5.25	136
A.6	Proof of Theorem 5.26	136
	Bibliography	137

List of Figures

1.1	Research contribution of the thesis	4
2.1	Architecture of a simple RNN.	14
3.1	Overview of <i>DeepStellar</i> and its typical applications.	22
3.2	An example FST representing two traces.	24
3.3	A set of concrete traces and their corresponding abstract state transition model.	26
3.4	Visualization of concrete traces of two audio over an RNN-based ASR model with the PCA-3 abstraction.	28
4.1	FST example.	51
4.2	The workflow overview of MARBLE.	54
4.3	A set of concrete traces and their corresponding abstract state transition model.	58
4.4	Attack success rates based on the robustness scores calculated by different methods. For the least robust word, the larger the attack success rate is, the better; for the most robust word, the smaller the attack success rate is, the better.	71
5.1	Time of Single Round Checking on Android	118
5.2	Time of Single Round Checking on Autonomous Vehicle	121

List of Tables

3.1	Subject model information.	36
3.2	Abstract model details under different configurations for each studied RNN.	38
3.3	Correlation of trace similarities and prediction diff.	39
3.4	Coverage criteria sensitivity to the slightly perturbed samples and adversarial samples.	41
3.5	AUROC results (%) of trace similarity based adversarial detection by configurations.	42
3.6	Coverage and unique adversarial samples detected.	44
4.1	Measures of MDP models refined with different configurations and strategies.	70
5.1	Standard CAN Frame Format	113
5.2	Details of Policies for Android	117
5.3	Time Overhead under Different Throughput for Android	119
5.4	Details of Policies for Autonomous Vehicle	121

Chapter 1

Introduction

Over the past decade, artificial intelligence (AI) has achieved tremendous success in many cutting-edge applications in the real world. Especially, deep learning has become an integral part of AI, which has enabled great leaps of progress in applications like image processing [1], natural language processing [2], speech recognition [3], and autonomous driving [4], game playing [5, 6], robotics, finance, medicine, and more.

Given the ability to deal with complex problems and massive amounts of data, Deep Neural Networks (DNN) have often exceeded the capabilities of humans. However, just like traditional software, AI systems could also have defects [7]. For example, imperceptible perturbations on the input will cause the intelligent system to misclassify, in spite of having achieved high testing accuracy. The attacker may evade detection, force a system to cause damage, or even reveal confidential information contained within its training data. Real-world examples, like the Uber tragedy accident and intelligent audio-bots manipulated with hidden commands [8–10], have demonstrated the severe consequences in the safety- and security- critical applications. This raises the significant concern about the security and robustness of AI systems, particularly in the mission-critical scenarios.

By now, a lot of techniques including adversarial attack (*e.g.*, FGSM [11] and C&W [12]) and defense techniques (*e.g.*, distillation based defense [13]) have been proposed. However, such techniques are ad-hoc and available on specific models.

As the result, researchers have been exploring the domain of adversarial machine learning, to understand and mitigate this threat. In this thesis, we try to develop systematic approaches to analyze the behaviors, especially erroneous behaviors, of the DL systems, to quantify the robustness and to provide real-time protection against adversarial attacks, aiming to build secure and robust DL system.

1.1 Motivations and Challenges

Given limited time budget and human resources, to produce DL systems of high engineering quality is challenging. Actually, the development of traditional software has been faced the same problem. At the current moment, researchers have developed mature analysis processes (*e.g.*, Software Development Life Cycle) and techniques for traditional software. However, these existing techniques and toolchains cannot be applied to deep learning software directly because of the fundamental differences in the programming models, as well as the decision logic representations of the software artifacts (*e.g.*, architectures) [14–16]. To bridge the gap, research on testing [14–19], verification [20], and adversarial sample detection [21–23] of Feed-forward Neural Networks (FNN), *e.g.*, Convolution Neural Networks (CNN) and fully connected neural networks, started to emerge recently. Researchers also call for the establishment of secure deep learning development lifecycle [24, 25] to provide guidance for each development step, in view of that security issues could be introduced at each stage, even during the deployment [26].

Yet, the existing techniques are not specially designed to be applicable to Recurrent Neural Network (RNN)-based stateful DL systems, which are widely used in audio, natural languages and video processing, etc. Particularly, in contrast to FNN, RNNs capture the temporal behaviors by loops and memorization with internal states to take into account the influence of previous (or future) observations. Although recent work mentions the possibility to analyze an RNN through direct unrolling, treating it as an FNN [15, 16, 18], such a strategy is still far from sufficient to handle inputs of varying lengths. Here we define the problem we would like to solve with this thesis.

Problem Definition. Given an RNN, to better address its security and robustness concerns in a systematic way, we try to answer the following questions.

(1) On the quantitative behavior analysis and testing:

- How should we model RNNs to capture the internal behaviors of the stateful neural networks?
- How to define proper metrics to have a quantitative measurement of the RNN behavior? *e.g.*, how adequately its interval behaviors have been examined during testing?

(2) On the robustness analysis:

- Can we further indicate the robustness property of RNN over the behavior model?
- How to quantify the RNN robustness?

(3) On the real-time protection against adversarial attacks:

- How to specify security policies capturing the abnormal behaviors triggered by adversarial examples?
- How to design an efficient monitoring algorithm to examine these policies during runtime to avoid too much overhead?

Now we highlight the key challenges for solving the above mentioned problem.

1. The stateful nature of RNN introduces a new dimension of complexity for the security analysis. Different from an FNN, where each layer has a fixed role in feature extraction, a layer in an unrolled RNN often does not preserve the same feature latent space function (or semantics) for different input sequences. Therefore, the same unrolling which works well for one input may not fit for another.
2. The behavior space of RNN models is typically huge and can be even larger for real-world systems. Furthermore, one needs to unroll the gigantic networks many times to fully understand its stateful behavior, which maybe exponentially

FIG. 1.1: Research contribution of the thesis.

increase the analysis difficulty. The size and inherited complexity pose a big challenge in the scalability of analytical solutions.

3. Quantitative measures over the RNN behaviors is of great important for us to be aware of the testing adequacy, *i.e.*, how many behaviors have been examined by the test case. To design and select the most effective measures requires substantial efforts on the experimental evaluation, over different RNN architectures, model size, and abstraction configurations, *etc.* The challenge is to make the balance between the abstraction scalability and the measurement effectiveness.
4. The definition of the security and robustness of RNN requires precise understanding and (formal) behavior abstraction of the networks. Unfortunately, how to model RNN is not clear at the moment, and model extraction of complex RNN could be a non-trivial process, not mentioning how to perform the corresponding analysis on the security and robustness.
5. DL models cannot be fully secured due to incomplete training data. It is very tedious and time-consuming to perform retaining on the endless adversary examples. How to provide immediate security on the imperfect RNN models is an unknown question.

1.2 Main Work

Figure 1.1 presents the research overview of this thesis. First, we use Discrete-Time Markov Chain (DTMC) to model the RNN and utilize the abstract model to perform generic quantitative analyses on coverage-guided testing and adversarial example detection. To further understand the robustness of RNN, we extend the model to Markov Decision Process (MDP) to capture the input-aware internal transitions between states. Particularly, the robustness analysis over MDP model enables more efficient attack generation. Neural networks are usually with large attack surface due to the incompleteness of the training data. Offline adversarial example generation and retraining are time consuming and cost continuous effort. To have an immediate and real-time protection, runtime security defense is necessary. To this end, we develop a runtime monitoring framework to monitor security policies efficiently. Then we apply it to conduct runtime monitoring of RNN attacks and other traditional systems like mobile and autonomous vehicles. With these techniques, we are able to answer the questions above confirmatively and address the challenges mentioned in the previous section.

To elaborate on the approaches, we start with addressing the first two questions on the modeling of the RNN behaviors. It is infeasible to accurately capture the internal behaviors of RNN models due to its complexity. Therefore, we propose to model RNN as an abstract state transition system, i.e., DTMC, to characterize the probabilistic nature of the internal behaviors. With this formalization, we design an algorithm to automatically construct the DTMC and apply Principle Component Analysis (PCA) [27] to reduce the dimensions of the state vectors to control the abstraction level. Based on the abstract model, we design two trace similarity metrics and five coverage criteria which enable the quantitative analysis of RNNs. We further propose two algorithms powered by the quantitative measures for adversarial sample detection and coverage-guided test generation. The evaluation results demonstrate that the abstract model is useful in capturing the internal behaviors of RNNs, and confirm that (1) the similarity metrics could effectively

capture the differences between samples even with very small perturbations and (2) the coverage criteria are useful in revealing erroneous behaviors.

With the DTMC model, we can describe the RNN behavior in a probabilistic manner but are not able to express the relation between states, inputs, and robustness. Hence we introduce a more expressive model, named Markov Decision Process (MDP), to capture the robustness of RNN models. The most important usage of the MDP model is that we can define robustness property over the MDP. Robustness depicts how vulnerable the RNN is under minor input perturbations, thus the input awareness is of great importance. We first profiles and perturbs training data to collect RNN internal state information. Based on this, we build a probabilistic model to compactly characterize the behaviors of the RNN, through abstraction. Furthermore, we propose a refinement algorithm to iteratively derive an abstract model that enables to accurately quantify the robustness of an RNN. We evaluate the effectiveness on an LSTM model and a GRU model trained separately with two popular natural language datasets. The results demonstrate that our refinement algorithm is more efficient in deriving an accurate model than the random strategy, and our approach enables quantitative robustness analysis, in rendering better efficiency, accuracy, and scalability than the state-of-the-art technique.

These above two works together are able to provide a concrete answer to the first four questions. To explore the online security support, we turn our eyes to runtime monitoring, to examine how often the transitions triggered over the MDP are with weak robustness. Once the frequency is above a threshold, we raise an alarm and let the RNN-based system developer decide whether he/she needs to terminate the current inference. Linear temporal logic (LTL) has been widely used to specify runtime policies. Traditionally this use of LTL is to capture the qualitative aspects of the monitored systems, but recent developments in metric LTL and its extensions with aggregate operators allow some quantitative policies to be specified. Our interests in LTL-based policy languages is driven by its efficient monitoring algorithm, the time and space complexity of which is independent of the length of the system event traces so that its performance does not degrade

as the traces grow, *i.e.*, with a trace-length independent performance. For the monitoring of DL models, the trace-length independence property is also desired as the DL models are sometimes deployed on mobile phone or IoT (Internet of Things) systems, where the computation and storage resource are limited, and requires least overhead to prevent too much performance sacrifice. We propose a policy language based on a past-time variant of LTL, extended with an aggregate operator called the counting quantifier to specify a policy based on the number of times some sub-policies are satisfied in the past. We show that a broad class of policies, but not all policies, specified with our language can be monitored in a trace-length independent way without sacrificing completeness, and provide a concrete algorithm to do so.

1.3 Contributions of the Thesis

The main contributions of this thesis are highlighted in the following aspects.

First, we model an RNN as a probabilistic state transition system in the form of DTMC to capture the RNN's internal behaviors. Then we develop an algorithm to extract a DTMC from the given RNN automatically. Based on DTMC model, we have designed two trace similarity metrics and five coverage criteria, which can support different quantitative analysis for RNNs. We have developed two applications by the quantitative measures for adversarial sample detection and coverage-guided test generation.

Secondly, we further abstract an RNN as a robustness-aware abstract model, in the form of Markov Decision Process (MDP), which allows describing the relations between state, input, and robustness. Based on the MDP, we are able to estimate the model robustness over any sample efficiently. We measure the estimation error in comparison with the robustness reflected by mutation testing, and with *mean squared error* as the estimator. We also propose a refinement algorithm to continuously refine the MDP for more accurate robustness estimation, such that the estimation error satisfies a user-defined threshold. Further, we perform an

in-depth evaluation to demonstrate the effectiveness and efficiency of MARBLE for the robustness estimation of RNN.

Thirdly, this thesis incorporates runtime monitoring as the last line of defense against the adversary in real-time. We specify properties over the extracted MDP model with temporal logic to capture the anomaly behaviors. To guarantee the expressiveness and efficiency at the same time, we develop a logic specification language, MTL_{cnt} , and derive a complete and efficient runtime verification algorithm with trace-length independent performance. MTL_{cnt} allows specification on the number of times some sub-policies are satisfied in the specified past time interval. With trace-length independence, the monitoring cost does not increase as the execution trace grows, which is crucial for monitoring RNN attacks.

1.4 List of Materials Related to the Thesis

The thesis mainly contains the content from the following papers [28–31].

1. **Xiaoning Du**, Yang Liu and Alwen Tiu. “Trace-Length Independent Runtime Monitoring of Quantitative Policies in LTL,” *The 20th International Symposium on Formal Methods (FM 2015)*, pp. 231–247, Oslo, Norway, Jun. 22–26, 2015.
2. **Xiaoning Du**, Alwen Tiu, Kun Cheng, and Yang Liu. “Trace-Length Independent Runtime Monitoring of Quantitative Policies,” *IEEE Transactions on Dependable and Secure Computing (TDSC)*, May 2019. <https://ieeexplore.ieee.org/document/8725514>.
3. **Xiaoning Du**, Xiaofei Xie, Yi Li, Lei Ma, Yang Liu and Jianjun Zhao. “DeepStellar: Model-Based Quantitative Analysis of Stateful Deep Learning Systems,” *27th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (FSE 2019)*, Tallinn, Estonia, 26–30 August 2019.

4. **Xiaoning Du**, Xiaofei Xie, Yi Li, Lei Ma, Yang Liu and Jianjun Zhao. “A Quantitative Analysis Framework for Recurrent Neural Network,” *The 34th IEEE/ACM International Conference on Automated Software Engineering (ASE 2019 Tool Demo Track)*, San Diego, CA, USA on November 11-15, 2019.
5. **Xiaoning Du**, Xiaofei Xie, Yi Li, Lei Ma, Yang Liu and Jianjun Zhao. “Model-Based Robustness Analysis of Stateful Deep Learning Systems,” *32nd International Conference on Computer-Aided Verification (CAV 2020)*, Jul 2020. (Under submission)

The following papers [32–37] are the related papers published or under submission during my Ph.D. study.

6. **Xiaoning Du**, Bihuan Chen, Yuekang Li, Jianmin Guo, Yaqin Zhou, Yang Liu and Yu Jiang. “Attack Surface Identification for Vulnerability Assessment through Program Metrics,” *The 41st International Conference on Software Engineering (ICSE 2019)*, Montreal, Canada, 25-31, May, 2019.
7. **Xiaoning Du**. “Marvel: a generic, scalable and effective vulnerability detection platform.” *In Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings (ICSE 2019)*, pp. 129-131. IEEE Press, 2019.
8. **Xiaoning Du**. “Towards Building a Generic Vulnerability Detection Platform by Combining Scalable Attacking Surface Analysis and Directed Fuzzing.” *International Conference on Formal Engineering Methods (ICFEM 2018)*, pp. 464-468, 2018.
9. Yaqin Zhou, Shangqing Liu, Jingkai Siow, **Xiaoning Du** and Yang Liu. “Devin: Effective Vulnerability Identification by Learning Comprehensive Program Semantics via Graph Neural Networks,” *The Thirty-third International Conference on Neural Information Processing Systems (NeurIPS 2019)*, pp. 10197-10207, 2019.

10. Xiyue Zhang, Xiaofei Xie, Lei Ma, **Xiaoning Du**, Qiang Hu, Yang Liu, Jianjun Zhao and Meng Sun. “Towards Characterizing Adversarial Defects of Deep Learning Software from the Lens of Uncertainty,” *In Proceedings of the 42nd International Conference on Software Engineering: Companion Proceedings (ICSE 2020), Korea, May, 2020*.
11. Guangke Chen, Sen Chen, Lingling Fan, **Xiaoning Du**, Zhe Zhao, Fu Song, Yang Liu. *Who is Real Bob? Adversarial Attacks on Speaker Recognition Systems*, 41st IEEE Symposium on Security and Privacy (S&P 2020), (Under submission).

1.5 Outline of the Thesis

The rest of this thesis is organized as the following structure. The thesis organization can also be seen in Figure 1.1 with the chapter numbers.

Chapter 2 gives the preliminaries of deep learning systems (with the focus on RNN), security analysis on deep learning systems, linear temporal logic and runtime monitoring.

Chapter 3, from Papers 3 and 4, presents the DTMC model of RNN and their applications in test case generation and adversary detection.

Chapter 4, from Paper 5, explains the approach to model RNN using MDP model and the corresponding robustness analysis.

Chapter 5, from Papers 1 and 2, elaborates on the details about the policy language and runtime monitoring framework, and demonstrates the application of runtime monitoring on the attack detection for RNN models, and two other traditional systems.

Chapter 6 concludes the thesis and presents the possible future research directions in the area of RNN network analysis and runtime verification.

Chapter 2

Background and Preliminaries

This chapter presents the background and preliminaries related to the thesis. We first introduce preliminaries related to neural networks, particularly different types of neural networks with the special emphasis on RNN models. Then we move to the related works on security and robustness analysis of deep learning systems. Lastly, we present the basics of formal logics for describing security policy and runtime monitoring techniques.

2.1 Neural Networks

In 2016, AlphaGo from Google made a huge success and impact, which used deep learning algorithms to learn the Go game, and won the world champion by beating the top Go player. Since then, deep learning is starting to change the world by moving towards the AI-empowered solutions. These techniques boost the progress in fields including image analysis, natural language processing, and sound/video processing. Essentially, deep learning itself is a technique intending to “remember” data with complex structures with a set of non-linear transformations. The basic blocks of DL models are the neural networks, many of which are combined together to form DNNs.

Based on the architecture, there are two major types of neural networks:

1. Convolutional Neural Networks (CNN), which are usually used for processing images, and
2. Recurrent Neural Networks (RNN), which are used to deal with sequential data including text, audio, and video.

The key ideas of neural network are based on the deep cascade of multiple layers with smart stochastic optimization algorithms, and the configuration of the network with proper initial values, and well designed network structure to learn the data.

In the following, we will give the basic definition of neural network (NN). A NN is a non-linear function f w.r.t. its parameters θ . To use the function, we apply an input x to the parameters θ to generate an output y , *i.e.*, $y = f(x, \theta)$. To simplify the presentation, we make an assumption that the function y is unidimensional, however in reality that y can be multidimensional as well. As one kind of statistical learning, deep learning estimates the parameters θ from training data. Since the objective function is not convex, we can only rely on local minimizers. The usage of NN is twofold: they can be deployed for classification or regression. The reason for NN's success originates from a universal approximation theorem by Cybenko and Hornik. Most importantly in 1986, an efficient method, called backpropagation, is proposed to calculate the gradient of an NN, which simplifies the calculation of the quadratic criterion for a local minimizer. In the last decade, due to the recent advances in hardware (particularly the GUP or TPU), deep learning has managed to achieve very practical results in image, text and voice processing, even without solid theoretical foundations to understand the complexity inside the network.

2.1.1 Convolutional Neural Networks

The traditional machine learning for image applications relies on feature engineering, which requires a lot of experience in image processing and problem understanding. With the development in deep learning, particularly the concept of CNN, imaging processing is advanced significantly due to the elimination of manual feature

extraction step. The reason of this success is because CNN can directly work on tensors for images with three RGB values. As a result, CNN is currently widely adopted for image classification, segmentation, face recognition, etc.

A CNN contains an input layer, multiple hidden layers as well as an output layer. The special part about CNN is that the hidden layers contain a number of convolutional layers that convolve with multiplication or other dot product. In the hidden layers, inputs and outputs of the neurons are masked by the activation function and final convolution. The final convolution involves the backpropagation to generate a better weight in the end product. Due to the convention, we call the layers colloquially as convolutions. Actually, in the mathematical terminology, it is, in fact, a sliding dot product or cross-correlation, which has significance for the indices in the matrix.

2.1.2 Recurrent Neural Networks

CNN is good at processing the image format, but not sequential data such as text or audio. Then RNN is proposed for addressing this problem. It was invented in the 1980s. The architecture of a simple RNN is shown in Fig. 2.1. A simple RNN is a network of neuron-like nodes organized into successive *iterations*. It takes as inputs both the data stream and the internal state vector maintained. Instead of taking the input data as a whole, RNN processes a small chunk of data as it arrives, and sequentially produces outputs in each iteration while updating the internal states. In other words, information in an RNN not only flows from front neural layers to the rear ones but also from the current iteration to the subsequent ones. The stateful nature of an RNN contributes to its huge success in handling sequential data, such as audios and natural languages. At present, Long Short-Term Memory (LSTM) [38] and Gated Recurrent Unit (GRU) [39] are among the state-of-the-art and most widely used RNNs, designed with enhancement to overcome the “vanishing gradient problem” [40] that exists in the training process of most DL systems, and is aggravated by the iterative design of vanilla RNNs.

FIG. 2.1: Architecture of a simple RNN.

2.2 Analysis of Deep Learning Systems

2.2.1 Abstraction of RNN

Several approaches have been proposed to model RNN, but mostly in the form of Finite State Automaton (FSA). FSA help to manifest the internal state transitions explicitly and thus can be used to interpret the underlying decision rules embedded in an RNN. DTMC are superior with the ability to capture state transition distributions, making it more suitable for quantitative analysis. Constructing an FSA from an RNN usually requires two steps: (1) state space partition and abstraction, and (2) transition abstraction and automaton construction. Various partitioning strategies and automaton construction algorithms have been proposed. Omlin and Giles [41] proposed to split each dimension of the state vector into equal intervals, so as to divide the state space into regular grids. Unsupervised classification algorithms were also applied for state space partitions. For example, k -means and its variants were studied in [42–45]. Weiss et al. [20] devised an algorithm to dynamically create partitions, where an SVM classifier with an RBF kernel is fitted to separate several state vectors from its original partitions, and established the deterministic finite automata (DFA) with the help of L^* algorithm [46]. Further, they extend this approach to extract deterministic weighted automata from RNNs with a language modeling perspective [47]. Recent studies [20, 43, 44] have focused more on the interpretability and visualization of RNN behaviors, and try to simplify the abstract model by reducing the size of the models. When applied to real-world tasks, including NLP and speech recognition, the state space of the trained RNN models could be tremendously large. This makes scalability an issue for partition

techniques such as k -means and kernel algorithms. However, we adopted a cheaper interval abstraction and could benefit from its flexibility in precision adjustment.

2.2.2 Robustness Analysis

Robustness analysis of DNNs has received much attention due to its safety and security implications. Until now, the robustness measures mostly fall into two categories: *attack-dependent measures* and *attack-independent measures*. Interestingly, most literature is from the attack-specific perspective, *i.e.*, to examine the robustness of the RNN against certain types of attacks. The rest of this section discusses the most relevant literature to our work in analyzing the robustness of RNNs. We also cover some most recent studies on the robustness verification of feed-forward neural networks, although to the best of our knowledge, there does not exist such robustness verification for RNNs.

Adversarial Attacks on RNNs. Natural language processing (NLP) and automatic speech recognition are the two typical domains where existing RNN robustness attack applies. To attack NLP models, Papernot *et al.* [48] proposed one of the early works to launch adversarial attacks on text classification task with Fast Gradient Sign Method (FGSM). Li *et al.* [49] provided a technique to locate important and sensitive words with reinforcement learning, *e.g.*, to facilitate attack generation with word deletion. Following the similar line, Jia and Liang [50] proposed to score each word with an importance metric and leverage word manipulations, including swap, substitution, deletion and insertion, to generate adversarial examples for reading comprehension systems. Except for text classification models, there have also been works [51, 52] on generating adversarial examples to fool general-purpose sequence-to-sequence models.

For attacks of speech recognition models, Gong and Poellabauer [53] presented an approach to generate untargeted audio attacks, while Cisse *et al.* [54] proposed a technique to generate phonetically similar phrases and make it possible to generate targeted attacks. Carlini and Wagner [10] further advanced the adversarial audio generation to produce imperceptible attacks for any given targets and evaluate

their approach on DeepSpeech [55], the state-of-the-art RNN-based speech to text model. The after-mentioned approaches all reply on gradient-based algorithms, which require the white-box information on the full parameters of the subject models. A GAN-based black-box attack generation is designed in [56] with the demonstrations on textual entailment and machine translation for untargeted attacks. Analyzing the difficulties of leveraging existing attacks to RNN through either direct application or transfer attacks enables to estimate robustness of an RNN against an input.

Robustness Verification and Estimation of DNN. Instead of answering the question whether there exists an attack that can compromise the DNN, robustness verification aims to identify the minimum adversarial distortion bound such that no attack exists with distortion under this lower bound.

Existing works along this direction make some progress to verify the robustness of feed-forward DNN. The minimum adversarial distortion calculation of ReLU networks is shown to be NP-hard [57]. Therefore, further works attempt to compute a non-trivial certified lower bound [58–60] or to find an estimation of the minimum adversarial distortion [61, 62]. Bastani *et al.* [62] proposed to encode the DNN as a linear programming problem, and defined a robustness metric based on the adversarial examples discovered. This metric depicts an upper bound of the minimum distortion and also depends on specific attack algorithms. In contrast, the CLEVER score [61] provides an attack-agnostic estimation of the lower bound. Moreover, researchers further investigate how to borrow experiences and techniques from the classic program verification to certify the robustness of DNN. Gehr *et al.* [63] developed the first sound analyzer for DNN with abstract interpretation [64, 65], to automatically prove robustness properties. The analyzer is relatively efficient, but still lacks scalability to examine complex real-world sized DNNs.

To the best of our knowledge, the only existing research on the robustness verification of RNN is POPQORN [66], which proposes a robustness quantification framework to develop a guaranteed lower bound of the minimum distortion in

RNN attacks. Its performance was evaluated on vanilla RNN, as well as advanced RNN variants including LSTM and GRU. However, this approach is limited to only classification problems. It also suffers from scalability issues, which cannot handle hundreds of hidden layers as demonstrated in our empirical evaluation.

2.2.3 Testing of DNN

The lack of robustness places a major threat to the commercialization and wide adoption of DL systems. Researchers have devoted a great amount of efforts to investigate effective and systematic approaches to test DL systems, led with a pioneering work of Pei et al. [14]. The authors designed the first testing criterion – *neuron coverage* – to measure how much internal logic of DNNs has been examined by a given set of test data. Several new criteria have been proposed since then, including a set of multi-granularity testing criteria proposed in DeepGauge [16], a set of adapted MC/DC test criteria [67], and combinatorial testing criteria [68]. So far, the proposed coverage criteria are used to guide the metamorphic mutation-based testing [15], concolic testing [17], differential testing [19] and coverage-guided testing of DNN [69, 70]. In addition, mutation testing technique is also proposed to evaluate the test data quality through injecting faults into DL models [71]. In [19], a black-box differential testing framework is proposed for detecting the disagreements between multiple models.

MC/DC criteria are limited in scalability, and other criteria are specific to the FNN architecture, even though applicable to RNN via unrolling. The results reported in [15] demonstrated that the neuron coverage works effectively on FNN but far from ideal on RNN when used to guide test generation. This indicates that RNN is beyond a simple folding of CNN, and existing criteria may not be well suited for it. Due to the page limit, we refer the interested readers to a comprehensive survey on machine learning testing [72].

2.2.4 Adversarial Sample Detection

Some techniques [21–23, 73–75] are proposed to detect adversarial samples that are predicted incorrectly. The authors found that adversarial samples are much more sensitive in model mutants, and proposed a sensitivity-based approach to detect adversarial samples for feed-forward models [21]. The technique in [76] augments the DNNs by adding a small subnetwork which obtains inputs from the intermediate feature representations of the DNN and is trained to detect adversarial samples. The authors proposed two features [23], *i.e.*, density estimates and Bayesian uncertainty estimates to show the differences between benign and adversarial samples. Xu et al. [74] adopt two types of feature squeezing, *i.e.*, reducing the color bit depth of each pixel and spatial smoothing to detect adversarial samples. These approaches mainly consider CNNs and the image classification domain.

In [22], the authors detect the adversarial samples based on that the adversarial samples have a relatively smaller softmax probability. Following this line, the technique proposed in [73] observes that the softmax probabilities between in- and out-of-distribution samples can be further enlarged by temperature scaling in the softmax function. Such methods can be used on RNNs but are limited to the classification problem. Compared with them, our approach can handle sequential outputs of RNNs (*e.g.*, the outputs in automated speech recognition) based on the abstract model.

2.3 Trace-length Independent Runtime Monitoring

Under the scenario of runtime monitoring, we assume a runtime monitor runs in parallel with the system to be monitored, and the monitor has access to certain events emitted by the system. The intention is to monitor the *violation* of specified properties at every state/world of the trace. Whenever a new event arrives, the monitor would check whether the required property is violated in the trace resulting

from extending the history with the new event. Possibly, it needs to keep the entire trace to check the satisfaction.

Any algorithm that implements the satisfiability relation in Def. 5.1 can be used as a monitoring algorithm. In general, an algorithm that checks the satisfiability relation $(\rho, \tau, \nu, i \models \psi)$ would need to walk through the event trace ρ from the beginning up to the world i . In a setting with limited storage and computation resource (*e.g.*, an OS kernel or embedded devices), a runtime monitoring algorithm that requires storage and inspection of an entire event trace may not be practical, even if its complexity is linear in the size of the trace, since the performance of the monitor will degrade as the event trace grows. The dependency on the size of the policies (*i.e.*, formulas) is not so much of a problem in practice since the policies are usually fixed for a given system, or at least updated less frequently compared to event traces. Ad hoc restrictions such as limiting the time window or enforcing bounded storage of events are not desirable as they may introduce incompleteness with respect to the policies being monitored, *i.e.*, there may be violations to the policies that can only be detected on a trace of events longer than what could fit in the storage. In early work [77], monitoring algorithms are designed so that they require constant memory and take linear time to the size of the trace for a given formula to report the satisfiability relation in each world. Their algorithm is complete with formulas specified in ptLTL, and one needs to maintain only two states of all the subformulas of a policy. Such monitoring algorithms are termed as *trace-length independent* (TLI) monitoring algorithms in [78]:

Definition 2.1. A monitoring algorithm $\mathcal{M}$ is trace-length independent for a specification language $\mathcal{L}$ if for any trace and formula specified in $\mathcal{L}$, the interpretation for all worlds can be done in linear time in the size of trace, with its constant depending only on the size of formula, and with constant memory space which also depends only on the size of formula.

In designing the algorithm, trace-length independence is achieved when only a bounded history is required for checking at each state, *i.e.*, in a recursive paradigm. Formally, there exists a fixed k such that the interpretation of a formula in the

current world i depends only on the interpretations of its strict subformulas in the same world, or the truth values of its subformulas in at most k previous worlds. i.e., the interpretations in the following set:

$$\begin{aligned} &\{i - j \vdash \varphi \mid \varphi \text{ is a subformula of } \psi \text{ and } 1 \leq j \leq k\} \\ &\cup \{i \vdash \varphi \mid \varphi \text{ is a } \textit{strict} \text{ subformula of } \psi.\}. \end{aligned}$$

For example, the TLI monitoring algorithm for ptLTL in [77] follows such routine, with $k = 1$.

In a monitoring scenario where one needs to process events one at a time as they arrive, a TLI algorithm has the advantage that it does not need to store the entire trace of events seen so far. Rather, it only has to remember the truth values of every subformulas of the policy up to at most k previous worlds.

Chapter 3

Model-based Quantitative Analysis of RNN

The iterative characteristics of RNN have made its understanding and analysis more challenging. To better characterize the internal behaviors of RNNs, this chapter presents *DeepStellar*, a general-purpose quantitative analysis framework for RNN-based DL systems. Considering its stateful nature, we first model an RNN as Discrete-Time Markov Chain (DTMC) to capture its statistical behaviors. Based on the DTMC model, we design two trace similarity metrics to quantify the prediction proximity of different inputs, and five coverage criteria to measure the adequacy of test data from different perspectives.

To further demonstrate the usefulness of *DeepStellar*, we develop algorithms for two applications based the quantitative analysis, namely, RNN testing and adversarial sample detection. We implemented *DeepStellar* and empirically evaluated the usefulness of the abstract model, similarity metrics and coverage criteria on four RNN-based systems from image classification to Automated Speech Recognition (ASR). Specifically, we first performed controlled experiments to evaluate the capability of the abstract model. The results demonstrate that, (1) the trace similarity metrics serve as good indicators of the discriminatory power of RNNs, and the abstract model is sensitive enough to distinguish inputs generated with very small perturbations; and (2) the coverage criteria derived from the abstract model are

FIG. 3.1: Overview of *DeepStellar* and its typical applications.

able to measure test adequacy and are effective in manifesting erroneous behaviors. Further, we applied the metrics and criteria on two applications, *i.e.*, adversarial sample detection and coverage-guided testing of RNNs. The results show that *DeepStellar* (1) detects 89% and 97% of the adversarial samples, respectively, for ASR and image classification systems, and (2) generates tests with high coverage and yields at most hundreds of times more adversarial samples than random testing and existing neuron coverage guided testing with unrolling [15].

3.1 Overview

Fig. 3.1 summarizes the workflow of our approach, including the *abstract model construction* of RNN, different *quantitative measures* defined over the abstract model, and two *applications* to detect and generate adversarial samples of RNNs.

The abstract model construction module takes a trained RNN as input and analyses its internal behaviors through profiling. The inputs for profiling are from the training data, which can best reflect the characteristics of a trained RNN model. Specifically, each input sequence is profiled to derive a *trace*, *i.e.*, a sequence of RNN state vectors. After the profiling, we can get a set of traces which record the states visited and transitions taken during the training stage.

In practice, the internal state space of an RNN and the number of traces enabled by the training data are often beyond our analysis capability. Therefore, we perform abstraction over the states and traces to obtain an abstract model that captures the global characteristics of the trained network. At the state level, we

apply Principle Component Analysis (PCA) [27] to reduce the dimensions of the state vectors and keeps the first k most dominant components. For each of the k dimensions, we further partition it into m equal intervals. At the transition level, we consolidate concrete transitions into abstract ones according to the abstract states. We also take into account the frequencies of different transitions at each state and effectively derive a Discrete-Time Markov Chain (DTMC) [79] model for the trained RNN.

Based on the abstract model, we design two metrics for evaluating the trace similarity induced by different inputs, and five coverage criteria to facilitate the systematic testing of RNNs. The metrics and coverage criteria are designed from both the state- and transition-level. Specifically, the trace similarity metrics include *state-based trace similarity* (SBTSIM) and *transition-based trace similarity* (TBTSIM). The coverage criteria include the *basic state coverage* (BSCov), *n-step state boundary coverage* (n-SBCov), *weighted state coverage* (WSCov), *basic transition coverage* (BTCov), and *weighted transition coverage* (WTCov).

We then apply the metrics and criteria on two applications, *i.e.*, the *adversarial sample detection* and *coverage-guided testing*, both of which aim to mitigate the threats from adversarial samples. With the similarity metrics, we propose an approach to detect adversarial samples at runtime. With the coverage criteria, we develop a testing framework to guide test generation with the aim to improve coverage and uncover defects for quality assurance. The two techniques are complementary to each other. The testing technique aims to generate unseen adversarial samples that help developers analyze and improve the robustness of the model. The adversarial sample detection technique is able to identify malicious inputs and prevent potential damages at runtime.

FIG. 3.2: An example FST representing two traces.

3.2 State Transition Modeling of RNN

3.2.1 RNN Internal States and State Transitions

Following [80], we represent a neural network abstractly as a differentiable parameterized function $f(\cdot)$. The input to an RNN is a sequence $\mathbf{x} \in \mathcal{X}^N$, where $\mathcal{X}$ is the input domain and N is the length of the sequence. Let $x_i \in \mathcal{X}$ be the i -th element of the sequence $\mathbf{x}$. Then, when passing $\mathbf{x}$ into an RNN, it maintains a state vector $\mathbf{s} \in \mathcal{S}^N$ with $s_0 = \mathbf{0}$ and $(s_{i+1}, y_i) = f(s_i, x_i)$, where $\mathcal{S}$ is the domain of the hidden state, $s_i \in \mathcal{S}$ is the hidden state of RNN at the i -th iteration, and $y_i \in \mathcal{O}$ is the corresponding output at that step. We use s_i^d to denote the d -th dimension of the state vector s_i .

Naturally, each input sequence $\mathbf{x}$ induces a finite sequence of state transitions $\mathbf{t}$, which we define as a *trace*. The i -th element in a trace $\mathbf{t}$, denoted by t_i , is the transition from s_i to s_{i+1} after accepting an input x_i and producing an output y_i . A Finite State Transducer (FST) [81] can be used to represent a collection of traces more compactly [82] as defined below.

Definition 3.1. An FST is a tuple $(\mathcal{S}, \mathcal{X}, \mathcal{O}, I, F, \delta)$ such that $\mathcal{S}$ is a non-empty finite set of states, $\mathcal{X}$ is the input alphabet, $\mathcal{O}$ is the output alphabet, $I \subseteq \mathcal{S}$ is the set of initial states, $F \subseteq \mathcal{S}$ is the set of final states, and $\delta \subseteq \mathcal{S} \times \mathcal{X} \times \mathcal{O} \times \mathcal{S}$ is the transition relation.

For example, Fig. 4.1 shows a simple FST representing two traces, namely, $s_0 s_1 s_2 s_3$ and $s_0 s_1 s'_2 s_3$ with s_0 being the initial state and s_3 being the final state. The first trace takes an input sequence $x_0 x_1 x_2$ and emits an output sequence $y_0 y_1 y_2$; the second trace takes an input sequence $x_0 x'_1 x'_2$ and emits an output sequence $y_0 y'_1 y'_2$.

3.2.2 Abstract State Transition Model

The number of states and traces enabled while training an RNN can be huge. To effectively capture the behaviors triggered by a large number of input sequences and better capture the global characteristics of the trained network, we introduce an *abstract state transition model* in this chapter. The abstract model overapproximates the observed traces induced of an RNN and has a much smaller set of states and transitions compared with the original one. The abstraction is also configurable – one can trade-off between the size and precision of the model so that the abstract model is still able to maintain useful information of the input sequences for particular analysis tasks. To obtain an abstract model for a trained RNN, we abstract over both the states and the transitions.

State Abstraction. Each *concrete state* s_i is represented as a vector $(s_i^1, \dots, s_i^l)$, usually in high dimension (*i.e.*, l could be a large number). Intuitively, an *abstract state* represents a set of concrete states which are close in space. To obtain such a state abstraction, we first apply the Principle Component Analysis (PCA) [27] to perform an orthogonal transformation on the concrete states – finding the first k principle components (*i.e.*, axes) which best distinguish the given state vectors and ignore their differences on the other components. This is effectively to project all concrete states onto the chosen k -dimensional component basis (denoted as PCA- k).

Then, we split the new k -dimensional space into m^k *regular grids* [83] such that there are m equal-length intervals on each axis:

$$e_i^d = [lb_d + i \times \frac{ub_d - lb_d}{m}, lb_d + (i + 1) \times \frac{ub_d - lb_d}{m}],$$

where e_i^d represents the i -th interval on the d -th dimension, lb_d and ub_d are the lower and upper bounds of all state vectors on the d -th dimension, respectively. In this way, all concrete states s_i which fall within the same grid are mapped to the same abstract state: $\hat{s} = \{s_i \mid s_i^1 \in e_-^1 \wedge \dots \wedge s_i^k \in e_-^k\}$. We denote the set of all

FIG. 3.3: A set of concrete traces and their corresponding abstract state transition model.

abstract states as $\hat{\mathcal{S}}$. Noticeably, the precision of the state abstraction can easily be configured by tuning the parameters k and m .

Let $j = I^d(\hat{s})$ be the index of $\hat{s}$ on the d -th dimension such that for all $s \in \hat{s}$, s^d falls in e_j^d ($0 \leq j < m$). For any two abstract states $\hat{s}$ and $\hat{s}'$, we define their *distance* as:

$$Dist(\hat{s}, \hat{s}') = \sum_{d=1}^k |I^d(\hat{s}) - I^d(\hat{s}')|.$$

This definition can also be generalized to include space beyond the lower and upper bounds.

Transition Abstraction. Once the state abstraction is computed, a concrete transition between two concrete states can be mapped as a part of an *abstract transition*. An abstract transition represents a set of concrete transitions which share the same source and destination abstract states. In other words, there is an abstract transition between two abstract states $\hat{s}$ and $\hat{s}'$ if and only if there exists a concrete transition between s and s' such that $s \in \hat{s} \wedge s' \in \hat{s}'$. The set of all abstract transitions is denoted as $\hat{\delta} \subseteq \hat{\mathcal{S}} \times \hat{\mathcal{S}}$.

For instance, Fig. 3.3(a) depicts three concrete traces, *i.e.*, $\mathbf{t}_1$, $\mathbf{t}_2$ and $\mathbf{t}_3$, where states are shown as dots and transitions are directed edges. The grids drawn in dashed lines represent the abstract states, *i.e.*, $\hat{s}_0$, $\hat{s}_1$, $\hat{s}_2$, and $\hat{s}_3$, each of which

is mapped to a set of concrete states inside the corresponding grid. The set of abstract transitions is, therefore, $\{(\hat{s}_0, \hat{s}_1), (\hat{s}_1, \hat{s}_0), (\hat{s}_1, \hat{s}_1), (\hat{s}_1, \hat{s}_2), (\hat{s}_1, \hat{s}_3), (\hat{s}_3, \hat{s}_3)\}$.

3.2.3 Trace Similarity Metrics

To precisely compare two input sequences, we define the *trace similarity metrics* to quantify the proximity of their induced state transitions on the abstract model. Given an abstract model M and an input $\mathbf{x}$, we denote the set of abstract states and transitions covered by $\mathbf{x}$ as $\hat{\mathcal{S}}_{\mathbf{x}}$ and $\hat{\delta}_{\mathbf{x}}$. Then, the *state- and transition-based trace similarity metrics* for the two inputs $\mathbf{x}$ and $\mathbf{y}$ are defined based on the Jaccard indices of their states and transitions covered, respectively:

$$\text{STSIM}_M(\mathbf{x}, \mathbf{y}) = \frac{|\hat{\mathcal{S}}_{\mathbf{x}} \cap \hat{\mathcal{S}}_{\mathbf{y}}|}{|\hat{\mathcal{S}}_{\mathbf{x}} \cup \hat{\mathcal{S}}_{\mathbf{y}}|}, \quad \text{TTSIM}_M(\mathbf{x}, \mathbf{y}) = \frac{|\hat{\delta}_{\mathbf{x}} \cap \hat{\delta}_{\mathbf{y}}|}{|\hat{\delta}_{\mathbf{x}} \cup \hat{\delta}_{\mathbf{y}}|}.$$

The trace similarity metrics range over $[0, 1]$, where 0 indicates disjoint sets (*i.e.*, traces induced by $\mathbf{x}$ and $\mathbf{y}$ are totally different), while 1 indicating equal sets (*i.e.*, the traces are similar).

Fig. 3.4 shows the concrete traces on an RNN-based ASR model induced by two speech input samples, Fig. 3.4(a) (“this book is about science”) and Fig. 3.4(b) (“this book is about literature”). The darker dots appear earlier in the sequence, and vice versa. We can see a clear difference of the two at later parts of the sequence. The two concrete traces are then projected onto the 3-dimensional space under the PCA-3 abstraction with 5 intervals on each dimension, to calculate the trace similarities. The state- and transition-based trace similarities of the two inputs are 0.71 and 0.64, respectively.

Each input sequence in the training set yields a concrete trace of the RNN model. The abstract state transition model captures all the concrete traces enabled from training data (or its representative parts) and other potential traces which have not been enabled. The defined state and transition abstraction make the resulting abstract model represent an over-approximation and generalization of the observed behaviors of the trained RNN model.

FIG. 3.4: Visualization of concrete traces of two audio over an RNN-based ASR model with the PCA-3 abstraction.

3.2.4 Representing Trained RNN as a Discrete-Time Markov Chain

To also take into account the likelihood of transitions at different states, we augment the abstract model with transition probabilities, effectively making it a Discrete-Time Markov Chain (DTMC).

Definition 3.2. A DTMC is a tuple $(\hat{\mathcal{S}}, I, \hat{\mathcal{T}})$, where $\hat{\mathcal{S}}$ is a set of abstract states, I is a set of initial states, and $\hat{\mathcal{T}} : \hat{\mathcal{S}} \times \hat{\mathcal{S}} \mapsto [0, 1]$ is the *transition probability function* which gives the probability of different abstract transitions.

We write $\Pr(\hat{s}, \hat{s}')$ to denote the conditional probability of visiting $\hat{s}'$ given the current state $\hat{s}$, such that $\sum_{\hat{s}' \in \hat{\mathcal{S}}} \Pr(\hat{s}, \hat{s}') = 1$. We define the transition probability as the number of concrete transitions from $\hat{s}$ to $\hat{s}'$ over the number of all outgoing concrete transitions from $\hat{s}$, *i.e.*,

$$\Pr(\hat{s}, \hat{s}') = \frac{|\{(s, s') \mid s \in \hat{s} \wedge s' \in \hat{s}'\}|}{|\{(s, -) \mid s \in \hat{s}\}|}.$$

For example, Fig. 3.3(b) shows the abstract state transition model for the concrete traces in Fig. 3.3(a) as a DTMC. The abstract transitions are labeled with their transition probabilities. For instance, since all outgoing transitions at $\hat{s}_0$ end in $\hat{s}_1$, the transition probability from $\hat{s}_0$ to $\hat{s}_1$ is 1. There are four possible outgoing

transitions at $\hat{s}_1$, *i.e.*, $\{(\hat{s}_1, \hat{s}_1), (\hat{s}_1, \hat{s}_2), (\hat{s}_1, \hat{s}_3), (\hat{s}_1, \hat{s}_0)\}$. Hence, the transition probability from $\hat{s}_1$ to $\hat{s}_2$ is computed as $\Pr(\hat{s}_1, \hat{s}_2) = \frac{1}{4}$. Computation for other abstract transitions are similar.

As is shown in the example, a DTMC model is constructed by first applying the state and transition abstractions on a set of concrete traces, and then computing transition probability distributions for each abstract state. The time complexity of the abstraction step depends on the number of concrete traces, while the complexity for computing the transition probabilities only depends on the number of abstract transitions.

3.3 Coverage Criteria for RNN

Inspired by traditional software testing [84–89], we propose a set of testing coverage criteria for RNNs based on the abstract state transition model. The goal of the RNN coverage criteria is to measure the sufficiency of test data in exercising the trained as well as the unseen behaviors. The state and transition abstractions are designed to reflect the internal network configurations at a certain point as well as the temporal behaviors of the network over time, respectively. Therefore, to maximize the chance of discovering defects in stateful neural networks, one should combine coverage criteria based on both the state and transition abstractions to systematically generate comprehensive and diverse test suites.

Let $M = (\hat{\mathcal{S}}, I, \hat{\mathcal{T}})$ be an abstract model of the trained RNN represented as a DTMC. Let $T = \{\mathbf{x}_0, \dots, \mathbf{x}_n\}$ be a set of test input sequences. We define both the *state-level* and *transition-level* coverage of T to measure how extensively T exercises the states and transitions of M , respectively.

3.3.1 State-Level Coverage Criteria

The state-level coverage criteria focus on the internal states of the RNN. The set of abstract states $\hat{\mathcal{S}}$ represents a space generalization of the visited states obtained

from training data (or its representative parts), which is referred to as the *major function region* [16]. The space outside the major function region is never visited by the training data, and thus represents the *corner-case region* [16]. The test data should cover the major function region extensively to validate the trained behaviors and cover the corner-case region sufficiently in order to discover defects in unseen behaviors.

Basic State Coverage. Given an RNN abstract model M and a set of test inputs T , the *basic state coverage* measures how thoroughly T covers the major function region visited while training. To quantify this, we compare the set of abstract states visited by the training inputs and the test inputs, denoted by $\hat{\mathcal{S}}_M$ and $\hat{\mathcal{S}}_T$, respectively. Then, the basic state coverage is given by the number of abstract states visited by both the training and the test inputs over the number of states visited by the training inputs,

$$\text{BSCov}(T, M) = \frac{|\hat{\mathcal{S}}_T \cap \hat{\mathcal{S}}_M|}{|\hat{\mathcal{S}}_M|}.$$

Weighted State Coverage. The basic state coverage treats every state with equal weights. During training, not all states are visited equally often and one may want to emphasize more on some states than the others. To take into account the frequencies of different states and be able to assign weights to states, we define the *weighted state coverage* and allow users to specify a *weight function*. The default weight of an abstract state $\hat{s}$ is defined as the relative frequency of it among all the abstract states, *i.e.*,

$$w(\hat{s}) = \frac{|\{s \mid s \in \hat{s}\}|}{|\mathcal{S}|},$$

where $\mathcal{S}$ is the set of all distinct concrete states. Then, the weighted state coverage is defined as:

$$\text{WSCov}(T, M) = \frac{\sum_{\hat{s} \in \hat{\mathcal{S}}_T \cap \hat{\mathcal{S}}_M} w(\hat{s})}{\sum_{\hat{s} \in \hat{\mathcal{S}}_M} w(\hat{s})}.$$

In practice, the weight function can be defined differently according to specific needs. For example, when a constant function is chosen, the weighted state coverage is

equivalent to the basic one. In § 3.5.3, we evaluate two weight functions, including the default and the reversed one which assigns larger weight to less visited states.

n -Step State Boundary Coverage. The test data may also trigger new states that are never visited during training. The n -step state boundary coverage measures how well the corner-case regions are covered by the test inputs T . The corner-case regions $\hat{\mathcal{S}}_{M^c}$ are the set of abstract states outside of $\hat{\mathcal{S}}_M$, which have non-zero distances from any states in $\hat{\mathcal{S}}_M$. Then $\hat{\mathcal{S}}_{M^c}$ can be further divided into different boundary regions defined by their distances from $\hat{\mathcal{S}}_M$. For example, the n -step boundary region, $\hat{\mathcal{S}}_{M^c}(n)$, contains all abstract states which have a minimal distance n from $\hat{\mathcal{S}}_M$, or more formally,

$$\hat{\mathcal{S}}_{M^c}(n) = \{\hat{s} \in \hat{\mathcal{S}}_{M^c} \mid \min_{\hat{s}' \in \hat{\mathcal{S}}_M} \text{Dist}(\hat{s}, \hat{s}') = n\}.$$

The n -step state boundary coverage is defined as the ratio of states visited by the test inputs in the boundary regions of at most n steps away from $\hat{\mathcal{S}}_M$:

$$n\text{-SBCov}(T, M) = \frac{|\hat{\mathcal{S}}_T \cap \bigcup_{i=1}^n \hat{\mathcal{S}}_{M^c}(i)|}{|\bigcup_{i=1}^n \hat{\mathcal{S}}_{M^c}(i)|}.$$

3.3.2 Transition-Level Coverage Criteria

The state-level coverage indicates how thorough the internal states of an RNN are exercised but it does not reflect the different ways transitions have happened among states in successive time steps. The transition-level coverage criteria targets at the abstract transitions activated by various input sequences and a higher transition coverage shows that the inputs are more adequate in triggering diverse temporal dynamic behaviors.

Basic Transition Coverage. To quantify transition coverage, we compare the abstract transitions exercised during both the training and testing stages, written as $\hat{\delta}_M$ and $\hat{\delta}_T$, respectively. The *basic transition coverage* is defined as:

$$\text{BTCov}(T, M) = \frac{|\hat{\delta}_T \cap \hat{\delta}_M|}{|\hat{\delta}_M|}.$$

Basic transition coverage subsumes basic state coverage. In other words, for any abstract model M , a test input T satisfies basic transition coverage regarding to M , also satisfies basic state coverage.

Weighted Transition Coverage. Similar as the state-level coverage, we can calculate the *weighted transition coverage* by considering the relative frequency of each transition. More formally,

$$\text{WTCov}(T, M) = \frac{\sum_{(\hat{s}, \hat{s}') \in \hat{\delta}_T \cap \hat{\delta}_M} w(\hat{s}, \hat{s}')}{\sum_{(\hat{s}, \hat{s}') \in \hat{\delta}_M} w(\hat{s}, \hat{s}')}.$$

where the weight function can be configured similarly as in the weighted state coverage. By default, the weight of a transition $(\hat{s}, \hat{s}')$ is computed as transition probability defined in § 3.2.4.

3.4 Applications

To demonstrate the usefulness of the abstract model and the proposed quality measures, we apply them in finding adversarial samples for RNNs in two scenarios: (1) adversarial sample detection to identify adversarial inputs at runtime, and (2) coverage-guided testing to generate unseen adversarial samples offline.

3.4.1 Adversarial Sample Detection for RNNs

Adversarial sample detection aims to check whether a given input is an adversarial sample at runtime. We propose to use the trace similarity metrics to measure the behavioral differences between two inputs. Based on this idea, we develop a new approach to detect adversarial samples for RNNs.

Given a target sample i , we define a *reference sample* r such that the RNN gives the same predictions for both i and r . The traces derived from the original and the reference samples are typically similar when i is benign. However, when i is adversarial, the trace difference between the two samples can be much larger.

With such insight, we propose to detect adversarial sample based on the trace difference from its reference sample.

We use a learning-based approach (Algorithm 1) to train a classifier. The inputs include an RNN-based system D and an abstract model M . We first collect a set of benign samples B and a set of adversarial samples A . The set of reference samples R are also prepared for comparison (more details in the next paragraph), and the trace similarities between benign/adversarial samples and the reference samples are calculated. For each benign sample b , the *select* function obtains a group of corresponding reference samples R' from the reference samples R . Based on this, we compute the trace similarity between b and each reference sample $r \in R'$ (Lines 6-10) and take the average similarity (Line 11) to represents the distance between b and the set of reference samples R' . The distance is added into the benign distance list dis_b (Line 12). Similarly, we compute the adversarial distance list dis_a (Line 13). With dis_a and dis_b , a linear regression classifier C is learned. Given a new input i , we compute the similarity d_i between i and the reference samples, and rely on $C(d_i)$ to indicate whether i is a benign or adversarial.

We apply and evaluate the detection algorithms on two domains, namely, ASR and image classification. The approaches used to generate reference samples are as follows: for ASR, given an audio input a that is transcribed to texts t by the RNN, we generate the reference audios using off-the-shelf text-to-speech engines (*e.g.*, Google cloud text-to-speech [90]) that generate the audio t with correct and clear pronouncing. For image classification, given a new image i with prediction result c (*i.e.*, the image i belongs to class c), the reference images are selected from the training data such that they share the same label as c .

3.4.2 Coverage-Guided Testing of RNNs

In this section, we propose a Coverage-Guided Testing (CGT) technique which aims to generate adversarial samples incorrectly recognized by an RNN. CGT uses the proposed coverage criteria (§ 3.3) to guide the test generation and evaluates the quality of the tests from different perspectives. During the testing process,

Algorithm 1: Training an Adversarial Detection Classifier

---

input : D : RNN-based DL system, M : Abstract model of D
output : C : A classifier for detecting adversarial samples

- 1 Prepare benign set B , adversarial set A and reference set R ;
- 2 $dis_b \leftarrow \emptyset$;
- 3 **for** $b \in B$ **do**
- 4 $R' \leftarrow select(R, b)$;
- 5 $vec \leftarrow \emptyset$;
- 6   **for** $r \in R'$ **do**
- 7 $(r_1, state\_vec1) \leftarrow predict(R, r)$;
- 8 $(r_2, state\_vec2) \leftarrow predict(R, b)$;
- 9 $j \leftarrow TraceSimilarity(state\_vec1, state\_vec2, M)$;
- 10 $vec \leftarrow vec \cup \{j\}$;
- 11 $d \leftarrow average(vec)$;
- 12 $dis_b \leftarrow dis_b \cup \{d\}$;
- 13 Compute dis_a similar with dis_b ;
- 14 $C \leftarrow LinearRegressionClassifier(dis_a, dis_b)$

CGT maintains a test queue. In each run, it selects a seed (*i.e.*, test case) from the queue and generates multiple mutants. A mutant is an adversarial sample if it is predicted incorrectly by the network. Otherwise, if the mutant improves the coverage, it is then retained as an interesting seed and added back to the queue.

Algorithm 2 presents the process to generate tests for RNNs. The inputs include the initial seeds I , the RNN-based DL system D and the abstract model M . The outputs are benign tests and failed tests for which D gives correct and incorrect inference respectively. The initial test queue contains a set of initial seeds. In each run, CGT selects one input a from the test queue (Line 2), and randomly picks a transformation function t with parameter p (Line 4). Then, a set of new samples A are generated under transformation t (Line 5). For each new sample a' , CGT first obtains the concrete trace by letting D do the inference (Line 7) and then calculates the coverage information over the abstract model (Line 8). If the inference is incorrect, a' is added into the failed test set F (Line 10). If a' is correctly predicted and covers new states or transitions, CGT puts it in the test queue and updates the coverage criteria of all tests currently in the queue (Line 12- 13).

A challenge in DL testing is the lack of oracle that tells the ground-truth label of

Algorithm 2: Coverage guided testing of RNN

```

input    :  $I$ : Initial seeds,  $D$ : RNN-based DL system,  $M$ : Abstract model of  $D$ 
output   :  $F$ : Failed tests,  $Q$ : Test queue
1  $F \leftarrow \emptyset$ ;
2  $Q \leftarrow I$ ;
3 while  $a \leftarrow \text{Select}(Q)$  do
4   Randomly pick transformation  $t$  with a random parameter  $p$  ;
5    $A = \text{mutate}(t, p, a)$ ;
6   for  $a' \in A$  do
7      $(\text{result}, \text{state\_vec}) \leftarrow \text{predict}(R, a')$ ;
8      $\text{cov} \leftarrow \text{CovAnalysis}(\text{state\_vec}, M)$ ;
9     if  $\text{Failed}(a', \text{result})$  then
10       $F \leftarrow F \cup \{a'\}$ 
11     else if  $\text{CoverageIncrease}(\text{cov}, Q)$  then
12       $Q \leftarrow Q \cup a'$ ;
13       $\text{UpdateCoverage}(Q)$ ;

```

---

any new mutant. Mutation operators are often specific to the application domains. For CGT, we mainly focus on image classification and apply the metamorphic mutation strategy [70] for generating new mutants that would keep the prediction meaning from the human’s perspective during testing.

3.5 Evaluation

To demonstrate the usefulness of the proposed techniques, we implemented *DeepStellar* in Pythonbased on the Keras (2.2.4) [91] with TensorFlow (1.4, 1.8 and 1.11) [92]. We first study whether the abstract model is able to characterize the stateful behaviors of RNNs (*i.e.*, **RQ1** & **RQ2**). Based on this, we further evaluate the usefulness of the proposed quantitative measures on the two applications (*i.e.*, **RQ3** & **RQ4**). Specifically, we leverage *DeepStellar* to investigate the following research questions:

RQ1: Are the proposed trace similarity metrics suitable indicators for the discriminatory power of RNNs (*i.e.*, sensitive to even small perturbations on inputs)?

RQ2: How sensitive are different coverage criteria for capturing erroneous behaviors of RNNs?

TABLE 3.1: Subject model information.

Subject Model	Kernel RNN		# Trainable Parameters	Acc. (%)	
	Type	State vec. shape		Train.	Test.
DeepSpeech 0.1.1	Bi-LSTM	(None, 4096)	122,740,765	-	-
DeepSpeech 0.3.0	LSTM	(None, 2048)	47,228,957	-	11.00
MNIST-LSTM	LSTM	(None, 128)	81,674	99.69	98.66
MNIST-GRU	GRU	(None, 128)	61,578	99.70	98.61

RQ3: How useful is the trace similarity-based detection algorithm for detecting adversarial samples of RNNs?

RQ4: How effective is the coverage-guided testing in achieving high coverage and generating adversarial samples of RNNs?

3.5.1 Experiment Settings

Models and Dataset. We selected four RNN-based DL models including two ASR models and two image classification models, which cover popular RNN variants (see Table 3.1). The trainable parameter size often reflects the complexity of models, and the models we select range from small-scale ones with about 60k parameters, to practical-sized ones with over 100 million parameters. The highest dimension of the RNN state vectors (Column “State vec. shape”) hits 4,096, indicating the high complexity of the models.

For the ASR tasks, we selected two versions of Mozilla pretrained DeepSpeech [55] (*i.e.*, 0.1.1 and 0.3.0) that are among the state-of-the-art open source ASR models with different types of RNN core. **DeepSpeech-0.1.1** adopts a bi-directional LSTM, and **DeepSpeech-0.3.0** uses a one-directional LSTM. The state vectors of both models are with a high dimension of 64-bit floating point type. For the image classification task, we followed the instructions and trained two RNN-based classifiers on MNIST dataset (*i.e.*, **MNIST-LSTM**, **MNIST-GRU**) that achieve competitive accuracy. These two models are relatively lightweight, whose internal states are conveyed via 128-dimensional vectors of 32-bit floating points.

Abstract Model Construction. For ASR models, we use *Common Voice* [93] training dataset to perform the profiling, which is used for the training of the DeepSpeech models. Overall, there are 193,284 audios in the dataset. Each sample is processed by both models to collect the state traces. As for the image classification models, we use the official MNIST training dataset that contains 60,000 images.

For the PCA transformation, due to the huge size of state vectors for ASR models, we randomly selected 20% of state vectors to fit the PCA model, and use it for all further analysis. For image models, we use all of the state vectors. The model abstraction parameters k and m can be configured to generate DTMC models with different granularity. Table 3.2 summarizes 13 different configurations we evaluated, as well as the number of abstract states and transitions in each obtained DTMC model. Note that we use (k, m) to represent a configuration with k dimensions and m partitions.

Data Preparation. For a comprehensive evaluation of the proposed metrics and coverage criteria, we prepared three types of samples: 1) original benign samples from the test data, 2) perturbed samples which are generated by a slight perturbation on the original benign samples, and 3) adversarial samples from the original benign samples. For ASR models, we use *word error rate* (WER) to measure the inference precision. In particular, benign samples are with WER of zero. The perturbed samples would have a relatively smaller WER while the targeted adversarial samples have a larger WER. For image classification tasks, the perturbed samples we generate are with slight perturbations but still remain benign.

Initially, we randomly selected 100 benign audios and 100 benign images separately from their test datasets, from which we generate the perturbed and adversarial samples. For each ASR model, we generate 10,000 perturbed audios from original benign ones with existing audio data augmentation techniques [94] (*i.e.*, speed and volume adjustment, low/high-frequency filtering, noise blending). Finally, we only successfully generate adversarial samples for DeepSpeech-0.1.1

TABLE 3.2: Abstract model details under different configurations for each studied RNN.

Config.	DeepSpeech-0.1.1		DeepSpeech-0.3.0		MNIST-LSTM		MNIST-GRU	
(k, m)	# St.	# Trans.	# St.	# Trans.	# St.	# Trans.	# St.	# Trans.
(2, 5)	26	187	27	159	28	162	28	192
(2, 10)	88	1,067	86	755	93	745	92	1,203
(2, 20)	325	8,185	308	4,958	340	4,531	334	8,740
(2, 40)	1,221	74,936	1,154	38,653	1,267	27,657	1,241	57,506
(2, 80)	4,700	733,007	4,397	327,480	4,719	105,529	4,656	167,640
(3, 5)	107	1,416	96	1,269	109	938	109	1,188
(3, 10)	595	13,643	533	10,836	632	6,186	606	9,973
(3, 20)	3,811	179,925	3,291	128,144	3,806	39,148	3,713	72,989
(3, 40)	26,154	2,480,927	21,970	1,708,455	21,458	128,225	22,829	183,478
(3, 80)	183,724	18,408,782	145,602	15,232,230	78,259	200,567	101,417	218,765
(6, 10)	65,550	2,567,944	55,055	2,040,096	24,276	96,749	34,423	139,595
(6, 20)	1,373,236	23,057,145	1,108,832	21,164,285	110,325	185,223	149,801	209,887
(6, 40)	15,804,002	38,324,511	14,258,035	37,594,508	194,765	213,978	215,680	223,505

because there exists a compatibility issue between DeepSpeech-0.3.0 and the adversarial attack tools [10] we used. It is worth noting that the generation of targeted adversarial audios is rather computationally intensive and time-consuming. To be specific, we select the 11 commands [95] as the targets and generate 1,100 (100 seeds $\times$ 11 targets) adversarial audios, which took about 12 days in total on 4 GPUs (*i.e.*, 48 days V100 GPU time). For each MNIST model, we also generate 10,000 benign perturbed samples with existing image transformation techniques [15] (*i.e.*, image contrast, brightness, translation, scaling, shearing, rotation and add white noise). Besides, we generate 10,000 adversarial samples with each state-of-the-art attack tool, including, *FGSM* [96], *BIM* [97] and *DeepFool* [98].

For both audio and image case, we set conservative parameters for transformation so that the perturbation on original samples is slight and imperceptible. Note that all the adversarial samples are also with minimal perturbations and not perceptible by human.

Coverage Criteria Instances. For the n -SBCov criteria, we empirically study two instances with $n = 3$ and $n = 6$, denoted as 3-SBCov and 6-SBCov, respectively. For the WSCov and WTCov, besides the default weight function, which assigns larger weights to states or transitions with high visiting frequency during profiling, we introduce another weight function to assign smaller weights to more frequently visited states and transitions by inverting the original weights. We use WSCov and

TABLE 3.3: Correlation of trace similarities and prediction diff.

Config.	DeepSpeech-0.1.1		DeepSpeech-0.3.0		MNIST-LSTM		MNIST-GRU	
(k, m)	$\rho(st.)$	$\rho(tr.)$	$\rho(st.)$	$\rho(tr.)$	$U(st.)$	$U(tr.)$	$U(st.)$	$U(tr.)$
(2, 5)	-0.25	-0.28	-0.36	-0.44	2.99.E+08	2.79.E+08	2.70.E+08	2.62.E+08
(2, 10)	-0.27	-0.35	-0.44	-0.50	3.03.E+08	2.76.E+08	2.53.E+08	2.48.E+08
(2, 20)	-0.35	-0.38	-0.46	-0.49	2.84.E+08	2.55.E+08	2.49.E+08	2.33.E+08
(2, 40)	-0.37	-0.26	-0.43	-0.47	2.50.E+08	2.30.E+08	2.29.E+08	2.05.E+08
(2, 80)	-0.30	-0.16	-0.40	-0.36	2.30.E+08	2.20.E+08	2.04.E+08	1.92.E+08
(3, 5)	-0.34	-0.38	-0.47	-0.52	3.11.E+08	2.92.E+08	2.89.E+08	2.74.E+08
(3, 10)	-0.39	-0.39	-0.52	-0.50	3.04.E+08	2.64.E+08	2.73.E+08	2.45.E+08
(3, 20)	-0.38	-0.34	-0.51	-0.53	2.74.E+08	2.35.E+08	2.44.E+08	2.09.E+08
(3, 40)	-0.31	-0.15	-0.51	-0.43	2.35.E+08	2.14.E+08	2.05.E+08	1.82.E+08
(3, 80)	-0.23	-0.09	-0.41	-0.25	2.17.E+08	2.11.E+08	1.93.E+08	1.89.E+08
(6, 10)	-0.54	-0.47	-0.62	-0.59	2.65.E+08	2.18.E+08	2.23.E+08	1.95.E+08
(6, 20)	-0.40	-0.24	-0.59	-0.43	2.25.E+08	2.14.E+08	1.89.E+08	1.90.E+08
(6, 40)	-0.06	-0.06	-0.40	-0.13	2.14.E+08	2.09.E+08	1.89.E+08	1.89.E+08

WTCov to denote the criteria with default weight function, and refer WSCov' and WTCov' to the inverted ones. The criteria allow observing how the test data cover states and transitions that have high/low visiting frequency.

All the experiments were run on a server with the Ubuntu 16.04 system with 28-core 2.0GHz Xeon CPU, 196 GB RAM and 4 NVIDIA Tesla V100 16G GPUs.

3.5.2 RQ1: Trace Similarity

Setup. We perform a statistical analysis on the correlation between the trace similarity and the prediction difference over the slightly perturbed samples and their original benign samples. The difference is difficult to capture because the sample and its slightly perturbed counterpart are perceived almost the same from human perceptions. We compute the prediction difference of ASR with the word-level Levenshtein distance [99] of their transcripts, and the prediction difference in image classification by checking whether they belong to different classes.

For audios, we use the 10,000 perturbed samples, which are with various Levenshtein distances compared with their original seeds. For the image case, we take the 10,000 perturbed samples and also randomly select another 10,000 samples from all the generated adversarial samples. This is to include both correctly and wrongly inferring perturbed images. For the statistical analysis, we use Spearman

rank-order correlation [100] (denoted as ρ), to analyze the monotonic association between two variables, for the ASR models; and we use Mann-Whitney U test [101] (denoted as U), to check the binary association, for MNIST models.

Results. Table 3.3 shows the results of the correlation between trace similarity and prediction difference measured over the perturbed data, where Column $\rho(st.)$ and Column $U(st.)$ represent the results of STSIM; and Column $\rho(tr.)$ and Column $U(tr.)$ represent the results of TTSIM. The best two results of each column are highlighted in bold font. All reported correlations are statistically significant (with $p < 0.01$). Negative association of Spearman correlation indicates that the larger the similarity metrics, the less different the predicted transcripts would be. For MNIST models, the Mann-Whitney U test results indicate that when measuring the trace similarity compared with the original benign samples, perturbed samples obtain significantly larger values than adversarial ones.

Answer to RQ1: Both state- and transition-level trace similarity metrics are capable of capturing the prediction difference even for slightly perturbed samples. Thus, trace similarity could be useful for detecting adversarial samples (See RQ4).

3.5.3 RQ2. Coverage Criteria

Setup. In this experiment, we evaluate the sensitivity of the proposed coverage criteria to adversarial samples. The abstraction configurations used in RQ2 are selected based on the RQ1 results. In order to better differentiate various coverage criteria, three configurations were selected from RQ1, which are most sensitive to minor perturbations according to either STSIM or TTSIM. We compare the coverage results of the 100 original benign samples (denote as O), and the coverage achieved by including perturbed samples (denoted as $O+P$) or adversarial samples (denoted as $O+A$) (see Column *Data* of Table 3.4). Note that the perturbed samples are not included for the MNIST models because they are all benign. For DeepSpeech-0.1.1, the number of adversarial samples is 1,100, and we also select

TABLE 3.4: Coverage criteria sensitivity to the slightly perturbed samples and adversarial samples.

Sub. mod.	Conf. (k, m)	Data	State (%)										Transition (%)					
			BSCov		WSCov		WSCov'		3-SBCov		6-SBCov		BTCov		WTCov		WTCov'	
DS1	(3, 10)	O	67.9		99.7		3.0		0.0		0.0		23.5		95.5		23.5	
		$O+P$	74.5	10	99.9	0	3.2	7	0.0	0.0	0.0	0.0	37.9	61	98.7	3	37.9	61
		$O+A$	74.1	9	99.9	0	3.2	7	0.0	0.0	0.0	0.0	44.0	87	98.6	3	44.0	87
	(6, 10)	O	13.7		81.1		0.4		0.0		0.0		0.8		29.6		0.8	
		$O+P$	26.0	90	92.9	15	0.7	75	0.0	0.0	0.0	0.0	3.4	325	49.4	67	3.4	325
		$O+A$	35.4	158	94.0	16	0.9	125	0.1	0.1	0.0	0.0	6.4	700	53.0	79	6.4	700
	(6, 20)	O	1.7		22.9		0.1		0.0		0.0		0.1		5.1		0.1	
		$O+P$	7.2	324	47.1	106	0.4	300	0.0	0.0	0.0	0.0	0.4	300	9.1	78	0.4	300
		$O+A$	13.9	718	51.8	126	0.8	700	0.2	0.2	0.1	0.1	0.4	300	6.2	22	0.4	300
ML	(2, 5)	O	85.7		100.0		14.2		0.0		0.0		57.1		99.0		56.8	
		$O+A$	85.7	0	100.0	0	14.2	0	0.0	0.0	0.0	0.0	85.3	49	100.0	1	85.2	50
	(3, 5)	O	71.6		99.4		8.2		0.0		0.0		32.1		95.6		32.0	
		$O+A$	89.0	24	99.9	1	10.2	24	0.4	0.4	0.2	0.2	76.7	139	99.4	4	75.8	137
	(3, 10)	O	54.6		96.4		5.6		0.0		0.0		15.1		77.8		15.1	
		$O+A$	81.8	50	99.4	3	8.2	46	1.5	1.5	0.7	0.7	60.3	299	95.0	22	60.0	297

* The coverage increase ratio/value w.r.t. its seed group are marked with grey background.

the same number of perturbed samples to make a fair comparison. For other models, we use all 10,000 perturbed/adversarial samples.

Results. Table 3.4 reports the coverage results using different coverage criteria on different dataset (see results of DeepSpeech-0.3.0 and MNIST-GRU on website [102]). The coverage increase ratio indicates the sensitivity of the coverage criteria to adversarial samples. We observed that finer-grained abstract models tend to have larger coverage increase ratio. This is because finer-grained state and transition information is more likely to distinguish adversarial samples from the benign samples. We also found that the sensitivities of various criteria to adversarial samples are rather different. For example, the increase ratio of WSCov/WTCov is relatively small because they mainly concern the frequently covered states and transitions (during profiling) which are often already fully covered by benign samples. In contrast, the increase ratio of WSCov'/WTCov' is larger, with a competitive performance as BSCov and BTCov, indicating rarely visited states and transitions tend to be covered by adversarial/perturbed samples. For the n -SBCov criteria, we present the *increase value* instead the increase ratio as the initial criteria are zero. The increase is not quite significant, as these states are really hard to cover even by adversarial samples. Furthermore, for DeepSpeech-0.1.1,

TABLE 3.5: AUROC results (%) of trace similarity based adversarial detection by configurations.

Config.	DeepSpeech-0.1.1				MNIST-LSTM			
(k, m)	STSIM	TTSIM	FSGM		BIM		DeepFool	
			STSIM	TTSIM	STSIM	TTSIM	STSIM	TTSIM
(2, 5)	49.78	67.67	77.97	79.8	74.61	74.85	75.11	73.55
(2, 10)	59.52	84.73	83.13	84.29	80.81	79.91	78.92	80.1
(2, 20)	80.90	81.24	83.37	84.71	80.42	80.42	79.09	80.27
(2, 40)	81.00	50.00	79.95	88.01	77.9	80.77	76.82	81.86
(2, 80)	70.25	50.00	90.54	96.55	82.38	92.12	84.43	92.46
(3, 5)	69.41	85.40	86.95	85.31	84.71	83.25	82.42	80.15
(3, 10)	89.26	85.19	90.05	89.74	86.18	85.28	85.23	83.78
(3, 20)	85.25	50.00	89.4	92.62	84.63	84.72	83.53	85.46
(3, 40)	50.49	50.00	90.56	93.79	85.97	86.62	84.05	87.47
(3, 80)	50.00	50.00	96.63	93.64	92.97	89.75	93.44	90.43
(6, 10)	75.82	50.00	89.81	86.68	85.18	78.04	84.95	80.59
(6, 20)	50.00	50.00	86.94	84.53	79.45	72.64	82.54	78.99
(6, 40)	50.00	50.00	85.94	92.17	78.64	85.08	79.96	87.79

we find that the coverage criteria of $O+A$ are generally higher than those of $O+P$. This is possibly due to the larger average WER of the adversarial audios.

Answer to RQ2: The test coverage criteria are more sensitive with finer-grained abstraction. All proposed coverage criteria are sensitive to erroneous behaviors in adversarial samples, among which, BSCov, and BTCov are the most sensitive ones.

3.5.4 RQ3. Adversarial Sample Detection

Setup. This section evaluates *DeepStellar* for adversarial sample detection on three models, namely, DeepSpeech-0.1.1, MNIST-LSTM, and MNIST-GRU. We first prepared the benign, adversarial, and reference samples (refer to B , A and R in Algorithm 1) to train the linear regression classifier. For the ASR model, we randomly selected 1,100 benign samples (800 for training and 300 for testing) from the test dataset, to make an equal number as the 1,100 generated adversarial samples. Specifically, the 11 target commands are divided into two sets C_1 and

C_2 , which contain 8 and 3 commands, respectively. The adversarial samples whose prediction results belong to C_1 are used as training data (*i.e.*, 100×8), while the other as test data (*i.e.*, 100×3). The reference samples are constructed by re-transcribing all the prediction results of both adversarial and benign samples to audios with the Google cloud text-to-speech [90]. For MNIST models, we take the 9,000 benign samples from the test data of MNIST and generate 9,000 adversarial samples with each of the three approaches, *i.e.*, *FGSM*, *BIM* and *DeepFool*, of which 70% are used for training and 30% are used for testing. For reference samples, we randomly selected 50 samples from the training data of MNIST for each of the 10 categories (*i.e.*, 50×10). With the constructed dataset, we trained a classifier for each model to detect adversarial samples.

Results. Table 3.5 shows the AUROC [103] results of adversarial sample detection using different trace similarity metrics (*i.e.*, STSIM and TTSIM) and DTMC models with different configurations (Column *Config.*). The best two results of each model are highlighted, *e.g.*, 89.26% for DeepSpeech-0.1.1, 96.63% for MNIST-LSTM, and 96.63% for MNIST-GRU (see website [102]). The best results indicate that state-based trace similarity is a bit more effective than transition-based one in many cases. The results of MNIST models on detecting attacks generated by different tools show that our algorithm is robustness to a wide range of attacks, respectively with accuracy of 97%, 93%, and 93%. Furthermore, the results under DTMC models with different configurations vary largely. With finer-grained model, the result is not necessarily better. Overall, the results confirm that the trace similarity-based method is effective for adversarial sample detection under carefully selected abstraction configurations, with more than 89% prediction accuracy.

Answer to RQ3: Similarity metric based method is useful for adversarial sample detection. The detection accuracy varies under different metrics and model configurations.

TABLE 3.6: Coverage and unique adversarial samples detected.

Criteria (%) /Crash (#)	MNIST-LSTM					MNIST-GRU				
	Seed	S-Guid.	T-Guid.	Ran.	DeepTest	Seed	S-Guid.	T-Guid.	Ran.	DeepTest
BSCov	54.59	97.78	97.78	86.23	65.35	63.20	95.87	96.04	88.78	68.98
WSCov	96.44	99.99	99.99	99.66	98.04	97.03	99.98	99.98	99.80	98.05
WSCov'	5.56	9.97	9.97	8.79	6.66	3.83	5.82	5.83	5.38	4.18
3-SBCov	0.00	1.50	2.00	1.86	0.07	0.00	0.78	1.24	0.85	0
6-SBCov	0.00	0.56	0.75	0.69	0.03	0.00	0.29	0.46	0.32	0
BTCov	15.13	53.43	96.43	73.88	26.23	14.42	42.80	93.89	71.82	21.32
WTCov	77.80	94.81	99.90	98.02	85.12	63.40	88.78	99.69	96.95	72.34
WTCov'	15.12	53.43	96.43	73.88	26.22	14.41	42.80	93.89	71.81	21.31
#Unique Cra.	-	87,596	41,614	2,219	300	-	69,777	35,228	19,738	244

* The last row presents the number of unique crashes discovered in each experiment.

3.5.5 RQ4. Coverage-guided Testing

Setup. We use the prepared 100 original benign samples as the initial seeds, which are correctly predicted by both MNIST-LSTM and MNIST-GRU. Based on the results of RQ2, we use the fined-grained configuration (3,10) for constructing the DTMC models, and select BSCov and BTCov as the testing guidance. Finally, we implement two testing strategies, *i.e.*, *S-Guid.* and *T-Guid.*. To further demonstrate the usefulness of the coverage guidance, we include random testing without coverage guidance and *DeepTest* [15], a neuron coverage guided testing tool for unrolled RNNs, as baseline approaches for comparison. Each testing configuration was run for 6 hours, upon which the studied coverage criteria tend to saturate. To counter the randomness of testing tool, each configuration is repeated 5 times and averaged results are reported.

Results. Table 3.6 summarizes the obtained coverage results for different coverage criteria and the unique adversarial samples detected with the testing tools. The first column lists the studied coverage criteria. Column *Seed* represents the coverage of the initial seeds. Columns *S-Guid.*, *T-Guid.*, *Random* and *DeepTest* are the coverage by different testing strategies. We can observe that all of the studied strategies improve the coverage to some extent. Transition coverage-guided strategy outperforms the other two strategies in achieving higher coverage under all criteria. Furthermore, state coverage-guided strategy is often more effective in generating adversarial samples although it does not obtain the highest coverage. The overall

results indicate that covering more new states could be potentially helpful in generating adversarial samples.

Answer to RQ4: The coverage-guided testing is generally useful in terms of achieving higher coverage and guiding adversarial sample exploration. Among the three strategies, transition coverage-guided method achieves higher coverage, while state coverage-guided method uncovers more unique adversarial samples.

3.5.6 Threats to Validity

We summarize factors that could affect the validity of our study. A major threat is related to the abstract model configuration settings. There could be many possible configurations in the abstract model. Due to the computation resource constraint, we tried our best to experiment with as many settings as possible. Even though our results may still not generalize beyond the considered settings.

The subject model selection could be another threat to generalizability. We mitigate this by choosing models with diverse complexities and different application domains, covering both simple cases and industrial-grade ASR applications. In addition, for the ASR model, the training sets for the adversarial detection classifier is relatively small, which might affect the the performance of the detection. Adopting a larger training set may help obtain even better detection performance, which we leave as future work. Furthermore, randomness is a threat in both sampling, testing across our studied research questions. To counteract this, we repeat the same setting for all experiments five times and average the results.

3.6 Chapter Summary

Vulnerabilities of DL systems are threatening the trust and mass adoption of these technologies. As such, much research efforts have been focused on the testing of feed forward networks to ensure their reliability and robustness. This work initiates the first step towards the quantitative analysis of stateful DL systems. We model an

RNN as an abstract model, based on which a set of similarity metrics and coverage criteria are proposed. We demonstrated the usefulness of the proposed models and quantitative measures on RNNs testing and adversarial sample detection. Our long term goal is to provide quality assurance for the DL system life-cycle [104].

Chapter 4

Model-based Robustness Analysis of RNN

Robustness analysis is a method for evaluating the capability of a DNN in tolerating input perturbations or noises that either naturally occur in the physical environment or intentionally generated for specific purposes (*e.g.*, adversarial attacks). In general, numerical scores can be assigned to indicate the robustness of a DNN model for a certain input. For example, given an input at runtime, robustness analysis enables to decide whether the decision made by a DNN is consistent and reliable when facing with noises in practice. If the analysis on robustness could be obtained at real-time, it offers the system a chance to in timely roll back to the classical solutions when the system becomes unrobust.

Up to present, the progress of DNN robustness analysis mainly falls into the following three categories: (1) robustness estimation by adversarial attacks [12, 96, 98], (2) robustness evaluation by testing [14, 15, 70], and (3) robustness guarantee by verification [63, 105] which aims to prove that the DNN is robust under a certain level of perturbations. While adversarial attacks consider generating any human imperceptible input perturbation to manipulate a DNN’s decision (*e.g.*, often gradient-based), testing-based methods focus more on simulating realistic noises and transformations that might occur in physical environment. The difficulty of generating examples that lead to incorrect predication results, either by adversarial

attack or testing, reflects the robustness of a model, *i.e.*, the more robust the model is, often the harder it is to generate such examples. On the other hand, the verification techniques attempt to prove the absence of these input examples under a certain constraint (*e.g.*, distance).

The existing methods face challenges in terms of precision, efficiency, applicability and scalability, especially to handle a large system with real-time requirement. In particular, the attacking- and testing-based techniques are often incomplete and the exploration of robustness is often unsystematic. To perform thorough evaluation of the robustness of a single input, these techniques often have to generate quite a few input examples, which is expensive and not applicable to real-time applications. While the existing robustness analysis techniques are mainly applicable only to feed-forward neural networks (FNNs), little progress has been made so far on the robustness analysis of recurrent neural networks (RNNs) that especially exceeds for sequential inputs processing (*e.g.*, natural languages, speech recognition). This is possibly due to the technical challenges brought by the significantly increased complexity of RNNs. Different from FNNs, the robustness analysis of RNNs has to be performed not only to the input as a whole, but also to each individual segment of the input sequence (*e.g.*, different words in a text input). To the best of our knowledge, POPQORN [106] is the first work on the robustness analysis of RNNs. It assumes continuous input domains and relies on expensive bound calculations through layers (*i.e.*, after unrolling RNNs). This makes it inapplicable to discrete domain input, *e.g.*, natural language processing, as well as real-time processing, *e.g.*, machine translation, speech recognition and cyber-physical system perception.

To address this, in this chapter, we propose a model-based approach, named MARBLE, for evaluating the robustness of RNNs effectively. The aim is to build a model offline and enable the light-weight robustness estimation online for real-time applications. To overcome the performance challenges and avoid performing many perturbations at runtime, we construct a robustness-aware abstract model for RNN to capture the behaviors of an RNN given various inputs in a compact form. Due to the stateful nature of RNNs, the output produced at each step is affected by both the input element and the context of the element. For example, in sentiment

analysis, the positive/negative opinion predicted at certain word within a sentence is also affected by words before and after it. Our abstract model maintains previously seen states and transitions among states, which are essential in characterizing how different inputs are handled under various contexts.

The contextual information can also be useful in estimating robustness. Intuitively, under a similar context, the robustness over the same input element is expected to be similar as well. For instance, when processing two reviews, “I love this movie” and “I love that movie”, an RNN often maintains very similar contexts before processing the final word “movie”. With this insight, we compute robustness measures for training inputs and encode this information into the abstract model, which is then used to estimate the robustness of new (previously unseen) test inputs. This makes real-time identification of unreliable predictions possible and could improve the overall quality of the decision making process.

The abstract models we adopt are in the form of a labeled Markov Decision Process (MDP). We also propose an refinement algorithm to iteratively improve its precision, if the calculated robustness from the abstract model is inconsistent with the robustness measures from dynamic input perturbations. The refinement process continues until the robustness estimation error is reduced under a threshold. We applied MARBLE to quantify the robustness of two RNNs which are trained for news title classification and the sentiment analysis, respectively. The results show that MARBLE is more effective in refining the abstract model than random strategy (*e.g.*, an average performance increase of 32.6%). Compared with the state-of-the-art technique POPQORN, MARBLE can quantify the robustness more efficiently (*e.g.*, the average time cost for an input is 0.02 seconds while the baseline POPQORN needs 52.8 minutes) and more precisely (*i.e.*, in achieving higher attack success rate on the least robust inputs and lower attack success rate on the most robust inputs).

The main contributions of this chapter are three-folds:

- We propose to abstract RNN as an MDP model to perform a quantitative robustness evaluation of an RNN that enables the analysis on both the whole

input and partial fragments of the input.

- We propose a mutation-based refinement technique to continuously refine the MDP for robustness estimation of an RNN.
- We further perform an in-depth evaluation to demonstrate the effectiveness of MARBLE by applying it to attack RNNs on two popular datasets.

The rest of this chapter is organized as follows. § 4.1 formulates the robustness quantification problem of RNN models and provides necessary background information. § 4.2 presents a customized abstraction of RNN based on labeled MDP for efficient robustness estimation. § 4.3 describes the model refinement algorithm in details. § 4.4 describes our implementation and evaluates its efficiency and effectiveness. § 4.5 concludes and outlines possible future directions.

4.1 Problem Formulation

This section provides the necessary background, and defines formal representations of RNN as well as its behaviors. We formulate the problem of quantifying the robustness of an RNN model, and overview our proposed solution.

4.1.1 Recurrent Neural Networks and Their Traces

Following [80], we represent an RNN abstractly as follows.

Definition 4.1 (Recurrent Neural Networks (RNN)). An RNN is a tuple $\mathcal{R} = (\mathcal{X}, \mathcal{S}, \mathcal{O}, f)$ such that $\mathcal{X}$, $\mathcal{S}$, and $\mathcal{O}$ denote the domains of the inputs, hidden states, and outputs, respectively, and f is a differentiable parameterized function. $\mathcal{R}$ takes as input a sequence $\mathbf{x} \in \mathcal{X}^n$ of length n , maintains a sequence of hidden states $\mathbf{s} \in \mathcal{S}^{n+1}$ of length $n + 1$, and applies the function f on each state and input pair (s_i, x_i) to produce an output sequence $\mathbf{y} \in \mathcal{O}^n$ such that $(s_{i+1}, y_i) = f(s_i, x_i)$, where x_i , s_i , and y_i denote the i -th element of the respective sequences.

FIG. 4.1: FST example.

More specifically, the hidden states $\mathcal{S} \subseteq \mathbb{R}^m$ are m -dimensional real number vectors and the initial state s_0 is a vector of m zeros. We use s_i^d to denote the d -th dimension of the state vector s_i , where $d \in [0, m)$.

Given an RNN, each input sequence $\mathbf{x}$ induces a finite sequence of state transitions, which forms a *trace* denoted by $\mathbf{t}(\mathbf{x})$. The i -th element of a trace, denoted by $t_i(\mathbf{x})$, is the transition from s_i to s_{i+1} after accepting an input element x_i and producing an output y_i . The traces of an RNN can be represented compactly as a Finite State Transducer (FST) [81], which is formally defined as follows.

Definition 4.2 (Finite State Transducer (FST)). Given an RNN $\mathcal{R} = (\mathcal{X}, \mathcal{S}, \mathcal{O}, f)$, we define its traces induced by a set of inputs $D \in 2^{\mathcal{X}^n}$ as a finite state transducer, $\mathcal{T}_{\mathcal{R}}(D) = (\mathcal{X}, \mathcal{S}, \mathcal{O}, s_0, F, \delta)$, where $\mathcal{S}$ is a non-empty finite set of states, $\mathcal{X}$ is the input alphabet, $\mathcal{O}$ is the output alphabet, $s_0 \in \mathcal{S}$ is the initial state, $F \subseteq \mathcal{S}$ is the set of final states, and $\delta \subseteq \mathcal{S} \times \mathcal{X} \times \mathcal{O} \times \mathcal{S}$ is the transition relation.

Example 4.1. We illustrate the FST with an example as shown in Fig. 4.1. Given a set of input sequences, $D = \{x_0x_1x_2, x_3x_4x_5\}$, two traces $\mathbf{t}(x_0x_1x_2)$ and $\mathbf{t}(x_3x_4x_5)$ are induced. They share the same initial state s_0 and each consists of three transitions. Transitions are represented by arrows, originating from one state and transit to another state, labeling with its input and output pair. For the transition from s_0 to s_1 , the label “ $x_1 : y_1$ ” denotes that this transition is induced by the input element x_1 and emits the output y_1 . For the FST defined over the two traces, the set of input alphabet is $\mathcal{X} = \{x_0, x_1, x_2, x_3, x_4, x_5\}$, the set of states is $\mathcal{S} = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6\}$, the set of outputs is $\mathcal{O} = \{y_0, y_1\}$, the initial state is s_0 , the set of final states is $F = \{s_3, s_6\}$, and the transition relation δ defines the set of six transitions $\{(s_0, x_0, y_0, s_1), (s_1, x_1, y_1, s_2), (s_2, x_2, y_0, s_3), (s_0, x_3, y_0, s_4), (s_4, x_4, y_0, s_5), (s_5, x_5, y_0, s_6)\}$.

4.1.2 Point-wise Robustness of an RNN

The quantification of CNN robustness has been widely studied in the literature [58, 59], whereas little work has been done to address the robustness of RNN models. Unlike CNN which takes an input as a whole, RNN consumes a sequential input element-by-element, and perturbations on each input element may cause different levels of impacts to the model outputs. To measure the robustness of an RNN model w.r.t. an input, we allow the input to be perturbed within a certain range, and observe their impacts on the induced traces.

To precisely quantify the point-wise robustness, we first specify how input elements are to be perturbed, formally defined by a *point-wise mutation function*. Note that, for a given set Q , the probability distribution over Q is a function $d : Q \mapsto [0, 1]$ such that $\sum_{q \in Q} d(q) = 1$. And we denote the set of all probability distributions over Q as $\text{Dist}(Q)$.

Definition 4.3 (point-wise Mutation Function). A point-wise mutation function $\mu \subseteq \mathcal{X} \times \text{Dist}(\mathcal{X})$ maps an input element $x \in \mathcal{X}$ to a probability distribution of the possible mutants $x' \in \mathcal{X}$, such that $\mu(x, x')$ is the probability of mutating x into x' .

The point-wise mutation function can be used to describe various types of perturbations to the inputs. For example, p -norm ball [62, 106] is often used in the literature to describe perturbations happened uniformly within a sphere around the given input. In general, the robustness of a model can be examined against different real-world attack patterns by simulating the ranges and frequencies of the potential mutations appeared in practice.

We define the *0-step point-wise robustness* of an RNN at an input element x_i as the likelihood that the immediate next transition t_i induced by x_i stays unchanged after some *perturbation* is applied to x_i . This definition can be further generalized to observe the impacts of the perturbation only after a certain number of transitions, instead of the immediate next one.

Definition 4.4 (k -Step Point-wise Robustness). Let $\mathcal{R}$ be an RNN and $\mathbf{x} \in \mathcal{X}^n$ be an input of length n , respectively. Let x_i be the input element at the i -th position

of $\mathbf{x}$, where $i < n$. Let μ be a point-wise mutation function generating a new input $\mathbf{x}[x_i \rightarrow x'_i]$ by mutating x_i . The k -step point-wise robustness of $\mathcal{R}$ w.r.t. $\mathbf{x}$ at the i -th position against perturbations defined by μ , denoted by $\gamma^k(\mathcal{R}, \mathbf{x}, i, \mu)$, is given as the probability that the trace induced by $\mathbf{x}'$ produces the same output at the $(i + k)$ -th transition, where $(i + k) \leq n$. More formally,

$$\gamma^k(\mathcal{R}, \mathbf{x}, i, \mu) = \sum_{x'_i \in \mathcal{X} \cdot t_{i+k}(\mathbf{x}) \stackrel{o}{=} t_{i+k}(\mathbf{x}[x_i \rightarrow x'_i])} \mu(x_i, x'_i),$$

where $\stackrel{o}{=}$ denotes the equality of outputs for both sides.

The point-wise robustness definition is parameterized by the number of steps, after which the impacts of the perturbation to the model output are observed. Specifically, $\mathbf{t}(\mathbf{x})$ and $\mathbf{t}(\mathbf{x}[x_i \rightarrow x'_i])$ represent the two traces induced by $\mathbf{x}$, before and after x_i is mutated, respectively. The k -step point-wise robustness only observes the outputs produced at the $(i + k)$ -th transition, denoted by $t_{i+k}(\mathbf{x})$, and ignores any intermediate states and transitions. In particular, for the 0-step point-wise robustness, we observe how the transition t_i induced by x_i is affected by perturbations over x_i . Instead of parameterizing point-wise robustness by the input sequence and position indices, we sometimes use an equivalent notation $\gamma^0(\mathcal{R}, s_i, x_i, \mu)$ such that,

$$\gamma^0(\mathcal{R}, s_i, x_i, \mu) = \sum_{x'_i \in \mathcal{X} \cdot f(s_i, x_i) \stackrel{o}{=} f(s_i, x'_i)} \mu(x_i, x'_i).$$

Example 4.2. Following Example 4.1, we briefly illustrate how the 0-step point-wise robustness is calculated with a simple example. Here we compute the robustness of $\mathcal{R}$ over the input sequence $x_0x_1x_2$ at its third input element, i.e., x_2 . For the mutation function μ , we assume that x_2 can be mutated into any element in $\{x_0, x_1, x_2\}$ with an equal probability, i.e., $\frac{1}{3}$. The perturbation of x_2 would affect the transition right after the state s_2 , and we assume the new transitions induced by the replacements are $(s_2, x_0, y_1, -)$, $(s_2, x_1, y_0, -)$, and (s_2, x_2, y_1, s_3) , respectively. Obviously, two out of the three new transitions still emit the same output y_1 as

FIG. 4.2: The workflow overview of MARBLE.

originally. Therefore, the 0-step robustness of $\mathcal{R}$ over the input sequence $x_0x_1x_2$ at x_2 is $\frac{2}{3}$.

4.1.3 Overview

Following Def. 4.4, the point-wise robustness of an RNN can be estimated through random sampling from mutations. One can randomly generate mutants according to the probabilistic distribution given by the mutation function, and observe their impacts on the model outputs directly. Yet, this sampling approach likely does not scale in practice. To get precise robustness measurements w.r.t. even a single input, the number of mutants required can be huge if the RNN model is non-trivial.

To overcome this challenge, we propose a model-based analysis technique, MARBLE, to efficiently compute the point-wise robustness of a given RNN with respect to an input element. Fig. 4.2 overviews the workflow of MARBLE. We first construct a robustness-aware abstract model by profiling a set of *mutants* generated based on the training data. Specifically, MARBLE takes in (partial) training input samples and generates mutated inputs following a point-wise mutation function. The mutation function is specific to the application domains (*e.g.*, audio, image). We then profile the RNN’s behaviors on both the training samples and their mutants, producing a set of traces of RNN internal states. We then build an abstract model, in the form of a Markov Decision Process (MDP), by abstracting the states and transitions observed in the collected traces.

Given an input, we can then compute a robustness score based on the MDP model. If the obtained score is inconsistent with the result of the dynamic *mutation*

sampling (i.e., their difference exceeds a threshold), we iteratively refine the abstract model so that the robustness can be more precisely quantified. Finally, we obtain a precise abstract model after the results of all training data from the abstract model is consistent with the *mutation sampling*. Based on the abstract model, we can analyze the robustness of the RNN against new inputs that follow a similar distribution of training data efficiently.

4.2 RNN Abstraction using Labeled MDP

As shown in Def. 4.4, the robustness of an RNN is defined with respect to a concrete input sequence at a certain position. Yet, the number of states and traces enabled during the training stage of an RNN can be huge. It is not practical, during runtime, to compute the robustness for each trace individually, with either classical mathematical estimation methods [106] or mutation testing techniques [107]. To make the computation more efficiently, we now introduce an RNN abstraction model based on labeled MDP.

4.2.1 Aggregated Pointwise Robustness

The key idea in deriving a suitable abstraction is to group the traces with similar robustness measures together, which can be used to estimate the robustness of an RNN for various inputs at various contexts in a more cost-effective manner. We define *aggregated pointwise robustness* to reflect the overall robustness of a set of input-induced traces against perturbations.

Definition 4.5 (Aggregated Pointwise Robustness). Let $\mathcal{R}$ be an RNN and $\mathcal{T}_{\mathcal{R}}(D) = (\mathcal{X}, \mathcal{S}, \mathcal{O}, s_0, F, \delta)$ be its traces induced by a set of inputs D . Let x be an input element and μ be a pointwise mutation function. Let $\tilde{S} \subseteq \mathcal{S}$ be a set of states, and $\forall s \in \tilde{S}, f(s, x)$ emits the same output. The aggregated pointwise robustness of $\mathcal{R}$ at $\tilde{S}$ w.r.t. x against perturbations defined by μ , denoted by $\Gamma(\mathcal{R}, \tilde{S}, x, \mu)$, is given as the average of the 0-step pointwise robustness of $\mathcal{R}$ w.r.t.

the states in $\tilde{S}$. More formally,

$$\Gamma(\mathcal{R}, \tilde{S}, x, \mu) = \frac{1}{|\tilde{S}|} \sum_{s \in \tilde{S}} \left(\sum_{x' \in \mathcal{X} \cdot f(s, x) \stackrel{o}{=} f(s, x')} \mu(x, x') \right). \quad (4.1)$$

The innermost sum in Equation (4.1) gives the 0-step pointwise robustness for a particular state. The aggregated pointwise robustness is essentially the average value of all pointwise robustness observations for states within a set. With this aggregation operator, we are able to represent the overall robustness of a set of states properly in a statistical manner.

4.2.2 Abstraction as Labeled MDP

Abstraction on states is essentially dividing the state space such that states with similar robustness measures are clustered. After such clustering, an abstract state may have multiple non-deterministic outgoing transitions even after receiving the same input. Markov Decision Process (MDP) [108] is widely used to describe the probabilistic decision process, and to characterize the probabilistic state transitions triggered by different input elements. Here, we use the labeled MDP to formalize the abstraction of RNN traces, and the detailed definition is given in Def. 4.6. The labeling function is used to record the robustness measure of each state and input element pair.

Definition 4.6 (Labeled Markov Decision Process). A labeled MDP, $\mathcal{M}$, is a tuple $(\hat{\mathcal{X}}, \hat{\mathcal{S}}, \hat{\mathcal{O}}, \hat{s}_0, \hat{\delta}, \eta)$ consisting of a finite set $\hat{\mathcal{S}}$ of states, a finite set $\hat{\mathcal{X}}$ of input alphabet, an initial state $\hat{s}_0$, a finite probabilistic transition relation $\hat{\delta} \subseteq \hat{\mathcal{S}} \times \hat{\mathcal{X}} \times \text{Dist}(\hat{\mathcal{S}})$, where states and inputs are in relation with distributions of successor states, and a labeling function $\eta \subseteq \hat{\mathcal{S}} \times \hat{\mathcal{X}} \times \text{Dist}(\mathcal{O})$ which maps the state and input pairs to distributions of outputs.

Now we demonstrate how to derive a labeled MDP from a set of observed traces represented as an FST. In order to represent the behaviors of an RNN in a more compact manner, we allow abstractions over the state domain, as well as the input

and output domains, and more details about the abstraction functions will be discussed in Sect. 4.3.1. Given an FST $\mathcal{T}_{\mathcal{R}}(D) = (\mathcal{X}, \mathcal{S}, \mathcal{O}, s_0, F, \delta)$ of an RNN $\mathcal{R}$ on the input set D , an input abstraction function $\lambda_I : \mathcal{X} \mapsto \hat{\mathcal{X}}$, a state abstraction function $\lambda_S : \mathcal{S} \mapsto \hat{\mathcal{S}}$ and an output abstraction function $\lambda_O : \mathcal{O} \mapsto \hat{\mathcal{O}}$, we can establish a labeled MDP $\mathcal{M} = (\hat{\mathcal{X}}, \hat{\mathcal{S}}, \hat{\mathcal{O}}, \hat{s}_0, \hat{\delta}, \eta)$ for $\mathcal{T}_{\mathcal{R}}(D)$. The input, state and output domains in $\mathcal{M}$ are abstracted from those in $\mathcal{T}_{\mathcal{R}}(D)$ with the abstraction functions. Specifically, $\hat{s}_0 = \lambda_S(s_0)$, and for each transition $(s, x, s', y) \in \delta$, it is abstracted as $(\hat{s} = \lambda_S(s), \hat{x} = \lambda_I(x), \hat{s}' = \lambda_S(s'), \hat{y} = \lambda_O(y))$ and included as an abstract transition in $\mathcal{M}$. With these abstractions, the probabilistic transition relation $\hat{\delta}$ and the labeling function η can be derived accordingly. $\hat{\delta}(\hat{s}, \hat{x}, \hat{s}')$ denotes the conditional probability of visiting $\hat{s}'$ given the current abstract state $\hat{s}$ and an abstract input element $\hat{x}$, and $\sum_{\hat{s}' \in \hat{\mathcal{S}}} \hat{\delta}(\hat{s}, \hat{x}, \hat{s}') = 1$. The transition probability is calculated as the number of concrete transitions from $\hat{s}$ to $\hat{s}'$ via $\hat{x}$ over the number of all outgoing concrete transitions from $\hat{s}$ accepting abstract input element $\hat{x}$, *i.e.*,

$$\hat{\delta}(\hat{s}, \hat{x}, \hat{s}') = \frac{|\{(s, x, s', -) \in \delta \mid s \in \hat{s} \wedge x \in \hat{x} \wedge s' \in \hat{s}'\}|}{|\{(s, x, -, -) \in \delta \mid s \in \hat{s} \wedge x \in \hat{x}\}|}.$$

The labeling function η records the 0-step aggregated pointwise robustness of $\mathcal{R}$ under a mutation function μ . Note that the robustness definition in Def. 4.5 is presented under the concrete input and output domain, and here we adjust it to the abstract counterparts. Different from Def. 4.5, given an abstract state $\hat{s}$ and an abstract input $\hat{x}$, the outputs of the corresponding concrete state and input pairs can vary. Therefore, we label the robustness under abstract states and abstract inputs as a distribution over the output domain, *i.e.*, to label the robustness for different outputs respectively. Formally, we define 0-step aggregated pointwise robustness at $\hat{s}$ over $\hat{x}$, when emitting the abstract output $\hat{y}$, as

$$\eta(\hat{s}, \hat{x}, \hat{y}) = \frac{1}{|S_{\hat{x}}|} \sum_{(s, x) \in S_{\hat{x}}} \left(\sum_{x' \in \mathcal{X} \cdot f(s, x') \in (-, \hat{y})} \mu(x, x') \right), \quad (4.2)$$

where $S_{\hat{x}} = \{(s, x) \mid \exists (s, x, -, -) \in \delta \text{ for } s \in \hat{s}, x \in \hat{x}\}$ is the set of state-and-input pairs which source from concrete states in $\hat{s}$ and accept concrete input elements in $\hat{x}$. Apparently, the inner sum calculates the robustness at s over x , while relaxing

FIG. 4.3: A set of concrete traces and their corresponding abstract state transition model.

the constraint on the expected output to be within the abstract output $\hat{y}$. Then, we take the average value for robustness observations over all pairs inside $S_{\hat{x}}$.

In practice, when estimating the labeling function $\eta(\hat{s}, \hat{x}, \hat{y})$, we perform as many input mutations (following the pointwise mutation function) as possible and obtain the output values after mutations. Here, for $(s, x) \in S_{\hat{x}}$, we do not necessarily get an output which happens to be within $\hat{y}$. Instead of calculating the pointwise robustness directly, we can record the probability distribution of emitting different outputs when mutating x at s instead, which is denoted by $\gamma^0(s, x)$. Note that we omit the parameters $\mathcal{R}$ and μ when they are clear from the context. Later, we can make use of this probability distribution to estimate the aggregated pointwise robustness in the abstract domain.

Example 4.3. Fig. 4.3 illustrates the abstract model with a simple example. Fig. 4.3(a) depicts three concrete traces, namely, t_1 , t_2 , and t_3 , where states and transitions are shown as dots and directed edges, respectively. We also highlight some input and output pairs on the edges. Fig. 4.3(b) shows the MDP abstraction of the concrete traces. The abstraction is carried out on two levels, namely, the input/output abstraction and the state/transition abstraction, which is useful especially when dealing with data from continuous space.

For the input/output abstraction, multiple concrete inputs/outputs can be represented by the same abstract inputs/outputs according to the abstraction functions. For instance, x_1 and x_2 are abstracted as $\hat{x}$, and y_1 , y_2 and y_3 are abstracted as $\hat{y}$. The full abstraction function is illustrated at the bottom of Fig. 4.3(a).

The grids drawn in dashed lines in Fig. 4.3(a) represent the abstract states, i.e., $\hat{s}_0$, $\hat{s}_1$, $\hat{s}_2$, and $\hat{s}_3$, each of which is mapped to a set of concrete states inside the corresponding grid. For each abstract state $\hat{s}$, we can then compute the transition function by deciding the set of abstract inputs accepted at $\hat{s}$ and the probabilistic distributions of the successor states under each abstract input. For example, there are four concrete transitions originated from $\hat{s}_1$, which lead to four different abstract successor states: $\hat{s}_0$, $\hat{s}_1$, $\hat{s}_2$, and $\hat{s}_3$. The abstract state $\hat{s}_1$ accepts two abstract inputs ($\hat{x}$ and $\hat{x}'$), each inducing a distribution of the successor states, i.e., $\{\delta(\hat{s}_1, \hat{x}, \hat{s}_2) = 0.5, \delta(\hat{s}_1, \hat{x}, \hat{s}_3) = 0.5\}$ and $\{\delta(\hat{s}_1, \hat{x}', \hat{s}_0) = 0.5, \delta(\hat{s}_1, \hat{x}', \hat{s}_1) = 0.5\}$, marked as labels to the corresponding transitions.

We now show how a labeling function is computed for the abstract state $\hat{s}_1$ with the abstract input $\hat{x}'$. Two concrete transitions are involved, i.e., the transitions annotated by “ $x_3 : y_3$ ” and “ $x_4 : y_4$ ”, respectively. Both transitions are originated from the abstract state $\hat{s}_1$ and accept abstract input $\hat{x}'$. Here we assume the probability distribution of emitting $\hat{y}$ and $\hat{y}'$ for these two transitions are (0.8, 0.2) and (0.4, 0.6), respectively. Note that the distribution is obtained with an assumed mutation function, and the new transitions induced by the perturbations are not included in the figure. According to Equation 4.2, we compute the aggregated pointwise robustness by taking the average over probabilities to emit $\hat{y}$ at $\hat{s}_1$ after receiving $\hat{x}'$, i.e., $\eta(\hat{s}_1, \hat{x}', \hat{y}) = \frac{0.8+0.4}{2} = 0.6$. Similarly, we also have $\eta(\hat{s}_1, \hat{x}', \hat{y}') = \frac{0.2+0.6}{2} = 0.4$.

4.2.3 k -step Aggregated Pointwise Robustness

Based on the abstracted MDP model, we define the k -step aggregated pointwise robustness. Firstly, we define traces over the MDP induced by an abstract input sequence starting from a designated abstract state. Given an MDP

$\mathcal{M} = (\hat{\mathcal{X}}, \hat{\mathcal{S}}, \hat{\mathcal{O}}, \hat{s}, \hat{\delta}, \eta)$, a start abstract state $\hat{s}_0 \in \hat{\mathcal{S}}$ and an abstract input $\hat{\mathbf{x}} \in \hat{\mathcal{X}}^n$ of length n , we denote the set of traces over $\mathcal{M}$ triggered by $\hat{\mathbf{x}}$ and starting from $\hat{s}_0$ as $\Pi(\hat{\mathbf{x}}, \hat{s}_0)$. The i -th element in a trace $\pi \in \Pi(\hat{\mathbf{x}}, \hat{s}_0)$ is the transition from $\hat{s}_i$ to $\hat{s}_{i+1}$ via $\hat{x}_i$. In particular, the 0-th transition is triggered by $\hat{x}_0$ and transits from $\hat{s}_0$ to $\hat{s}_1$. Moreover, we use $\rho(\pi)$ to denote the trace probability of π , which is the product of transit probabilities for transitions in π , formally,

$$\rho(\pi) = \prod_{i=0}^{n-1} \hat{\delta}(\hat{s}_i, \hat{x}_i, \hat{s}_{i+1}).$$

Specially, we define ϵ as the empty abstract input element, and for any abstract state, the self-transition via ϵ happens with probability 1. Hence, given ϵ as the input sequence of length 1, we have $\Pi(\epsilon, \hat{s}_0)$ contains the *only* trace with a single self-transition of $\hat{s}_0$. To facilitate the notations, we use $\mathbf{x}[i: j]$ to denote the subsequence of a sequence $\mathbf{x}$, which starts from its i -th element and ends at the $(j-1)$ -th element; if $i \geq j$, the subsequence is ϵ . Also, we use π_r to denote the reached state of its last transition. We give the definition of k -step aggregated pointwise robustness in Def. 4.7.

Definition 4.7 (k -step Aggregated Pointwise Robustness). Given an RNN $\mathcal{R}$, let $\mathcal{M} = (\hat{\mathcal{X}}, \hat{\mathcal{S}}, \hat{\mathcal{O}}, \hat{s}_0, \hat{\delta}, \eta)$ be a labeled MDP established for it. Let $\mathbf{x} \in \mathcal{X}^n$ be an input of length n , and the corresponding abstract input is $\hat{\mathbf{x}}$ with $\hat{x}_i = \lambda_I(x_i)$ for $i < n$. The k -step aggregated pointwise robustness of $\mathcal{R}$ w.r.t. $\mathbf{x}$ at the i -th ($i < n$) position can be calculated over $\mathcal{M}$ as

$$\Gamma^k(\mathcal{M}, \mathbf{x}, i) = \frac{1}{|\Pi|} \sum_{\pi \in \Pi} \rho(\pi) \eta(\pi_r, \hat{x}_{i+k}, \hat{y}),$$

where the source state of $t_i(\mathbf{x})$ is s_i , which maps to abstract state $\hat{s}_i$, the output of $t_{i+k}(\mathbf{x})$ maps to the abstract output $\hat{y}$, and $\Pi(\hat{\mathbf{x}}[i: i+k], \hat{s}_i)$ (simply noted as Π) is the set of traces over $\mathcal{M}$ initiating from $\hat{s}_i$ and triggered by the abstract input sub-sequence $\hat{\mathbf{x}}[i: i+k]$.

The above definition is consistent with our 0-step aggregated pointwise robustness defined under the abstract domain in Equation 4.2. In that case, the

sub-sequence $\hat{\mathbf{x}}[i:i+k]$ becomes ϵ , thus $\Pi(\epsilon, \hat{s}_i)$ contains only *one* trace which consists of a single self-transition of $\hat{s}_i$ via ϵ , and we have $\Gamma^k(\mathcal{M}, \mathbf{x}, i) = \eta(\hat{s}_i, \hat{x}_i, \hat{y})$.

4.3 Model Refinement Through State Clustering

With RNN abstracted as a labeled MDP, we hope to perform robustness computation over the abstraction for better efficiency. To derive an abstraction which allows for accurate estimation of the robustness as obtained from mutation testing (see Def. 4.5), we design a refinement process to reduce the estimation errors iteratively. The aim is to reduce the estimation errors from the k -step aggregated point-wise robustness and gradually approaches the ground-truth robustness reflected by mutation testing.

In the following, we assume appropriate input and output abstraction functions are predefined, and present the refinement algorithm focusing on the refinement of the state abstraction function in Sect. 4.3.2.

4.3.1 Input and Output Abstraction

For the input abstraction function, it should be aligned with the point-wise mutation function, as defined in Def. 4.3. The mutation function is also defined by users and is designed according the specific domain properties of the inputs, including whether the domain is continuous (*e.g.*, speech audios) or discrete (*e.g.*, natural languages), and which type of distance measures (*e.g.*, l_p norm distance or cosine distance) is more suitable to restrict the mutations. The aim of input abstraction is to gather similar input elements together. Here, we stipulate that concrete input elements can be mapped into the same abstract input element if they share the same mutation probability distribution. Taking an NLP application for example, words are grouped together to form synonym clusters if the cosine distance between them is within a threshold. We assume a mutation function that allows uniform mutation

sampling among all the synonyms, thus, words inside a synonym cluster share the same mutation probability distribution. For the input abstraction, words from the same synonym cluster can be mapped into the same abstract input element.

For the output abstraction function, it should be designed specifically for different types of deep learning tasks. Similarly, the abstraction on the output domain aims to gather similar outputs together, based on how tolerant the users are about the output variation. For classification tasks, identity function can be used for the output abstraction, since any classification result other than the truth label is regarded as a failure. For tasks attempting to predict a value from continuous domain, *e.g.*, the steering angle of autonomous cars, an abstraction function can be designed to map the outputs to a finite discrete domain with techniques such as predicate abstraction [109]. The predictions of such tasks are deemed as correct as long as they are within certain range of the truth label. We believe, in most cases, the output abstraction functions can be easily derived by RNN domain application developers.

4.3.2 Refinement of State Abstraction

Now we introduce an algorithm for refining the state abstraction. The overall goal of the refinement is to reduce the gap between the k -step aggregated point-wise robustness (Def. 4.7) and the individual k -step point-wise robustness (Def. 4.4) within an abstract state. From Def. 4.7, we see that the k -step aggregated point-wise robustness can be deduced from the robustness labeling function.

Def. 4.8 gives the *mean squared error* (MSE) [110] of the estimation calculated using an MDP abstraction compared with the point-wise robustness obtained by mutating individual concrete inputs. The accuracy of the MDP abstraction in measuring robustness is determined by how large the abstract states are and how concrete states are clustered. Ideally, we would like to design the state abstraction function such that states with similar robustness are clustered in an abstract state.

Definition 4.8 (Mean Squared Error). Given an RNN $\mathcal{R}$, a set of samples D , let $\mathcal{T}_{\mathcal{R}}(D) = (\mathcal{X}, \mathcal{S}, \mathcal{O}, s_0, F, \delta)$ and $\mathcal{M} = (\hat{\mathcal{X}}, \hat{\mathcal{S}}, \hat{\mathcal{O}}, \hat{s}_0, \hat{\delta}, \eta)$ be the FST and the

Algorithm 3: Refinement algorithm for a labeled MDP.

```

input :  $\mathcal{R} = (\mathcal{X}, \mathcal{S}, \mathcal{O}, f)$ : RNN,  $\mu$ : mutation function,  $\theta$ : threshold,  $D$ :
        samples
output :  $\mathcal{M}$ : labeled MDP
1 Prepare input/output/state abstraction functions  $\lambda_I, \lambda_O, \lambda_S$ ;
2 Mutation sampling count  $N$ ;
3  $\mathcal{T}_{\mathcal{R}}(D) = (\mathcal{X}, \mathcal{S}, \mathcal{O}, s_0, F, \delta)$ ; // finite state transducer of  $\mathcal{R}$ 
4 for  $\mathbf{x} \in D$  do // generate mutants and add them to  $\mathcal{T}_{\mathcal{R}}$ 
5   for  $i$  in  $[0, |\mathbf{x}|)$  do
6     for  $j$  in  $[0, N)$  do
7        $x'_i \leftarrow \text{selectMutation}(\mu, x_i)$ ;
8        $\delta \leftarrow \delta \cup t_i(\mathbf{x}[x_i \rightarrow x'_i])$ ;
9 do
10  $\mathcal{M} = (\hat{\mathcal{X}}, \hat{\mathcal{S}}, \hat{\mathcal{O}}, \hat{s}_0, \hat{\delta}, \eta) \leftarrow \text{build\_mdp}(\mathcal{T}_{\mathcal{R}}(D), \lambda_I, \lambda_O, \lambda_S)$ ;
11  $\text{refinable} \leftarrow \text{False}$ ;
12 for  $\hat{s} \in \hat{\mathcal{S}}$  do // refine each abstract state
13   for  $(\hat{s}, \hat{x}, \_) \in \hat{\delta}$  do // refine the first refinable transition
14      $B_{pw}, S_{pw} \leftarrow \emptyset, \emptyset$ ; // initialize robustness and concrete states
15     for  $s \in \hat{s}$  do
16        $\mathbf{b} \leftarrow [0]^{|\hat{\mathcal{O}}|}$ ;
17       for  $\hat{y} \in \hat{\mathcal{O}}$  do // point-wise robustness for each abstract output
18          $\mathbf{b}[\hat{y}] = \frac{|\{(s, x, \_, y) \in \delta \mid x \in \hat{x}, y \in \hat{y}\}|}{N}$ 
19        $B_{pw}, S_{pw} \leftarrow B_{pw} \cup \mathbf{b}, S_{pw} \cup s$ ;
20       if  $\frac{1}{|B_{pw}|} \sum_{b \in B_{pw}} \|\eta(\hat{s}, \hat{x}) - b\|^2 > \theta$  then // require refinement
21          $\mathcal{K} \leftarrow \text{fitKmeansCluster}(B_{pw}, \text{clusters} = 2)$ ;
22          $\mathcal{C} \leftarrow \text{fitSVMClassifier}(S_{pw}, \mathcal{K}.labels)$ ;
23          $\lambda_S \leftarrow \text{addSubAbstracter}(\lambda_S, \hat{s}, \mathcal{C})$ ;
24          $\text{refinable} \leftarrow \text{True}$ ;
25         break
26 while  $\text{refinable}$ ;
27 return  $\mathcal{M}$ ;

```

---

labeled MDP established accordingly. The mean squared error is calculated as

$$\frac{1}{|\delta|} \sum_{\hat{s} \in \hat{\mathcal{S}}} \left(\sum_{(\hat{s}, \hat{x}, \_) \in \hat{\delta}} \left(\sum_{s \in \hat{s} \wedge x \in \hat{x} \wedge (s, x, \_, \_) \in \delta} \|\gamma^0(s, x) - \eta(\hat{s}, \hat{x})\|^2 \right) \right),$$

where $s \in \hat{s}$ and $x \in \hat{x}$.

Next, we elaborate how to refine a state abstraction function in order to satisfy an MSE threshold θ , which is sketched in Algorithm 3. Given an RNN model $\mathcal{R}$, a mutation function μ , a threshold θ and a set of samples D , the algorithm produces

a labeled MDP, with which the estimated 0-step aggregated point-wise robustness of samples in D would achieve an estimation error no larger than θ . Here, the point-wise robustness computed by mutation testing with the mutation function μ is used as the ground truth reference. We assume that λ_I and λ_O are predefined by users; and λ_S is initialized such that all concrete states are mapped to a single abstract state.

Firstly, we profile the RNN model $\mathcal{R}$ with samples in D and represent the traces as an FST, $\mathcal{T}_{\mathcal{R}}(D)$ (Line 3). We then prepare N mutants for each sample and store them in D (Lines 4–8), to facilitate the calculation of reference point-wise robustness later. For each element x_i from an input, we obtain N mutants according to the mutation function μ , and include the new traces triggered by replacing x_i by x'_i into $\mathcal{T}_{\mathcal{R}}(D)$.

Lines 9–26 show the core refinement steps for the state abstraction function. In each iteration, we attempt to improve the accuracy of the abstraction function for each abstract state in $\mathcal{M}$. As the first step, a labeled MDP is computed from $\mathcal{T}_{\mathcal{R}}(D)$ using the input, output, and state abstractions functions. For each abstract state $\hat{s}$ and abstract input element $\hat{x}$ triggering a transition sourcing from $\hat{s}$ (Line 13), we examine whether the estimation error of this pair is below θ (Line 20). We calculate the mutation-based robustness at each concrete state, mapped to $\hat{s}$, over the abstract input $\hat{x}$ with Lines 14–19, and the results are stored in B_{pw} . We also keep the concrete state vectors in S_{pw} which one-to-one map to those robustness in B_{pw} . The 0-step aggregated point-wise robustness estimated for the $(\hat{s}, \hat{x})$ pair is $\eta(\hat{s}, \hat{x})$, which equals to the average of the values in B_{pw} . If the MSE of this subset of observations is greater than θ (Line 20), we make a refinement on $\hat{s}$ such that the overall MSE on D is reduced (Theorem 4.9).

We design a two-step refinement strategy to achieve this goal, to narrow the distance between the estimated aggregated robustness and every corresponding mutation testing robustness. We firstly cluster the concrete state in S_{pw} according to their mutation-based robustness (Line 21), such that states with similar robustness would be gathered together. Technologically, we leverage k-means to make the (two) clusters. However, the requirement is to make abstraction over the state

space, instead of the robustness domain, thus we need to further calculate the state abstraction which can fulfill the robustness domain clustering to the most extend. To address this problem, by treating the cluster index as the label, we employ SVC classifier $\mathcal{C}$ to approximate the decision boundary over state space (Line 22). Finally, we append $\mathcal{C}$ as a sub-abstraction function of state $\hat{s}$ and update the state abstraction function λ_S (Line 23). After examining all abstract state and input element pairs, if there exists any pair requiring a refinement, we will re-build the MDP mode $\mathcal{M}$ with the updated λ_S and continue the iteration. Otherwise, if neither of the abstract states requires further refinement, the procedure would terminate and return the refined MDP model.

Theorem 4.9 (Error Reduction). *Given an RNN $\mathcal{R}$, a set of samples D , let $\mathcal{M}$ and $\mathcal{M}'$ be labeled MDPs obtained from $\mathcal{R}$ and D , before and after the execution of the refinement step (Lines 21–25 in Algo 3). We have $MSE_{\mathcal{M}'} \leq MSE_{\mathcal{M}}$, where $MSE_{\mathcal{M}}$ and $MSE_{\mathcal{M}'}$ are the mean squared errors of $\mathcal{M}$ and $\mathcal{M}'$, respectively.*

Proof. From the MSE definition in Def. 4.8, we can see that the outer most sum iterates over all the abstract states, the middle sum iterates over all the abstract input elements accepted by the abstract state, and inner most sum iterates over all the corresponding concrete state and input element pairs. The fraction in the front divides the total number of concrete transitions, which remains constant during the execution of Algorithm 3.

Here, we focus on the inner most sum, *i.e.*, the estimation error for a specific abstract state and abstract input element pair $(\hat{s}, \hat{x})$, and prove that the squared error decreases if $\hat{s}$ is divided into two states. To make the proof succinct, we represent the 0-step pointwise robustness of the concrete state and input element pairs as a set of vectors, $X = \{\mathbf{x}_i\}_{i=1}^n$, and we define $\bar{X}$ as the mean of the set. More formally,

$$\bar{X} = \frac{1}{|X|} \sum_{\mathbf{x}_i \in X} \mathbf{x}_i.$$

The squared error of X is given as,

$$\sum_{i=1}^n \|\mathbf{x}_i - \bar{X}\|^2. \quad (\text{SE}_{\text{total}})$$

After the abstract state $\hat{s}$ is divided into two states, *e.g.*, $\hat{s}_1$ and $\hat{s}_2$, two vectors X_1 and X_2 can be obtained accordingly. We assume them as $X_1 = \{\mathbf{x}_i\}_{i=1}^k$ and $X_2 = \{\mathbf{x}_i\}_{i=k+1}^n$, and denote their means as $\bar{X}_1$ and $\bar{X}_2$, respectively. Now the 0-step aggregated pointwise robustness for the pair $(\hat{s}_1, \hat{x})$ is $\bar{X}_1$, and for the pair $(\hat{s}_2, \hat{x})$ is $\bar{X}_2$. The new squared error sums up the error of each pair as,

$$\sum_{i=1}^k \|\mathbf{x}_i - \bar{X}_1\|^2 + \sum_{i=k+1}^n \|\mathbf{x}_i - \bar{X}_2\|^2. \quad (\text{SE}_1 + \text{SE}_2)$$

Now we prove that the squared error is reduced after the division, *i.e.*, $\text{SE}_1 + \text{SE}_2 - \text{SE}_{\text{total}} \leq 0$.

$$\begin{aligned} & \text{SE}_1 + \text{SE}_2 - \text{SE}_{\text{total}} \\ &= \sum_{i=1}^k \|\mathbf{x}_i - \bar{X}_1\|^2 + \sum_{i=k+1}^n \|\mathbf{x}_i - \bar{X}_2\|^2 - \sum_{i=1}^n \|\mathbf{x}_i - \bar{X}\|^2 \\ &= \sum_{i=1}^k \|\mathbf{x}_i - \bar{X}_1\|^2 - \sum_{i=1}^k \|\mathbf{x}_i - \bar{X}\|^2 + \sum_{i=k+1}^n \|\mathbf{x}_i - \bar{X}_2\|^2 - \sum_{i=k+1}^n \|\mathbf{x}_i - \bar{X}\|^2 \\ &= \underbrace{\sum_{i=1}^k (\|\mathbf{x}_i - \bar{X}_1\|^2 - \|\mathbf{x}_i - \bar{X}\|^2)}_{P_1} + \underbrace{\sum_{i=k+1}^n (\|\mathbf{x}_i - \bar{X}_2\|^2 - \|\mathbf{x}_i - \bar{X}\|^2)}_{P_2} \end{aligned}$$

We firstly prove $P_1 \leq 0$.

$$\begin{aligned}
P_1 &= \sum_{i=1}^k (\mathbf{x}_i^2 + \bar{X}_1^2 - 2\mathbf{x}_i \bar{X}_1 - \mathbf{x}_i^2 - \bar{X}^2 + 2\mathbf{x}_i \bar{X}) \\
&= \sum_{i=1}^k (\bar{X}_1^2 - 2\mathbf{x}_i \bar{X}_1 - \bar{X}^2 + 2\mathbf{x}_i \bar{X}) \\
&= k(\bar{X}_1^2 - \bar{X}^2) + \sum_{i=1}^k (-2\mathbf{x}_i \bar{X}_1 + 2\mathbf{x}_i \bar{X}) \\
&= k(\bar{X}_1^2 - \bar{X}^2) + \sum_{i=1}^k 2\mathbf{x}_i \bar{X} - \sum_{i=1}^k 2\mathbf{x}_i \bar{X}_1 \\
&= k(\bar{X}_1^2 - \bar{X}^2) + 2\bar{X} \sum_{i=1}^k \mathbf{x}_i - 2\bar{X}_1 \sum_{i=1}^k \mathbf{x}_i \\
&= k(\bar{X}_1^2 - \bar{X}^2) + 2k\bar{X}\bar{X}_1 - 2k\bar{X}_1^2 \\
&= -k(\bar{X}_1^2 + \bar{X}^2 - 2\bar{\mathbf{x}}\bar{X}_1) \\
&= -k\|\bar{X}_1 - \bar{X}\|^2 \leq 0
\end{aligned}$$

Similarly, we can prove $P_2 \leq 0$, thus $\text{SE}_1 + \text{SE}_2 - \text{SE}_{\text{total}} \leq 0$. We have shown that the state splitting strategy used in our refinement step reduces the mean square error. □

With Theorem 4.9, the MSE of $\mathcal{M}$ is reduced after each refinement step. Since there is only a finite number of concrete states in $\mathcal{S}$, and the number of abstract states $\hat{\mathcal{S}}$ is guaranteed to increase after each iteration, $MSE_{\mathcal{M}}$ is eventually approaching zero. Therefore the threshold will be reached after finite number of iterations, *i.e.*, Algorithm 3 always terminates.

4.4 Evaluation

We implemented MARBLE in Python based on PyTorch (1.2.0) framework and conducted evaluation on two real-world RNN subject models (for NLP tasks) to evaluate the refinement strategy and the robustness quantification. All the experiments were run on a desktop with the Ubuntu 18.04 system with Intel(R)

Xeon(R) CPU E5-2697 v3 @ 2.60GHz, 64 GB RAM and one NVIDIA GeForce RTX 2080Ti 11G GPU.

In the first experiment, we measured the performance and effectiveness of the refinement strategy under various hyperparameter configurations compared with a random strategy as the baseline. Note that, in order to better handle high-dimensional state vectors, we apply PCA [111] to reduce the whole dimension to k major components in a similar way as *DeepStellar*. In the second experiment, we evaluated the robustness estimation accuracy by inspecting how the attack efficiency can be boosted/inhibited under the guidance of robustness calculated by MARBLE. In particular, we attack an input at the segment positions where our abstract model reveals the least robust and the most robust of an RNN, and measure the attack success rate, *i.e.*, the portion of samples being successfully attacked at positions with different robustness. Furthermore, compare MARBLE with POPQORN [106], the state-of-the-art approach for RNN robustness analysis, in terms of accuracy and scalability.

Subject Datasets and RNN Models. We select two datasets and corresponding RNN models as follows.

1. **The UIUC’s CogComp QC dataset** [112] consists of news titles, labeled with topic classifications. There are 20,000 samples for training and 8,000 samples for testing, and there are 15 words in each sample on average. Over the dataset, we follow the same configurations as in [106] to train an LSTM model (test accuracy 83.3%) with 32 hidden nodes. MARBLE is capable of scaling to larger datasets and models.
2. **The sentiment analysis dataset**, IMDB [113], where the samples are movie reviews labeled with binary sentimental classifications. There are 25,000 training data and 25,000 test data, where each sample contains 216 words on average. We trained a GRU model with 300 hidden nodes that achieves 90.7% test accuracy. In particular, we leveraged the pre-trained word embedding vectors GloVe [114], for the above two models under analysis to boost the training process.

Implementation of Input Abstraction. For NLP applications, the inputs are natural language sentences, and one of the most commonly used metamorphic mutation for NLP is synonym replacement [115, 116], where the semantics of the sentence stays unchanged whenever a word is replaced with its synonyms. In general, *cosine similarity* is often used to measure the semantic distance between words. Based on the GloVe embedding vector, we performed the input abstraction by making synonym groups, among which the cosine similarity is above a threshold. In our evaluation, we set the threshold to 0.75 and obtain 3572 synonym groups with an average of 8.6 words in each group. For the mutation function, we confined random mutations only to words inside the same group by uniform sampling.

Experiment I: effectiveness on the refinement. In this experiment, we examined the performance of our refinement strategy under different configuration instances (k, θ) . As a baseline for comparison, we also implemented a random-split based strategy for abstract state refinement, and demonstrated the effectiveness and advantage of our refinement strategy.

Table 4.1 summarizes the results of refinement obtained from the models of news title classification and sentiment analysis. Column “Config.” shows the evaluated configuration instances (*i.e.* (k, θ)), which are used in both of MARBLE and random strategy. The other columns include the number of iterations used to complete the refinement (column “#Iteration”), the number of states (column “#State.”), the number of transitions (column “#Transition”) of the refined MDP, and the number of missed transitions (column “#Miss”) when processing the testing dataset. We call a transition missed if it is not included in the MDP. The results of the MARBLE strategy and the random strategy are listed under column “MARBLE” and “Random”, respectively.

The results confirm the advantages of the refinement strategy of MARBLE compared with a random-split approach in that: 1) MARBLE terminates with less iterations across configurations. For example, in news title classification, the average iterations of MARBLE is 20 while random strategy is 30. 2) MARBLE refines a better MDP model, which not only has both less states and transitions

TABLE 4.1: Measures of MDP models refined with different configurations and strategies.

Dataset	Config	Refine	#Iteration	#State	#Transition	#Miss	Time (s)
News Title Classification	(16, 0.3)	MARBLE	20	950	82,953	12,597	13.6
		Random	30	2,500	105,256	16,365	15.9
	(16, 0.6)	MARBLE	20	685	78,547	11,637	16.2
		Random	27	1,762	99,752	15,501	12.9
	(16, 1.0)	MARBLE	20	354	70,204	9,940	21.3
		Random	22	799	90,773	13,567	14.3
	(32, 0.3)	MARBLE	22	1,026	85,047	12,834	17.0
		Random	44	2,687	105,725	16,528	24.4
	(32, 0.6)	MARBLE	18	755	79,415	11,772	14.2
		Random	33	1,912	99,749	15,240	18.7
	(32, 1.0)	MARBLE	20	369	70,937	9,955	22.4
		Random	22	801	92,657	13,462	17.8
Sentiment Analysis	(32, 0.3)	MARBLE	46	20,828	1,696,536	62,349	750.1
		Random	96	66,848	2,500,136	99,195	1,104.7
	(32, 0.6)	MARBLE	44	14,539	1,512,339	52,261	756.7
		Random	40	52,422	2,472,142	96,276	653.4
	(32, 1.0)	MARBLE	33	6,409	1,297,227	39,072	859.6
		Random	35	25,132	2,280,291	78,947	638.0
	(64, 0.3)	MARBLE	68	21,303	1,650,063	60,304	1,038.1
		Random	100	68,873	2,578,434	104,178	887.4
	(64, 0.6)	MARBLE	53	13,393	1,405,890	47,281	741.8
		Random	59	56,258	2,534,159	100,753	645.1
	(64, 1.0)	MARBLE	42	5,406	1,194,095	34,731	881.3
		Random	44	26,794	2,351,163	83,777	645.3

but also raises less transition misses when dealing with new samples from the test dataset. Under the same configuration of k , the larger the threshold θ is, the less complicated the refined MDP becomes. Under the same configuration of θ , a larger k often leads to a model with slightly more states and transitions, and less missed transitions. A coarse MDP often endows better generalization capability when processing new samples, leading to less missed transitions.

Experiment II: effectiveness on the robustness. We evaluate the effectiveness of the obtained robustness by conducting the adversarial attack under the robustness results. For the CogComp QC dataset, besides the comparison with the random baseline, we also compared the performance with the POPQORN. For the IMDB dataset, POPQORN failed to scale to the size/complexity of the model, forcing us unable to include it for performance comparison.

To make a comparison on the CogComp QC dataset, we run POPQORN to calculate the robustness scores for sentences from the CogComp QC dataset. Due

FIG. 4.4: Attack success rates based on the robustness scores calculated by different methods. For the least robust word, the larger the attack success rate is, the better; for the most robust word, the smaller the attack success rate is, the better.

to the performance limitation of POPQORN, we only succeed in calculating the robustness estimation on these 42 sentences. Then, MARBLE is used to calculate the robustness scores for the same set of sentences. Based on the robustness score, we then adopted the mutation-based technique to attack the specific word in each sentence. Specifically, for each sentence, we performed the attacks on the least robust word and the most robust word to evaluate the accuracy of the robustness quantification, and allow a fixed buffer of chances (here we allow 5 times) to make the mutations. We also selected a random strategy that randomly selects one word position to attack. For each target word (*i.e.*, the least robust or the most robust) in the sentence, we randomly changed the word to one of its synonyms and check whether the result is changed. We repeated the attack for 50 times and computed the average of the attack success rate.

The results of average attack success rates are shown in Fig. 4.4. For the CogComp QC dataset, we present the average success rate among the 42 sentences. Compared with POPQORN and Random, MARBLE achieves the highest success rate (19.0%) when attacking the least robust word and the lowest success rate (2.0%) when attacking the most robust word, respectively. The results confirm that MARBLE calculates more accurate robustness scores. Random strategy achieves the worst result as its attack has no robustness guidance. Specifically, for attacks on the least robust word, MARBLE outperforms POPQORN and Random for a better

success rate, with 27.5% and 48.4% improvement, respectively; for attacks on the most robust word, MARBLE achieves the least success rate, with 80.0% and 84.1% decrease compared with POPQORN and Random. For evaluation on the IMDB dataset, 1650 sentences are used, and under both attack scenarios, MARBLE reveals a better performance. For attacks on the least robust word, MARBLE outperforms Random with 178.5% increase on the success rate; for attacks on the most robust word, the success rate of MARBLE is 66.7% less than that of Random.

Performance. We also evaluated the performance of MARBLE on refinement and robustness quantification. Column “Time” in Table 4.1 shows the time used for the refinement with different configurations. The results demonstrate that MARBLE can generate a MDP model efficiently. For example, in news title classification, the average time of MARBLE on all configuration is 17.41 seconds, while the average time of random strategy is 17.28 seconds. With the constructed abstract model, then the robustness of an input could be calculated quickly. In Experiment II, we present the time used to quantify the robustness over every element in an input to demonstrate the efficiency. On average, MARBLE takes 0.02 second while POPQORN takes 52.8 minutes. In addition, we had already removed some cases which POPQORN failed to return the results after 12 hours. The results indicate that MARBLE is much more efficient than the state-of-the-art (*i.e.*, POPQORN) and it is especially useful in real-time online robustness checking.

In summary, MARBLE is with better scalability for models with more complicated structures and taking longer inputs. We can also see that MARBLE is very efficient in calculating the robustness of an input and reveals better accuracy in identifying the least/most robust position.

4.5 Conclusion

In this chapter, we proposed MARBLE, a model-based technique for quantitative robustness analysis of RNN-based DL systems. We first model an RNN and its internal state transition behaviors with a labeled MDP through profiling of the

training data. On top of that, we design a refinement strategy to iteratively derive an accurate abstraction by minimizing the estimation errors of the pointwise robustness. Our evaluation suggests that, compared with the state-of-the-art technique, MARBLE enables more accurate and efficient quantification of the robustness of RNNs. In the future, we plan to conduct more comprehensive studies on the effect parameters k (in the k -step trace-based robustness) on the robustness quantification and apply MARBLE on more diverse applications, such as image classification and automatic speech recognition.

Chapter 5

Trace-length Independent Runtime Monitoring

To support the runtime protection of RNN-based DL systems, it requires expressive languages or logics to capture the malicious behavior (*i.e.*, the adversarial attacks) and also the efficient monitoring algorithms to make the solution feasible and practical. In this chapter, we present the theory and algorithms of the proposed effective runtime monitoring framework.

Linear temporal logic (LTL) [117] and metric linear-time logic (MTL) [118] and are widely used as a specification logics or language to describe runtime properties of various systems. Typically, the use of temporal logic is mainly concerned with qualitative properties, for example the relative ordering of events/action, or eventuality of events/action, etc. MTL is an extension of LTL, where operators are constrained by the time intervals, and is able to specify timing-dependent properties. Such application in runtime monitoring is mainly because the efficient monitoring algorithms are usually available for such logic specification language, and the monitoring scripts can be automatically generated for any given policies. However, there exist some attack patterns that cannot be stated as pure MTL formulas. For example, to detect the adversarial samples of RNN, we need to stipulate how many transitions with weak robustness are visited during the past several steps or the past time period. The demand of quantitative policies is

even extensive in transitional software systems, *e.g.* Android, where we need to monitor the frequency of events of interests, like SMS sending and socket opening, to detect DDos attacks. Currently, MTL lacks such quantitative semantics in policy specification.

One approach to solve this problem is to build into MTL a notion of counting of events [119], or more generally, aggregate operators [120]. But monitoring algorithms for such extensions have not been extensively studied in the literature. Particularly, they are expected to be time and space efficient considering the restricted storage and computation resources in Android. Extension of expressiveness always brings more challenges to the design of efficient algorithms. The monitoring problem for even a simple extension of LTL with the counting quantifier, as studied in [119], is already PSPACE complete (in the size of policy and the trace), and PTIME (in the size of the trace) when the policy is fixed. In the online monitoring, where near real-time decisions need to be made, the dependence of the monitor on the size of the trace would make it impractical even if its complexity is PTIME (assuming the policy is fixed), as its performance would degrade as the trace length grows. Our goal is thus to balance the expressivity of the policy specification language and the complexity of the monitoring algorithm.

In this work, we propose an extension of past-time MTL (ptMTL) [121], called MTL_{cnt} , to support a *metric temporal counting quantifier*, or metric counting quantifier for short, which is motivated by and extended from the counting quantifier proposed in [119]. In addition, we also introduce arithmetic relations to allow various restrictions on the count. MTL_{cnt} considers only the fragment of MTL with past time operators, as this is sufficient for our purpose to enforce history-sensitive access control. For monitoring on resource restricted platforms, once we fix the policy to be monitored, the space requirement of the monitoring algorithm is expected to be constant, *i.e.*, independent of the length of the system event trace. Such type of monitoring algorithms are called as *trace-length independent* (TLI) monitoring algorithms [78]. In this work, we call a policy whose satisfiability can be checked by a TLI monitoring algorithm as a TLI-monitorable policy. Note that we require the TLI algorithm generated from a policy to be complete w.r.t. the policy,

that is, every trace of events that adheres to the policy should also be accepted by the monitor, and every trace of events that violates the policy specification should be rejected. Otherwise the problem would be trivial as one could simply make various ad hoc restrictions such as restricting the time window for the monitoring.

A complete TLI monitoring algorithm for past-time LTL (ptLTL) is developed in [77]. For more expressive logics, such as first-order logics [78, 119] and our MTL_{cnt} , trace-length independence is not always possible, *i.e.*, there are formulas for which the monitor needs to store the entire history of events and hence the space used to store critical information will increase as the traces grow. For example, with MTL_{cnt} , one can write a formula that compares the numbers of occurrences of two events, say e_1 and e_2 . Let x and y denote the number of past occurrences of events of e_1 and e_2 , respectively. To check the relation $x < y$ at any state, we would need to keep track of the difference between x and y ; such a difference may grow as the trace grows, so the space requirement for monitoring this formula is not bounded. However, there are also instances of complex quantitative policies that are TLI-monitorable (see § 5.6). One main contribution of our work is in deriving classes of policies that are TLI-monitorable.

5.1 Specification Language MTL_{cnt}

In this section, we propose a logic specification language MTL_{cnt} by extending ptMTL [121] with a *metric temporal counting quantifier*. The counting quantifier helps to specify the times that a sub-policy has been satisfied in a past time period and checks whether the count satisfies some constraints.

5.1.1 Syntax of MTL_{cnt}

We denote with AP the set of atomic propositional variables. Elements in AP are ranged over by p, q and s . We assume a countably infinite set of constants and use a, b, c and d to range over integer constants. We assume an infinite set $\mathcal{V}$ of variables of integers, whose elements are ranged over by x, y and z . MTL_{cnt}

also admits common arithmetic operators such as $+$, $-$ and $\times$, and comparison operators like $=$, $<$, $>$, $\leq$ and $\geq$. Terms are built from constants, variables and arithmetic operators, and denoted by s , t , u and v .

The time model of MTL_{cnt} follows the point-based time model adopted in [121]. We denote the time domain with $\mathbb{T}$, and in this work, we consider only discrete time domain with $\mathbb{T} = \mathbb{N}$. $\mathbb{T}$ is assumed with the standard ordering $\leq$. An *interval* is a non-empty set $I \subseteq \mathbb{T}$ such that, for all $\tau, \tau' \in I$ and $\kappa \in \mathbb{T}$, if $\tau < \kappa < \tau'$ then $\kappa \in I$. We denote with $\mathbb{I}$ the set of all time intervals. The lower bound and the upper bound of an interval I are denoted by $\ell(I)$ and $r(I)$. To simplify exposition, we require the time intervals to be left-closed and right-open. For a natural number n , we define $I \pm n = \{y \pm n | y \in I\} \cap [0, +\infty)$. A *time sequence* $(\tau_i)_{i \in \mathbb{N}_{\geq 1}}$ is a sequence of elements $\tau_i \in \mathbb{T}$ with $\tau_k \leq \tau_j$ for all $k \leq j$ and $k, j \in \mathbb{N}_{\geq 1}$. Here the time sequence is not required to be strictly increasing, that is successive elements in the sequence can have identical timestamps.

The syntax of MTL_{cnt} is defined via the grammar:

$$\phi := \perp \mid p \mid (t > 0) \mid \neg\phi \mid \phi \vee \phi \mid \odot_I \phi \mid \phi \mathcal{S}_I \phi \mid \mathcal{C}_I x : \langle \phi, \phi \rangle . \phi,$$

where ϕ is any MTL_{cnt} formula, $p \in AP$, $I \in \mathbb{I}$, t ranges over terms and x ranges over variables. The operators appearing above are those of ptMTL except for the counting quantifier $\mathcal{C}$ and the relation $t > 0$. The operators subscripted with I are *metric temporal operators* and the subscripts can be omitted when $I = [0, +\infty)$ for succinctness. We assume the reader is familiar with the notion of free and bound variables. In the formula $\mathcal{C}_I x : \langle \alpha, \beta \rangle . \phi$, the variable x is a bound variable, whose scope is over ϕ , so x is not free in either formulas α or β . Roughly, the variable x is a placeholder that would contain a number associated with how many times β is satisfied within the interval I since α is satisfied. The formula α here acts as a counter *reset* condition, i.e., we only start counting once α is satisfied. The counting quantifier is generalized from a similar modality introduced in [119], and a detailed discussion about the difference will be given in §5.1.3. We also assume that bound variables in a formula are pairwise distinct. We write $\phi(x_1, \dots, x_n)$ to

mean that the free variables of ϕ are in $\{x_1, \dots, x_n\}$ and we write $\phi(c_1, \dots, c_n)$ to denote the instance of $\phi(x_1, \dots, x_n)$ where c_i is the substitution for x_i .

In the definition of formulas, we have kept a minimum number of logical operators. The omitted operators can be derived using the given operators, *e.g.*, propositional operators such as $\top$ (truth), $\wedge$ (conjunction), $\rightarrow$ (implication), and modal operators such as $\Diamond_I$ (sometimes in the past I), which is defined as $\Diamond_I \phi \equiv \top \mathcal{S}_I \phi$, and $\Box_I$ (always in the past I), which can be defined as $\Box_I \phi \equiv \neg \Diamond_I \neg \phi$. Note also that all other arithmetic relations can be derived from the relation of the form $(t > 0)$ and logical connectives: $s > t \equiv (s - t) > 0$, $s \leq t \equiv \neg(s > t)$, $s = t \equiv (s \leq t) \wedge (t \leq s)$, $s \geq t \equiv s > t \vee s = t$, and $s < t \equiv s \leq t \wedge \neg(s = t)$.

5.1.2 Semantics of MTL_{cnt}

A *trace* is a finite sequence of *states*, where each state itself consists of a set of atomic propositions. These atomic propositions correspond to events of interests that are being monitored in a system. We assume an *interpretation function* ξ which maps constant symbols to integers, and arithmetic operators and relation symbols to their corresponding semantic counterparts. We assume the usual arithmetic operators, and in addition, depending on applications, we may assume a fixed set of function symbols denoting computable functions over the integer domain. Since terms can contain variables, we additionally need to interpret these variables. This is done via a *valuation function*, *i.e.*, a function from variables to integers. Formally, given an interpretation function ξ and a valuation function ν , the interpretation of a term t , written $t^{\xi, \nu}$ is defined as in first-order logic [122]. For example, $(5 + x)^{\xi, \nu}$ is defined as $5^\xi +^\xi \nu(x)$. If ν maps x to 2, then $(5 + x)^{\xi, \nu}$ is 7. Relations in integer arithmetic are interpreted similarly, *e.g.*, $(x > 0)^{\xi, \nu}$ is interpreted as $(\nu(x) >^\xi 0^\xi)$. To simplify the presentation, we drop the superscript ξ for familiar arithmetic symbols, *e.g.*, instead of writing $+^\xi$ and $(5 + x)^{\xi, \nu}$, we simply write $+$ and $(5 + x)^\nu$ respectively.

A *model* for MTL_{cnt} is a triple (ρ, τ, ν) , where ρ is a trace, τ a time sequence, and ν is a valuation function, with $|\rho| = |\tau|$. For ρ (or τ), we write ρ_i (or τ_i) to

denote its i -th state/world (or timestamp). Each state ρ_i is timestamped with τ_i . For a valuation ν , we write $\nu[x \mapsto n]$ to denote the function which is identical to ν except for the valuation of x , *i.e.*, $\nu[x \mapsto n](y) = \nu(y)$, when $y \neq x$, and $\nu[x \mapsto n](x) = n$.

Definition 5.1. The satisfiability relation between a model (ρ, τ, ν) , a world $i \in \mathbb{N}_{\geq 1}$ and a formula ψ , written as $\rho, \tau, \nu, i \models \psi$, is defined by induction on ψ as below, where $\rho, \tau, \nu, i \not\models \psi$ if $\rho, \tau, \nu, i \models \psi$ is false.

- $\rho, \tau, \nu, i \not\models \psi$ if $i < 1$ or $i > |\rho|$.
- $\rho, \tau, \nu, i \not\models \perp$.
- $\rho, \tau, \nu, i \models p$ iff $p \in \rho_i$.
- $\rho, \tau, \nu, i \models t > 0$ iff $t^\nu > 0$ is true.
- $\rho, \tau, \nu, i \models \neg\phi$ iff $\rho, \tau, \nu, i \not\models \phi$.
- $\rho, \tau, \nu, i \models \phi_1 \vee \phi_2$ iff $\rho, \tau, \nu, i \models \phi_1$ or $\rho, \tau, \nu, i \models \phi_2$.
- $\rho, \tau, \nu, i \models \odot_I \phi$ iff $\rho, \tau, \nu, i - 1 \models \phi$ and $\tau_i - \tau_{i-1} \in I, i \geq 2$.
- $\rho, \tau, \nu, i \models \phi_1 \mathcal{S}_I \phi_2$ iff there exists $j \leq i$ such that $\tau_i - \tau_j \in I, \rho, \tau, \nu, j \models \phi_2$ and $\rho, \tau, \nu, k \models \phi_1$ for all $j < k \leq i$.
- $\rho, \tau, \nu, i \models \mathcal{C}_I x : \langle \alpha, \beta \rangle . \phi$ iff $\rho, \tau, \nu[x \mapsto n], i \models \phi$ where $n = |\{j \mid \rho, \tau, \nu, j \models \beta \text{ for } \tau_i - \tau_j \in I \text{ and } m < j \leq i\}|$ and $m = \max(\{j \mid \rho, \tau, \nu, j \models \alpha, \tau_i - \tau_j \in I, 1 \leq j \leq i\} \cup \{0\})$.

We simply write $i \models \psi$ when the trace ρ , the time sequence τ and the valuation function ν are clear from the context.

Intuitively, the meaning of $\mathcal{C}_I x : \langle \alpha, \beta \rangle . \phi$ is as follows: suppose that β is true at exactly n states within the past time period I since the latest state where α holds; then the instance of ϕ with x mapped to n must also be true. Here is an example to give a more intuitive explanation about the semantics of the counting quantifier.

Example 5.1. For an authentication server (e.g., bank) which validates a user's credential, a common login policy can be that if a user fails to enter the correct password three times in a row, then the user's account is temporarily disabled. Let us consider only two system events: a correct password was entered (cp), and a wrong password was entered (wp) by a particular user. The logic policy can be specified as follows:

$$\Box [\neg(cp \wedge wp) \wedge (\mathcal{C}x : \langle cp, wp \rangle . x < 3)]. \quad (5.1)$$

The first conjunct expresses a consistency property, i.e., a password entered cannot be both correct and wrong at the same time. The variable x stores the number of times a wrong password was entered since the last time a correct password was entered (or since the beginning of the trace, if no correct password has been entered so far). Consider the event trace

$$\rho = \langle \{wp\}; \{cp\}; \{wp\}; \{wp\}; \{cp\}; \{wp\} \rangle.$$

Since the formula is free of time intervals, there is no need to consider the time sequence. Then formula (5.1) above is true at every state. For example, at ρ_4 , the variable x gets instantiated to 2, since there are exactly two wp events since the latest cp event at this point. The wp event at ρ_1 is not counted in this case since it happened before the reset condition cp .

Intuitively, the counting quantifier can be used to express quantitative properties within a ‘session’ (e.g., an authentication session, a life cycle of a process, etc). One could introduce two events, *start* and *end*, to mark the beginning and the end of a session. Then to check that the number of occurrences of an event e within a session is less than n , for example, one can simply use the formula $\mathcal{C}x : \langle start, e \wedge \neg end \rangle . x < n$ in conjunction with other formulas expressing the well-formedness of a session (e.g., every *end* corresponds to a *start*, etc). If e is a simple event (e.g., the wp event in Example 5.1), one could encode this in MTL using standard temporal operators, but at the expense of conciseness, i.e., one needs to expand the parameter n into n instances of $e \wedge \neg end$. For example,

Example 5.1 can be alternatively specified as

$$\Box[\neg(cp \wedge wp) \wedge \neg(wp \wedge \odot(wp \wedge \odot wp))].$$

That is, there cannot be three consecutive wp events any time in the past. However, this is the case only when there are no events being monitored other than cp and wp . When other events are possible, then we need to specify that events other than cp can happen in between two consecutive wp events. In general, in a formula $\mathcal{C}x : \langle \alpha, \beta \rangle . \phi$, any of the α , β and ϕ could be a complicated temporal formula, *e.g.*, it could involve nested counting quantifiers and other temporal operators. In such a case, the encoding into pure MTL becomes less obvious and less concise. Therefore, it is necessary to introduce the counting quantifier for not only a simple way to compose formulas but also the possibility of designing an efficient monitoring algorithm.

5.1.3 Language Comparison

In [119], Bauer et. al. introduce a counting quantifier $\mathcal{N}$ (defined in Def. 5.2) to LTL, which is similar to the non-metric fragment of $\mathcal{C}$. We prove that the two quantifiers are with equivalent expressiveness, when $\mathcal{C}$ is with interval $[0, +\infty)$, as shown in Prop. 5.3.

Definition 5.2. ($\mathcal{N}$ counting quantifier) Given a trace ρ , valuation ν , a world i and the formula $\mathcal{N}x : \psi.\varphi$, the satisfiability is defined as:

$$\begin{aligned} \rho, \nu, i \models \mathcal{N}x : \psi.\varphi \text{ iff } \rho, \nu[x \mapsto n], i \models \varphi \text{ where} \\ n = |\{j \mid \rho, \nu, j \models \psi \text{ and } 1 \leq j \leq i\}|. \end{aligned}$$

Proposition 5.3. The counting quantifiers $\mathcal{C}$ (with interval $[0, +\infty)$) and $\mathcal{N}$ are equivalent, *i.e.*, one can be defined in terms of the other.

Proof. Obviously, the quantifier $\mathcal{N}$ can be encoded using $\mathcal{C}$ as follows:

$$\mathcal{N}x : \beta.\phi \equiv \mathcal{C}x : \langle \perp, \beta \rangle.\phi. \quad (5.2)$$

That is, for $\mathcal{C}$, we can set the time interval to $[0, +\infty)$ and disable the reset condition to achieve the same semantics with $\mathcal{N}$. In this case, the time interval constraints in the semantic definition will not have any effect.

By Def. 5.1, we have:

$$\begin{aligned} \rho, \tau, \nu, i \models \mathcal{C}x : \langle \perp, \beta \rangle.\phi &\text{ iff } \rho, \tau, \nu[x \mapsto n], i \models \phi, \text{ where} \\ n &= |\{j \mid \rho, \tau, \nu, j \models \beta \text{ and } m < j \leq i\}| \text{ and} \\ m &= \max(\{j \mid \rho, \tau, \nu, j \models \perp, 1 \leq j \leq i\} \cup \{0\}). \end{aligned}$$

Since $\{j \mid \rho, \tau, \nu, j \models \perp, 1 \leq j \leq i\} = \emptyset$, we have $m = 0$. Thus,

$$\begin{aligned} \rho, \tau, \nu, i \models \mathcal{C}x : \langle \perp, \beta \rangle.\phi &\text{ iff } \rho, \tau, \nu[x \mapsto n], i \models \phi, \\ \text{where } n &= |\{j \mid \rho, \tau, \nu, j \models \beta \text{ and } 0 < j \leq i\}|. \end{aligned}$$

And with a little adjustment, $n = |\{j \mid \rho, \tau, \nu, j \models \psi \text{ and } 1 \leq j \leq i\}|$. This exactly follows the semantic definition of $\mathcal{N}$.

Conversely, $\mathcal{C}$ can also be encoded with $\mathcal{N}$ as follows:

$$\mathcal{C}x : \langle \alpha, \beta \rangle.\phi \equiv \mathcal{N}z : \alpha. \mathcal{N}x : (\beta \wedge \mathcal{N}y : \alpha.y = z). \phi. \quad (5.3)$$

Note that the subformula $\mathcal{N}y : \alpha.y = z$ actually acts as a counter reset condition. According to the definition of $\mathcal{N}$, we have

$$\begin{aligned} \rho, \nu, i \models \mathcal{N}z : \alpha. \mathcal{N}x : (\beta \wedge \mathcal{N}y : \alpha.y = z). \phi \\ \text{iff } \rho, \nu[z \mapsto n_1, x \mapsto n_2], i \models \phi, \text{ where} \\ n_1 &= |\{h \mid \rho, \nu, h \models \alpha, 1 \leq h \leq i\}| \text{ and} \\ n_2 &= |\{j \mid \rho, \nu, j \models \beta \text{ and } \rho, \nu, j \models \mathcal{N}y : \alpha.y = n_1, 1 \leq j \leq i\}|. \end{aligned} \quad (5.4)$$

Let $m_j = |\{k \mid \rho, \nu, k \models \alpha, 1 \leq k \leq j\}|$. Then n_2 can be equivalently stated as

$$n_2 = |\{j \mid \rho, \nu, j \models \beta, \rho, \nu[y \mapsto n_3], j \models (m_j = n_1), \\ \text{and } 1 \leq j \leq i\}|$$

We define $m = \max(\{h \mid \rho, \nu, h \models \alpha, 1 \leq h \leq i\} \cup \{0\})$, and given the same trace, if $m_j = n_1$ with $j \leq i$, i.e.,

$$|\{k \mid \rho, \nu, k \models \alpha, 1 \leq k \leq j\}| = |\{h \mid \rho, \nu, h \models \alpha, 1 \leq h \leq i\}|,$$

obviously, we can get $j \geq m$.

Because with the definition of m , we know

$$\begin{aligned} \{h \mid \rho, \nu, h \models \alpha, m < h \leq i\} &= \emptyset, \\ \{h \mid \rho, \nu, h \models \alpha, 1 \leq h \leq i\} &= \{h \mid \rho, \nu, h \models \alpha, 1 \leq h \leq m\}. \end{aligned}$$

Given $j \geq m$, since $j \leq i$, we can also get

$$\begin{aligned} & \{k \mid \rho, \nu, k \models \alpha, 1 \leq k \leq j\} \\ &= \{k \mid \rho, \nu, k \models \alpha, 1 \leq k \leq m\} \cup \{k \mid \rho, \nu, k \models \alpha, m < k \leq j\} \\ &= \{k \mid \rho, \nu, k \models \alpha, 1 \leq k \leq m\} \cup \emptyset \\ &= \{k \mid \rho, \nu, k \models \alpha, 1 \leq k \leq m\}. \end{aligned} \tag{5.5}$$

That is $m_j = n_1$.

Therefore, $j \geq m$ is equivalent to the condition $m_j = n_1$. Thus,

$$\begin{aligned} & \rho, \nu, i \models \mathcal{N}z : \alpha. \mathcal{N}x : (\beta \wedge \mathcal{N}y : \alpha.y = z). \phi \\ & \equiv \rho, \nu[x \mapsto n_2], i \models \phi \text{ with } n_2 = |\{j \mid \rho, \nu, j \models \beta \text{ and } m \leq j \leq i\}| \end{aligned}$$

which equals to $\mathcal{C}x : \langle \alpha, \beta \rangle. \phi$.

Finally, the proposition follows. □

Although $\mathcal{C}$ can be encoded using $\mathcal{N}$ as in equation (5.3), it introduces nested occurrences of $\mathcal{N}$ and one needs to compare at least two counting variables. In general, policies involving two or more counting variables are hard or impossible to be monitored in a trace-length independent way, like $x < y$, where we need to keep track of the difference between two variables. We could have simply used $\mathcal{N}$, but we would then have to use the encoding above to compare two or more variables. Such encodings would thus obscure the underlying structure of the problem, and makes it harder to systematically generate TLI monitors from a given specification. For instance, the policy described in Example 5.1 uses only one counting variable when expressed using $\mathcal{C}$, and results in § 5.3 would guarantee the existence of TLI monitors for that particular policy. Had we chosen to encode it using $\mathcal{N}$, we would have to make more efforts in order to show that the policy is in fact TLI monitorable. Moreover, the counting quantifier we propose is capable of specifying metric temporal counting policies while $\mathcal{N}$ fails to. We also mention that counting operators defined in [120, 123] are encountered with the same problem of complicating the encoding, though they are able to specify metric temporal constraints.

The quantifier $\mathcal{C}$ achieves equivalent expressiveness but with a more concise representation owing to the reset component. In [124], a similar design of counting operator with reset is given but under the non-metric semantics. To the best of our knowledge, $\mathcal{C}$ is the first metric counting quantifier with an elegant and beneficial design of reset condition, especially for the study of TLI monitoring algorithms.

5.2 Preliminaries

5.2.1 Trace-length Independent Monitoring of ptMTL

Beyond ptLTL, TLI monitoring algorithms for its metric extension, ptMTL, have also been investigated in [121, 125]. Before exploring the TLI monitoring algorithm for MTL_{cnt} , we first prepare the readers with basic ideas on the TLI monitoring of the vanilla ptMTL. Recursive definitions of satisfiability relations typically lead

to efficient dynamic programming based algorithms for checking membership of a trace in the set of traces defined by a formula [126]. For the TLI-monitoring of ptMTL, the core problem is to devise a recursive semantics for the metric *since* operator, and it is firstly investigated and solved in [125]:

$$\begin{aligned} i \models \phi_1 \mathcal{S}_I \phi_2 \text{ iff } 0 \in I \text{ and } i \models \phi_2, \text{ or} \\ i > 1, i \models \phi_1 \text{ and } i \models \phi_1 \mathcal{S}_{I'} \phi_2 \text{ where } I' = I - (\tau_i - \tau_{i-1}). \end{aligned}$$

As can be seen that the recursive semantics is dependent on the interpretation of a new formula with the same structure, but a different time interval, which shifts by the difference between the current and the previous timestamps. In a general setting, difference between two successive timestamps (*e.g.*, τ_i and τ_{i-1}) in a time sequence varies, so we need to simultaneously monitor the set of formulas arising from all possible interval shifts. Such formulas are termed as “*interval-skewed subformulas*” in [127]. Let $\mathcal{SF}(\phi)$ denote the set of subformulas of ϕ that are defined in a usual manner, the set of its interval-skewed subformulas is defined as:

$$\mathcal{SF}^k(\phi) = \{\phi_1 \mathcal{S}_{I-n} \phi_2 \mid \phi_1 \mathcal{S}_I \phi_2 \in \mathcal{SF}(\phi), n \in [1, \max(I)]\}.$$

Example 5.2 gives a demonstration of the interval-skewed subformula concept.

Example 5.2. For $\phi = p_1 \mathcal{S}_{[2,4)} p_2$, since $\{[2,4) - n \mid n \in [1,3]\} = \{[1,3), [0,2), [0,1)\}$, we get

$$\mathcal{SF}^k(\phi) = \{p_1 \mathcal{S}_{[1,3)} p_2, p_1 \mathcal{S}_{[0,2)} p_2, p_1 \mathcal{S}_{[0,1)} p_2\}.$$

Specially, if the *since* operator is bounded with an infinite time interval, *e.g.*, $I = [3, +\infty)$, then $(I - n) \in \{[0, +\infty), [1, +\infty), [2, +\infty)\}$.

We denote the expanded set of subformulas with $\mathcal{SF}^+(\phi)$, and define it as:

$$\mathcal{SF}^+(\phi) = \mathcal{SF}(\phi) \cup \mathcal{SF}^k(\phi).$$

For metric logic, timestamps can be infinitely large and recording any of them will interfere with the TLI property. As this is a characteristic of the logic itself and beyond our topic, following [121] we assume that timestamps can be represented by bounded number of bits. In practice, this assumption is often reasonable since arbitrarily precise clocks do not exist physically. We stick to the assumption in this chapter, not only to restrain bound space consumption, but also guarantee calculations on timestamps to be carried out in constant time. Our monitoring algorithm is extended from the one presented in [125].

5.2.2 Isolated Variables

For a counting formula, *e.g.*, $C_I x : \langle \alpha, \beta \rangle . \phi$, the constraint part ϕ may consist of multiple complicated components, like arithmetic relations, *since* operators, counting operators, conjunction operators and etc, thus x can be used in more than one arithmetic relations. In this case, these relations must be considered as a whole and complicate the analysis. To ease the exposition and make the description more succinct, we define isolated variables as in Def. 5.4.

Definition 5.4. Given a counting formula $C_I x : \langle \alpha, \beta \rangle . \phi$, if variable x is used in exactly one arithmetic relation, we call it an isolated variable.

In reality, this is not a limitation to the formulas. Particularly, if there is no variable in ϕ , the counting formula is logically equivalent to ϕ . In case ϕ is with more than one arithmetic relations involving x , it can be rewritten into an equivalent formula where x is isolated as shown in Lemma 5.5.

Lemma 5.5. *Given a counting formula $C_I x : \langle \alpha, \beta \rangle . \phi$ with x occurring in more than one arithmetic relations, it can be encoded to an equivalent formula where all variables are isolated.*

Proof. According to the syntax of MTL_{cnt} , all direct occurrences of variables are in arithmetic relations, and we assume x is used in terms $t_1, \dots, t_m$. The rewriting is done as follows:

1. We substitute the quantifier $\mathcal{C}_I x : \langle \alpha, \beta \rangle$ with $\mathcal{C}_I x_1 : \langle \alpha, \beta \rangle \dots \mathcal{C}_I x_m : \langle \alpha, \beta \rangle$.
2. For t_j with $1 \leq j \leq m$, x is substituted with x_j .

With the rewriting, it is guaranteed any x_j ($1 \leq j \leq m$) occurs in only one term. More importantly, formulas before and after the rewriting are equivalent because the quantification conditions — target condition, trigger condition and time interval — for x and its substitutions are the same, thus keeping the same valuation for these variables, and further the same valuation for the terms. $\square$

With Lemma 5.5, we can assume without loss of generality that all the variables in MTL_{cnt} formulas are isolated and restricted by a particular counting quantifier. Furthermore, given an execution trace, the valuation of a variable is determined by the counting quantifier, and this is different from the traditional usage of first-order quantifiers. From this point of view, we do not deem MTL_{cnt} as an extension of the first-order temporal logic, where variables always have alternative valuations.

5.3 Trace-length Independent Monitoring of MTL_{cnt}

With the extension of counting quantifier and arithmetic relation, MTL_{cnt} is more expressive and designing a TLI monitoring algorithm for it is desirable but challenging. From experiences on TLI monitoring of other past time temporal logics, it is profitable to explore the recursive semantics of the counting formula. Afterwards, algorithms can be designed accordingly. In the following, all variables are required to be bound, which means for any occurrence of an arithmetic relation in a formula, we can always identify the corresponding counting quantifiers. For instance, if a relation $t(x_1, \dots, x_n) > 0$ occurs as a subformula, then there must exist quantifiers $\mathcal{C}_{I_1} x_1 : \langle \alpha_1, \beta_1 \rangle, \dots, \mathcal{C}_{I_n} x_n : \langle \alpha_n, \beta_n \rangle$ restricting the valuation of its variables. Moreover, we assume all variables are isolated unless otherwise stated.

There are two parts in the semantics of counting formula: the valuation of the variable and the interpretation of the arithmetic relation under this valuation. In

runtime monitoring, the valuation of a variable is calculated via interpreting the counting quantifier against the execution trace. For non-metric counting quantifiers (i.e., those with interval $[0, +\infty)$), the number of times a formula is satisfied in the history can never decrease, unless the reset condition is triggered, which means $t^{\nu[x \mapsto d]} > 0$ is always evaluated in worlds behind $t^{\nu[x \mapsto (d-1)]} > 0$. This statement is still true for metric temporal counting quantifiers when interpreted on the *absolute* time intervals, and details can be found in § 5.3.4. This observation is the key inspiration for us to unveil the recursive semantics of counting formulas. Here we need to mention that recording the actual count and updating it according to the coming event would break the TLI property as the space needed grows. On the other hand, storing partial history is also not practical, as we do not specify any upper bound of the number of events that can emerge within a time period. As a result, we focus our attention on how to interpret the arithmetic relations recursively, assuming the counts grow irreversibly unless reset.

In this section, we first identify the class of arithmetic relations that are TLI-monitorable and then give the recursive semantics for counting formulas with such arithmetic relations. Specifically, in § 5.3.1 we look at the univariate case where relations are with arity 1, and then generalize it to the multivariate case in § 5.3.2. If all relations in a counting formula are TLI-monitorable, it is straightforward that formula itself is TLI-monitorable. For such TLI-monitorable counting formulas, we develop the recursive semantics to interpret the counting quantifiers and derive the valuation of the formulas based on results from bounded previous worlds. In § 5.3.3, we present the recursive description of the semantics of TLI-monitorable counting formulas with non-metric counting quantifiers, and make a generalization to metric temporal counting quantifiers in § 5.3.4.

5.3.1 TLI-Monitorable Arithmetic Relations

Intuitively, TLI-monitorable arithmetic relations are those for which the valuation of $t(x_1, \dots, x_n) > 0$ can be solved incrementally as events keep coming. In this

subsection, we look into univariate relations, i.e., $t(x) > 0$, and leave the multivariate cases to the next subsection. If a variable x is quantified with the time interval $[0, +\infty)$, we can infer that whenever an event occurs, the valuation of x either remains the same or increases by 1, or is cleared to 0 if the reset condition is triggered. This means the interpretation of $t^{\nu[x \mapsto (d-1)]} > 0$ is always calculated ahead of the interpretation of $t^{\nu[x \mapsto d]} > 0$ for any count number d . For counting quantifiers with general time intervals, we also observe such an interpretation order, after some transformation on the time intervals, as we shall discuss § 5.3.4. If we can determine the truth values of $t^{\nu[x \mapsto d]} > 0$ based on a finite number of interpretations of $t^{\nu[x \mapsto d-1]} > 0, \dots, t^{\nu[x \mapsto d-k]} > 0$, then it only needs to maintain the previous k status of this interpretation. Moreover, there is no need to store the actual value of the variable x nor the value of $t(x)$ during the monitoring; all that matters is the truth value of $t(x) > 0$. Thus the space required for monitoring such relations remains constant irrespective of the value of x during the monitoring. The formal definition of TLI-monitorable relations is given in Def. 5.6. In this definition, we do not require the valuation of $t(x) > 0$ to be recursive from 0 because it is when x becomes large that the constant space restriction is hard to fulfill.

Definition 5.6. Given a term $t : \mathbb{N} \rightarrow \mathbb{R}$, $t > 0$ is *TLI-monitorable* if there are two constants $c, k \in \mathbb{N}$, with $c \geq k \geq 1$, and a total computable Boolean function H , such that for any $x \geq c$:

$$f_t(x) = H(f_t(x-1), \dots, f_t(x-k)).$$

Definition 5.7. Let $t : \mathbb{N}^n \rightarrow \mathbb{R}$. We define its characteristic function f_t as follows:

$$f_t(x_1, \dots, x_n) = \begin{cases} 0 & \text{if } t(x_1, \dots, x_n) \leq 0, \\ 1 & \text{otherwise.} \end{cases}$$

Definition 5.8. Given a function $f : \mathbb{N} \rightarrow \mathbb{R}$, it is said to be *periodic* over interval $I \subseteq \mathbb{N}$ with period $T \in \mathbb{N}_{>0}$ if we have

$$f(x) = f(x+T)$$

for all values of $x \in I$, such that $(x + T) \in I$.

Definition 5.9. A total function $f : \mathbb{N} \rightarrow \mathbb{R}$ is said to be *lower-bounded periodic*, or *lb-periodic* for short, if there is a $b \in \mathbb{N}$ such that f is periodic on interval $[b, +\infty)$.

Lemma 5.10. *Given a term $t : \mathbb{N} \rightarrow \mathbb{R}$, $t > 0$ is TLI-monitorable if f_t is lb-periodic.*

Proof. Since f_t is lb-periodic, there is a b such that f_t is periodic on $[b, +\infty)$. Suppose the period is k , and then for any $x \geq b + k$ we have $f_t(x) = f_t(x - k)$. So,

$$f_t(x) = \begin{cases} t(x) > 0 & \text{if } x < b + k, \\ f_t(x - k) & \text{otherwise.} \end{cases}$$

Hence, the Boolean function H can be easily constructed as: for $x \geq b + k$,

$$H(f_t(x - 1), \dots, f_t(x - k)) = f_t(x - k).$$

Obviously, H is a standard primitive recursive projection function and according to Def. 5.6, $t > 0$ is TLI-monitorable. $\square$

We characterize the class of relations which are TLI-monitorable according to Def. 5.6. Some auxiliary definitions are presented in Def. 5.7-5.9 to define the characteristic function and to describe the *lb-periodic* property of a function. These definitions help to specify the characterizations of TLI-monitorable relations. The sufficiency condition for a relation to be TLI-monitorable is shown and proved in Lemma 5.10. Example 5.3 is presented to illustrate the definitions and lemma above.

Example 5.3. *Let $t(x) = x^2 - 8x + 15, x \in \mathbb{N}$. In this case, the characteristic function of t is*

$$f_t(x) = \begin{cases} 0 & \text{if } 3 \leq x \leq 5, \\ 1 & \text{otherwise.} \end{cases}$$

Since $t(x) \leq 0$ holds only if $3 \leq x \leq 5$, function f_t is lb-periodic on $[6, +\infty)$, with lower bound 6 and period 1. So $t > 0$ is TLI-monitorable according to Lemma 5.10.

In the next, we present in Lemma 5.11 the necessary condition for a relation to be TLI-monitorable and give the proofs.

Lemma 5.11. *Given a term $t : \mathbb{N} \rightarrow \mathbb{R}$, f_t is lb-periodic if $t > 0$ is TLI-monitorable.*

Proof. As f_t is TLI-monitorable, there exist two constants c and k , with $c \geq k \geq 1$ and $c, k \in \mathbb{N}$, such that

$$f_t(x) = \begin{cases} t(x) > 0 & \text{if } x < c, \\ H(f_t(x-1), \dots, f_t(x-k)) & \text{otherwise.} \end{cases}$$

for some given function H .

We denote the value of $f_t(x)$ as a_x for $x \geq c$, and then we can get a string $a = a_c a_{c+1} a_{c+2} \dots$ with infinite length. We define a subsequence of a as $A = a a_{i+1} \dots a_{i+k-1}$, with $i \geq c$ and $i \in \mathbb{N}$. A is of length k . It is obvious that A is composed of only 0 and 1, so $A \in \{0, 1\}^k$.

Now we can get an infinite list of sequences $A_c, A_{c+1}, A_{c+2}, \dots$. For the first $(2^k + 1)$ sequences, there must be at least one pair of identical sequences according to the *Pigeonhole Principle*. We pick out the first sequence A_p which has a repeat, and then there must exist $q > p$ so that $A_p = A_q$; and for any $p < t < q$, $A_p \neq A_t$. More explicitly, $A_p = A_q$ means that $a_p = a_q, \dots, a_{p+k-1} = a_{q+k-1}$.

With some adaption on representation, we have $a_x = H(a_{x-1}, \dots, a_{x-k})$, then it can be inferred that

$$H(a_{p+k-1}, \dots, a_p) = H(a_{q+k-1}, \dots, a_q) \Rightarrow a_{p+k} = a_{q+k}.$$

Along with $a_{p+k} = a_{q+k}$, we get

$$H(a_{p+k}, \dots, a_{p+1}) = H(a_{q+k}, \dots, a_{q+1}) \Rightarrow a_{p+k+1} = a_{q+k+1},$$

and so on and so forth.

Via the deduction we get the conclusion that for $x \geq q$, $a_x = a_{x-(q-p)}$, i.e., $f_t(x) = f_t(x - (q - p))$. That is, $f_t(x) = f_t(x - (q - p))$ on $[q, +\infty)$. Therefore, f_t is lb-periodic on $[p, +\infty)$, and this lemma follows. $\square$

With the sufficient and necessary conditions proved in Lemma 5.10 and Lemma 5.11 separately, we get the following theorem:

Theorem 5.12. *Given a term $t : \mathbb{N} \rightarrow \mathbb{R}$, $t > 0$ is TLI-monitorable iff f_t is lb-periodic.*

The abstract characterization in Theorem 5.12 is in a way quite obvious from the definition of TLI-monitorable relations. The important part is that monitorability is associated with periodic characteristic function f_t rather than the term t itself. That is, t may not be periodic yet still be TLI-monitorable. In concrete applications, since we are usually only given the term t but not the characteristic function f_t , the difficulty is in deciding whether f_t is lb-periodic. In the following, we show several classes of functions for which their characteristic functions are lb-periodic.

Theorem 5.13. *Given a term $t : \mathbb{N} \rightarrow \mathbb{R}$, $t > 0$ is TLI-monitorable if t satisfies one of the following conditions:*

1. t is lb-periodic.
2. t is monotonically increasing/decreasing.
3. t is a univariate polynomial function.

Proof. We show the proofs for each case.

Case(1): Supposing t is lb-periodic on interval $[b, +\infty)$ with period T , we have, for $\forall m \in [b, +\infty)$, $t(m) = t(m + T)$. If $t(m) \leq 0$, naturally $t(m + T) \leq 0$, and then

$$f_t(m) = f_t(m + T) = 0.$$

Else if $t(m) > 0$, it can be inferred that $t(m + T) > 0$ and

$$f_t(m) = f_t(m + T) = 1.$$

Therefore, $f_t(x)$ is lb-periodic on interval $[b, +\infty)$. According to Theorem 5.12, $t > 0$ is TLI-monitorable.

Case(2): We first consider the scenario where term t is monotonically increasing. If $t(x) \leq 0$ for $\forall m \in \mathbb{N}$, then $f_t(x)$ remains 0, i.e., it is lb-periodic on $\mathbb{N}$ with period 1. Else, if $\exists m_0 \in \mathbb{N}$, we have $t(x) > 0$. Then for any $m > m_0$, we have $t(m) > t(m_0) > 0$, that is $f_t(x)$ remains 1 on $[m_0, +\infty)$. According to Theorem 5.12, $t > 0$ is TLI-monitorable.

In the other case F is monotonically decreasing, with similar proofs we can get to the same results.

Case(3): Suppose t is a univariate polynomial of degree n and $t(x)$ has at most n roots. If $t(x)$ has no roots, according to the Continuity of polynomial functions, t remains positive or negative on the domain. Else, we assume the roots are $c_1, \dots, c_k$, for some $1 \leq k \leq n$. (Note that these roots are not necessary to be integers.) Let c be a positive integer larger than $\lceil \max(c_1, \dots, c_k) \rceil$. Since all polynomial functions are continuous, we have either $t(x) > 0$ for $\forall x \geq c$ or $t(x) < 0$ for $\forall x \geq c$. In either case, f_t is lb-periodic on $[c, \infty)$ with period 1, and therefore $t > 0$ is TLI-monitorable. $\square$

There also exist univariate relations that are not TLI-monitorable, like $t(x) = \sin\sqrt{x}$, for which the characteristic function is not lb-periodic.

5.3.2 TLI-Monitorable Multivariate Relations

We now look at the case where the arithmetic relations can be multivariate. The formal definition of TLI-monitorable multivariate relations is given in Def. 5.14. Theorem 5.16 implies that φ is TLI-monitorable if the period of a projection of F into one of its parameter is independent of the other parameter, once the value of that parameter exceeds a certain threshold. This allows us to quotient the values of each parameters into their own equivalence classes independently of each other.

Definition 5.14. Let $t : \mathbb{N}^n \rightarrow \mathbb{R}$ be a term, then $t > 0$ is TLI-monitorable if there are constants $c_1, \dots, c_n$ and $k_1, \dots, k_n$, with $c_i \geq k_i \geq 1$ and $c_i, k_i \in \mathbb{N}$ for all

$1 \leq i \leq n$, such that $f_t(x_1, \dots, x_n) =$

$$\begin{cases} t(x_1, \dots, x_n) > 0, & \text{if } \forall i : x_i < c_i, \\ H(\{f_t(x_1 - u_1, \dots, x_n - u_n) \mid \forall i : 0 \leq u_i \leq k_i\} \setminus \\ f_t(x_1, \dots, x_n)), & \text{otherwise.} \end{cases}$$

Here H is a total computable Boolean function.

Definition 5.15. Let $t : \mathbb{N}^n \rightarrow \mathbb{R}$, we define the set of the i -th dimension slices of t as:

$$\Pi_i(t) = \{x \rightarrow t(a_1, \dots, a_{i-1}, x_i, a_{i+1}, \dots, a_n) \mid a_1, \dots, a_{i-1}, a_{i+1}, \dots, a_n \in \mathbb{N}\}.$$

Theorem 5.16. Given a term $t : \mathbb{N}^n \rightarrow \mathbb{R}$, for any $1 \leq i \leq n$, if the characteristic functions of functions in $\Pi_i(t)$ are all lb-periodic with the same lower bound and period, then $t > 0$ is TLI-monitorable.

Proof. For any $1 \leq i \leq n$, we assume the characteristic functions of functions in $\Pi_i(t)$ are lb-periodic and share the same period k_i and lower bound c_i . If $x_i < k_i + c_i$ for any $1 \leq i \leq n$, then $t(x_1, \dots, x_n)$ is calculated accordingly and get the interpretation of $t > 0$. Otherwise, supposing there exists j such that $x_j \geq k_j + c_j$ and $x_m < k_m + c_m$ for all $1 \leq m < j$, since the characteristic functions of functions in $\Pi_j(t)$ are lb-periodic, we have,

$$f_t(x_1, \dots, x_j, \dots, x_n) = f_t(x_1, \dots, x_j - k_j, \dots, x_n).$$

Following similar lines, function H can be constructed as $f_t(x_1, \dots, x_n) =$

$$\left\{ \begin{array}{ll} t(x_1, \dots, x_n) > 0 & \text{if } x_i < k_i + c_i \text{ for all } 1 \leq i \leq n, \\ f_t(x_1 - k_1, \dots, x_n) & \text{if } x_1 \geq k_1 + c_1. \\ \dots & \dots \\ f_t(x_1, \dots, x_j - k_j, \dots, x_n) & \text{if } x_j \geq k_j + c_j \text{ and } x_i < k_i + c_i \\ & \text{for all } 1 \leq i < j \leq n. \\ \dots & \dots \\ f_t(x_1, \dots, x_n - k_n) & \text{if } x_n \geq k_n + c_n \text{ and } x_i < k_i + c_i \\ & \text{for all } 1 \leq i < n. \end{array} \right.$$

According to the Def. 5.14, $t > 0$ is TLI-monitorable. $\square$

5.3.3 Recursive Semantics of Non-metric Temporal Counting Quantifier

For a counting formula, the counting semantics comes from the counting quantifiers and the limitation semantics comes from the arithmetic relations. If the counting quantifier is with time interval $[0, +\infty]$, we call it non-metric temporal counting quantifier. In this section, we investigate how to interpret the counting formula with non-metric temporal counting quantifiers in a recursive way.

Recall (from § 5.2.2) that all variables in a counting formula are isolated, which means each counting variable appears in exactly one term. We start the analysis with a particular arithmetic relation, and see how to translate the counting semantics for all its bound counting quantifiers. Generally, the count for a target formula never decreases unless the reset condition is triggered. As the count is finally checked in an arithmetic relation, if the arithmetic relation is TLI-monitorable, we can track the valuation recursively instead of keeping its real value which can be arbitrarily large. For a univariate TLI-monitorable relation $t(x) > 0$, its truth value repeats periodically when x exceeds the lower bound of its characteristic

function, therefore we can quotient its domain based on the period to form a finite set of equivalence classes. As for multivariate relations, *e.g.*, $t(x_1, \dots, x_n) > 0$, surprisingly we can get the equivalence classes following similar line. The main difference between the univariate and the multivariate case is finding the right lower bound and the periods for a particular variable, then we can quotient one dimension of its domain to form a finite set of equivalence classes. The definition of equivalence classes can be found in Def. 5.17, and the number equivalence classes is finite according to Prop. 5.18.

Definition 5.17. Given a term $t : \mathbb{N}^n \rightarrow \mathbb{R}$, and for any $1 \leq i \leq n$ the characteristic functions of functions in $\Pi_i(t)$ are lb-periodic over $[b_i, +\infty)$ with period T_i , we define the *equivalence class* on the i -th dimension domain of t as:

$$[e] = \begin{cases} \{e\} & \text{if } e < b_i, \\ \{a \mid (a - b_i) \bmod T_i + b_i = e, a \geq b_i\} & \text{otherwise.} \end{cases}$$

Proposition 5.18. For equivalence classes defined in Def. 5.17, $[e]$ is empty iff $e \geq b_i + T_i$.

Proof. Please refer to Appendix A.1. □

Proposition 5.19. For equivalence classes of the i -th dimension defined in Definition 5.17, $t > 0$ has the same interpretation for elements in the same equivalence class.

Proof. For the equivalence class $[e]$, and $\forall a \in [e]$:

- (1) If $e < b_i$, since $[e] = \{e\}$, obviously, $a = e$ and $t^{\nu[x_i \mapsto a]} > 0 \equiv t^{\nu[x_i \mapsto e]} > 0$.
- (2) If $e \geq b_i$, since

$$[e] = \{a \mid (a - b_i) \bmod T_i + b_i = e, a \geq b_i, a \in \mathbb{N}\},$$

thus we have $(a - b_i) \bmod T_i + b_i = e$, and with some equivalent transformation, we have

$$\begin{aligned}
 & (a - b_i) \bmod T_i + b_i = e \\
 \Leftrightarrow & (a - b_i) \bmod T_i = e - b_i \\
 \Leftrightarrow & a - b_i = (e - b_i) + m * T_i (\exists m \in \mathbb{N}) \\
 \Leftrightarrow & a = e + m * T_i
 \end{aligned}$$

Since the characteristic functions of functions in $\Pi_i(t)$ is lb-periodic over $[b_i, +\infty)$ with period T_i , we assume f'_t the characteristic function of the mapping

$$x \rightarrow t(x_1, \dots, x_{i-1}, x, x_{i+1}, \dots, x_n),$$

where $x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n$ can be instantiated by any valid valuation functions.

Then we have

$$f'_t(a) = f'_t(e + m * T_i) = f'_t(e + (m - 1) * T_i) = \dots = f'_t(e).$$

That is, $t^{\nu[x_i \mapsto a]} > 0 \equiv t^{\nu[x_i \mapsto e]} > 0$.

Now, it is proved that for any $a \in [e]$, we have $t^{\nu[x_i \mapsto a]} > 0 \equiv t^{\nu[x_i \mapsto e]} > 0$. Thus the proposition follows. $\square$

As proved in Prop. 5.19, the interpretation of $t > 0$ is the same for valuations in the same equivalence class; we can achieve a recursive semantics by managing the valuation of variables in t as indexes to equivalence classes. For simplicity, each equivalence class is indexed with the minimal element in the set. In this case the valuation of variables is limited to a finite range and make trace-length independence possible. The reset condition in the quantifier is orthogonal to the issue of quotienting the domain of corresponding variable; when it is satisfied, the count is reset to 0, i.e., the index is reset to 0. We write $\mathbf{r}(\rho, \tau, \nu, i, \mathcal{C}_I x : \langle \alpha, \beta \rangle)$ to denote the index of the equivalence class indicated by the valuation x quantified by

$\mathcal{C}_I x : \langle \alpha, \beta \rangle$, and the value is computed under model (ρ, τ, ν) and world i . When ρ, τ and ν are clear from the context, we also write as $\mathfrak{r}(i, \mathcal{C}_I x : \langle \alpha, \beta \rangle)$ for simplicity.

As in [77, 128], the key to achieve the trace-length independence property is to express the semantics of all logical operators in a recursive form, e.g., the interpretation of ϕ at state i is generated only based on the interpretation of its subformulas at current world and/or the interpretation of ϕ at world $i - 1$. Semantics of all operators in MTL_{cnt} except the counting formula is already in recursive form. Our aim is to calculate to which equivalence class the current valuation of the variable belongs based on the previous records. Theorem 5.20 shows that the semantics of a TLI-monitorable non-metric counting formula also admits a recursive form. For world 1, i.e., the base case, we uniformly give the semantics of general (temporal and non-temporal) counting quantifier in Theorem 5.21 (§5.3.4).

Theorem 5.20. *Given a model (ρ, τ, ν) and a counting formula $\mathcal{C}x : \langle \alpha, \beta \rangle . \phi(x)$, if every arithmetic relation in ϕ is TLI-monitorable, then for every $1 < j \leq |\rho|$,*

$$j \models \mathcal{C}x : \langle \alpha, \beta \rangle . \phi(x) \text{ iff } j \models \phi^{\nu[x \mapsto \mathfrak{r}(j, \mathcal{C}x : \langle \alpha, \beta \rangle)]},$$

where for variable x , we assume the characteristic functions of slices on x are lb-periodic on $[b, +\infty]$ with period T , and the index of equivalence class is calculated via:

1. If $j \models \alpha$, then $\mathfrak{r}(j, \mathcal{C}x : \langle \alpha, \beta \rangle) = 0$;
2. If $j \not\models \alpha$, and $j \models \beta, r < b$, where $r = \mathfrak{r}(j - 1, \mathcal{C}x : \langle \alpha, \beta \rangle)$, then $\mathfrak{r}(j, \mathcal{C}x : \langle \alpha, \beta \rangle) = r + 1$;
3. If $j \not\models \alpha$, and $j \models \beta, r \geq b$, where $r = \mathfrak{r}(j - 1, \mathcal{C}x : \langle \alpha, \beta \rangle)$, then $\mathfrak{r}(j, \mathcal{C}x : \langle \alpha, \beta \rangle) = (r + 1 - b) \bmod T + b$;
4. If $j \not\models \alpha$, and $j \not\models \beta$, then $\mathfrak{r}(j, \mathcal{C}x : \langle \alpha, \beta \rangle) = \mathfrak{r}(j - 1, \mathcal{C}x : \langle \alpha, \beta \rangle)$.

Proof. For the counting quantifier, based on whether the reset condition and target formula are satisfied at current stage, the valuation of x determined by $\mathcal{C}x : \langle \alpha, \beta \rangle$

is discussed under the following three scenarios with distinct conditions, among which the union is complete and the intersection of any two is empty:

- (i) $j \models \alpha$;
- (ii) $j \not\models \alpha$ and $j \models \beta$;
- (iii) $j \not\models \alpha$ and $j \not\models \beta$.

Since we assume the counting formula is TLI-monitorable, x is used either in a TLI-monitorable univariate relation whose characteristic function is lb-periodic on $[b, +\infty)$ and with period T , or a TLI-monitorable multivariate relation for which the characteristic functions of slices on x are lb-periodic on $[b, +\infty)$ and with period T .

For a non-metric counting quantifier its semantics can be simplified as:

- $j \models \mathcal{C}x : \langle \alpha, \beta \rangle . \phi$ iff $\nu[x \mapsto l], j \models \phi$, where $l = |\{h \mid h \models \beta, m < h \leq j\}|$, $m = \max(\{h \mid h \models \alpha, 1 \leq h \leq j\} \cup \{0\})$.

For $\mathcal{C}x : \langle \alpha, \beta \rangle$, we can define

$$\mathbb{B} = \{h \mid h \models \beta, m < h \leq j\},$$

$$\mathbb{A} = \{h \mid h \models \alpha, 1 \leq h \leq j\}.$$

$$l = |\mathbb{B}|,$$

$$m = \max(\mathbb{A} \cup \{0\}).$$

Thus we only need to prove l always belongs to the equivalence class $[\mathbf{r}(j, \mathcal{C}x : \langle \alpha, \beta \rangle)]$.

Under condition (i), if $j \models \alpha$, then $j \in \mathbb{A}$ and j is the largest value in $\mathbb{A}$. Naturally, $m = j$ and $\mathbb{B}$ is empty because there is no h greater than j . It follows that $l = 0$, and obviously $l \in [\mathbf{r}(j, \mathcal{C}x : \langle \alpha, \beta \rangle)]$.

Under condition (ii), if $j \not\models \alpha$ and $j \models \beta$, it means that there is a new occurrence of β which should be updated to the count as the reset condition is not triggered.

To deal with the update of the count, we need to check whether the index of the equivalence class to which the count belongs at the current state is beyond the lower bound. We define, at state $j - 1$,

$$\begin{aligned}\mathbb{B}' &= \{h \mid h \models \beta, m' < h \leq j - 1\}, \\ \mathbb{A}' &= \{h \mid h \models \alpha, 1 \leq h \leq j - 1\}, \\ l' &= |\mathbb{B}'|, \\ m' &= \max(\mathbb{A}' \cup \{0\}).\end{aligned}$$

With the condition, we can get

$$\begin{aligned}j \not\models \alpha &\Rightarrow j \notin \mathbb{A} \Rightarrow \mathbb{A} = \mathbb{A}' \Rightarrow m = m', \\ j \models \beta &\Rightarrow j \in \mathbb{B} \Rightarrow \mathbb{B} = \mathbb{B}' \cup \{j\} \text{ (with } m = m').\end{aligned}$$

Since $j \notin \mathbb{B}'$, it can be implied that $|\mathbb{B}| = |\mathbb{B}'| + 1 \Rightarrow l = l' + 1$.

After getting the valuation l of x , we need to compute to which the equivalence class l belongs. If $l' < b$, we have $l' \in [l']$. It can also be inferred that $l' + 1 \leq b$, and also

$$\begin{aligned}l' + 1 < b &\Rightarrow l' + 1 \in [l' + 1]; \\ l' + 1 = b &\Rightarrow l' + 1 \in [(b - b) \bmod T + b] \Rightarrow l' + 1 \in [b].\end{aligned}$$

So we can always get $l \in [l' + 1]$. With $r = \mathbf{r}(j - 1, \mathcal{C}x : \langle \alpha, \beta \rangle) = l'$, we have $\mathbf{r}(j, \mathcal{C}x : \langle \alpha, \beta \rangle) = r + 1 = l' + 1$, that is $l \in [\mathbf{r}(j, \mathcal{C}x : \langle \alpha, \beta \rangle)]$.

Otherwise, if $l' \geq b$, then $l' \in [(l' - b) \bmod T + b]$. Obviously, $l = l' + 1 \geq b$ and $l \in [(l - b) \bmod T + b]$. As $\mathbf{r}(j, \mathcal{C}x : \langle \alpha, \beta \rangle) = ((l' - b) \bmod T + b + 1 - b) \bmod T + b$, if $T = 1$,

$$\mathbf{r}(j, \mathcal{C}x : \langle \alpha, \beta \rangle) = b = (l - b) \bmod T + b,$$

thus we have $l \in [\mathbf{r}(j, \mathcal{C}x : \langle \alpha, \beta \rangle)]$.

If $T > 1$, then

$$\begin{aligned}
& ((l' - b) \bmod T + b + 1 - b) \bmod T + b \\
&= ((l' - b) \bmod T + 1) \bmod T + b \\
&= ((l' - b) \bmod T + 1 \bmod T) \bmod T + b \\
&= (l' - b + 1) \bmod T + b \\
&= (l - b) \bmod T + b
\end{aligned}$$

that is $l \in [\mathbf{r}(j, \mathcal{C}x : \langle \alpha, \beta \rangle)]$.

Under condition (iii), if $j \not\models \alpha$ and $j \not\models \beta$, then we know that $j \notin \mathbb{B}$ and $j \notin \mathbb{A}$, thus yielding

$$\begin{aligned}
\mathbb{A} &= \{h \mid h \models \alpha, 1 \leq h \leq j - 1\} = \mathbb{A}'. \\
m &= \max(\mathbb{A} \cup \{0\}) = m', \\
\mathbb{B} &= \{h \mid h \models \beta, m < h \leq j - 1\} = \mathbb{B}'.
\end{aligned}$$

It follows that $l = l'$. Since $l' \in [\mathbf{r}(j - 1, \mathcal{C}x : \langle \alpha, \beta \rangle)]$, it is obviously $l \in [\mathbf{r}(j, \mathcal{C}x : \langle \alpha, \beta \rangle)]$.

The theorem follows for cases (i) (ii) (iii). $\square$

5.3.4 Recursive Semantics of Metric Temporal Counting Quantifier

In this section, we present the recursive semantics of counting quantifier bounded with general intervals. Since the time model used in this chapter is non-strictly increasing, which means there can be multiple occurrences of events with same timestamp, the count for certain formulas can be extremely large even in a short time interval. In this case, simply recording the exact count for metric counting quantifiers can still break the trace-length independence. Therefore, we need to seek for the recursive semantics of metric counting quantifiers with general time

intervals in order to pave the way for TLI monitoring. We present the proposed formal definition in Theorem 5.22 and the base case (world 1) in Theorem 5.21. The key idea is to look at the time intervals in an absolute way. If there is a $\Delta\tau$ time shifting between the current and previous timestamps, current observation of events within time interval I is consistent with previous observation in skewed interval $I - \Delta\tau$. In this way we can accumulate the count incrementally.

Theorem 5.21. *Given a model (ρ, τ, ν) and a counting formula $\mathcal{C}_I x : \langle \alpha, \beta \rangle . \phi(x)$, at the world 1,*

$$1 \models \mathcal{C}_I x : \langle \alpha, \beta \rangle . \phi(x) \text{ iff } 1 \models \phi^{\nu[x \mapsto \mathfrak{r}(1, \mathcal{C}_I x : \langle \alpha, \beta \rangle)]},$$

where for variable x , we assume the characteristic functions of slices on x is lb-periodic on $[b, +\infty]$ with period T , then the index of equivalence class is calculated via:

1. If $0 \notin I$, $\mathfrak{r}(1, \mathcal{C}_I x : \langle \alpha, \beta \rangle) = 0$;
2. If $0 \in I$,
  - (a) if $1 \models \alpha$, then $\mathfrak{r}(1, \mathcal{C}_I x : \langle \alpha, \beta \rangle) = 0$;
  - (b) if $1 \not\models \alpha$ and $1 \not\models \beta$, then $\mathfrak{r}(1, \mathcal{C}_I x : \langle \alpha, \beta \rangle) = 0$.
  - (c) if $1 \not\models \alpha$, $1 \models \beta$ and $b > 1$, then $\mathfrak{r}(1, \mathcal{C}_I x : \langle \alpha, \beta \rangle) = 1$;
  - (d) if $1 \not\models \alpha$, $1 \models \beta$ and $b \leq 1$, then $\mathfrak{r}(1, \mathcal{C}_I x : \langle \alpha, \beta \rangle) = (1 - b) \bmod T + b$.

Proof. Please refer to Appendix A.2. □

Theorem 5.22. *Given a model (ρ, τ, ν) and a counting formula $\mathcal{C}_I x : \langle \alpha, \beta \rangle . \phi(x)$, if every arithmetic relation in ϕ is TLI-monitorable, then for every $1 < j \leq |\rho|$,*

$$j \models \mathcal{C}_I x : \langle \alpha, \beta \rangle . \phi(x) \text{ iff } j \models \phi^{\nu[x \mapsto \mathfrak{r}(j, \mathcal{C}_I x : \langle \alpha, \beta \rangle)]}, \quad (5.6)$$

where for variable x , we assume the characteristic functions of slices on x is lb-periodic on $[b, +\infty]$ with period T , and $I' = I - (\tau_j - \tau_{j-1})$, then the index of equivalence class is calculated via:

1. If $0 \notin I$ and $I' = \emptyset$, $\mathbf{r}(j, \mathcal{C}_I x : \langle \alpha, \beta \rangle) = 0$;
2. If $0 \notin I$ and $I' \neq \emptyset$, $\mathbf{r}(j, \mathcal{C}_I x : \langle \alpha, \beta \rangle) = \mathbf{r}(j - 1, \mathcal{C}_{I'} x : \langle \alpha, \beta \rangle)$;
3. If $0 \in I$ and $I' = \emptyset$:
 If $j \not\models \alpha$ and $j \models \beta$, $\mathbf{r}(j, \mathcal{C}_I x : \langle \alpha, \beta \rangle) = 1$ if $b > 1$, and $\mathbf{r}(j, \mathcal{C}_I x : \langle \alpha, \beta \rangle) = (1 - b) \bmod T + b$ if $b \leq 1$. Otherwise $\mathbf{r}(j, \mathcal{C}_I x : \langle \alpha, \beta \rangle) = 0$;
4. If $0 \in I$ and $I' \neq \emptyset$:
 (a) if $j \models \alpha$, then $\mathbf{r}(j, \mathcal{C}_I x : \langle \alpha, \beta \rangle) = 0$;
 (b) if $j \not\models \alpha$ and $j \models \beta$ and $r = \mathbf{r}(j - 1, \mathcal{C}_{I'} x : \langle \alpha, \beta \rangle) < b$, then $\mathbf{r}(j, \mathcal{C}_I x : \langle \alpha, \beta \rangle) = r + 1$;
 (c) if $j \not\models \alpha$ and $j \models \beta$ and $r = \mathbf{r}(j - 1, \mathcal{C}_{I'} x : \langle \alpha, \beta \rangle) \geq b$, then $\mathbf{r}(j, \mathcal{C}_I x : \langle \alpha, \beta \rangle) = (r + 1 - b) \bmod T + b$;
 (d) if $j \not\models \alpha$ and $j \not\models \beta$, then $\mathbf{r}(j, \mathcal{C}_I x : \langle \alpha, \beta \rangle) = \mathbf{r}(j - 1, \mathcal{C}_{I'} x : \langle \alpha, \beta \rangle)$.

Proof. In this proof, for each counting quantifier, the valuation of x determined by $\mathcal{C}_I x : \langle \alpha, \beta \rangle$ is analyzed under the four cases based on whether $0 \in I$ and I' is empty. Since we assume the counting formula is TLI-monitorable, x is used either in a TLI-monitorable univariate relation whose characteristic function is lb-periodic on $[b, +\infty)$ and with period T , or a TLI-monitorable multivariate relation for which the characteristic functions of slices on x are lb-periodic on $[b, +\infty)$ and with period T . According to the semantics of metric temporal counting quantifier defined in § 5.1, for $\mathcal{C}_I x : \langle \alpha, \beta \rangle$ we denote

$$\begin{aligned}\mathbb{B} &= \{h \mid h \models \beta, \text{ for } \tau_j - \tau_h \in I, m < h \leq j\}, \\ \mathbb{A} &= \{h \mid h \models \alpha, \text{ for } \tau_j - \tau_h \in I, 1 \leq h \leq j\}, \\ l &= |\mathbb{B}|, \\ m &= \max(\mathbb{A} \cup \{0\}).\end{aligned}$$

Thus we only need to prove that l always belongs to the equivalence class $[\mathbf{r}(j, \mathcal{C}_I x : \langle \alpha, \beta \rangle)]$.

Case(1): If $0 \notin I$ and I' is empty,

$$\begin{aligned}
0 \notin I &\Rightarrow \tau_j - \tau_j \notin I \Rightarrow j \notin \mathbb{B}, \\
I' = \emptyset &\Rightarrow (\tau_j - \tau_{j-1}) > \max(I) \\
&\Rightarrow \forall h < j - 1, \tau_j - \tau_h > \max(I) \\
&\Rightarrow \forall h < j - 1, \tau_j - \tau_h \notin I
\end{aligned}$$

then we can conclude that $\mathbb{B} = \emptyset \Rightarrow l = 0$. Hence, $l \in [\mathbf{r}(j, \mathcal{C}_I x : \langle \alpha, \beta \rangle)]$.

Case(2): If $0 \notin I$ and $I' \neq \emptyset$, for $\mathcal{C}_{I'} x : \langle \alpha, \beta \rangle$, we denote at state $j - 1$

$$\begin{aligned}
\mathbb{B}' &= \{h \mid h \models \beta, \text{ for } \tau_{j-1} - \tau_h \in I', m' < h \leq j - 1\}, \\
\mathbb{A}' &= \{h \mid h \models \alpha, \text{ for } \tau_{j-1} - \tau_h \in I', 1 \leq h \leq j - 1\}, \\
l' &= |\mathbb{B}'|, \\
m' &= \max(\mathbb{A}' \cup \{0\}).
\end{aligned}$$

For $0 \notin I$, similar with Case(1), we know $j \notin \mathbb{A}$ and $j \notin \mathbb{B}$. Therefore, we can get

$$\begin{aligned}
\mathbb{A} &= \{h \mid h \models \alpha, \text{ for } \tau_j - \tau_h \in I, 1 \leq h \leq j - 1\} \\
&= \{h \mid h \models \alpha, \text{ for } \tau_j - \tau_h - (\tau_j - \tau_{j-1}) \in I - (\tau_j - \tau_{j-1}), \\
&\quad 1 \leq h \leq j - 1\} \\
&= \{h \mid h \models \alpha, \text{ for } \tau_{j-1} - \tau_h \in I', 1 \leq h \leq j - 1\} = \mathbb{A}' \\
&\Rightarrow m = m',
\end{aligned}$$

$$\begin{aligned}
\mathbb{B} &= \{h \mid h \models \beta, \text{ for } \tau_j - \tau_h \in I, m < h \leq j - 1\} \\
&= \{h \mid h \models \beta, \text{ for } \tau_j - \tau_h - (\tau_j - \tau_{j-1}) \in I - (\tau_j - \tau_{j-1}), \\
&\quad m < h \leq j - 1\} \\
&= \{h \mid h \models \beta, \text{ for } \tau_{j-1} - \tau_h \in I', m < h \leq j - 1\} \\
&= \{h \mid h \models \beta, \text{ for } \tau_{j-1} - \tau_h \in I', m' < h \leq j - 1\} = \mathbb{B}' \\
&\quad (m = m') \\
&\Rightarrow l = l'.
\end{aligned}$$

Hence, $l \in [\mathbf{r}(j, \mathcal{C}_I x : \langle \alpha, \beta \rangle)]$.

Case(3): If $0 \in I$ and $I' = \emptyset$, similar with Case(1),

$$I' = \emptyset \Rightarrow \forall h \in [1, j), h \notin \mathbb{B}, h \notin \mathbb{A}.$$

Thus the only possible member for $\mathbb{B}$ and $\mathbb{A}$ is j because $\tau_j - \tau_j = 0 \in I$, and further $l \leq 1$. Easily, we can get

$$\begin{aligned}
l = 1 &\Leftrightarrow \mathbb{B} = \{j\} \Leftrightarrow j \models \beta \text{ and } m < j \Leftrightarrow j \models \beta \text{ and } j \notin \mathbb{A} \\
&\Leftrightarrow j \models \beta \text{ and } j \not\models \alpha.
\end{aligned}$$

Hence, if $j \models \beta$ and $j \not\models \alpha$, then $l = 1$, and naturally $l \in [\mathbf{r}(j, \mathcal{C}_I x : \langle \alpha, \beta \rangle)]$ no matter $b > l$ or $b \leq l$. Otherwise, $l = 0$, and also $l \in [\mathbf{r}(j, \mathcal{C}_I x : \langle \alpha, \beta \rangle)]$.

Case(4): If $0 \in I$ and $I' \neq \emptyset$, following the proof of Theorem 5.20, cases are analyzed based on the satisfaction of $j \models \beta$ and $j \models \alpha$.

For case(a),

$$j \models \phi \Rightarrow j \in \mathbb{A} \Rightarrow m = j(0 \in I) \Rightarrow \mathbb{B} = \emptyset \Rightarrow l = 0$$

Thus, $l \in [\mathbf{r}(j, \mathcal{C}_I x : \langle \alpha, \beta \rangle)]$.

Algorithm 4: $Monitor(\rho, \tau, \nu, i, \phi)$

```

1 Init( $\rho, \nu, \phi, cur, c\_cnt$ );
2 for  $j = 2$  to  $i$  do
3   Iter( $\rho, \tau, \nu, j, \phi, prev, cur, p\_cnt, c\_cnt$ );
4 return  $cur[idx(\phi)]$ ;

```

For case(d), similar with case(2),

$$\begin{aligned}
j \not\models \alpha &\Rightarrow j \notin \mathbb{A} \\
&\Rightarrow \mathbb{A} = \{h \mid h \models \alpha, \text{ for } \tau_j - \tau_h \in I, 1 \leq h \leq j-1\} = \mathbb{A}' \\
&\Rightarrow m = m',
\end{aligned}$$

and

$$\begin{aligned}
j \not\models \beta &\Rightarrow j \notin \mathbb{B} \\
&\Rightarrow \mathbb{B} = \{h \mid h \models \beta, \text{ for } \tau_j - \tau_h \in I, m < h \leq j-1\} = \mathbb{B}' \\
&\Rightarrow l = l'
\end{aligned}$$

Hence, $l \in [\mathbf{r}(j, C_{Ix} : \langle \alpha, \beta \rangle)]$.

For case(b) and case(c), it is easy to get $j \not\models \alpha \Rightarrow m = m'$, and

$$\begin{aligned}
j \models \beta &\Rightarrow j \in \mathbb{B}(0 \in I) \\
&\Rightarrow \mathbb{B} = \{h \mid h \models \beta, \text{ for } \tau_j - \tau_h \in I, m < h \leq j-1\} \cup \{j\} \\
&= \mathbb{B}' \cup \{j\} \\
&\Rightarrow l = l' + 1.
\end{aligned}$$

This is the same case as shown in the proof of Theorem 5.20 for condition (ii), so we do not repeat here.

In conclusion, the theorem follows. $\square$

Algorithm 5: $Init(\rho, \nu, \phi, cur, c\_cnt)$

```

1 for  $k = 1$  to  $m$  do
2   switch  $\phi_k$  do
3     case  $\perp$  do  $cur[k] \leftarrow false$ ;
4     case  $p$  do  $cur[k] \leftarrow p \in \rho_1$ ;
5     case  $\neg\psi$  do  $cur[k] \leftarrow \neg cur[idx(\psi)]$ ;
6     case  $t > 0$  do  $cur[k] \leftarrow t^\nu > 0$ ;
7     case  $\psi_1 \vee \psi_2$  do  $cur[k] \leftarrow cur[idx(\psi_1)] \vee cur[idx(\psi_2)]$ ;
8     case  $\odot_I \psi$  do  $cur[k] \leftarrow false$ ;
9     case  $\psi_1 \mathcal{S}_I \psi_2$  do  $cur[k] \leftarrow 0 \in I \wedge cur[idx(\psi_2)]$ ;
10    case  $\mathcal{C}_I x : \langle \alpha, \beta \rangle . \varphi$  do
11      for  $(Q = \mathcal{C}_{I'} x : \langle \alpha, \beta \rangle) \in \mathcal{IS}_Q(\mathcal{C}_I x : \langle \alpha, \beta \rangle)$  do
12        if  $!cur[idx(\alpha)] \wedge cur[idx(\beta)] \wedge 0 \in I'$  then
13           $lb \leftarrow tli[idx_v(x)].lowerBound$ ;
14           $pd \leftarrow tli[idx_v(x)].period$ ;
15          if  $lb > 1$  then  $c\_cnt[idx_c(Q)] \leftarrow 1$ ;
16          else  $c\_cnt[idx_c(Q)] \leftarrow (1 - lb) \bmod pd + lb$ ;
17        else  $c\_cnt[idx_c(Q)] \leftarrow 0$ ;
18       $cur[k] \leftarrow cur[idx(\varphi^{\nu[x \mapsto c\_cnt[idx_c(\mathcal{C}_I x : \langle \alpha, \beta \rangle)]})]$ ;
19 return  $cur[idx(\phi)]$ ;

```

5.4 Monitoring Algorithm of MTL_{cnt}

Now each logical operators of MTL_{cnt} can be handled recursively, and dynamic programming can be employed to design the TLI monitoring algorithm dealing with formulas composed by these operators. A key for the monitoring is to guarantee subformulas are always interpreted ahead of its root formula. Hence, the set of subformulas of a given MTL_{cnt} formula must be formulated and organized in a well-order. For quantified counting formulas, however, its subformulas would contain free variables, and their truth values would thus depend on the valuation of variables. To avoid encoding valuation of variables explicitly in the monitoring algorithm, we need to instantiate variables to concrete values before computing their interpretation. Given a TLI-monitorable formula $\mathcal{C}_I x : \langle \alpha, \beta \rangle . \phi(x)$, we compute a finite set of equivalence classes for the domain, particularly on the dimension of x . Suppose this set has n elements $\{[e_1], \dots, [e_n]\}$. We define its immediate subformulas as: α , β , and $\phi^{\nu[x \mapsto e_i]}$ for $1 \leq i \leq n$. Given a formula ψ , we use $\mathcal{SF}^*(\psi)$ to denote the extension of $\mathcal{SF}^+(\psi)$ to include the instantiated subformulas

of counting formulas, and it is defined as:

$$\mathcal{SF}^*(\psi) = \mathcal{SF}^+(\psi) \cup \{\phi^{\nu[x \mapsto e_i]} \mid \mathcal{C}_I x : \langle \alpha, \beta \rangle . \phi(x) \in \mathcal{SF} \text{ and } 1 \leq i \leq n\}.$$

Algorithm 6: $Iter(\rho, \tau, \nu, i, \phi, prev, cur, p\_cnt, c\_cnt)$

```

1  prev ← cur;
2  p_cnt ← c_cnt;
3  for k = 1 to m do
4      switch  $\phi_k$  do
5          case  $\perp$  do cur[k] ← false;
6          case p do cur[k] ←  $p \in \rho_i$ ;
7          case  $\neg\psi$  do cur[k] ←  $\neg cur[idx(\psi)]$ ;
8          case  $t > 0$  do cur[k] ←  $t^\nu > 0$ ;
9          case  $\psi_1 \vee \psi_2$  do cur[k] ←  $cur[idx(\psi_1)] \vee cur[idx(\psi_2)]$ ;
10         case  $\odot_I \psi$  do cur[k] ←  $prev[idx(\psi)] \wedge (\tau_i - \tau_{i-1} \in I)$ ;
11         case  $\psi_1 \mathcal{S}_I \psi_2$  do
12              $I' \leftarrow I - (\tau_i - \tau_{i-1})$ ;
13             if  $0 \in I \wedge cur[idx(\psi_2)]$  then cur[k] ← true ;
14             if  $I' \neq \emptyset \wedge cur[idx(\psi_1)] \wedge prev[idx(\psi_1 \mathcal{S}_{I'} \psi_2)]$  then cur[k] ← true ;
15             else cur[k] ← false ;
16         case  $\mathcal{C}_I x : \langle \alpha, \beta \rangle . \varphi(x)$  do
17             for  $Q = \mathcal{C}_{I'} x : \langle \alpha, \beta \rangle \in \mathcal{IS}_{\mathcal{Q}}(\mathcal{C}_I x : \langle \alpha, \beta \rangle)$  do
18                  $I'_\Delta \leftarrow I' - (\tau_i - \tau_{i-1})$ ;
19                  $Q' = \mathcal{C}_{I'_\Delta} x : \langle \alpha, \beta \rangle$ ;
20                  $c\_cnt[idx_c(Q)] \leftarrow 0$ ;
21                 if  $0 \notin I' \wedge I'_\Delta \neq \emptyset$  then  $c\_cnt[idx_c(Q)] \leftarrow p\_cnt[idx_c(Q')]$  ;
22                 else if  $0 \in I' \wedge I'_\Delta = \emptyset$  then
23                     if  $!cur[idx(\alpha)] \wedge cur[idx(\beta)]$  then
24                          $lb \leftarrow tli[idx_v(x)].lowerBound$ ;
25                          $pd \leftarrow tli[idx_v(x)].period$ ;
26                         if  $lb > 1$  then  $c\_cnt[idx_c(Q)] \leftarrow 1$ ;
27                         else  $c\_cnt[idx_c(Q)] \leftarrow (1 - lb) \bmod pd + lb$ ;
28                 else if  $0 \in I' \wedge I'_\Delta \neq \emptyset$  then
29                     if  $!(cur[idx(\alpha)] \vee cur[idx(\beta)])$  then
30                          $c\_cnt[idx_c(Q)] \leftarrow p\_cnt[idx_c(Q')]$ ;
31                     else if  $cur[idx(\beta)]$  then
32                          $n \leftarrow ++p\_cnt[idx_c(Q')]$ ;
33                          $lb \leftarrow tli[idx_v(x)].lowerBound$ ;
34                          $pd \leftarrow tli[idx_v(x)].period$ ;
35                         if  $n \geq lb + pd$  then
36                              $c\_cnt[idx_c(Q)] \leftarrow (n - lb) \bmod pd + lb$ ;
37                         else  $c\_cnt[idx_c(Q)] \leftarrow n$ ;
38             cur[k] ←  $cur[idx(\varphi^{\nu[x \mapsto c\_cnt[idx_c(\mathcal{C}_I x : \langle \alpha, \beta \rangle)])})]$ 
39 return cur[idx( $\phi$ )];

```

From the recursive semantics presented in Theorem 5.22, we can see the interpretation of the counting formulas depends not only on its immediate subformulas, but also the valuation of its interval-skewed counting quantifiers. We define the interval-skewed counting quantifiers of $\mathcal{C}_I x : \langle \alpha, \beta \rangle$ as:

$$\mathcal{IS}_Q(\mathcal{C}_I x : \langle \alpha, \beta \rangle) = \{\mathcal{C}_{I-m} x : \langle \alpha, \beta \rangle \mid m \in [0, \max(I)]\}.$$

These interval-skewed counting quantifiers are no included in the set of subformulas but their valuation of variables are maintained respectively. We define a well-order $\prec$ over the set of interval-skewed quantifiers that respects the following rule:

- if Q' is an interval-skewed counting quantifier of Q and $Q' \neq Q$, then $Q' \prec Q$.

And we assume the quantifiers in the set are always well ordered.

Now we present how monitoring can be done for a general MTL_{cnt} formula ϕ , given a trace ρ and a time sequence τ , with $|\rho| = |\tau|$. Let $\phi_1, \phi_2, \dots, \phi_m$ be an enumeration of $\mathcal{SF}^*(\phi)$ respecting the $\prec$ order. Simply, we deem ϕ_j ($1 \leq j \leq m$) is with index j and the index can be referred to via $idx(\phi_j)$. We maintain two Boolean arrays $prev[1, \dots, m]$ and $cur[1, \dots, m]$ to record the interpretation of formulas in $\mathcal{SF}^*(\phi)$ at two successive states. Under the runtime monitoring scenario, the algorithm is required to output the interpretation of ϕ at each state. With dynamic programming approach, the monitoring algorithm outputs the verdict of all subformulas at a world following the $\prec$ order; and it outputs the verdict for all worlds incrementally from the beginning to the end. That is, with the results of world $i - 1$ stored in $prev[1, \dots, m]$, we can calculate the results of world i and keep them in $cur[1, \dots, m]$. Array cur is updated with $prev$ at the start of next iteration, and the iteration stops when reaching the end of the trace.

For quantified counting subformulas, we assume they are with TLI-monitorable univariate arithmetic relations and/or TLI-monitorable multivariate relations as specified in Def. 5.14. In this case, we need to maintain the lower bound and period for each univariate relation and every slice of the multivariate relations. We also assume all the quantified subformulas in $\mathcal{SF}^*(\phi)$ to be variable-isolated. Since each

variable occurs in exactly one relation, we use variable names to index the set of lower bounds and periods. These information is stored in array tli , whose size is equivalent to the number of variables in the formula. For variable x , the index can be referred via $idx_v(x)$, and its lower bound and period are $tli[idx_v(x)].lb$ and $tli[idx_v(x)].pd$. From the recursive semantics shown in Theorem 5.20 and Theorem 5.22, we know it is necessary to keep track of the valuation determined by counting quantifiers. For counting quantifiers in the subformulas, the corresponding valuations for the current and previous state are stored in arrays $c\_cnt[1, \dots, l]$ and $p\_cnt[1, \dots, l]$. Here l is equivalent to the number of interval-skewed quantifiers of all quantifiers in ϕ . Two counting quantifiers are deemed to be equivalent if they are with the same interval, variable, reset condition and target formula. For example, the valuation of $\mathcal{C}_I x : \langle \alpha, \beta \rangle$ can be accessed via $c\_cnt[idx_c(\mathcal{C}_I x : \langle \alpha, \beta \rangle)]/p\_cnt[idx_c(\mathcal{C}_I x : \langle \alpha, \beta \rangle)]$. Updating on the valuation arrays follows Theorem 5.20 and Theorem 5.22, and is done as a first step (see line 17-36 in Algo. 6) when dealing with quantified subformulas.

The main monitoring algorithm (Algo. 4) can be divided into two steps on the top level: the initialization procedure (Algo. 5) and the for-loop calling iterative procedure (Algo. 6). The initialization procedure works to initialize the cur and c_cnt arrays at the first state and the iterative procedure will help to maintain the arrays as the state progressing. In the pseudocode of the algorithms, we overload some logical symbols to denote operators on Boolean values. It is straightforward to see that, once the formula to be monitored is fixed, the space required to run the algorithm does not grow with the length of traces. Moreover, the correctness of our monitoring algorithm is stated in Theorem 5.26, with auxiliary Lemma 5.23, Lemma 5.24, and Lemma 5.25. Proofs of them are included in Appendix A.5. The worst case time and space complexity of the monitoring algorithm is discussed and proved in Theorem 5.27.

Lemma 5.23. *Given a model (ρ, τ, ν) with $(|\rho| = |\tau|) \geq 1$, after procedure $Init(\rho, \nu, \phi, cur, c\_cnt)$, for all $\phi_k \in \mathcal{SF}^*(\phi)$, we have $1 \models \phi_k$ iff $cur[k] \equiv true$.*

Lemma 5.24. *Given a model (ρ, τ, ν) with $(|\rho| = |\tau|) \geq 1$, for all $i \geq 2$, if we assume $i - 1 \models \phi_k$ iff $cur[idx(\phi_k)] = true$ for any $\phi_k \in \mathcal{SF}^+(\phi)$, then after*

executing $Iter(\rho, \tau, \nu, i, \phi, prev, cur, p\_cnt, c\_cnt)$, for any $\phi_k \in \mathcal{SF}^*(\phi)$, we have $i \models \phi_k$ iff $cur[k] \equiv true$.

Lemma 5.25. *Given a model (ρ, τ, ν) with $(|\rho| = |\tau|) \geq 1$, after executing $Monitor(\rho, \tau, \nu, i, \phi)$, for all $\phi_k \in \mathcal{SF}^+(\phi)$, we have $i \models \phi_k$ iff $cur[k] = true$.*

Theorem 5.26. *Given a model (ρ, τ, ν) , world $i \geq 2$, and a TLI-monitorable formula ϕ , the procedure $Monitor(\rho, \tau, \nu, i, \phi)$ returns true iff $i \models \phi$.*

Theorem 5.27. *Given a model (ρ, τ, ν) , the procedure $Monitor(\rho, \tau, \nu, i, \phi)$ takes space $m + \log(b + T)_{max}^n + \log(max(b_{max}, T_{max}))^k$ and time $|\rho|ml$, where $m = |\mathcal{SF}^*(\phi)| \leq |\mathcal{SF}^+(\phi)| + (b + T)_{max}^k$, $n = \sum_{\phi_C} k$, $k = |\mathcal{SF}(\phi)|$, and $l = max(\phi_C)$.*

Proof. As can be seen from the monitoring algorithm, we need to store the array of the truth value of all subformulas of ϕ , the array of valuations of all the interval-skewed quantifiers, and the array of the lower bound and period of all the counting quantifiers occurring in ϕ . These are the three parts in the space complexity. Here k is number of usual subformulas of ϕ , m is the number of all subformulas of ϕ including not only the interval-skewed subformulas of the $\mathcal{S}$ formulas but also the instantiated subformulas of counting formulas. To record the valuations of counting quantifiers, we need an array of size n , where n is the number of all the interval-skewed quantifiers and it is calculated via summing up all the maximum numeric constants associated to each occurrence of $\mathcal{C}_I$ in ϕ . For example, if $I = [a, d]$, then the constant is d ; if $I = [a, +\infty)$, then a . Particularly, the valuation is bounded by the number of corresponding equivalence classes, which is $b + T$ as shown in Prop. 5.18, and here we set it as the maximum one. We also need to store the lower bounds and periods for all variables used in ϕ . The size of the array is bounded by k and the element size is bounded by the maximum among all lower bounds and periods.

For the time complexity, we can see from the algorithm that the maximum nesting level of loops is three, and the number of iterations are $|\rho|$, m and l , where l is the maximum number of interval-skewed counting quantifiers of a particular counting quantifier occurring in ϕ . □

5.5 Case Studies

Runtime monitoring provides effective realtime understanding and protection of the target system. Policies used in realtime monitoring may involve counting or frequency of some events. In this section, we first summarize the policies used to detect adversarial attacks for RNN, which are defined based on the MDP. Then, we list some runtime monitoring applications in areas like anomaly detection, performance bug analysis and hardware-level malware detection. Afterwards, we study and present some concrete security policies for runtime Android malware detection.

Adversarial attack is a major concern for the deployment of DL systems in safety- and security-critical applications. Here we propose several policies to detect adversarial attacks of RNN based on the robustness-aware abstract model MDP, as introduced in Chapter 4. Given a sample, we execute it with the RNN and monitor its concrete execution trace. With the input/state/output abstraction functions, each transition in the concrete execution trace can be mapped to an abstract transition in the MDP and get its robustness score with the labeling function. Here we deem a transition with robustness score less than a certain threshold (*e.g.*, 0.5) as unrobust, and denote the event as V , otherwise we denote the event as R . We monitor the number of continuous unrobust transitions triggered in the past, and if the number is greater than a threshold (*e.g.*, 5), we raise an alarm for suspicious adversarial sample. The policy is formally presented as

$$\Box \neg [\mathcal{C}x : \langle R, V \rangle. (x > 5)].$$

Apart from the adversarial sample detection of RNN applications, we also provide a policy specification for CNN applications, as inspired by the research of Ziv and Yuval [129]. They observed the activation space of each hidden layer in CNN, and calculated a priori probability of classification for each layer based on its activation space. They found that, for normal samples, the decision sequence along the layers tends to alter frequently at the beginning but becomes stable at the rear layers. However, for adversarial samples, the decisions made by rear layers keep changing,

i.e., will not converge to a stable decision. Here, we specify a policy as

$$\Box \neg [\mathcal{C}y : \langle \mathcal{C}x : \langle \perp, \top \rangle . (x < 7), F \rangle . (y > 5)].$$

Intuitively, we count how many times the decision flipping happens after the 7th layer, and if the count exceeds 5, we raise an alarm. Here, the inner counter counts how many decisions has been made (*i.e.*, after how many hidden layers), and acts as the reset condition of the outer counter. The outer counter will be reset to zero as long as the inner counter does reach 7. In such a way, only flippings happening at the rear layers are included.

Anomaly detection is extensively used in cyber physical systems (CPSs) [130] and network intrusion detection [131]. In the anomaly detection scenario, various metrics are used to build the norm profile of the target system in order to detect anomalies. Frequency is among one of the most widely used metrics. Concerned frequencies include link utilization, CPU usage, login failure, as well as traffic rate and the rate of connections.

DoS attacks are the most common one [132–134] against intra-vehicle bus in automobile systems. A Controller Area Network (CAN) is a vehicle bus standard designed to allow microcontrollers and devices to communicate with each other in applications without a host computer. As shown in Table 5.1, each standard CAN frame consists of the following fields, 1-bit start of frame (SOF), 11-bit identifier (ID), 7-bit control field including data length code, up to 64-bit data field (Data), 16-bit CRC checksum, 2-bit ACK field and 7-bit end of frame (EOF). Each frame has a unique id and carries different information for such as acceleration, braking, engine revolutions per minute (RPM) and odometer readings. Frame ID is also used for arbitration when transmission conflicts, where the one with smaller ID always wins. That means, if an adversary injects smaller ID frames in a high frequency, it is possible to force transmission pending of normal frames (DoS attacks). Given the fact that most CAN traffic is transmitted at a fixed rate [135], [136] proposed a frequency check against anomaly communication on CAN bus by comparing historical statistics with the current data. Hyun Min Song et al. [137] analyzed

TABLE 5.1: Standard CAN Frame Format

SOF	ID	Control	Data	CRC	ACK	EOF
1 bit	11 bits	7 bits	0-64 bits	16 bits	2 bits	7 bits

the time interval of single id traffic to detect anomaly changes when frames were injected into CAN bus.

Performance bugs [138, 139] are drawing more and more attention recently. Different from traditional functional bugs, performance bugs degrade performance and waste computation resource. GUI lagging [139] is one of the most commonly found performance bugs, where user events cannot be handled in a timely manner. A convenient and effective approach [140] to identify potential GUI lagging bugs is to measure the number of frames a graphics-rendering unit can produce per second. When the target application, *e.g.*, Firefox explore, scores less than 60 frames per second, it is likely that there exist GUI lagging bugs and further optimizations have to be made.

RowHammer [141] is a kind of disturbance error found in dynamic random-access memory. Physically, it results from the increasingly denser memory cells and smaller cell isolation margins, and in this case, higher-than-natural access rate of certain memory rows can possibly cause bit flips in adjacent rows. The first exploit of RowHammer was released by Google’s Project Zero team in [142], which is able to gain kernel privileges. Recently, Gruss *et al.* [143] demonstrated an JavaScript-based exploit that can launch Rowhammer attack with the eviction rate up to 99.99%. This is also the first remote software-induced hardware-fault attack. A countermeasure to detect suspicious RowHammer attack with hardware performance counter is proposed by [144] in 2015. The core technique is to observe high numbers of last-level cache (LLC) miss events within short periods of time.

Android malware is a major threat to the ecosystem [145, 146], and sometimes difficult to identify with current static detection approaches [147, 148]. However, runtime monitoring techniques can help to capture stealthy behaviors during the execution. Here, we put more attention on several security policies extensively discussed in the Android system and some concrete policies are provided as case

studies for MTL_{cnt} . In the rest of this chapter, we assume the following atomic propositions in Android OS.

- S_i (or E_i) : the app with UID i starts to run (or stops running).
- M_i : the app with UID i sends a message.
- I_i : the app with UID i opens an Internet connection socket.
- F_i : the app with UID i forks a new child process.

The following policies refer to the malicious access patterns that are forbidden in Android systems. As for the bounded time interval and the limit on a specific count variable, we just give a rough estimation for illustration purpose. The unit adopted for the time intervals is second. Every policy given below takes the form $\Box \neg \phi$ where ϕ describes a violation of the policy at a given time. The prefix $\Box$ ensures that globally such a violation does not occur.

1. SMS-based Botnet Attack

$$\Box \neg [\mathcal{C}_{[0,1800)}x : \langle \perp, M_i \rangle . (x > 30)]$$

This policy describes that an app with UID i cannot send more than 30 SMS messages within any 30 minutes. This is a requirement for the SMS sending rate from Android 4.1 [149], and for older systems the limit is 100 SMS messages per hour. Arzt *et al.* [150] also suggest stopping unintended SMS transmissions by limiting the SMS sending frequency. In [151], authors find that current Android botnets are exploiting SMS messages to gather money by sending SMS to premium-rate numbers. With the specified policy, it helps to discover possible tainted Android bots.

2. DDos Attack

$$\Box \neg [\mathcal{C}_{[0,3)}x : \langle \perp, I_i \rangle . (x > 5)]$$

This policy specifies that an app cannot open the Internet connection socket for more than 5 times in 3 seconds. If an Android app aims at flooding a

targeted server to launch a DDoS attack, it usually tries to open massive Internet connections. This policy can help to control the amount of Internet connections, thus preventing some potential malware.

3. RageAgainstTheCage

$$\Box \neg [\mathcal{C}x : \langle S_i, F_i \wedge \neg E_i \rangle . (x > 2^{16})]$$

This policy says that during the life cycle of an app, it is not allowed to create more than 2^{16} child processes to exhaust the domain of *pid*, *i.e.*, the process identifier, in Linux kernel. *RageAgainstTheCage* [152] is a well-known exploit on Android, which can perform unauthorized privileged actions by gaining the root access. This malware uses a vulnerability in Android kernel to get the root privilege by keeping forking the child process to 2^{16} . With this policy, a constraint is set on how many child processes an app can fork during a single run.

5.6 Evaluation

5.6.1 Application on Android

We implement the monitoring algorithm for MTL_{cnt} and evaluated it on LogicDroid platform [128], which is a modified version of Android 4.1. Instrumentation on IPC (Inter-process communication) calls, like opening Internet socket, sending SMS and accessing contact database, is deployed in LogicDroid. These IPC calls compose the event trace, and whenever they happen during the system execution, the monitoring module in the kernel is notified and runtime examination will proceed. Aiming to demonstrate the trace-length independence property, we measure the time used during the monitoring. We also measure the overhead caused by the RM mechanism under different event throughput and it turns out the delays are negligible for normal use.

For the benchmarks used, since hooks on the start or end of a process or an app have not been implemented in LogicDroid, some of the reset conditions in the example policies are not applicable. Policy 3) in Section 5.5 is also excluded from the evaluation due to the lack of support for process forking instrumentation in LogicDroid. Consequently, the benchmark policies used in our experiments are:

$$\text{P1: } \Box \neg [\mathcal{C}_{[0,1800)}x : \langle \perp, M_i \rangle . (x > 30)]$$

$$\text{P2: } \Box \neg [\mathcal{C}_{[0,3)}x : \langle \perp, I_i \rangle . (x > 5)]$$

$$\text{P3: } \Box \neg [\mathcal{C}_{[0,30)}x : \langle \perp, I_i \rangle . (x > 5)]$$

$$\text{P4: } \Box \neg [\mathcal{C}_{[0,300)}x : \langle \perp, I_i \rangle . (x > 5)]$$

$$\text{P5: } \Box \neg [\mathcal{C}_{[0,3)}x : \langle \perp, I_i \rangle . (x > 50)]$$

$$\text{P6: } \Box \neg [\mathcal{C}_{[0,3)}x : \langle \perp, I_i \rangle . (x > 500)]$$

P7: $\Box \neg [\mathcal{C}_{[0,3)}x : \langle \perp, M_i \rangle . \mathcal{C}_{[0,3)}y : \langle \perp, I_i \rangle . K(x, y) > 0]$ where function K is defined as follows:

$$K(x, y) = \begin{cases} 3x - 4y & \text{if } x < 4 \text{ and } y < 5, \\ K(x - 3, y) & \text{if } x \geq 4 \text{ and } y < 5, \\ K(x, y - 2) & \text{if } x < 4 \text{ and } y \geq 5, \\ K(x - 3, y - 2) & \text{otherwise.} \end{cases}$$

Note that each projection of function K becomes periodic once $x \geq 1$ and $y \geq 3$, so it is easy to show that φ_K is TLI-monitorable.

Note that in the implementation in LogicDroid, the runtime monitor is used to prevent execution that could lead to violation of a given security policy. This means that in practice, at every stage, the runtime monitor can assume that no violation of the given policy has occurred in the past. In this setting, the job of the runtime monitor, when enforcing a given policy of the form $\Box \neg \phi$ in the current world and given an event generated by the app (*i.e.*, a request to some resource), is

TABLE 5.2: Details of Policies for Android

Policy	Module Size(KB)	$ \mathcal{SF}^*(\phi) $	$\max(\phi_c)$	Quantifiers		
				variable	lb	pd
P1	407	35	1800	x	31	1
P2	405	10	3	x	6	1
P3	405	10	30	x	6	1
P4	405	10	300	x	6	1
P5	408	55	3	x	51	1
P6	439	505	3	x	501	1
P7	407	28	3	x	1	3
				y	3	2

to check whether the formula ϕ would be satisfied or not in the (hypothetical) next world where the event is allowed to happen. If ϕ would be satisfied in this next world, which means the policy would be violated, the monitor would terminate the request. So in practice, only the truth value of ϕ needs to be computed, and the truth value of $\Box\neg\phi$ is implicit in the way the runtime monitor is implemented and deployed.

The first two policies P1 and P2 are those introduced in § 5.5, and others are artifacts used to evaluate the effectiveness of our approach, as interval ranges and arithmetic relations (with different *period* and *lower bound*) may affect the performance. We summarize key characteristics of the policies according to the metrics used in the complexity analysis, including the number of subformulas, the max number of interval-skewed counting quantifiers of a particular counting quantifier, and the lower bound and period of corresponding variables, as listed in column 3-7 in Table 5.2. The policies cover a wide range of each complexity metrics, e.g., the number of subformulas varies from 10 to 505. We also include a multivariate arithmetic relations as in P7.

The monitor of each policy is implemented in C language according to the algorithm described in § 5.4 and is compiled as an Android kernel module. For quantified variables in the tested policies, the calculation of its corresponding *period* and *lower bound* is currently done manually. Since there is no dynamic memory allocation in the monitoring algorithm, the total memory occupied by a monitor is just the size of the monitor after loaded to the kernel, as shown in the second column in Table 5.2. There is no obvious expansion in the size

FIG. 5.1: Time of Single Round Checking on Android

of the monitor as the number of subformulas increases by dozens of times. The overall memory in Android emulator is around 840MB, in this case, the memory overhead of the monitor is just around 0.5%. We run LogicDroid on an emulator and a prepared kernel module is inserted to the system once its initialization is finished. We measure the performance of each policy separately, which means only one kernel module is inserted at each experiment. The monitor module is aware of every occurrence of the instrumented events, and is able to continuously and simultaneously check whether the system behavior so far complies with the policy stipulated. To facilitate the experiment, we also implement a fuzzy app to trigger IPC calls randomly at user-specified throughput.

All the experiments are conducted on a PC with 64-bit Ubuntu 16.04LTS system, 16GB RAM and a 3.50GHz Intel Xeon(R) CPU E5-1650 v3. Our implementation and the modules used in this section are all available on [153]. We first measure the time used by each iteration of the for-loop as shown in Algo. 4, and we call it iteration time. The fuzzy app is set to send an IPC call every 0.1s and we show the results when the event trace length grows to 10,000. For each policy, we record the time used for each iteration and calculate the average time for every 1000 calls. Note that the time measured here is the pure execution time of the monitor excluding that used to finish the user requested tasks or switch between

TABLE 5.3: Time Overhead under Different Throughput for Android

Event Interval (ms)	Overhead(%)						
	P1	P2	P3	P4	P5	P6	P7
0	8.77	0.72	1.53	4.78	4.21	1.48	1.69
1	3.59	1.28	1.69	3.15	2.54	1.23	1.43
10	0.07	-0.14	-0.24	-0.42	-0.27	0.15	0.04

user mode and kernel mode. To get a statistically significant result, we take the average of five times' measurement for all the experiments. From the results shown in Fig. 5.1, we can see that the iteration time for each policy keeps constant as trace length increasing, which is a shred of strong evidence for the trace-length independence. The y-axis is in logarithm scale. Policies P2, P3 and P4 are with increasing max instants by the magnitude of 10. Policies P2, P5 and P6 are with increasing number of subformulas resulting from the increasing lower bounds, and the magnitude is also around 10. For both cases, the increase in examination time is finally around 5 times as can be seen from the figure. Among these seven policies, P1 requires the longest examination time, approximately 500 microseconds, mainly due to the max number of interval-skewed quantifiers. We also conduct a contrast experiment as a baseline, where the event trace is recorded by the monitor for future queries. It is not implemented as a specific policy but only to record and query, which is the basic functionality in a naive monitoring algorithm. The UIDs of the caller and callee are stored as integers and the timestamps are stored as a long integer in the memory. Whenever the trace grows, we measure the time used to traverse the whole history. The result is shown as the *BL1* line in Fig. 5.1. We can see that when the trace length grows from 0 to 10,000, the time used for each iteration roughly increases by 8 times. This is a significant defect for realtime runtime monitoring.

We also measure how much time overhead is introduced after adding the monitoring mechanism to Android. Since different IPC calls may take different time to finish, instead of triggering several IPC calls randomly, here we just trigger Internet socket requests. The fuzzy app is set to open Internet socket every 0, 1, and 10 milliseconds respectively for 10,000 times. We measure the time used to finish this task by the emulator with and without the monitoring module and the

overhead results are shown in Table 5.3. For the largest throughput, the worst performance is observed for P1, 8.77%, which is rarely spotted by users. When the interval between events drops to 10 milliseconds, the time overhead becomes less than 0.5%. Here are several negative overheads due to some minor deviation in time measurement. It also reflects that the overhead is extremely low and maybe affected by measurement error. In normal usage of mobiles, the events throughput is even smaller, and we can claim that the overhead of our monitoring mechanism is negligible.

5.6.2 Application on Autonomous Vehicle

Our intrusion detection system for the autonomous vehicles on the CAN bus has been built on Robot Operating System (ROS) with Ubuntu 16.04. We use USB-CAN adapters to build a real CAN bus system for evaluation. In our setup, the CAN bus system connects a computer running the autonomous driving suite and issues control commands over the CAN bus to the corresponding electronic control units (ECUs), and ECUs could send feedback to the driving system via the bus. In the meantime, our monitor links to the CAN bus to detect any abnormal traffic. It sniffs CAN frames with libpcap and checks the communication features of each frame ID. To detect any DoS attacks, the monitor counts and compares the frequency of each CAN frame traffic with a threshold periodically for any sudden increasing. F_i is used to denote the CAN frame with ID i and the policies used in the evaluation are as follows:

$$\text{P8: } \Box \neg [\mathcal{C}_{[0,10)}x : \langle \perp, F_i \rangle . (x > 20)]$$

$$\text{P9: } \Box \neg [\mathcal{C}_{[0,50)}x : \langle \perp, F_i \rangle . (x > 20)]$$

$$\text{P10: } \Box \neg [\mathcal{C}_{[0,250)}x : \langle \perp, F_i \rangle . (x > 20)]$$

$$\text{P11: } \Box \neg [\mathcal{C}_{[0,10)}x : \langle \perp, F_i \rangle . (x > 100)]$$

$$\text{P12: } \Box \neg [\mathcal{C}_{[0,10)}x : \langle \perp, F_i \rangle . (x > 500)]$$

FIG. 5.2: Time of Single Round Checking on Autonomous Vehicle

TABLE 5.4: Details of Policies for Autonomous Vehicle

Policy	Size(B)	$ \mathcal{SF}^*(\phi) $	$\max(\phi_c)$	Quantifiers		
				variable	lb	pd
P8	336,472	25	10	x	21	1
P9	336,472	25	50	x	21	1
P10	336,472	25	250	x	21	1
P11	336,476	105	10	x	101	1
P12	336,500	505	10	x	501	1

The monitor is implemented as an independent process and just “listening” to the CAN frames on the CAN bus, which means that it has no effect on the timing of CAN communication. Hence, it makes no sense to investigate the delay caused by the monitor under different throughput, and we only look into its performance with different policies. The time intervals are in millisecond for high precision measurement. Policies P8, P9 and P10 are with increasing time interval (by 5 times), and policies P8, P11 and P12 are with increasing lower bound (by around 5 times). Detailed characteristics of the policies and memory usage are available in Table 5.4. This is designed to measure the performance of policies under different complexity. We calculate the average time used for every 1000 CAN frames, and the results are shown in Fig. 5.2. From the results we can see, the size of memory taken is largely related to the number of subformulas, but the increment is slow compared with the increment of subformulas. As there is no allocation of extra memory, the memory usage by the monitor itself would remain at this amount,

which is negligibly small. Basically, the monitor takes 1 to 5 milliseconds to do a single round checking, which is low enough for monitoring. When the size of the time interval increases by 25 times, the time spent increases by around 3 times. When the number of the subformulas increases by 25 times, the time spent increases by around 5 times. The overall increment is slow, which means the monitor can potentially provide good performance for even more complicated policies. Similar to the previous evaluation on Android, we implement a baseline monitor which records each frame with its ID and timestamp, and simply traverses through the entire trace for each new world, i.e., not implementing any specific policy. As shown in the *BL2* line in Fig. 5.2, the time increases rapidly as the trace length grows. With more and more delays, the monitor fails to react in a realtime manner.

5.7 Related Work

Instead of listing all the related works, we discuss the most related ones in this section, which will cover a brief history of trace-length independent RM and also a bunch of specification languages with counting semantics but unfortunately fail to have a trace-length independent monitor.

TLI monitors. The concept of trace-length independence is proposed in 2013 [78], where a TLI monitoring algorithm based on spawning automaton for a first-order temporal logic, LTL^{FO} , is devised. Actually, there have been some works provided with TLI property before that. In 2002, [77] gives a TLI solution for the monitoring of past-time LTL with dynamic programming. Later, the same problem is discussed in terms of MTL, the metric temporal extension of LTL. Thati and Roşu [125] present a TLI monitoring algorithm for MTL with both past and future fragments, but restraining the interpretation of future to truncated finite traces. Their algorithm is also designed with dynamic programming based on the derivatives of formulas. The time and space complexity is linear to the size of the formula for MTL with only past time operators, and exponential for full fragment MTL. TLI monitoring algorithms for ptMTL with pointed-based and interval-based time models are discussed in [121]. For the point-based ptMTL semantics, their

algorithm provides a better performance on time complexity compared to that in [125]. However, the list of timestamps used in their algorithms dealing with $\mathcal{S}$ operator can grow boundlessly large when the time sequence is not strictly increasing, that is, it is dependent on the event rate. This problem also exists in the monitor of full-fragment MTL proposed in [154]. This restriction on time sequence is released in [128], where a TLI monitoring algorithm for ptMTL extended with recursive definition is designed while restricting all the time intervals bounded with the temporal operators to starting from 0. Recently, there also arises some discussion on event-rate independent monitoring MTL [127] and metric dynamic logic [155].

Counting Semantics. There are some works focusing particularly on the counting semantics, and also some others on general aggregation semantics. The most related counting quantifier is proposed in [119] but their algorithm is PSPACE-complete in the size of history length. Laroussinie *et al.* [156] present a quantitative extension of LTL, called CLTL, allowing to specify the number of states satisfying certain sub-formulas. The semantics is the same with ours on the non-metric part. According to their investigation, even though CLTL formulas can be translated into classical LTL, an exponential blow-up in formula size is inevitable. As for the satisfiability and model-checking problem of CLTL, it turns out to be EXPSPACE-complete, and PSPACE-complete when restricting to a fragment. Actually this fragment belongs to set of the TLI-formulas stipulated in our work, for which the valuation of the term will grow monotonously with the count increasing.

Here are some works extending specification languages with general aggregation semantics. Basin *et al.* [120] extend metric first-order temporal logic (MFOTL) with aggregation operators, like SUM, CNT, MAX and AVG, and propose a monitoring algorithm where policies are translated to an extended relational algebra and checked via SQL-like processing. SOLOIST [123] is a many-sorted first-order metric temporal logic extended with new temporal modalities that support aggregate operators for events occurring in a certain time window. On its monitoring problem, Bianculli *et al.* [157] and Bersani *et al.* [158] give some solutions, but all are dependent on SMT-solver, and evaluations in these two

works show that increasing time and memory are needed when trace length grows. Other specification languages with support for different kinds of aggregations are LarvaSat [159], LOLA [160], as well as rule-based EAGLE [161], RULER [162] and LOGFIRE [163], and also one based on algebraic alternating automata [164]. However, monitoring algorithms for the above languages all need to record the specific count values, even though most of them avoided storing the entire trace history. In principle, these counts can increase indefinitely, so the space required is not constant.

To the best of our knowledge, this is the first work on TLI monitoring algorithms for logics equipped with metric temporal counting quantifier. For readers interested in exploration on trade-off between the expressiveness and monitoring efficiency, we provide further references on monitoring with parameters [165, 166]. Specifically, the parameters may include data quantification [167–169], timed pattern matching [170, 171], or both data and time parameters [172].

5.8 Chapter Summary

We presented a formal policy specification language MTL_{cnt} that allows expressions of temporal quantitative policies. We considered the questions of when a formula is trace-length independent monitorable. For univariate relations, we obtained sufficient and necessary conditions for the relations to be TLI-monitorable. We then discussed an extension to the multivariate relations. Assuming the relations are all TLI-monitorable, we constructed a TLI monitoring algorithm for MTL_{cnt} .

Chapter 6

Conclusion and Future Work

This chapter summarizes the research work that we have completed in the thesis, and discusses our on-going and future research directions.

6.1 Summary of the Thesis

Artificial intelligence has recently achieved tremendous success in a variety of cutting-edge applications. Particularly, deep learning is the key technology driving this boom. The key reason preventing deploying deep learning on safety- and security-critical applications is the lack of security and robustness. These challenges have motivated researchers to investigate on testing, verification and attack detection of feed-forward neural networks, but recurrent neural network (RNN)-based stateful DL systems have not been explored.

This thesis initiates the first major effort on the security analysis for RNN using a model-based approach. To further protect the DL system, we propose an online monitoring approach to protect DL systems against attacks at runtime, based on the models extracted.

First, we model RNN as a Discrete-Time Markov Chain (DTMC) model to characterize its internal behaviors. Based on the extracted abstract DTMC model, we design two trace similarity metrics and five coverage criteria which enable the

quantitative analysis of RNNs to quantitative measures for adversarial sample detection and coverage-guided test generation, respectively. The experimental results demonstrate that the proposed DTMC model is effective in capturing the internal behaviors of RNNs, and confirm that (1) the similarity metrics could effectively capture the differences between samples even with very small perturbations and (2) the coverage criteria are useful in revealing erroneous behaviors.

Secondly, to study the robustness of RNN, we extend our model to Markov Decision Process (MDP) for RNN as an input-aware probabilistic abstract state transition system. Robustness depicts how vulnerable the RNN is under minor input perturbations, thus the input awareness is of great importance. The model learning process starts with an abstract model, for which there could exist some inconsistency between the ‘real’ robustness value. The real value may not be accessible, so we rely on mutation testing to obtain an approximation. With such oracle, we are able to refine the rough MDP model to get it close to the real model. With the learned accurate model, we are able to statically analyze and identify the vulnerable locations of the RNN for a given input. We demonstrate the usefulness of the abstract model with applications to more efficient attack generation.

Thirdly, we rely on runtime monitoring as the last line of defense against adversary during the runtime, since even known defects in DL systems are difficult to fix. We specify properties over the extracted abstract models with temporal logic to detect anomaly behaviors. To guarantee the expressiveness and efficiency at the same time, we propose a logic specification language, MTL_{cnt} , and derive a complete and efficient runtime verification algorithm with trace-length independent performance. With trace-length independence (TLI), the monitoring cost does not increase as the execution trace grows, which is crucial for monitoring on devices with restricted computation and storage resources. To demonstrate its usefulness in specifying security policies, we have conducted a wide spectrum case studies on RNN adversarial example detection, mobile malware detection and AV anomaly detection. Further, we implement and test our algorithm on both Android platform and AV platform.

6.2 Future Work

This thesis starts the first step towards quantitative behavior analysis and robustness quantification of RNN based on formal modeling. We firmly believe it opens a new research direction on white-box analysis for deep learning systems. There are several immediate topics to continue as discussed below.

1. As an immediate future work, more effective security policies to detect adversarial behaviors of DL systems will be studied and extensive evaluations of the runtime monitoring algorithm implemented for DL systems will be conducted.
2. Currently, our robustness analysis is to identify the weakness of the RNN. One possible continuation along this topic could study how to enhance the robustness of the RNN after the detection.
3. Our based-based analysis focuses only on the security and robustness of the RNN models. With the capability of precisely modeling of RNN, we can study other interesting properties effortlessly, examples like fairness and performance. This is because we can track and trace how the training or testing data are executed during the system with the help of the abstracted model. This white box understanding of the runtime behavior will enable various analysis.
4. For the analysis of the stateful behaviors, researchers have also made significant progress on the summarization [173–175] and invariant analysis [176, 177] of the loop structures in traditional program. It is worth exploration on how these advances can help to provide a better understanding of RNNs.
5. We want to extend this research to the modeling and analysis of deep reinforcement learning (DRL). Because DRL has a stateful nature as RNN. The challenge will be the modeling of the environment and their interactions with the DRL system, which will create one more dimension of complexity.

Appendix A

Appendix

A.1 Proof of Proposition 5.18

Proof. ($\Leftarrow$) Supposing $e \geq b_i + T_i$, accordingly we have

$$[e] = \{a \mid (a - b_i) \bmod T_i + b_i = e, a \geq b_i, a \in \mathbb{N}\}.$$

According to $(a - b_i) \bmod T_i + b_i = e$ and $a \geq b_i$ we can get

$$\begin{aligned} (a - b_i) \bmod T_i &\in [0, T_i) \\ \Leftrightarrow (a - b_i) \bmod T_i + b_i &\in [b_i, b_i + T_i) \\ \Leftrightarrow e &\in [b_i, b_i + T_i) \end{aligned}$$

This is contradictory to $e \geq b_i + T_i$, so $\nexists a \geq b_i. (a - b_i) \bmod T_i + b_i = e$, i.e., $[e]$ is empty.

($\Rightarrow$) Given $[e]$ is empty, according to the definition of $[e]$,

$$[e] = \begin{cases} \{e\} & \text{if } e < b_i, \\ \{a \mid (a - b_i) \bmod T_i + b_i = e, a \geq b_i\} & \text{otherwise.} \end{cases}$$

since $e \notin [e]$, we can get $e \geq b_i$, such that

$$\begin{aligned}
[e] &= \{a \mid (a - b_i) \bmod T_i + b_i = e, a \geq b_i, a \in \mathbb{N}\} = \emptyset \\
&\Leftrightarrow \forall a \geq b_i. (a - b_i) \bmod T_i + b_i \neq e, \\
&\Leftrightarrow \forall a \geq b_i. (a - b_i) \bmod T_i \neq e - b_i \\
&\Rightarrow e - b_i \geq T_i, (e > b_i) \\
&\Rightarrow e \geq b_i + T_i.
\end{aligned}$$

Now the proposition follows. $\square$

A.2 Proof of Theorem 5.21

Proof. In this proof, the valuation of x determined by $\mathcal{C}_I x : \langle \alpha, \beta \rangle$ is analyzed under the two cases of whether $0 \in I$. Since we assume the counting formula is TLI-monitorable, x is used either in a TLI-monitorable univariate relation whose characteristic function is lb-periodic on $[b, +\infty)$ and with period T , or a TLI-monitorable multivariate relation for which the characteristic functions of slices on x are lb-periodic on $[b, +\infty)$ and with period T . According to the semantics of metric temporal counting quantifier defined in Section 5.1, for $\mathcal{C}_I x : \langle \alpha, \beta \rangle$ we denote

$$\begin{aligned}
\mathbb{B} &= \{h \mid h \models \beta, \text{ for } \tau_1 - \tau_h \in I, m < h \leq 1\}, \\
\mathbb{A} &= \{h \mid h \models \alpha, \text{ for } \tau_1 - \tau_h \in I, 1 \leq h \leq 1\}, \\
l &= |\mathbb{B}|, \\
m &= \max(\mathbb{A} \cup \{0\}).
\end{aligned}$$

Thus we only need to prove l always belongs to the equivalence class $[\mathbf{r}(1, \mathcal{C}_I x : \langle \alpha, \beta \rangle)]$.

Case(1): If $0 \notin I$, then

$$0 \notin I \Rightarrow \tau_1 - \tau_1 \notin I \Rightarrow 1 \notin \mathbb{B} \Rightarrow \mathbb{B} = \emptyset.$$

We can conclude that

$$\mathbb{B} = \emptyset \Rightarrow l = 0.$$

Hence, $l \in [\mathbf{r}(1, \mathcal{C}_I x : \langle \alpha, \beta \rangle)]$.

Case(2): If $0 \in I$, cases are analyzed based on the satisfaction of $1 \models \beta$ and $1 \models \alpha$.

For case(a),

$$1 \models \alpha \Rightarrow 1 \in \mathbb{A} \Rightarrow m = 1(0 \in I) \Rightarrow \mathbb{B} = \emptyset \Rightarrow l = 0$$

Thus, $l \in [\mathbf{r}(1, \mathcal{C}_I x : \langle \alpha, \beta \rangle)]$.

For case(b),

$$1 \not\models \alpha \Rightarrow \mathbb{A} = \emptyset \Rightarrow m = 0,$$

and

$$1 \not\models \beta \Rightarrow 1 \notin \mathbb{B} \Rightarrow \mathbb{B} = \emptyset \Rightarrow l = 0.$$

Hence, $l \in [\mathbf{r}(1, \mathcal{C}_I x : \langle \alpha, \beta \rangle)]$.

For case(c) and case(d), it is easy to get

$$1 \not\models \alpha \Rightarrow m = 0,$$

and

$$1 \models \beta \Rightarrow 1 \in \mathbb{B}(0 \in I) \Rightarrow \mathbb{B} = \{1\} \Rightarrow l = 1.$$

Following the definition of equivalence class in Def. 5.17, we can easily know $l \in [\mathbf{r}(1, \mathcal{C}_I x : \langle \alpha, \beta \rangle)]$.

In conclusion, the theorem follows for the cases above. $\square$

A.3 Proof of Lemma 5.23

Proof. We assume all subformulas in $\mathcal{SF}^+(\phi)$ are well-ordered. The procedure *Init* will assign verdict to the subformulas respecting the partial order, i.e., for any $\phi_i \prec \phi_j$, ϕ_i will be valuated before ϕ_j . For each subformula, we conduct inductive analysis based on its structure. For ϕ_k :

(Base cases)

Case $\perp$: Obviously $1 \not\models \perp$ and with $cur[k] = false$, we have

$$1 \models \perp \text{ iff } cur[k] \equiv true.$$

Case p : p is an atomic proposition, and for state ρ_1 , $cur[k] = p \in \rho_1$, obviously

$$1 \models p \text{ iff } cur[k] \equiv true.$$

Case $t > 0$: This is an arithmetic relation. Since the valuations of all variables are determined initially, the truth value of this relation can be got following normal calculation rules, that is $cur[k] = t^\nu > 0$, so

$$1 \models t > 0 \text{ iff } cur[k] \equiv true.$$

Case $\odot_I \psi$: Since there is no previous states for the first state, $1 \not\models \odot_I \psi$. With $cur[k] = false$, naturally

$$1 \models \odot_I \psi \text{ iff } cur[k] \equiv true.$$

(Induction cases)

Case $\neg \psi$: Since $\psi \prec \neg \psi$ and it has been proved

$$1 \models \psi \text{ iff } cur[idx(\psi)] \equiv true,$$

then with $cur[k] = \neg cur[idx(\psi)]$ we have

$$1 \models \neg \psi \text{ iff } cur[k] \equiv true.$$

Case $\psi_1 \vee \psi_2$: Since $\psi_1 \prec \phi_k$, $\psi_2 \prec \phi_k$ and

$$1 \models \psi_1 \text{ iff } cur[idx(\psi_1)] \equiv true,$$

$$1 \models \psi_2 \text{ iff } cur[idx(\psi_2)] \equiv true,$$

then with $cur[k] = cur[idx(\psi_1)] \vee cur[idx(\psi_2)]$, we have

$$1 \models \psi_1 \vee \psi_2 \text{ iff } cur[k] \equiv true.$$

Case $\psi_1 \mathcal{S}_I \psi_2$: At state ρ_1 , according to the semantics defined in § 5.1.2, the only possible value of j is 1, in which this case there exists no valid k . The model relation holds if and only if $\tau_i - \tau_i = 0 \in I$ and $1 \models \psi_2$. Since $\psi_2 \prec \phi_k$, we have

$$1 \models \psi_2 \text{ iff } cur[idx(\psi_2)] \equiv true.$$

With $cur[k] = 0 \in I \wedge cur[idx(\psi_2)]$ we have

$$1 \models \psi_1 \mathcal{S}_I \psi_2 \text{ iff } cur[k] \equiv true.$$

Case $\mathcal{C}_I x : \langle \alpha, \beta \rangle . \varphi(x)$: For the valuation of x , the correctness has been proved in Theorem 5.21. We mention here since $\alpha \prec \phi_k$, $\beta \prec \phi_k$ and $\varphi^{\nu[x \mapsto c\_cnt[idx_c(\mathcal{C}_I x : \langle \alpha, \beta \rangle)]]} \prec \phi_k$, it follows that

$$1 \models \alpha \text{ iff } cur[idx(\alpha)] \equiv true,$$

$$1 \models \beta \text{ iff } cur[idx(\beta)] \equiv true,$$

$$1 \models \varphi' \text{ iff } cur[idx(\varphi')] \equiv true,$$

with $\varphi' = \varphi^{\nu[x \mapsto c\_cnt[idx_c(\mathcal{C}_I x : \langle \alpha, \beta \rangle)]]}$.

With the above inductive analysis, the lemma follows. $\square$

A.4 Proof of Lemma 5.24

Proof. In *Iter* procedure, firstly *prev* array is updated with *cur* array from previous iteration. Accordingly, the assumption becomes,

$$\text{for all } \phi_k \in \mathcal{SF}^+(\phi), i - 1 \models \phi_k \text{ iff } \text{prev}[\text{idx}(\phi_k)].$$

Formulas in $\mathcal{SF}^+(\phi)$ are well-ordered and each formula will be judged after its subformulas. Since the formulas are constructed inductively, we will conduct inductive analysis on the structure of each formula to obtain the truth value. Then for ϕ_k :

(Base cases)

Case $\perp$: Obviously $i \not\models \perp$ and with $\text{cur}[k] = \text{false}$, we have

$$i \models \perp \text{ iff } \text{cur}[k] \equiv \text{true}.$$

Case p : p is an atomic proposition, and for state ρ_1 , $\text{cur}[k] = p \in \rho_1$, obviously

$$i \models p \text{ iff } \text{cur}[k] \equiv \text{true}.$$

Case $t > 0$: This is an arithmetic relation. Since the valuations of all variables are determined initially, the truth value of this relation can be got following normal calculation rules, that is $\text{cur}[k] = t' > 0$, so

$$i \models t > 0 \text{ iff } \text{cur}[k] \equiv \text{true}.$$

(Induction cases)

Case $\neg\psi$: According to the semantics,

$$i \models \neg\psi \text{ iff } i \not\models \psi.$$

Since $\psi \prec \neg\psi$ and

$$i \models \psi \text{ iff } \text{cur}[\text{idx}(\psi)] \equiv \text{true},$$

then with $\text{cur}[k] = \neg\text{cur}[\text{idx}(\psi)]$ we can get

$$i \models \neg\psi \text{ iff } \text{cur}[k] \equiv \text{true}.$$

Case $\psi_1 \vee \psi_2$: According to the semantics,

$$i \models \psi_1 \vee \psi_2 \text{ iff } i \models \psi_1 \text{ or } i \models \psi_2.$$

Since $\psi_1 \prec \psi_1 \vee \psi_2$ and $\psi_2 \prec \psi_1 \vee \psi_2$, and also

$$i \models \psi_1 \text{ iff } \text{cur}[\text{idx}(\psi_1)] \equiv \text{true},$$

$$i \models \psi_2 \text{ iff } \text{cur}[\text{idx}(\psi_2)] \equiv \text{true},$$

then with $\text{cur}[k] = \text{cur}[\text{idx}(\psi_1)] \vee \text{cur}[\text{idx}(\psi_2)]$ we can get

$$i \models \psi_1 \vee \psi_2 \text{ iff } \text{cur}[k] \equiv \text{true}.$$

Case $\odot_I \psi$: According to the semantics,

$$i \models \odot_I \psi \text{ iff } i - 1 \models \psi \text{ and } \tau_i - \tau_{i-1} \in I.$$

Since

$$i - 1 \models \psi \text{ iff } \text{prev}[\text{idx}(\psi)] \equiv \text{true},$$

then with $\text{cur}[k] = \text{prev}[\text{idx}(\psi)] \wedge (\tau_i - \tau_{i-1}) \in I$ we can get

$$i \models \odot_I \psi \text{ iff } \text{cur}[k] \equiv \text{true}.$$

Case $\psi_1 \mathcal{S}_I \psi_2$: According to the semantics,

$$\begin{aligned} \nu, i \models \phi_1 \mathcal{S}_I \phi_2 & \text{ iff } 0 \in I \text{ and } \nu, i \models \phi_2, \text{ or} \\ & i > 1, \text{ and } \nu, i \models \phi_1 \text{ and } \nu, i \models \phi_1 \mathcal{S}_{I'} \phi_2 \\ & \text{ where } I' = I - (\tau_i - \tau_{i-1}). \end{aligned}$$

Since $\psi_1 \prec \psi_1 \mathcal{S}_I \psi_2$, $\psi_2 \prec \psi_1 \mathcal{S}_I \psi_2$ and $\psi_1 \mathcal{S}_{I'} \psi_2 \prec \psi_1 \mathcal{S}_I \psi_2$,

$$\begin{aligned} i \models \psi_1 & \text{ iff } \text{cur}[\text{idx}(\psi_1)] \equiv \text{true}, \\ i \models \psi_2 & \text{ iff } \text{cur}[\text{idx}(\psi_2)] \equiv \text{true}, \\ i - 1 \models \psi_1 \mathcal{S}_{I'} \psi_2 & \text{ iff } \text{prev}[\text{idx}(\psi_1 \mathcal{S}_{I'} \psi_2)] \equiv \text{true}, \end{aligned}$$

then we know it follows that

$$i \models \odot_I \psi \text{ iff } \text{cur}[k] \equiv \text{true}.$$

Case $\mathcal{C}_I x : \langle \alpha, \beta \rangle . \varphi(x)$: Basically, the algorithm deals with the metric counting quantifier following the recursive semantics given in Theorem 5.22 and the correctness has been proved. We mention here since $\alpha \prec \phi_k$, $\beta \prec \phi_k$ and $\varphi^{\nu[x \mapsto c\_cnt[\text{idx}_c(\mathcal{C}_I x : \langle \alpha, \beta \rangle)]]} \prec \phi_k$, it follows that

$$\begin{aligned} i \models \alpha & \text{ iff } \text{cur}[\text{idx}(\alpha_j)] \equiv \text{true}, \\ i \models \beta & \text{ iff } \text{cur}[\text{idx}(\beta_j)] \equiv \text{true}, \\ i \models \varphi' & \text{ iff } \text{cur}[\text{idx}(\varphi')] \equiv \text{true}, \end{aligned}$$

with $\varphi' = \varphi^{\nu[x \mapsto c\_cnt[\text{idx}_c(\mathcal{C}_I x : \langle \alpha, \beta \rangle)]]}$.

With the above induction analysis, the lemma follows. $\square$

A.5 Proof of Lemma 5.25

Proof. The procedure *Monitor* first calls *Init*, according to Lemma 5.23, we have, for all $\phi_k \in \mathcal{SF}^+(\phi)$,

$$1 \models \phi_k \text{ iff } cur[k] \equiv true.$$

Then for $i \geq 2$, we assume, for all $\phi_k \in \mathcal{SF}^+(\phi)$,

$$i - 1 \models \phi_k \text{ iff } cur[k] \equiv true.$$

According to Lemma 5.24, after executing *Iter*($\rho, \tau, \nu, i, \phi, prev, cur, p\_cnt, c\_cnt$), we have, for all $\phi_k \in \mathcal{SF}^+(\phi)$,

$$i \models \phi_k \text{ iff } cur[k] \equiv true.$$

This lemma follows. □

A.6 Proof of Theorem 5.26

Proof. According to Lemma 5.25, after executing *Monitor*(ρ, τ, ν, i, ϕ), we have, for all $\phi_k \in \mathcal{SF}^+(\phi)$,

$$i \models \phi_k \text{ iff } cur[idx(\phi_k)] \equiv true.$$

Since $\phi \in \mathcal{SF}^+(\phi)$, naturally we can get

$$i \models \phi \text{ iff } cur[idx(\phi)] \equiv true.$$

This theorem follows. □

Bibliography

- [1] D. Ciregan, U. Meier, and J. Schmidhuber, “Multi-column deep neural networks for image classification,” in *CVPR*, 2012, pp. 3642–3649.
- [2] A. M. Rush, S. Chopra, and J. Weston, “A neural attention model for abstractive sentence summarization,” *arXiv preprint arXiv:1509.00685*, 2015.
- [3] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, “Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups,” *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.
- [4] B. Huval, T. Wang, S. Tandon, J. Kiske, W. Song, J. Pazhayampallil, M. Andriluka, P. Rajpurkar, T. Migimatsu, R. Cheng-Yue, F. Mujica, A. Coates, and A. Y. Ng, “An empirical evaluation of deep learning on highway driving,” *CoRR*, vol. abs/1504.01716, 2015. [Online]. Available: <http://arxiv.org/abs/1504.01716>
- [5] D. Ye, Z. Liu, M. Sun, B. Shi, P. Zhao, H. Wu, H. Yu, S. Yang, X. Wu, Q. Guo *et al.*, “Mastering complex control in moba games with deep reinforcement learning,” *arXiv preprint arXiv:1912.09729*, 2019.
- [6] Y. Zheng, X. Xie, T. Su, L. Ma, J. Hao, Z. Meng, Y. Liu, R. Shen, Y. Chen, and C. Fan, “Wuji: Automatic online combat game testing using evolutionary deep reinforcement learning,” in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 772–784.

- [7] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli, “Evasion attacks against machine learning at test time,” in *Joint European conference on machine learning and knowledge discovery in databases*. Springer, 2013, pp. 387–402.
- [8] N. Carlini, P. Mishra, T. Vaidya, Y. Zhang, M. Sherr, C. Shields, D. Wagner, and W. Zhou, “Hidden Voice Commands,” in *25th {USENIX} Security Symposium ({USENIX} Security 16)*. Austin, TX: {USENIX} Association, 2016, pp. 513–530. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/carlini>
- [9] G. Zhang, C. Yan, X. Ji, T. Zhang, T. Zhang, and W. Xu, “Dolphinattack: Inaudible voice commands,” in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS ’17. New York, NY, USA: ACM, 2017, pp. 103–117. [Online]. Available: <http://doi.acm.org/10.1145/3133956.3134052>
- [10] N. Carlini and D. Wagner, “Audio Adversarial Examples: Targeted Attacks on Speech-to-Text,” jan 2018. [Online]. Available: <http://arxiv.org/abs/1801.01944>
- [11] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” *ICLR*, 2015.
- [12] N. Carlini and D. Wagner, “Towards evaluating the robustness of neural networks,” in *Security and Privacy (SP), IEEE Symposium on*, 2017, pp. 39–57.
- [13] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, “Distillation as a defense to adversarial perturbations against deep neural networks,” in *Security and Privacy (SP), IEEE Symposium on*, 2016, pp. 582–597.
- [14] K. Pei, Y. Cao, J. Yang, and S. Jana, “Deepxplore: Automated whitebox testing of deep learning systems,” in *SOSP*, 2017, pp. 1–18.

- [15] Y. Tian, K. Pei, S. Jana, and B. Ray, “Deeptest: Automated testing of deep-neural-network-driven autonomous cars,” in *ICSE*. ACM, 2018, pp. 303–314.
- [16] L. Ma, F. Juefei-Xu, F. Zhang, J. Sun, M. Xue, B. Li, C. Chen, T. Su, L. Li, Y. Liu *et al.*, “Deepgauge: Multi-granularity testing criteria for deep learning systems,” in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. ACM, 2018, pp. 120–131.
- [17] Y. Sun, M. Wu, W. Ruan, X. Huang, M. Kwiatkowska, and D. Kroening, “Concolic Testing for Deep Neural Networks,” 2018.
- [18] J. Kim, R. Feldt, and S. Yoo, “Guiding deep learning system testing using surprise adequacy,” in *Proceedings of the 41st International Conference on Software Engineering*, ser. ICSE ’19, 2019, pp. 1039–1049.
- [19] X. Xie, L. Ma, H. Wang, Y. Li, Y. Liu, and X. Li, “Diffchaser: Detecting disagreements for deep neural networks,” in *IJCAI*, 2019.
- [20] G. Weiss, Y. Goldberg, and E. Yahav, “Extracting automata from recurrent neural networks using queries and counterexamples,” *arXiv preprint arXiv:1711.09576*, 2017.
- [21] J. Wang, G. Dong, J. Sun, X. Wang, and P. Zhang, “Adversarial sample detection for deep neural network through model mutation testing,” *arXiv preprint arXiv:1812.05793*, 2018.
- [22] D. Hendrycks and K. Gimpel, “A baseline for detecting misclassified and out-of-distribution examples in neural networks,” *arXiv preprint arXiv:1610.02136*, 2016.
- [23] R. Feinman, R. R. Curtin, S. Shintre, and A. B. Gardner, “Detecting adversarial samples from artifacts,” *arXiv preprint arXiv:1703.00410*, 2017.
- [24] Y. Liu, L. Ma, and J. Zhao, “Secure deep learning engineering: A road towards quality assurance of intelligent systems,” in *International Conference on Formal Engineering Methods*. Springer, 2019, pp. 3–15.

- [25] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann, “Software engineering for machine learning: A case study,” in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 2019, pp. 291–300.
- [26] Q. Guo, S. Chen, X. Xie, L. Ma, Q. Hu, H. Liu, Y. Liu, J. Zhao, and X. Li, “An empirical study towards characterizing deep learning development and deployment across different frameworks and platforms,” in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 810–822.
- [27] I. Jolliffe, “Principal Component Analysis,” in *International Encyclopedia of Statistical Science*. Springer, 2011, pp. 1094–1096.
- [28] X. Du, Y. Liu, and A. Tiu, “Trace-length independent runtime monitoring of quantitative policies in ltl,” in *FM*, 2015, pp. 231–247.
- [29] X. Du, A. Tiu, K. Cheng, and Y. Liu, “Trace-length independent runtime monitoring of quantitative policies,” *IEEE Transactions on Dependable and Secure Computing*, 2019.
- [30] X. Du, X. Xie, Y. Li, L. Ma, Y. Liu, and J. Zhao, “Deepstellar: Model-based quantitative analysis of stateful deep learning systems,” in *FSE*, 2019, pp. 477–487.
- [31] —, “A quantitative analysis framework for recurrent neural network,” in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 1062–1065.
- [32] X. Du, B. Chen, Y. Li, J. Guo, Y. Zhou, Y. Liu, and Y. Jiang, “Leopard: Identifying vulnerable code for vulnerability assessment through program metrics,” in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 60–71.

- [33] X. Du, “Marvel: a generic, scalable and effective vulnerability detection platform,” in *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 2019, pp. 129–131.
- [34] —, “Towards building a generic vulnerability detection platform by combining scalable attacking surface analysis and directed fuzzing,” in *International Conference on Formal Engineering Methods*. Springer, 2018, pp. 464–468.
- [35] Y. Zhou, S. Liu, J. Siow, X. Du, and Y. Liu, “Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks,” in *Advances in Neural Information Processing Systems*, 2019, pp. 10 197–10 207.
- [36] X. Zhang, X. Xie, L. Ma, X. Du, Q. Hu, Y. Liu, J. Zhao, and M. Sun, “Towards characterizing adversarial defects of deep learning software from the lens of uncertainty,” in *The 42nd International Conference on Software Engineering (ICSE 2020)*, 2020.
- [37] G. Chen, S. Chen, L. Fan, X. Du, Z. Zhao, F. Song, and Y. Liu, “Who is real bob? adversarial attacks on speaker recognition systems,” *arXiv preprint arXiv:1911.01840*, 2019.
- [38] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [39] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, “On the properties of neural machine translation: Encoder-decoder approaches,” *arXiv preprint arXiv:1409.1259*, 2014.
- [40] S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber *et al.*, “Gradient flow in recurrent nets: the difficulty of learning long-term dependencies,” 2001.
- [41] C. W. Omlin and C. L. Giles, “Extraction of rules from discrete-time recurrent neural networks,” *Neural networks*, vol. 9, no. 1, pp. 41–52, 1996.

- [42] A. L. Cechin, D. R. P. Simon, and K. Stertz, “State Automata Extraction from Recurrent Neural Nets Using k-Means and Fuzzy Clustering,” in *Proceedings of the XXIII International Conference of the Chilean Computer Science Society*, 2003, p. 73.
- [43] Q. Wang, K. Zhang, A. G. Ororbia, II, X. Xing, X. Liu, and C. L. Giles, “An empirical evaluation of rule extraction from recurrent neural networks,” *Neural Comput.*, vol. 30, no. 9, pp. 2568–2591, Sep. 2018.
- [44] B.-J. Hou and Z.-H. Zhou, “Learning with Interpretable Structure from RNN,” oct 2018. [Online]. Available: <http://arxiv.org/abs/1810.10708>
- [45] T.-D. B. Le and D. Lo, “Deep specification mining,” in *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*. ACM, 2018, pp. 106–117.
- [46] D. Angluin, “Learning regular sets from queries and counterexamples,” *Information and computation*, vol. 75, no. 2, pp. 87–106, 1987.
- [47] G. Weiss, Y. Goldberg, and E. Yahav, “Learning deterministic weighted automata with queries and counterexamples,” in *Advances in Neural Information Processing Systems 32*, 2019, pp. 8558–8569.
- [48] N. Papernot, P. D. McDaniel, A. Swami, and R. Harang, “Crafting adversarial input sequences for recurrent neural networks,” in *35th IEEE Military Communications Conference, MILCOM 2016*. Institute of Electrical and Electronics Engineers Inc., 2016, pp. 49–54.
- [49] J. Li, W. Monroe, and D. Jurafsky, “Understanding neural networks through representation erasure,” *arXiv preprint arXiv:1612.08220*, 2016.
- [50] R. Jia and P. Liang, “Adversarial examples for evaluating reading comprehension systems,” in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2017, pp. 2021–2031.

- [51] M. Cheng, J. Yi, H. Zhang, P.-Y. Chen, and C.-J. Hsieh, “Seq2sick: Evaluating the robustness of sequence-to-sequence models with adversarial examples,” *arXiv preprint arXiv:1803.01128*, 2018.
- [52] J. Ebrahimi, A. Rao, D. Lowd, and D. Dou, “Hotflip: White-box adversarial examples for text classification,” *arXiv preprint arXiv:1712.06751*, 2017.
- [53] Y. Gong and C. Poellabauer, “Crafting Adversarial Examples For Speech Paralinguistics Applications,” *ArXiv e-prints*, Nov. 2017.
- [54] M. Cisse, Y. Adi, N. Neverova, and J. Keshet, “Houdini: Fooling Deep Structured Prediction Models,” jul 2017. [Online]. Available: <http://arxiv.org/abs/1707.05373>
- [55] “Mozilla’s DeepSpeech,” <https://github.com/mozilla/DeepSpeech>, 2018.
- [56] Z. Zhao, D. Dua, and S. Singh, “Generating natural adversarial examples,” *arXiv preprint arXiv:1710.11342*, 2017.
- [57] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, “Reluplex: An efficient smt solver for verifying deep neural networks,” in *International Conference on Computer Aided Verification*. Springer, 2017, pp. 97–117.
- [58] A. Boopathy, T.-W. Weng, P.-Y. Chen, S. Liu, and L. Daniel, “Cnn-cert: An efficient framework for certifying robustness of convolutional neural networks,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 3240–3247.
- [59] T.-W. Weng, H. Zhang, H. Chen, Z. Song, C.-J. Hsieh, D. Boning, I. S. Dhillon, and L. Daniel, “Towards fast computation of certified robustness for relu networks,” *arXiv preprint arXiv:1804.09699*, 2018.
- [60] G. Singh, T. Gehr, M. Mirman, M. Püschel, and M. Vechev, “Fast and effective robustness certification,” in *Advances in Neural Information Processing Systems*, 2018, pp. 10 802–10 813.

- [61] T.-W. Weng, H. Zhang, P.-Y. Chen, J. Yi, D. Su, Y. Gao, C.-J. Hsieh, and L. Daniel, “Evaluating the robustness of neural networks: An extreme value theory approach,” *arXiv preprint arXiv:1801.10578*, 2018.
- [62] O. Bastani, Y. Ioannou, L. Lampropoulos, D. Vytiniotis, A. V. Nori, and A. Criminisi, “Measuring neural net robustness with constraints,” in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, ser. NIPS’16. USA: Curran Associates Inc., 2016, pp. 2621–2629. [Online]. Available: <http://dl.acm.org/citation.cfm?id=3157382.3157391>
- [63] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev, “Ai2: Safety and robustness certification of neural networks with abstract interpretation,” in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 3–18.
- [64] P. Cousot and R. Cousot, “Abstract interpretation frameworks,” *Journal of logic and computation*, vol. 2, no. 4, pp. 511–547, 1992.
- [65] —, “Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints,” in *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*. ACM, 1977, pp. 238–252.
- [66] C.-Y. Ko, Z. Lyu, L. Weng, L. Daniel, N. Wong, and D. Lin, “POPQORN: Quantifying robustness of recurrent neural networks,” in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. Long Beach, California, USA: PMLR, 09–15 Jun 2019, pp. 3468–3477. [Online]. Available: <http://proceedings.mlr.press/v97/ko19a.html>
- [67] Y. Sun, X. Huang, and D. Kroening, “Testing deep neural networks,” *arXiv preprint arXiv:1803.04792*, 2018.
- [68] L. Ma, F. Juefei-Xu, M. Xue, B. Li, L. Li, Y. Liu, and J. Zhao, “Deepct: Tomographic combinatorial testing for deep learning systems,” in *2019 IEEE*

- 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, Feb 2019, pp. 614–618.
- [69] A. Odena and I. Goodfellow, “Tensorfuzz: Debugging neural networks with coverage-guided fuzzing,” *arXiv preprint arXiv:1807.10875*, 2018.
- [70] X. Xie, L. Ma, F. Juefei-Xu, M. Xue, H. Chen, Y. Liu, J. Zhao, B. Li, J. Yin, and S. See, “Deephunter: A coverage-guided fuzz testing framework for deep neural networks,” in *ISSTA*, 2019.
- [71] L. Ma, F. Zhang, J. Sun, M. Xue, B. Li, F. Juefei-Xu, C. Xie, L. Li, Y. Liu, J. Zhao, and Y. Wang, “Deepmutation: Mutation testing of deep learning systems,” in *29th IEEE International Symposium on Software Reliability Engineering (ISSRE), Memphis, USA, Oct. 15-18, 2018*, pp. 100–111.
- [72] J. M. Zhang, M. Harman, L. Ma, and Y. Liu, “Machine Learning Testing: Survey, Landscapes and Horizons,” *arXiv e-prints*, p. arXiv:1906.10742, Jun 2019.
- [73] S. Liang, Y. Li, and R. Srikant, “Enhancing the reliability of out-of-distribution image detection in neural networks,” *arXiv preprint arXiv:1706.02690*, 2017.
- [74] W. Xu, D. Evans, and Y. Qi, “Feature squeezing: Detecting adversarial examples in deep neural networks,” *arXiv preprint arXiv:1704.01155*, 2017.
- [75] S. Ma, Y. Liu, G. Tao, W.-C. Lee, and X. Zhang, “Nic: Detecting adversarial samples with neural network invariant checking,” in *NDSS*, 2019, pp. 24–27.
- [76] J. H. Metzen, T. Genewein, V. Fischer, and B. Bischoff, “On detecting adversarial perturbations,” *arXiv preprint arXiv:1702.04267*, 2017.
- [77] K. Havelund and G. Rosu, “Synthesizing monitors for safety properties,” in *TACAS*, 2002, pp. 342–356.
- [78] A. Bauer, J.-C. Küster, and G. Vegliach, “From propositional to first-order monitoring,” in *RV*, 2013, pp. 59–75.

- [79] J. R. Norris, *Markov Chains*, ser. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 1997.
- [80] P. Rastogi, R. Cotterell, and J. Eisner, “Weighting Finite-State Transductions with Neural Context,” in *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2016, pp. 623–633.
- [81] A. Gill, *Introduction to the Theory of Finite-State Machines*, ser. McGraw-Hill electronic sciences series. McGraw-Hill, 1962. [Online]. Available: <https://books.google.com.sg/books?id=2IzQAAAAMAAJ>
- [82] B. G. Horne, C. L. Giles, P. C. Collingwood, S. O. Computing, M. Sci, P. Tino, and P. Tino, “Finite State Machines and Recurrent Neural Networks – Automata and Dynamical Systems Approaches,” in *Neural Networks and Pattern Recognition*. Academic Press, 1998, pp. 171–220.
- [83] J. F. Thompson, B. K. Soni, and N. P. Weatherill, *Handbook of Grid Generation*. CRC press, 1998.
- [84] J. Wang, B. Chen, L. Wei, and Y. Liu, “Skyfire: Data-driven seed generation for fuzzing,” in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 579–594.
- [85] Y. Li, B. Chen, M. Chandramohan, S.-W. Lin, Y. Liu, and A. Tiu, “Steelix: program-state based binary fuzzing,” in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 2017, pp. 627–637.
- [86] T. Su, G. Meng, Y. Chen, K. Wu, W. Yang, Y. Yao, G. Pu, Y. Liu, and Z. Su, “Guided, stochastic model-based gui testing of android apps,” in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 2017, pp. 245–256.
- [87] H. Chen, Y. Xue, Y. Li, B. Chen, X. Xie, X. Wu, and Y. Liu, “Hawkeye: Towards a desired directed grey-box fuzzer,” in *Proceedings of the 2018 ACM*

- SIGSAC Conference on Computer and Communications Security*, 2018, pp. 2095–2108.
- [88] J. Wang, B. Chen, L. Wei, and Y. Liu, “Superion: Grammar-aware greybox fuzzing,” in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 724–735.
- [89] Y. Li, Y. Xue, H. Chen, X. Wu, C. Zhang, X. Xie, H. Wang, and Y. Liu, “Cerebro: context-aware adaptive fuzzing for effective vulnerability detection,” in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 533–544.
- [90] “Google cloud text-to-speech,” 2019. [Online]. Available: <https://cloud.google.com/text-to-speech/>
- [91] F. Chollet *et al.*, “Keras,” <https://github.com/fchollet/keras>, 2015.
- [92] M. Abadi, P. Barham, J. Chen, Z. Chen, and et al, “Tensorflow: A system for large-scale machine learning,” in *OSDI*, 2016, pp. 265–283.
- [93] “Mozilla Common Voice,” <https://voice.mozilla.org/en>, 2018.
- [94] “kaggle: Audio data augmentation,” 2019. [Online]. Available: <https://www.kaggle.com/CVxTz/audio-data-augmentation>
- [95] T. Du, S. Ji, J. Li, Q. Gu, T. Wang, and R. Beyah, “Sirenattack: Generating adversarial audio for end-to-end acoustic systems,” *arXiv preprint arXiv:1901.07846*, 2019.
- [96] I. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” in *International Conference on Learning Representations*, 2015. [Online]. Available: <http://arxiv.org/abs/1412.6572>
- [97] A. Kurakin, I. Goodfellow, and S. Bengio, “Adversarial examples in the physical world,” *arXiv preprint arXiv:1607.02533*, 2016.

- [98] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, “Deepfool: A simple and accurate method to fool deep neural networks,” *CVPR*, pp. 2574–2582, 2016.
- [99] “Levenshtein Distance,” 2019. [Online]. Available: https://en.wikipedia.org/wiki/Levenshtein_distance
- [100] C. Spearman, “The proof and measurement of association between two things,” *The American journal of psychology*, vol. 100, no. 3/4, pp. 441–471, 1987.
- [101] H. B. Mann and D. R. Whitney, “On a test of whether one of two random variables is stochastically larger than the other,” *The annals of mathematical statistics*, pp. 50–60, 1947.
- [102] “DeepStellar,” 2019. [Online]. Available: <https://sites.google.com/view/deepstellar/home>
- [103] T. Fawcett, “An introduction to roc analysis,” *Pattern recognition letters*, vol. 27, no. 8, pp. 861–874, 2006.
- [104] L. Ma, F. Juefei-Xu, M. Xue, Q. Hu, S. Chen, B. Li, Y. Liu, J. Zhao, J. Yin, and S. See, “Secure Deep Learning Engineering: A Software Quality Assurance Perspective,” *arXiv e-prints*, p. arXiv:1810.04538, Oct. 2018.
- [105] M. Balunovic, M. Baader, G. Singh, T. Gehr, and M. Vechev, “Certifying geometric robustness of neural networks,” in *Advances in Neural Information Processing Systems 32*, 2019.
- [106] C.-Y. Ko, Z. Lyu, L. Weng, L. Daniel, N. Wong, and D. Lin, “Popqorn: Quantifying robustness of recurrent neural networks,” in *International Conference on Machine Learning*, 2019, pp. 3468–3477.
- [107] J. Gao, J. Lanchantin, M. L. Soffa, and Y. Qi, “Black-box generation of adversarial text sequences to evade deep learning classifiers,” in *2018 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2018, pp. 50–56.
- [108] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, 1st ed. New York, NY, USA: John Wiley & Sons, Inc., 1994.

- [109] B. Wachter, L. Zhang, and H. Hermanns, “Probabilistic model checking modulo theories,” in *fourth international conference on the quantitative evaluation of systems (QEST 2007)*. IEEE, 2007, pp. 129–140.
- [110] E. L. Lehmann and G. Casella, *Theory of point estimation*. Springer Science & Business Media, 2006.
- [111] H. Abdi and L. J. Williams, “Principal component analysis,” *WIREs Computational Statistics*, vol. 2, no. 4, pp. 433–459, 2010.
- [112] X. Li and D. Roth, “Learning question classifiers,” in *Proceedings of the 19th international conference on Computational linguistics-Volume 1*. Association for Computational Linguistics, 2002, pp. 1–7.
- [113] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, “Learning word vectors for sentiment analysis,” in *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies-volume 1*. Association for Computational Linguistics, 2011, pp. 142–150.
- [114] J. Pennington, R. Socher, and C. D. Manning, “Glove: Global vectors for word representation,” in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.
- [115] R. Jia and P. Liang, “Adversarial examples for evaluating reading comprehension systems,” in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, Sep. 2017, pp. 2021–2031.
- [116] Y. Belinkov and Y. Bisk, “Synthetic and natural noise both break neural machine translation,” in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018.
- [117] A. Pnueli, “The temporal logic of programs,” in *18th Annual Symposium on Foundations of Computer Science*, 1977, pp. 46–57.

- [118] R. Alur and T. A. Henzinger, “Real-time logics: complexity and expressiveness,” *Information and Computation*, vol. 104, no. 1, pp. 35–77, 1993.
- [119] A. Bauer, R. Goré, and A. Tiu, “A first-order policy language for history-based transaction monitoring,” in *ICTAC*, 2009.
- [120] D. Basin, F. Klaedtke, S. Marinovic, and E. Zălinescu, “Monitoring of temporal first-order properties with aggregations,” *Formal methods in system design*, vol. 46, no. 3, pp. 262–285, 2015.
- [121] D. Basin, F. Klaedtke, and E. Zălinescu, “Algorithms for monitoring real-time properties,” in *RV*, 2012, pp. 260–275.
- [122] M. Fitting, *First-Order Logic and Automated Theorem Proving*. Springer, 1996.
- [123] D. Bianculli, C. Ghezzi, and P. San Pietro, “The tale of soloist: a specification language for service compositions interactions,” in *FACS*, 2013, pp. 55–72.
- [124] A. P. Sistla and O. Wolfson, “Temporal conditions and integrity constraints in active database systems,” in *ACM SIGMOD/PODS*, 1995, pp. 269–280.
- [125] P. Thati and G. Roşu, “Monitoring algorithms for metric temporal logic specifications,” *ENTCS*, vol. 113, pp. 145–162, 2005.
- [126] N. Markey and P. Schnoebelen, “Model checking a path,” in *CONCUR*, 2003, pp. 251–265.
- [127] D. Basin, B. N. Bhatt, and D. Traytel, “Almost event-rate independent monitoring of metric temporal logic,” in *TACAS*, 2017, pp. 94–112.
- [128] H. Gunadi and A. Tiu, “Efficient runtime monitoring with metric temporal logic: A case study in the android operating system,” in *FM*, 2014, pp. 296–311.
- [129] Z. Katzir and Y. Elovici, “Detecting adversarial perturbations through spatial behavior in activation spaces,” in *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2019, pp. 1–9.

- [130] D. Yang, A. Usynin, and J. W. Hines, “Anomaly-based intrusion detection for scada systems,” in *NPIC & HMIT*, 2006, pp. 12–16.
- [131] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, “Anomaly-based network intrusion detection: Techniques, systems and challenges,” *computers & security*, vol. 28, no. 1, pp. 18–28, 2009.
- [132] E. Evenchick, “Hopping on the can bus,” in *Black Hat Asia 2015*, 2015, pp. 1–31.
- [133] C. Miller and C. Valasek, “Adventures in automotive networks and control units.”
- [134] V. L. L. Thing and J. Wu, “Autonomous vehicle security: A taxonomy of attacks and defences,” in *iThings/GreenCom/CPSCoM/SmartData*, 2016, pp. 164–170.
- [135] C. Miller and C. Valasek, “A survey of remote automotive attack surfaces,” *black hat USA*, vol. 2014, 2014.
- [136] A. Taylor, N. Japkowicz, and S. Leblanc, “Frequency-based anomaly detection for the automotive can bus,” in *WCICSS*, 2015, pp. 45–49.
- [137] H. M. Song, H. R. Kim, and H. K. Kim, “Intrusion detection system based on the analysis of time intervals of can messages for in-vehicle network,” in *ICOIN*, 2016, pp. 63–68.
- [138] G. Jin, L. Song, X. Shi, J. Scherpelz, and S. Lu, “Understanding and detecting real-world performance bugs,” in *PLDI*, 2012, pp. 77–88.
- [139] Y. Liu, C. Xu, and S.-C. Cheung, “Characterizing and detecting performance bugs for smartphone applications,” in *ICSE*, 2014, pp. 1013–1024.
- [140] “Firefox frame rate meter tool webpage,” <http://blog.mozilla.org/devtools/tag/framerate-monitor/>, 2011.

- [141] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, “Flipping bits in memory without accessing them: An experimental study of dram disturbance errors,” in *ISCA*, 2014, pp. 361–372.
- [142] M. Seaborn, “Exploiting the dram rowhammer bug to gain kernel privileges,” <http://googleprojectzero.blogspot.sg/2015/03/exploiting-dram-rowhammer-bug-to-gain.html>, 2015.
- [143] D. Gruss, C. Maurice, and S. Mangard, “Rowhammer.js: a remote software-induced fault attack in javascript,” *arXiv preprint arXiv:1507.06955*, 2015.
- [144] N. Herath and A. Fogh, “These are not your grand daddys cpu performance counters - cpu hardware performance counters for security,” 2015.
- [145] G. Meng, M. Patrick, Y. Xue, Y. Liu, and J. Zhang, “Securing android app markets via modelling and predicting malware spread between markets,” *TIFS*, 2018.
- [146] G. Meng, Y. Xue, M. Chandramohan, A. Narayanan, Y. Liu, J. Zhang, and T. Chen, “Mystique: Evolving android malware for auditing anti-malware tools,” in *AsiaCCS*, 2016, pp. 365–376.
- [147] Z. Tang, M. Xue, G. Meng, C. Ying, Y. Liu, J. He, H. Zhu, and Y. Liu, “Securing android applications via edge assistant third-party library detection,” *Computers & Security*, pp. 257 – 272, 2019.
- [148] G. Meng, Y. Xue, Z. Xu, Y. Liu, J. Zhang, and A. Narayanan, “Semantic modelling of android malware for effective malware comprehension, detection, and classification,” in *ISSTA*, 2016, pp. 306–317.
- [149] “Sms rate limit,” <http://support.telerivet.com/customer/portal/articles/1205288-customizing-app-setttings>, 2011.
- [150] S. Arzt, K. Falzon, A. Follner, S. Rasthofer, E. Bodden, and V. Stolz, “How useful are existing monitoring languages for securing android apps?” in *ATPS*, 2013, pp. 107–122.

- [151] H. Pieterse and M. S. Olivier, “Android botnets on the rise: Trends and characteristics,” in *ISSA2*, 2012, pp. 1–5.
- [152] “Rage against the cage,” <http://thesnkchrnr.wordpress.com/2011/03/24/rageagainstthecage>, 2011.
- [153] “Trace-length independent runtime monitoring of quantitative policies,” <https://sites.google.com/view/mtlcnt/home>, 2018.
- [154] H.-M. Ho, J. Ouaknine, and J. Worrell, “Online monitoring of metric temporal logic,” in *RV*, 2014, pp. 178–192.
- [155] D. Basin, S. Krstić, and D. Traytel, “Almost event-rate independent monitoring of metric dynamic logic,” in *RV*, 2017, pp. 85–102.
- [156] F. Laroussinie, A. Meyer, and E. Petonnet, “Counting ltl,” in *TIME*, 2010, pp. 51–58.
- [157] D. Bianculli, S. Krstic, C. Ghezzi, and P. San Pietro, “From soloist to ctlb (d): Checking quantitative properties of service-based applications,” 2013.
- [158] M. M. Bersani, D. Bianculli, C. Ghezzi, S. Krstić, and P. San Pietro, “Smt-based checking of soloist over sparse traces,” in *FASE*, 2014, pp. 276–290.
- [159] C. Colombo, A. Gauci, and G. J. Pace, “Larvastat: Monitoring of statistical properties,” in *RV*, 2010, pp. 480–484.
- [160] B. D’Angelo, S. Sankaranarayanan, C. Sanchez, W. Robinson, B. Finkbeiner, H. B. Sipma, S. Mehrotra, and Z. Manna, “Lola: Runtime monitoring of synchronous systems,” in *TIME*, 2005, pp. 166–174.
- [161] H. Barringer, A. Goldberg, K. Havelund, and K. Sen, “Rule-based runtime verification,” in *VMCAI*, 2004, pp. 44–57.
- [162] H. Barringer, D. Rydeheard, and K. Havelund, “Rule systems for run-time monitoring: from eagle to ruler,” *Journal of Logic and Computation*, vol. 20, no. 3, pp. 675–706, 2010.

- [163] K. Havelund, “Rule-based runtime verification revisited,” *STTT*, vol. 17, no. 2, pp. 1–28, 2012.
- [164] B. Finkbeiner, S. Sankaranarayanan, and H. B. Sipma, “Collecting statistics over runtime executions,” *Form. Methods Syst. Des.*, vol. 27, no. 3, pp. 253–274, 2005.
- [165] E. Asarin, A. Donzé, O. Maler, and D. Nickovic, “Parametric identification of temporal properties,” in *International Conference on Runtime Verification*. Springer, 2011, pp. 147–160.
- [166] A. Bakhirkin, T. Ferrère, and O. Maler, “Efficient parametric identification for stl,” in *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (part of CPS Week)*. ACM, 2018, pp. 177–186.
- [167] G. Reger, H. C. Cruz, and D. Rydeheard, “Marq: monitoring at runtime with qea,” in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2015, pp. 596–610.
- [168] D. Basin, F. Klaedtke, S. Müller, and E. Zălinescu, “Monitoring metric first-order temporal properties,” *Journal of the ACM (JACM)*, vol. 62, no. 2, p. 15, 2015.
- [169] K. Havelund, D. Peled, and D. Ulus, “First-order temporal logic monitoring with bdds,” *Formal Methods in System Design*, pp. 1–21, 2019.
- [170] C. Bertrand, H. Klaudel, M. Latapy, and F. Peschanski, “Pattern matching in link streams: A token-based approach,” in *International Conference on Applications and Theory of Petri Nets and Concurrency*. Springer, 2018, pp. 227–247.
- [171] C. Bertrand, F. Peschanski, H. Klaudel, and M. Latapy, “Pattern matching in link streams: Timed-automata with finite memory,” 2018.
- [172] M. Waga, É. André, and I. Hasuo, “Symbolic monitoring against specifications parametric in time and data,” in *CAV*, 2019.

- [173] X. Xie, B. Chen, Y. Liu, W. Le, and X. Li, “Proteus: computing disjunctive loop summary via path dependency analysis,” in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 2016, pp. 61–72.
- [174] X. Xie, B. Chen, L. Zou, Y. Liu, W. Le, and X. Li, “Automatic loop summarization via path dependency analysis,” *IEEE Transactions on Software Engineering*, 2018.
- [175] X. Xie, Y. Liu, W. Le, X. Li, and H. Chen, “S-looper: Automatic summarization for multipath string loops,” in *Proceedings of the 2015 International Symposium on Software Testing and Analysis*. ACM, 2015, pp. 188–198.
- [176] S.-W. Lin, J. Sun, H. Xiao, Y. Liu, D. Sanán, and H. Hansen, “Fib: Squeezing loop invariants by interpolation between forward/backward predicate transformers,” in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2017, pp. 793–803.
- [177] J. Li, J. Sun, L. Li, Q. L. Le, and S.-W. Lin, “Automatic loop-invariant generation and refinement through selective sampling,” in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2017, pp. 782–792.