Probabilistic model checking for hybrid systems with hybrid concolic testing and importance sampling

Kong, Pingfan

2016

Kong, P. (2016). Probabilistic model checking for hybrid systems with hybrid concolic testing and importance sampling. Master's thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/65915>

<https://doi.org/10.32657/10356/65915>

PROBABILISTIC MODEL CHECKING FOR HYBRID SYSTEMS WITH HYBRID CONCOLIC TESTING AND IMPORTANCE SAMPLING

KONG PINGFAN

School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfilment of the requirement for
the degree of Master of Engineering

2016

Acknowledgments

I would love to thank my supervisor Prof. Yang Liu for his patient instruction on my research topic and his intuitive training on my critical thinking ability and paper writing skills.

I would also love to thank Prof. Jun Sun from Singapore University of Technology and Design for helping me get into the domain of model checking and for his selfishless guidance.

This work is also a sincerest and most solemn memorial to my grandfather who passed away during my staying and studying in Singapore. Though relevant works were done towards Atrial Fibrillation analyzing in terms of hybrid system verification, there's still a long way to go. I have great confidence in its potential of discovering and precaution of such diseases and saving more precious lives.

Contents

Acknowledgments	i
List of Figures	iv
List of Tables	v
Abstract	1
1 Introduction	2
2 Related Work	4
2.1 Statistical Model Checking Tools for Hybrid Systems	4
2.2 Symbolic Tools and Theorem Provers	6
2.3 Importance Sampling and Importance Splitting Algorithms	7
2.4 summary	7
3 Preliminaries	9
3.1 Hybrid Automata	9
3.2 Discrete Time Markov Chain	11
3.3 Bounded Linear Temporal Logic	11
3.4 Importance Sampling	13
3.5 Hybrid Concolic Testing	14
3.6 dReach Introduction	15
3.7 Summary	16
4 Hybrid System Verification based on Importance Sampling	17
4.1 Hybrid Automata to Discrete Time Markov Chain	17
4.2 Introducing Hybrid Concolic Testing into Random Trajectory Generation	20
4.3 Use dReach to Find Importance Region	21

4.4	Importance Sampling according to Importance Region	21
4.5	HyChecker Algorithm	22
4.6	Summary	23
5	Experiments and Evaluation	25
5.1	Sewerage System	25
5.2	Room Heating System	27
5.3	Navigation Benchmark	29
5.4	Secure Water Testbed (SWaT)	31
5.5	Evaluation and Discussion	33
6	Conclusions and Future Works	35
	References	36

List of Figures

3.1	Dynamic Behavior and Hybrid Automaton of <i>Bouncing Ball</i>	10
3.2	Simple Example of Markov Chain	12
3.3	The BLTL Properties	13
3.4	Hybrid Concolic Testing	15
4.1	Major Flow Graph of Our Method	18
4.2	The Markov Chain	19
4.3	The Hybrid Concolic Guard Covering	20
5.1	Hybrid Automaton Form of Sewerage System	26
5.2	The Room Heating System	28
5.3	The Navigation System	31
5.4	The Schematic of Secure Water Testbed (SWaT)	33

List of Tables

5.1	Sewerage System Checking Outcome	27
5.2	Room Heating System Checking Outcome	30
5.3	Navigation Benchmark Outcome	32
5.4	SWaT Outcome	34

Summary

Hybrid systems are dynamic systems that exhibit both continuous and discrete behavior. Many real-world engineering problems can be categorized as hybrid systems, including part of the typical cyber-physical systems. Hybrid systems are known to be hard to analyze and verify as they can both flow with differential equation dynamics and jump like a control graph or similar to program statements. In this master thesis, we investigate the problem of systematically analyzing hybrid systems and build a tool accordingly. Through investigation of related papers, we discover that there are two major problems that arise from hybrid system characteristics. One is to decide the time points when mode jumps happen. This often needs solving the ordinary differential equations with specific guard conditions. It can always be time-consuming with absence of closed form value state functions. The other obstacle lies in the fact that the set of hybrid automata trajectories is often too large. So it is not easy to achieve completeness while checking Bounded Linear Temporal Logic (BLTL) formulas against it. In previous work, a method for analyzing the dynamics of a hybrid automaton H in terms of a Markov Chain M was proposed. Our new algorithm is based on this method. We stress our new algorithm on the problem of rare events with which traditional statistical methods are often hard to deal. We implement our algorithm in a parallel manner to accelerate computing on multi-core machines and compare traces generated by different cores to decide if there exists any guard condition that was never satisfied. Borrowing the idea of hybrid concolic testing in program verification, we call dReach to test this guard's satisfiability. We developed a new importance sampling mechanism based on the satisfiable region retrieved by dReach to improve our efficiency of random testing. We built this new hybrid system checker by the name of *HyChecker*. We tested several hybrid systems with our tool. Our experiments show that *HyChecker* has higher efficiency in finding counterexamples without altering the original probabilistic guarantee.

Chapter 1

Introduction

In the area of hybrid system verification, people are often faced with two notoriously annoying obstacles. The first is that the closed forms of the Ordinary Differential Equations (ODEs) inside the modes are often not available, so symbolically solving out the relations between value states and time domain bears very high time consumption [24, 9, 13]. The second one lies in the fact that hybrid systems have interaction between ODE dynamics with discrete mode jumps. So it is not easy to decide where mode jumps happen on time domain precisely. To solve the above two questions, stochastic model checking is introduced [20], and there are also many other works to be categorized in this domain. Recently, a probabilistic verification method of hybrid systems was proposed in [11]. In this work, the authors built Markov Chain M out of a given hybrid automaton H , then stated that if the built Markov Chain $M \models \psi$, we could equally obtain that $H \models \psi$. This is a very efficient algorithm, as it adopts hypothesis testing to harness the power of numerical solvers to approximately answer the original question whether a hybrid automata satisfies a certain property.

However, statistical methods often have a weakness in discovering *rare events* meaning the stochastic events with extremely low probability of occurrence. We informally define a rare event to be guards which are logically satisfiable while extremely hard to be covered in statistical methods unless tested against high sampling rate. In the counterpart, symbolic methods are able to cover all modes, while the time consumption is unbearable. These rare events can be very important if a rich set of modes would follow or when they are the destination modes in our reachability problems.

In this thesis, we develop a new algorithm especially targeted at verification of hybrid systems with rare events. The general idea is to do random testing to cover regular modes. Then for uncovered modes, we call dReach [9] to test whether they can be satisfied. We then adopt the importance sampling method based on the returned satisfiable region. Importance sampling, unlike normal sampling, picks samples with unequal probability. We boost the probability of the satisfiable region returned by dReach, and lower that of the complement time set. And give formal proof of this method.

With hybrid concolic testing as an our algorithm's structure, we use dReach to spot importance regions for rare events and do the importance sampling based on the importance regions. Our algorithm takes in hybrid automata models and generates Markov Chain trajectories. The trajectories will be used to test the BLTL properties and a *true* or *false* assertion will be given accordingly. We thus built our tool *HyChecker* in *python* language based on our algorithm. We adopted *HyChecker* to test 4 hybrid systems with different scales. The hybrid systems are sewerage system model, room heating model, navigation benchmark model and the Secure Water Treatment test bed. Our test results showed the feasibility and efficiency of our tool.

This report is organized as follows. In Chapter 2, we will summarize some related works for probabilistic model checking tools, symbolic verification methods and some other mechanisms dealing with inefficient sampling. In Chapter 3, We review hybrid automata, Markov Chain, bounded linear temporal logic, importance sampling, concolic testing in program verification and give introduction to the tool dReach [17]. In Chapter 4, we first introduce how to build a Markov Chain based on the original hybrid automaton input and why through studying this Markov Chain we can equally prove the properties specified in the hybrid automaton. We then talk about how to adopt dReach to return an importance region. Next we introduce our main contributions of a new importance sampling mechanism based on the generated importance region to find rarely touched modes. Then we give our general algorithm for this method and introduce our tool *HyChecker*. In Chapter 5 we test several hybrid systems with rarely satisfied modes in them. We show in our results how our tool is able to locate counterexamples compared to pure random testing. In Chapter 6, we make our conclusion and discuss our direction for possible future works.

Chapter 2

Related Work

As our main idea is to probabilistically check hybrid systems combining both statistical and symbolic methods, and target our work in the rare events, we investigated related works to our three important components in this master thesis. The first are statistical and probabilistic model checking tools. The second are symbolic tools and theorem provers. The third are importance sampling and importance splitting algorithms. The detailed investigation results are shown below.

2.1 Statistical Model Checking Tools for Hybrid Systems

There were many probabilistic and statistical model checkers proposed in the past.

PRISM [18, 19] is a cross-platform probabilistic model checking tool that covers a wide range of application domains. The tool's GUI allows manual exploring models and Monte-Carlo discrete-event simulation techniques for approximate model analysis. This tool also supports distributed simulation. Similarly we also implement parallelly our tool *HyChecker*. This tool allows user to build probabilistic models and verify quantitative properties on them. While our tool *HyChecker* allows user to directly build hybrid automata and automatically convert to Markov Chains.

PRISM 4.0 [21] is an upgrade of the PRISM tool. A new feature of quantitative verification of probabilistic timed automata is implemented. This tool can handle probabilistic, real-time characteristics and nondeterminism. This tool can be used to verify embedded controllers in automotive systems, which are part of cyber-physical systems.

Our tool *HyChecker* is also able to verify automotive systems with embedded controllers. Our experimental results of the Secure Water Treatment (SWaT) test bed show that our tool is able to verify cyber-physical systems in large scale.

COSMOS [5] is another statistical verification tool which uses Hybrid Automata Stochastic Logic. Unlike other tools that intake only Markovian probabilistic models, this tool is able to verify models that might have infinite states. Compared with PRISM and other tools with statistical verification functionalities, the tool COSMOS can handle infinite state space and has higher expressivity of the logic.

The specialty of d/dt [3] lies in the fact that it was once the only tool at the moment supporting switching controller synthesis, that is, the tool allows controllers to satisfy certain safety requirements by automatic jumping between continuous modes. However, the tool d/dt is only able to handle continuous dynamics that are polynomials. Our tool *HyChecker* can handle transcendental functions, which are common in cyber-physical systems.

VESTA [27] is a statistical analysis tool for probabilistic systems. It tests properties specified by probabilistic Computational Tree Logic (CTL) or continuous stochastic against probabilistic systems. It provides computation of the distribution of specified value of expressions with a query language named QUAEX. VESTA takes in as input the probabilistic model M and a formula ϕ . While this tool can handle continuous ordinary differential equations, it cannot be applied on complex cyber-physical systems like our tool *HyChecker* does.

PVESTA [1] is an extension of VESTA. Compared to its predecessor, PVESTA adds writing the model as discrete or continuous Markov Chains. Similar as our tool *HyChecker*, PVESTA is able to increase performance with parallelization techniques. But like many statistical model checking tools, PVESTA is not efficient when verifying properties related to rare events.

Ymer [28] deals with stochastic discrete event systems with probabilistic transients. This tool has two main features, one is adopting distributed acceptance sampling, the other is nested probabilistic operators. Unlike our tool *HyChecker*, Ymer does not handle continuous ordinary differential equations. This makes Ymer unable to verify complex real world hybrid systems.

Uppaal [22] is a very useful tool intended to be built for engineers working with embedded systems. It is an informal tool which does not guarantee complete proving. But it would provide useful model checking algorithms with its own model syntax and semantics. Uppaal provides both textual and graphical user interfaces. While our tool currently only provides the textual user interface. Uppaal uses symbolic verification techniques. While our tool *HyChecker* adopts a hybrid concolic testing algorithm combining both symbolic and statistical verification techniques.

2.2 Symbolic Tools and Theorem Provers

In his book [24], Platzer presented a theorem prover for hybrid systems. Using differential dynamic logic to describe ODEs, his model is capable of covering a small subset of standard hybrid systems. But this requires the prover to find a sound and complete proof. It works well on some small but extremely safety critical systems like small aircrafts. However, when it comes to large or complicated systems, it seems not powerful enough. While our tool *HyChecker* has been proven to work on large scale hybrid systems.

dReach [9] is another work on verification of non-linear hybrid systems. It is based on the SMT solver *dReal* [10] and focuses on bounded δ -complete reachability analysis, where the definition of δ -completeness will be given in Chapter 3.6. The tool supports an easy-to-use interface for describing a hybrid automaton and tells whether it is δ -safe under given safety demands. However, as a pure symbolic solution, the time cost increases rapidly along with the system scale and complexity of ODEs. In comparison, *dReal* is adopted as the solver but completeness is given up to reduce the time cost. Our tool *HyChecker* uses *dReach* as a crucial part to symbolically check the satisfiability of uncovered edges during random testing.

HyTech [13] is a tool adopting symbolic model checking techniques. Its input model must be linear hybrid automata. It is one of the very first tools checking hybrid systems. Though with limited capability, HyTech is able to perform parametric analysis, which is always a very competitive topic among hybrid system verification tools. Compared to HyTech, our tool *HyChecker* is capable of handling complex ordinary differential functions.

2.3 Importance Sampling and Importance Splitting Algorithms

Rare events always come in the way of random testing or random test case generation when a system is statistically verified. Traditional random testing methods will be very inefficient in such cases. For dealing with rare events, two approaches were adopted. One is importance sampling, the other is importance splitting.

In [6], the rare event is considered as the major drawback of statistical model checking, as traditional statistical methods' time consumption will increase by magnitudes. In this work, a structural analysis of the input model is required. And no numerical computation is required because the usage of coupling theory. This work inspires us to take notice of the rare events in hybrid systems. Because of the complex interaction between discrete mode jumps and continuous ODEs, locating rare events in hybrid systems is also very important and challenging.

[7] is an further extension of the work in [6]. Works were done to further extend timed temporal formula on Continuous Time Markov Chains (CTMCs). It is able to generate true confidence interval on events with very small probability. It has also proposed a framework on reduction of the variance.

In [15], the authors derived a score function for both fixed and adaptive level splitting for rare events. Three case studies were given in this work. This work tried to spot rare events in another perspective. Further work using this method towards hybrid systems could be done with our tool *HyChecker*.

2.4 summary

In this chapter, we firstly reviewed existing statistical model checking tools. Most of them take probabilistic models as input. While our tool *HyChecker* is able to verify hybrid automata, which can be generated directly from complex hybrid systems. The other tools can take in timed automata forms as input, but they are not scalable with large systems, and can only handle polynomial functions. While our *HyChecker* can handle large systems with complex continuous dynamics with transcendental functions. We then looked into symbolic tools and theorem provers. Compared to statistical methods, they

are able to give accurate assertions, but the time consumption is generally very high due to the natural drawback of concolic algorithms. Based on the above two means, we are inspired by the hybrid concolic testing idea in program verification proposed in [23] and developed the hybrid concolic verification tool *HyChecker* for hybrid systems. And the rare event problem raised in [6, 7] helped us to focus on importance sampling to solve the critical problem of rare events in hybrid systems. Overlooking the safety-critical rare events when verifying hybrid systems can lead to inefficient verification process. Our experiments in Chapter 5 shows our tool *HyChecker*'s advantage of solving out these problems.

Chapter 3

Preliminaries

In this chapter, the fundamental background knowledge is introduced. First, we introduce hybrid automaton and review its formal definition, we adopt a simple jumping ball example to demonstrate how to model a hybrid automaton from a hybrid system. Then we introduce the definition of Markov Chain. Then we introduce the Bounded Linear Temporal Logic (BLTL). BLTL will serve to describe system safety-critical properties. Next we introduce importance sampling and discuss the situations where this method is useful. Then we introduce the concolic testing idea in program verification and discuss how this idea could be adopted in hybrid automata verification. Next we give some introduction into the tool dReach and review the definition of the δ -completeness.

3.1 Hybrid Automata

Hybrid automata [2] are used to describe systems that have both continuous behavior (usually described with *ordinary differential equations*, ODEs) and discrete behavior. Bouncing-ball (see in Fig. 3.1) is a common example of a hybrid system. It includes 2 state variables (v indicates speed and x indicates altitude) and 2 modes (rising and falling). When the ball reaches the maximum altitude (speed down to zero) it will switch to falling mode and after it touches the ground it will return to rising mode.

Definition 3.1 (hybrid automata) *A hybrid automaton can be described as a tuple $H = (Q, q_{in}, V, \dot{V}, INIT, f, G)$,*

- Q is the finite set of modes

Figure 3.1: Dynamic Behavior and Hybrid Automaton of *Bouncing Ball*

- q_{in} is the initial mode
- V is the set of continuous variables. $V = \{x_1, x_2, \dots, x_n\}$
- $\dot{V}$ is the set of the first derivatives of continuous variables. $\dot{V} = \{\dot{x}_1, \dot{x}_2, \dots, \dot{x}_n\}$
- $INIT \subseteq \mathbb{R}^n$ indicates the initial value of V . For simplicity, we use a series of interval $(l_1, u_1), (l_2, u_2), \dots, (l_n, u_n)$ (where l_i and u_i are real numbers and $l_i \leq u_i$) to describe $INIT$.
- f is a function that maps mode to its corresponding set of ODEs.
- $G : Q \rightarrow \mathcal{G}$ denotes the set of guards where $\mathcal{G} = \mathbb{R}^n \rightarrow \text{Bool}$. We use $G(q)$ as the set of guards in mode q .

The Definition 3.1 was first seen in [12]. But we discard the definition of the next state variable value A' in discrete mode jumps. Because to suffice the proving in [11], the variable value assignment is not permitted.

Definition 3.2 (evaluation) An evaluation among $\{x_i\}$ is defined as a mapping function $v : V \rightarrow \mathbb{R}$ where the value of x_i is $v(i)$. We use $E(V)$ to denote all evaluations on variable set V .

Definition 3.3 (trajectory) A finite trajectory is a sequence.

$$\tau = (q_0, v_0)(q_1, v_1) \cdots (q_n, v_n)$$

where q_i is the mode of the i^{th} time unit and v_i is the state variables' evaluation of the i^{th} time unit.

3.2 Discrete Time Markov Chain

We know in hybrid systems, the complexity of ODEs leads to non closed form solutions. Numerical solvers like ODE45, ODE23 can be used to numerically solve out the state given a certain time. But the reverse, to find out the time points when jump conditions are satisfied, can be time inefficient. A useful way is to convert hybrid automata into Markov Chains. Markov Chains are transition systems where the state propagation is a stochastic event. A discrete time Markov Chain is a state transition system, while unlike decidable transition systems, the state change happens stochastically. Given a current state, the future state is traversed based on a certain probability distribution. Below is the formal definition of Markov Chain in [4].

Definition 3.4 (Markov Chain) *A Markov Chain can be described as a tuple $M = (S, P, l_{init}, AP, L)$,*

- S is a countable, nonempty set of states,
- $P : S \times S \rightarrow [0, 1]$ is the transition probability function such that for all state s :

$$\sum_{s' \in S} P(s, s') = 1,$$
- $l_{init} : S \rightarrow [0, 1]$ is the initial distribution, such that $\sum_{s \in S} l_{init}(s) = 1$, and
- AP is a set of atomic propositions
- $L : S \rightarrow 2^{AP}$

As shown in Fig. 3.2, we can select state *eat* as our initial state, then the next state could be anyone among *rest*, *jog*, *play* based on their transition probability. Note that the sum of the probability values of the edges coming out of the same node should equal to 1.

3.3 Bounded Linear Temporal Logic

To test our properties against the generated trajectories, we adopt Bounded Linear Temporal Logic. Linear means that the time behavior in our trajectories only have one

Figure 3.2: Simple Example of Markov Chain

direction. Unlike other temporal logics like Computational Tree Logic (*CTL*), that has branching behavior in terms of time. Temporal means that we only observe at discrete time points of a transition system TS , this is suitable as we adopt a discrete time Markov Chain instead of a continuous one. Bounded means that there is an upper bound of the trace that we generate and test out of the transition systems.

The set of atomic propositions AP contains element of state labels, meaning the numerical and logical assertions on a single state. For example, based on the variables x and y of the interested system, $x \geq a$, $y \leq b$ are both atomic propositions. Considering all such atomic propositions on the trajectory, we are able to test the BLTL properties. Basic BLTL properties are divided in *boolean* and *timed*.

Let's assume ψ_1 and ψ_2 are two atomic propositions, then the boolean properties are listed below:

$$\psi_1 \wedge \psi_2 \quad |\psi_1 \vee \psi_2 \quad |\neg\psi \quad (3.1)$$

Other boolean operators could all be derived from the above basic boolean operators.

For the properties of the trace, there are mainly 4 operators. U as until, where $\psi_1 U \psi_2$ means ψ_1 holds until ψ_2 holds. $\bigcirc\psi$ means next state ψ holds. $\Diamond\psi$ means eventually ψ will

Figure 3.3: The BLTL Properties

hold. This is often used in reachability problems. $\Box\psi$ means globally ψ holds. Detailed explanations could be seen in Fig. 3.3. The original picture is from [4].

Other complex properties could all be obtained by the above basic properties.

3.4 Importance Sampling

We write $P(X) \sim X$ to show that $P(X)$ is the probability distribution according to the random variable X . The Monte Carlo's method is often adopted to approximate the distribution. In Eq. 3.2, r represents our sample rate, $\hat{p}$ is our estimate of the probability and p is our true probability value. We can see that $\hat{p} \rightarrow p$ when sample rate tends to infinity.

$$\lim_{r \rightarrow \infty} \hat{p} = p \quad (3.2)$$

However, due to limited computing resource, we have an upper bound B_r on sample rate. There are common situations when we need to use Monte Carlo's method to estimate $E(f(x))$ according to $P(X) \sim X$. While the value $f(x)$ on $A \subseteq X$ is significantly larger than that on A^C , where A^C is the complement of set A . Or when we use fixed sample rate while $f(x)$ is a periodic function where its period equals to sample period. In these situations, we need to introduce the *importance sampling* method. The basic idea is to replace the original normal distribution with new probability distribution $Q(X) \sim X$,

while maintaining our desired expectation value $E_p(f(x))$. As depicted in Eq. 3.3 based on normal distribution and Eq. 3.4 based on importance sampling, it is easy to tell that $E_p(f(x)) = E_q(\frac{f(x)p(x)}{q(x)})$.

$$E_p(f(x)) = \int_X f(x)p(x)dx \quad (3.3)$$

$$E_q(\frac{f(x)p(x)}{q(x)}) = \int_X \frac{f(x)p(x)}{q(x)}q(x)dx \quad (3.4)$$

There is no doubt that the new distribution $Q(X) \sim X$ should also satisfy that $\int_X q(x)dx = 1$, while the sample rate on A should exceed that on A^C for $Q(X) \sim X$. Next we have our importance sampling estimate with n samples given in Eq. 3.5, where $x_i \sim q$ means the random variable x_i follows the probability distribution q .

$$\hat{\mu}_q = \frac{1}{n} \sum_{i=1}^n \frac{f(x_i)p(x_i)}{q(x_i)}, x_i \sim q \quad (3.5)$$

We thus have the $E_q(\hat{\mu}_q) = \mu$, and we have

$$\sigma_q^2 = \int \frac{(f(x)p(x))^2}{q(x)}q(x)dx - \mu^2 \quad (3.6)$$

As a good sample $Q(X) \sim X$ is with a smaller σ_q^2 , and μ is a fixed value, we need only get a smaller $\int_X \frac{(f(x)p(x))^2}{q(x)}q(x)dx$. We will introduce our design of $Q(X) \sim X$ in Chapter 4.4.

3.5 Hybrid Concolic Testing

Hybrid Concolic Testing (also called dynamic symbolic execution) is a program verification method initially raised by K. Sen and G. Agha in [23, 25, 26].

Random testing and *Symbolic execution* [16] are the main branches in program verification. Random testing gives a concrete random value to any variable that needs an input. It is easy for this method to reach deep program states in a relatively short period of time. The industry often adopts this method as it is handy and efficient.

Symbolic execution, on the other hand, is capable of covering every state but its time consumption is prodigious. It does not give any concrete value to variables but only

Figure 3.4: Hybrid Concolic Testing

record constraints on the execution path (e.g. conditions from *if* and *switch*, etc.). With the help of SMT solvers, symbolic execution can find if a path is reachable. However, it does not work well in large programs because the time consumption is unacceptable.

Hybrid Concolic testing is a mixed proposal of random testing and symbolic execution. As can be seen in Fig. 3.4 originated from [23]. Starting from the *INIT*, hybrid concolic testing starts from random trajectory generation, then it turns to symbolic execution when random testing does not hit new states in a given period of time. After solving out new states, the algorithm continues with random testing to reach deeper of the states.

3.6 dReach Introduction

dReal [10] is an SMT solver for first-order logic formulas over the reals. It can handle formulas in the framework of δ -complete decision procedure. In theorem proving and formal verification, SMT formulas can be hard to verify when nonlinear functions come in the way. In this work, a decision procedure is said to be δ -complete if for any φ from set S of SMT formulas the procedure returns below answers:

- unsat: φ is unsatisfiable.
- δ -sat: φ^δ is satisfiable.

The original φ is a $\sum_1$ -sentence in the form $\varphi := \exists^I x (\wedge_i^m (\vee_{j=1}^{k_i} f_{ij}(x) = 0))$, while the symbol φ^δ is defined as the δ -weakening of φ as $\varphi^\delta := \exists^I x (\wedge_i^m (\vee_{j=1}^{k_i} |f_{ij}(x)| \leq \delta))$. Here δ is a user specified positive real number meant for a mathematical relaxation on the

original φ . With this relaxation, dReal improves numerical algorithms' efficiency for nonlinear problems and provide correctness guarantees at the same time.

dReach [9] is a hybrid system safety verification tool which encodes safety critical questions into SMT formulas and then calls dReal to implement the δ -complete decision procedure. So dReach performs bounded δ -complete reachability analysis.

The input file for dReach can be written in a specified *.drh* file which depicts the mode transitions, mode related ODEs and safety properties. The *.drh* file allows for value assignment during mode transitions.

3.7 Summary

In this chapter, we introduced the definition of hybrid automata and used a simple jumping ball example to demonstrate. We also introduced discrete time Markov Chain. In next chapter, we will review the method in [11] of converting hybrid automata to Markov Chain. We then introduced Bounded Linear Temporal Logic (BLTL). We will adopt BLTL to describe the safety-critical properties in the coming chapters. We then discussed the importance sampling and why it is a useful method in spotting rare events. In the next chapter, we will introduce how to implement this method to alter the traditional sampling method based on normal distribution. We then introduced the hybrid concolic testing idea borrowed from program verification. We will implement this idea in our tool *HyChecker* shown in the next chapter. At last we introduced the tool dReach. In the next chapter, we will discuss how to adopt this tool to solve the satisfiability problems of mode transitions.

Chapter 4

Hybrid System Verification based on Importance Sampling

As shown in Fig. 4.1, our method is summarized in this flow graph. Firstly, we convert our hybrid automata H into the Markov Chain M , then randomly generate trajectories based on the transition probability. We would then call dReach to try to check satisfiability of uncovered edges. We use importance sampling to adjust our sample distribution. And then convert back to random trajectory until we gain the required BLTL length.

This chapter is organized as listed below. In Chapter 4.1, we talk about how to convert hybrid automata to the discrete time Markov Chain. In Chapter 4.2, we discuss how to borrow the hybrid concolic testing idea in program verification in our method. In Chapter 4.3, we mention dReach’s capability of symbolically checking satisfiability problems. In Chapter 4.4, we give our mathematical proof of importance sampling. In Chapter 4.5, we introduce our algorithm.

4.1 Hybrid Automata to Discrete Time Markov Chain

The basic idea in [11] is to consider mode transitions in the hybrid automata models as stochastic events and build Markov Chains accordingly. By testing generated trajectories of this Markov Chain, we can tell if $M \models \psi$. According to Theorem 4.1, which is the main result in [11], the test equally reflects whether $H \models \psi$.

Theorem 4.1 $H \models \psi$ iff $M \models \psi$

Figure 4.1: Major Flow Graph of Our Method

From [11], we know that the set of all possible trajectories in Hybrid Automaton H equals to that of the Markov Chain M . So we can convert the problem of testing Hybrid Automaton H to testing trajectories generated in Markov Chain. The Markov Chain is established using a probability propagation method. Let's start from *INIT* in q_{in} , assume we pick random points on *INIT* according to normal distribution. Other distributions could also be adopted, this is for simplicity purpose. Then for every point we pick, we sample from $(0, 1)$ on time T and use numerical solvers like ODE23, ODE45 to calculate the state, and then test them against outgoing mode guards and generate the state on time 1 according to the new mode characteristics as shown in Eq. 4.1. It shows the value of Φ when $t = 1$ by first simulating the value of $\Phi_q(t, v)$ according to dynamics of mode q , then making a sudden mode jump at time t and then simulating the rest time $(t, 1)$ by dynamics of mode q_i .

$$\Phi_{t=1} = \Phi_{q_j}(1 - t, \Phi_q(t, v)) \quad (4.1)$$

Then we do integration for every picked points on *INIT* to get a new probability distribution on each possible outgoing modes. Then we carry on this method to generate a Markov Chain accordingly.

$$P_{X_j}(Y) = \int_{v \in X} \int_{t \in T_j(v)} 1_{(\Phi_{q_j}(1-t, \Phi_q(t, v)) \cap Y)} dP_{T_j(v)} dP(X) \quad (4.2)$$

Figure 4.2: The Markov Chain

As Eq. 4.2 suggests, where $P_{X_j}(Y)$ is the probability distribution in original mode, $T_j(\vec{v})$ is given in Eq. 4.3 and g_j is the set of all states that would satisfy mode j , and 1_Z is the indicator function of set Z .

$$T_j(\vec{v}) = \{t \mid \Phi_q(t, \vec{v}) \in g_j\} \quad (4.3)$$

As depicted in Fig. 4.2 from [11] for the Markov Chain, every node in this tree structure only has one father, while it is allowed to have multiple children. Here, ρ represents the current mode, and it should have multiple outgoing modes. X_j represents the set of all possible states calculated by Eq. 4.1, and P_{X_j} represents the probability distribution on X_j .

The transition probability p_j is calculated in Eq. 4.4. In this equation, $\mu(T_j(\vec{v}))$ is the measure of $T_j(\vec{v})$.

$$p_j = \int_{\vec{v} \in X} \frac{\mu(T_j(\vec{v}))}{\sum_{l=1}^m \mu(T_l(\vec{v}))} dP(X) \quad (4.4)$$

The behavior of H and M are depicted using bounded linear-time temporal logic (BLTL), this work is able to handle formulas like ψ , ψ' , $F^{\leq l}\psi$, $G^{\leq l}\psi$, $\psi U^{\leq l}\psi'$.

Figure 4.3: The Hybrid Concolic Guard Covering

4.2 Introducing Hybrid Concolic Testing into Random Trajectory Generation

Similar to hybrid concolic testing in program verification, we introduce the exhaustive search guard satisfiability test during the process of trajectory generation. As can be seen in Fig. 4.3, when the trajectory generation process hits state A , it has three outgoing modes. For all the trajectories, it will then randomly sample points based on normal distribution and test the guards' satisfiability for each value vector generated.

As in Fig. 4.3, the vectors represent guards. The guards in solid lines are covered at least once and the guards in dotted lines are never satisfied. Now we find that in trajectory 1, only state B was covered. In trajectory 2, only C was covered. In trajectory 3, B and C were both covered, leaving only D uncovered. Note that in this case, we only check the guard D 's satisfiability for all 3 trajectories but not B or C , as they were covered at least once in all trajectories.

4.3 Use dReach to Find Importance Region

For a given entrance value state $\vec{v} \in V$ in a specific mode, the default way in [11] is to sample based on normal distribution. While we know that for a mode whose guard conditions are rarely satisfied, there exists a set $A \subset (0, 1)$ with $\mu(A) \ll \mu((0, 1))$. Our goal is to find A or any subset of A . As shown in the previous chapter, we now call dReach to symbolically generate the answer. This coverage problem could be equally defined as: starting from the initial point v_{init} , whether the next mode D is reachable in the given time step dt . dReach would be called to solve this simple reachability problem, if no such time set is returned, dReach would generate unsat. If this reachability property holds, dReach would return a small satisfiable region for us to carry on with importance sampling.

4.4 Importance Sampling according to Importance Region

After finding the importance region $A \subset (0, 1)$, we will adjust our sampling from $N(t) \sim t$ to $R(t) \sim t$. Recall in Eq. 4.2 we had the inner integration based on $T_j(\vec{v})$, now we do some modification on this equation. We note that the value of indicator function 1_Z equals to 0 for any $t \notin T_j(\vec{v})$, so we can equally change our integration region to $(0, 1)$. Then we propose our importance sampling distribution as given in Eq. 4.5.

$$R(t) = \begin{cases} \frac{a}{(a-1)\mu(A)+\mu(B)} & t \in A \\ \frac{1}{(a-1)\mu(A)+\mu(B)} & t \notin A \end{cases} \quad (4.5)$$

Here a represents the ratio of sampling rate for $t \in A$ compared to that where $t \notin A$. And $(a-1)\mu(A) + \mu(B)$ is a normalization factor to make $\int_0^1 R(t)dt = 1$. And so we can reform the integration as shown in Eq. 4.6. It is easy to tell that Eq. 4.2 and Eq. 4.6 are equivalent.

$$P_{X_j}(Y) = \int_{\vec{v} \in X} \int_{t \in (0,1)} \frac{1_{(\Phi_{q_j}(1-t, \Phi_q(t, v)) \cap Y)} N(t)}{R(t)} dR(t) dP(X) \quad (4.6)$$

Here $N(t)$ is our nominal distribution, $R(t)$ is the importance distribution and the adjustment factor $N(t)/R(t)$ is called the likelihood ratio. We can see that $N(t) = 1$ for any $t \in (0, 1)$, so we can shorten Eq. 4.6 to get Eq. 4.7.

$$P_{X_j}(Y) = \int_{\vec{v} \in X} \int_{t \in (0,1)} \frac{1_{(\Phi_{q_j}(1-t, \Phi_q(t,v)) \cap Y)}}{R(t)} dR(t) dP(X) \quad (4.7)$$

And so we will sample the value of $\frac{1_{(\Phi_{q_j}(1-t, \Phi_q(t,v)) \cap Y)}}{R(t)}$ according to $R(t) \sim t$ and get $E_r(\frac{1_{(\Phi_{q_j}(1-t, \Phi_q(t,v)) \cap Y)}}{R(t)})$ for the inner integration.

4.5 HyChecker Algorithm

So now we are able to generate our Algorithm 1.

From lines 1 $\sim$ 2, the algorithm takes in hybrid automata and BLTL formula, and it will get the information from the machine the number of cores L . At line 3, it will take L samples uniformly from the *INIT*. Lines 4 $\sim$ 20 are our main body of the algorithm. We process the L sampled points, and firstly do random point generation to cover regular guards in lines 5 $\sim$ 8. Then we do checking on set of guards to see if there's any guard that has never been covered in lines 9 $\sim$ 15, if yes, we shall return an importance region for testing. In lines 16 $\sim$ 20, we pick a guard to jump in every trajectory based on the probability we gained. In lines 21, we shall get the whole trajectory and test our BLTL property for the L traces. In lines 22 $\sim$ 27, we check our outcome, if the BLTL is violated in any of the L traces, we shall give False assertion and stop our algorithm immediately. If the BLTL is not violated, we shall carry on with the next L traces until all N samples are done and we give the True assertion.

The important symbols are explained below. τ is our trajectory. v is the state points. q is the modes in the hybrid automata and Markov Chain. g is the set of guards in the hybrid system. $\hat{T}$ is the set of time points that would satisfy certain guards. L is number of cores of any machine we run on our program. N is the total number of points we check, it's a user configurable parameter.

Algorithm 1: HyChecker Procedure

Data: Hybrid automaton $H = (Q, q_{in}, V, INIT, f, G)$, BLTL formula, maximum time step K

Result: Trajectory $\tau_0, \dots, \tau_{n-1}$. True or False assertion on BLTL formula

```

1 Initialization;
2 Get information on number of cores  $L$  of the machine running the program.;
3 Sample  $L$  points  $v_0 \dots v_{c-1}$  uniformly from  $INIT$ ;
4 for  $k:=1, \dots, K$  do
5     for  $i:=0, \dots, (L-1)$  do
6         Generate time points  $T := t_1, \dots, t_J$  uniformly in  $(0, 1)$ ;
7         Simulate  $v_i^j := \Phi_{q_i^{k-1}}(t_j, \vec{v}_i^{k-1})$ , for  $j \in \{1, \dots, J\}$ ;
8         Let  $\hat{T}_j := \{t \in T : \vec{v}_i^j \in g_j\}$  be the time points where  $g_j$  is enabled;
9         if there are outgoing guards that are never satisfied then
10             call dReach to check whether they could be satisfied;
11             if sat then
12                 return importance region;
13             else
14                 do nothing
15             Importance sampling according to importance region to get  $\hat{T}_{rare}$ ;
16         for  $i:=0, \dots, (L-1)$  do
17             Pick  $g_l$  randomly according to probability
18              $\{p_j := |\hat{T}_j| / (\sum_{u=1}^m |\hat{T}_u| + |\hat{T}_{rare}|)\}$ ;
19             Pick  $t_l$  uniformly at random from  $\hat{T}_l$ ;
20             Simulate  $\vec{v}_i' := \Phi_{q_i'}(1 - t_l, \vec{v}_i^l)$ , where  $q_i'$  is the target of  $g_l$ ;
21             Set  $q_i^k := q_i'$ ,  $\vec{v}_i^k := \vec{v}_i'$ , and extend  $\tau_i := (q_i^0, \vec{v}_i^0) \dots (q_i^k, \vec{v}_i^k)$ ;
22 return  $\tau_0, \dots, \tau_c$  for BLTL checking;
23 if Found counterexample then
24     Give assertion: False;
25 else
26     Continue with next round  $L$  samples;
27 if Finished with all  $N$  samples then
28     Give assertion: True;
```

4.6 Summary

In this chapter, we looked back to the method of converting a hybrid automaton to a discrete time Markov Chain proposed in [11]. We then discussed our method of intro-

ducing hybrid concolic testing into random trajectory generation. We further introduced how to use dReach to spot importance region where rare events happen and how to do importance sampling according to the importance region. At the end of this chapter, we wrote the algorithm of our tool *HyChecker*. In the next chapter, we will test our tool with 4 hybrid systems in different scales. We will compare the results of our importance sampling method and the original method proposed in [11].

Chapter 5

Experiments and Evaluation

In this Chapter, we will use *HyChecker* to check three hybrid systems. The sewerage system is a simple model we come out. It contains a rare event in terms of a storm weather. The room heating system and the navigation benchmark are both from [8]. They are popular benchmarks among many hybrid system checking algorithms and tools.

We shall check the reachability problem for all three cases. These reachability problems are either safety critical or give guidance in solving real world problems. The below experiments are done in Ubuntu 14.04 with Intel(R) Xeon(R) CPU E5-1650 v2 @3.50GHz and 16 GB of memory.

5.1 Sewerage System

Let us consider a simple sewerage system whose hybrid automaton form is depicted in Fig. 5.1.

Our *INIT* mode depicts the behavior of sewerage water flow. The ODE is shown in Eq. 5.1. If flow exceeds threshold *thre_high*, the hybrid system would jump to *mode 1* with more pipes open. If flow falls below threshold *thre_low*, the hybrid system would jump to *mode 2* with pipes closed. However, when storms come, the water flow would increase to a high value extremely quickly and exceed *thre_storm*, and the system should jump to *mode 3* with emergent drain off scheme. The time span of storm is very short, and we consider it as our *rare event*.

$$\frac{dflow}{dt} = -\frac{t - \mu}{\sigma^2} max_storm \exp\left(-\frac{(t - \mu)^2}{2\sigma^2}\right) + \omega max_normal \cos(\omega t) \quad (5.1)$$

Figure 5.1: Hybrid Automaton Form of Sewerage System

To explicitly analyze the system, we firstly solve out the primitive function in closed form as shown in Eq. 5.2. $flow_{t=0}$ is our initial value of $flow$.

$$flow = max_storm \exp\left(-\frac{(t - \mu)^2}{2\sigma^2}\right) + max_normal \sin(\omega t) + flow_{t=0} \quad (5.2)$$

Then we write our guard towards *mode 3* in Eq. 5.3, through which we can tell that the solution region is approximately $(\mu - \sigma, \mu + \sigma)$. And for settling the parameter σ , we define 4σ as the time ratio of the storm event, which is 10 *min* in our setting. For convenience, we normalize the region $(0, 24)$ *hour* down to $(0, 1)$. According to our calculation, in random testing the sample rate J should exceed 288 in order to surely find the region satisfying guard 3.

$$flow \geq thre_storm = max_storm \exp\left(-\frac{(-\sigma)^2}{2\sigma^2}\right) + flow_{t=0} \quad (5.3)$$

The parameters were set as follows. Time step dt was set to be 1, total sample rate J is set to 10, the max_len was set to 3. Thus the BLTL formula we tested were shown in Eq. 5.4.

$$\Box(\neg(mode\ 2)) \quad (5.4)$$

Table 5.1: Sewerage System Checking Outcome

Test No.	Importance Sampling Outcome/time(s)	Random Sampling Outcome/time(s)
1	False/2.7	True/623.4
2	True /21.5	True/624.3
3	False/13.3	True/667.9
4	False/5.7	False/404.1
5	False/2.7	True/666.2
6	False/2.8	True/650.8
7	False/3.9	True/648.2
8	False/4.5	True/657.9
9	False/6.8	True/650.6
10	False/4.1	True/653.3

We provide parameter settings for *HyChecker* so that it can run in both importance sampling mode and in pure random testing mode. We did 10 experiments for the sewerage system with the BLTL formula in Eq. 5.4. Our results are shown in Table. 5.1. As can be seen, importance sampling mode were able to give the False assertion in 9 of the total 10 runs, while we only failed once with importance sampling. As for pure random sampling, we got the True assertion for 9 runs, which means that the pure random mode were not able to discover the rare guard. We only got the False assertion for 1 run.

Note that our algorithm involves randomness, so the time consumption and result could differ between different runs.

We also tested in pure random sampling method with J to be 100, and the algorithm gave the False assertion to the tested BLTL property, which fulfills our expectation. However, though this assertion is correct, the time consumption is 2509.2 second. It's generally much longer than the importance sampling method.

5.2 Room Heating System

The room heating system was proposed in [8]. This benchmark denotes an n room house's thermodynamics behavior. Here n is set to 3, larger numbers could also be applied. The temperature inside each room is affected by the outside temperature, the temperature

Figure 5.2: The Room Heating System

difference with the adjacent room and whether there is a heater inside. The heater will be turned off if the temperature exceeds the upper bound *high_temp* or turned on if the temperature falls beyond the lower bound *low_temp*. If the current room fulfills the below conditions: the temperature in current room is smaller than *get_heater*, the temperature difference with the adjacent room is larger than *dif_temp*, the current room does not have a heater inside and the adjacent room has a heater inside, then the heater in the adjacent room could be transferred to the current room.

As can be seen in Fig. 5.2 drawn in *GeoGebra* [14], that depicts the whole heater moving scheme, the heater can only be moved between adjacent rooms. So the mode jump can only happen between the lower left state and the middle state or between the lower right state and the middle state. In the 3 major states, there are 4 sub states, as both the 2 heaters have state of *on* or *off*. If heater movement happens, the moved heater in the new room should be immediately *on* regardless of its original state.

The ODE functions of the room heating system could be seen in Eq. 5.5. x_{init} takes value on normal distribution from $([20, 20.5] \times [20.20.5] \times [20, 20.5])^T$. Further, if $x_i \geq temp_high_i$, the heater in the room will be turned off. If $x_i \leq temp_low_i$, the heater

will be turned on. For a room to take a heater from the adjacent room, the following state conditions must hold: room i has no heater inside, room j has a heater inside, the temperature in room i is significantly low as $x_i \leq fetch_i$ and the temperature difference between i and j should be large enough, having $x_j - x_i \geq diff_{ij}$. We thus set the parameters to be, $temp_low_i = 20$, $temp_high_i = 21$, $fetch_i = 1$, with i ranging from $i = 0, 1, 2$.

$$\dot{x} = \begin{pmatrix} -0.9 & 0.5 & 0 \\ 0.5 & -1.3 & 0.5 \\ 0 & 0.5 & -0.9 \end{pmatrix} x + \begin{pmatrix} 0.4 \\ 0.3 \\ 0.4 \end{pmatrix} u + diag(6, 7, 8)h \quad (5.5)$$

We use *HyChecker* to test against the property shown in Eq. 5.6. Where the initial condition is both heaters inside room $R1$ and room $R2$. So the BLTL property tests that in 5 days both heaters will not leave the original rooms. We have done 10 test runs with importance sampling and random testing. The results are listed in Table. 5.2. It is clear that though the average time consumption is relatively higher than that with the pure random testing mode, *HyChecker* is able to discover counterexample in all the 10 test runs. So we can give the assertion that property $C1$ will be violated. That in 5 days, one of the heaters will eventually leave the original room.

$$C1: \Box^{\leq 5}([Heater \text{ in } R1] \wedge [Heater \text{ in } R2]) \quad (5.6)$$

5.3 Navigation Benchmark

The navigation benchmark was firstly proposed in [8]. As shown in Fig. 5.3, the arrows are pointing different directions with unequal vector length. These are the target velocity v_d , whose value ranges with $(\sin(i\pi/4), \cos(i\pi/4))$, where $i = 0, 1, \dots, 7$. Imagine that an object initiates at some velocity in one of the grid. It will move in the grid with the acceleration given in Eq. 5.7. However, if the object travels to another grid, it will change its acceleration rate because the v_d in the grid is possibly different. In other situations, the object could go out of boundary of the 5×5 matrix. In such cases, the v_d will be the same of nearest grid.

Table 5.2: Room Heating System Checking Outcome

Test No.	Importance Sampling Outcome/time(s)	Random Sampling Outcome/time(s)
1	False/178.4	True/93.9
2	False /177.0	True/95.7
3	False/180.9	True/92.3
4	False/179.2	True/89.4
5	False/227.2	True/92.8
6	False/177.9	True/116.1
7	False/178.3	True/116.2
8	False/178.1	True/110.3
9	False/186.9	True/109.9
10	False/227.2	True/108.4

$$\frac{d\vec{v}}{dt} = A(\vec{v} - \vec{v}_d) \quad (5.7)$$

Here the matrix A is the eigenvalue matrix to make sure the velocity converges to v_d . We inherit the same parameters for A as shown in Eq. 5.8

$$A = \begin{pmatrix} -1.2 & -0.2 \\ -0.1 & -0.8 \end{pmatrix} \quad (5.8)$$

In this hybrid system, we have 4 variables. Variables x_0 and x_1 denotes the velocity at x and y dimension of the 2D plane respectively. Variables x_2 and x_3 are location variables. We divide the hybrid system modes based on the x_2 and x_3 values, and write jump conditions also according to them. The ODEs for x_0 and x_1 are different based on which mode the system is in, while the x_2 and x_3 are always the same as shown in Eq. 5.9.

$$\frac{dx_2}{dt} = x_0 \quad \frac{dx_3}{dt} = x_1 \quad (5.9)$$

Note that in Fig. 5.3, we have the destination grid A . We thus test the reachability problem in Eq. 5.10.

Figure 5.3: The Navigation System

$$\Box(\neg atom(A)) \quad (5.10)$$

We have tested that starting from location (3.5, 3.5) with the initial speed taking any value combination from $[-0.2, 0.2] \times [-0.2, 0.2]$, whether the property expressed in Eq. 5.10 would be violated. In the random testing mode, all 10 test runs are *true* assertion, while in the importance sampling mode, we are able to find counterexample in 9 runs, while only 1 run failed to spot counterexample.

5.4 Secure Water Testbed (SWaT)

The Secure Water Treatment (SWaT) test bed is a raw water purification laboratory located at Singapore University of Technology and Design.

Fig. 5.4 depicts the simplified version of SWaT system. S represents the raw water inlet source, which we use Eq. 5.11 to describe. mi means the *max inlet* water ranging in $(0.0004, 0.0012) m^3/s$. The circles represent the motorised pumps. The arrows depict the direction of water flows. Dirty water is firstly collected in the raw water tank. The dirty water is then pumped to tank C , which is the ultrafiltration feed tank. Water will then be pumped out of tank C to go through the ultrafiltration cycle, which consists of 4 processes. The ultrafiltration process takes water from tank C . The filtrated water

Table 5.3: Navigation Benchmark Outcome

Test No.	Importance Sampling Outcome/time(s)	Random Sampling Outcome/time(s)
1	False/1049.0	True/32.2
2	False /879.0	True/33.3
3	False/1626.2	True/92.3
4	False/816.7	True/32.9
5	False/243.3	True/32.2
6	False/87.8	True/22.5
7	True/3162.7	True/22.9
8	False/1180.3	True/32.9
9	False/884.9	True/33.9
10	False/770.8	True/33.3

goes to tank RF , which is the reverse osmosis feed tank, and the dirty water goes to drain 2. The backwash process will pump water from the backwash tank, and force the accumulated particles to be drained to $D1$. Then in the drain off process, the backwash tank pump will stop, let the UF module to drain off. In the refill process, water will be pumped from tank C and refill the UF module. Then the UF module will carry on with the next cycle. The reverse osmosis cycle contains 2 processes. In the pressurizing process, water pressure will gradually go up. In the reverse osmosis process, clean water will be produced and collected in the permeate tank P , and the dirty water containing ions will be collected in backwash tank.

This simplified SWaT system can be written in the hybrid automaton form. We use the actuators to divide discrete modes. Actuators in this system infer to motorised valves and motorised pumps with the states of *on* or *off*, both can be controlled by the programmable logic control (PLC). If we use totally n motorised actuators in this simplified model, we will have 2^n modes in the hybrid automaton. The ODEs in each mode depicts the hydraulic characteristics.

$$mi * \sin(\omega * t) + 0.6 * mi \quad (5.11)$$

In the simplified model, a safety critical property will arise. The backwash tank will collect dirty water with ions in the reverse osmosis process, and pump out water in the

Figure 5.4: The Schematic of Secure Water Testbed (SWaT)

backwash process. For the correct functioning of the system, we need to know whether the water level in this tank will stay in the range $(level_{low}, level_{high})$. If the water level is out of this range, the system needs to be shut down and checked before reboot. So we write our property in Eq. 5.12.

$$\neg \Diamond^{\leq 30}(H \leq 0.4 \text{ or } H \geq 1.2) \quad (5.12)$$

We did 10 test runs with our *HyChecker*. In random testing mode, we are not able to spot a violation. While in importance sampling mode, we found counterexample in 9 runs. Only 1 run failed to spot a counterexample.

We have in this test bed the parameters as listed: $J = 10$, $dt = 15$, $maxlen = 30$.

5.5 Evaluation and Discussion

Our experiments focused on spotting counterexamples in reachability problems.

In the sewerage benchmark, we are able to spot the storm event in *HyChecker*. While given same computing budget, the pure random testing method fails. If we boost the sample rate, the random test method is able to spot the counterexample, but the time consumption is about 250 times longer than our algorithm. In the room heating system, we assert the heater will eventually move in the 5 day period, where the random testing method was not able to find. In the navigation benchmark, we are also able to assert

Table 5.4: SWaT Outcome

Test No.	Importance Sampling Outcome/time(s)	Random Sampling Outcome/time(s)
1	False/27.9	True/321.4
2	True /250.1	True/321.1
3	False/395.1	True/327.3
4	False/12.1	True/ 322.1
5	False/26.4	True/324.3
6	False/39.9	True/320.9
7	False/193.3	True/ 318.9
8	False/26.4	True/ 326.2
9	False/242.2	True/330.2
10	False/12.4	True/323.7

that the grid A is reachable. In transportation safety testing, this could be very useful results. And in the simplified SWaT system results, we show that our method is scalable and efficient in complex systems. With our importance sampling method, the True assertion, which is incorrect in our experiments settings, each appears once in the experiments results of the sewerage system, navigation benchmark and the SWaT system. This arises from the fact that our method bears randomness. Our method can heavily boost the possibility of spotting counterexamples compared to random sampling method, but there is still possibility that the counterexample is not spotted in one test run. In the counterpart, the random sampling method is able to generate the False assertion in the sewerage system test run, which is the correct result in our experiment setting. This reflects that random sampling is still able to spot the counterexample, though with much less possibility. Generally, our tool *HyChecker* is able to spot counterexamples quicker compared to the random testing algorithm given same computing budget. Our experiment results with the SWaT system shows that our method is scalable and can be adopted on large scale systems. It is often the case that compositional verification is applied on verifying large scale hybrid systems. While with our method, we only need to obtain the hybrid automata from the entire hybrid system and do verification on the given BLTL formulas.

Chapter 6

Conclusions and Future Works

In this master thesis, we studied hybrid system verification in probabilistic model checking techniques. We investigated the major stochastic checking algorithms and tools. We based our work on the previous work of checking hybrid systems in terms of Markov Chains. We focused on the rare event problem that would cause inefficiency and inaccuracy in the previous work. We thus proposed a revised algorithm stressed on spotting rare events in probabilistic checking for hybrid systems. We built a model checking tool *HyChecker* based on our algorithm. *HyChecker* was built in a parallel manner that allows us to fully adopt the computing power of multi core machines so that our algorithm is scalable for large system verification. We tested our method with *HyChecker* on three typical hybrid systems: the sewerage system, the room heating system and the navigation benchmark. In our experiments, we found that our tool is able to detect counterexamples when checking the reachability problems expressed in BLTL formula. While comparably, traditional pure random testing algorithms are either time consuming or not capable of.

Based on our results, we could assert that *HyChecker* is generally more powerful in terms of spotting counterexamples. It is useful in generating safety guarantees for real world engineering problems and in verifying cyber physical systems.

Future works could be done in our framework. For example, spotting the rare traces that has a very small measure in *INIT* is very challenging, as it needs to consider tracing back during trajectory generation. And to gain completeness guarantees for all traces, one could do trace abstraction before adopting our algorithm, or target our algorithm only at particular partition of *INIT* interested.

References

- [1] M. AlTurki and J. Meseguer. PVESTA: A parallel statistical model checking and quantitative analysis tool. In *Algebra and Coalgebra in Computer Science*, pages 386–392. Springer, 2011.
- [2] R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Lecture Notes in Computer Science*, pages 209–229, 1993.
- [3] E. Asarin, T. Dang, and O. Maler. The d/dt tool for verification of hybrid systems. In *Computer Aided Verification*, pages 365–370. Springer, 2002.
- [4] C. Baier, J.-P. Katoen, et al. *Principles of model checking*, volume 26202649. MIT press Cambridge, 2008.
- [5] P. Ballarini, H. Djafri, M. DufLOT, S. Haddad, and N. Pekergin. Cosmos: a statistical model checker for the hybrid automata stochastic logic. In *Quantitative Evaluation of Systems (QEST), 2011 Eighth International Conference on*, pages 143–144. IEEE, 2011.
- [6] B. Barbot, S. Haddad, and C. Picaronny. Coupling and importance sampling for statistical model checking. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 331–346. Springer, 2012.
- [7] B. Barbot, S. Haddad, C. Picaronny, et al. Importance sampling for model checking of continuous time markov chains. In *International Conference on Advances in System Simulation (SIMUL)*, pages 30–35, 2012.

REFERENCES

- [8] A. Fehnker and F. Ivančić. Benchmarks for hybrid systems verification. In *Hybrid Systems: Computation and Control*, pages 326–341. Springer, 2004.
- [9] S. Gao, S. Kong, W. Chen, and E. Clarke. Delta-complete analysis for bounded reachability of hybrid systems. *arXiv preprint arXiv:1404.7171*, 2014.
- [10] S. Gao, S. Kong, and E. M. Clarke. drealm: An smt solver for nonlinear theories over the reals. In *Automated Deduction—CADE-24*, pages 208–214. Springer, 2013.
- [11] B. M. Gyori, B. Liu, S. Paul, R. Ramanathan, and P. Thiagarajan. Approximate probabilistic verification of hybrid systems. *arXiv preprint arXiv:1412.6953*, 2014.
- [12] T. A. Henzinger. The theory of hybrid automata. In *Proc. 11th IEEE Symp. Logic in Computer Science*, pages 278–292. Springer, 2000.
- [13] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. Hytech: A model checker for hybrid systems. In *Computer Aided Verification*, pages 460–463. Springer, 1997.
- [14] M. Hohenwarter et al. Geogebra. <https://www.geogebra.org/>. Accessed: 2015-08-13.
- [15] C. Jegourel, A. Legay, and S. Sedwards. Importance splitting for statistical model checking rare properties. In *Computer Aided Verification*, pages 576–591. Springer, 2013.
- [16] J. C. King. Symbolic execution and program testing. In *Communications of the ACM*, volume 19, pages 385–394. ACM, 1976.
- [17] S. Kong, S. Gao, W. Chen, and E. Clarke. dreach: δ -reachability analysis for hybrid systems. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 200–205. Springer, 2015.
- [18] M. Kwiatkowska, G. Norman, and D. Parker. Prism: Probabilistic symbolic model checker. In *Computer performance evaluation: modelling techniques and tools*, pages 200–204. Springer, 2002.

REFERENCES

- [19] M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking with prism: A hybrid approach. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 52–66. Springer, 2002.
- [20] M. Kwiatkowska, G. Norman, and D. Parker. Stochastic model checking. In *Formal methods for performance evaluation*, pages 220–270. Springer, 2007.
- [21] M. Kwiatkowska, G. Norman, and D. Parker. Prism 4.0: Verification of probabilistic real-time systems. In *Computer aided verification*, pages 585–591. Springer, 2011.
- [22] K. G. Larsen, P. Pettersson, and W. Yi. Uppaal in a nutshell. *International Journal on Software Tools for Technology Transfer (STTT)*, 1(1):134–152, 1997.
- [23] R. Majumdar and K. Sen. Hybrid concolic testing. In *Software Engineering, 2007. ICSE 2007. 29th International Conference on*, pages 416–426. IEEE, 2007.
- [24] A. Platzer. *Logical analysis of hybrid systems: proving theorems for complex dynamics*. Springer Science and Business Media, 2010.
- [25] K. Sen. Concolic testing. In *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, pages 571–572. ACM, 2007.
- [26] K. Sen and G. Agha. Cute and jcute: Concolic unit testing and explicit path model-checking tools. In *Computer Aided Verification*, pages 419–423. Springer, 2006.
- [27] K. Sen, M. Viswanathan, and G. A. Agha. Vesta: A statistical model-checker and analyzer for probabilistic systems. In *International Conference on Quantitative Evaluation of Systems (QEST)*, volume 5, pages 251–252, 2005.
- [28] H. L. Younes. Ymer: A statistical model checker. In *Computer Aided Verification*, pages 429–433. Springer, 2005.