Scalable safety verification of stochastic hybrid systems

by

Ratan Lal

M.Tech., Indian Statistical Institute, Kolkata, 2014

AN ABSTRACT OF A DISSERTATION

submitted in partial fulfillment of the
requirements for the degree

DOCTOR of PHILOSOPHY

Department of Computer Science
Carl R. Ice College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2021

Abstract

Stochastic hybrid systems consist of software-controlled physical processes, where uncertainties manifest due to either disturbance in the environment in which the physical systems operate or the noise in sensors/actuators through which they interact with the software. The safety analysis of such systems is challenging due to complex dynamics, uncertainties, and infinite state space. This thesis introduces fully automated methods for bounded/unbounded safety analysis of certain subclasses of the stochastic hybrid systems against given safety specifications.

Our first contribution is to compute the maximum/minimum bounded probability of reachability of polyhedral probabilistic hybrid systems, where plant dynamics are expressed as a set of linear constraints over the rate of state variables, and uncertainties are involved in discrete transitions represented as discrete probability distributions over the set of locations/modes. Our broad approach is to encode all possible bounded probabilistic behaviors into an appropriate logic along with the given safety specifications. Then, we perform optimization over all possible behaviors for the maximum/minimum probability via state-of-the-art optimization solvers.

The second contribution is to present fully automatic unbounded safety analysis of the polyhedral probabilistic hybrid systems (PHS). We present a novel counterexample guided abstraction refinement (CEGAR) algorithm for polyhedral PHS. Developing a CEGAR algorithm for the polyhedral PHS is complex owing to the uncertainties in the discrete transitions, and the infinite state space due to the real-valued variables. We present a practical algorithm by choosing a succinct representation for counterexamples, an efficient validation algorithm and a constructive method for refinement that ensures progress towards the elimination of a spurious abstract counterexample.

The third contribution is to extend unbounded safety analysis to the class of linear probabilistic hybrid systems (PHS). Developing an abstraction for the linear PHS is a challenge when the dynamics is linear, because the solutions are exponential and require solving exponential constraints to construct the finite state MDP. Hence, we consider a hierarchical abstraction, where we first abstract a linear PHS to a polyhedral PHS using hybridization and then apply predicate abstraction to construct the finite state MDP.

Finally, we consider uncertainties in plant dynamics, and develop an abstraction based method for both bounded/unbounded safety analysis of linear stochastic systems. The bounded safety analysis is similar to the encoding in bounded model checking, where we encode bounded stochastic behaviors instead of continuous behaviors and solve the encoding using a semi-definite program solver. For the unbounded safety analysis, we abstract the linear stochastic system into a finite state system, and analyze its safety using graph search algorithms.

Scalable safety verification of stochastic hybrid systems

by

Ratan Lal

M.Tech., Indian Statistical Institute, Kolkata, 2014

A DISSERTATION

submitted in partial fulfillment of the
requirements for the degree

DOCTOR of PHILOSOPHY

Department of Computer Science
Carl R. Ice College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2021

Approved by:

Major Professor
Dr. Pavithra Prabhakar

Copyright

Abstract

Stochastic hybrid systems consist of software-controlled physical processes, where uncertainties manifest due to either disturbance in the environment in which the physical systems operate or the noise in sensors/actuators through which they interact with the software. The safety analysis of such systems is challenging due to complex dynamics, uncertainties, and infinite state space. This thesis introduces fully automated methods for bounded/unbounded safety analysis of certain subclasses of the stochastic hybrid systems against given safety specifications.

Our first contribution is to compute the maximum/minimum bounded probability of reachability of polyhedral probabilistic hybrid systems, where plant dynamics are expressed as a set of linear constraints over the rate of state variables, and uncertainties are involved in discrete transitions represented as discrete probability distributions over the set of locations/modes. Our broad approach is to encode all possible bounded probabilistic behaviors into an appropriate logic along with the given safety specifications. Then, we perform optimization over all possible behaviors for the maximum/minimum probability via state-of-the-art optimization solvers.

The second contribution is to present fully automatic unbounded safety analysis of the polyhedral probabilistic hybrid systems (PHS). We present a novel counterexample guided abstraction refinement (CEGAR) algorithm for polyhedral PHS. Developing a CEGAR algorithm for the polyhedral PHS is complex owing to the uncertainties in the discrete transitions, and the infinite state space due to the real-valued variables. We present a practical algorithm by choosing a succinct representation for counterexamples, an efficient validation algorithm and a constructive method for refinement that ensures progress towards the elimination of a spurious abstract counterexample.

The third contribution is to extend unbounded safety analysis to the class of linear probabilistic hybrid systems (PHS). Developing an abstraction for the linear PHS is a challenge when the dynamics is linear, because the solutions are exponential and require solving exponential constraints to construct the finite state MDP. Hence, we consider a hierarchical abstraction, where we first abstract a linear PHS to a polyhedral PHS using hybridization and then apply predicate abstraction to construct the finite state MDP.

Finally, we consider uncertainties in plant dynamics, and develop an abstraction based method for both bounded/unbounded safety analysis of linear stochastic systems. The bounded safety analysis is similar to the encoding in bounded model checking, where we encode bounded stochastic behaviors instead of continuous behaviors and solve the encoding using a semi-definite program solver. For the unbounded safety analysis, we abstract the linear stochastic system into a finite state system, and analyze its safety using graph search algorithms.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Embedded systems consist of software-controlled physical processes that are integral parts of automotive systems (cruise control[1;2], lane assistants[3–5], self-driving cars[6–8]), medical devices (pacemakers[9;10], infusion pumps[11;12]), robotics (Roomba[13;14], surgery robots[15;16]), aeronautics (autopilot[17;18], collision avoidance modules[19;20]), and smart grid[21;22], where stochastic disturbances manifest due to either disturbance in the environment in which the physical systems operate or the noise in sensors/actuators through which they interact with the software. These systems manifest uncertainties either in physical systems or digital controllers that arise due to either noise in the environment, modeling errors, or sensors/actuators' error. We also refer such systems to "probabilistic/stochastic hybrid systems" which formally extend finite state automaton with probabilistic transitions and continuous (stochastic) differential equations. One of the grand challenges is to build safe and reliable stochastic hybrid systems that can be deployed in the real world with high confidence. Towards this end, we need rigorous formal methods and accompanying software tools for the analysis to be automated and scalable.

1.2 Formal methods

Formal methods[23;24] are a special kind of mathematically rigorous techniques that have been used for the modeling of complex systems, the specification development and its verification for the systems. The goal of formal methods is to verify system's properties in a more thorough fashion than the empirical testing. While empirical testing increases confidence about the performance of the system, formal methods provide strong guarantees about the correctness of the systems. The formal techniques are mainly classified into two categories Model Checking[25–27] and Theorem Proving[28–31].

Model Checking verifies specifications against a system model to prove that system's behaviors conform to the specification. More precisely, given a mathematical model of the system, and a specification, it provides correctness of the system's behaviors with respect to the specification. This techniques suffer from the state space exploration due to exponential growth in number of states with the number of variables. To overcome the problem, researchers have developed techniques for the reduction of the state space; thus allowing verification to the large scale systems. The popular available Model Checking tools are UPPAAL[32], NUSMV[33], SPIN[34], and probabilistic Model Checking tools are PRISM[35], STORM[36].

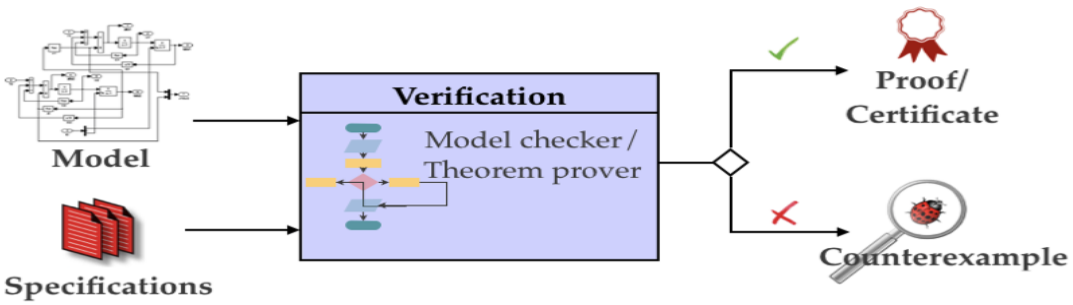

Figure 1.1: Formal methods

Theorem Proving is used for evaluating higher order logic that corresponds to the system's behaviors. It uses mathematical structure to build the logic. Theorem Proving does

not deal with the states, but with formulas. Although it does not take system's model as an input explicitly, it requires high degree of knowledge about the system under verification. The popular available Theorem Proving tools are Theorem Proving: ACL2[37], Coq[38], Isabelle/HOL[39], STeP[40], PVS[41] and Z3[42].

A general framework for the verification is shown in Figure 1.1. This framework takes as input either system model and a formal specification or a formal specification, and outputs either proof/certificate for the correctness if the specification is true or a counterexample, that is, a system's behavior violating the specification, which supports the correction of bugs if the specification is false.

1.3 Modeling of stochastic hybrid systems

Stochastic systems[43–45] have received extensive attention in recent years toward modeling and analyzing uncertain systems. In general, stochastic systems are modeled as a mixture

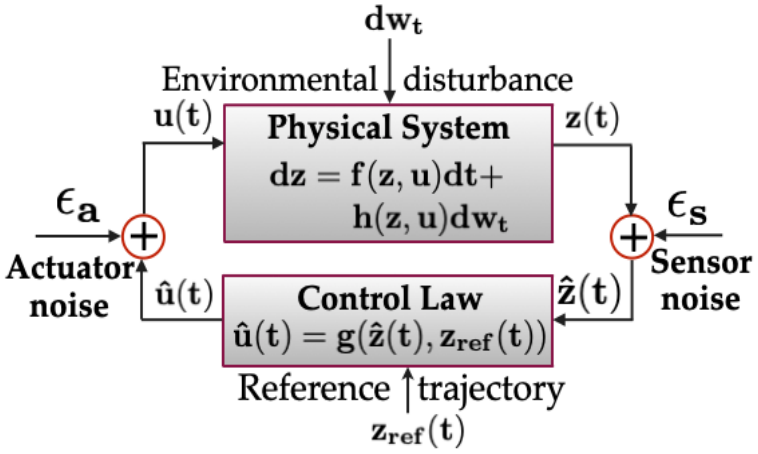

Figure 1.2: Stochastic hybrid systems

of continuous and discrete dynamics, where the continuous dynamics capture stochastic behaviors, which is described by stochastic differential/difference equations as shown in Figure 1.2, where $\mathbf{w_t}$ is a zero mean Gaussian process, and the discrete dynamics capture the control law, which is described by probabilistic systems, such as Markov chains, Markov

decision processes, where sensor/actuator noises are captured as probability distributions. In this thesis, we develop methods/techniques for the safety analysis of probabilistic/stochastic hybrid systems and its subclasses.

1.4 Main results of the thesis

We consider safety verification problems of probabilistic/stochastic hybrid systems that consist of either stochastic processes or probabilistic discrete transitions. The safety problem turns into probabilistic safety, that is, whether probability of reaching an unsafe region is below a given probability threshold. Probabilistic bounded/unbounded safety analysis of probabilistic/stochastic hybrid systems relies heavily on probabilistic reachability analysis. In general, the reachability analysis of stochastic hybrid systems is undecidable, however, it is decidable for certain subclass of the systems[46–48]. Probabilistic reachability problem for the general class of probabilistic hybrid system has been solved using abstraction techniques[49;50] applied in papers[51;52], that give the bound on the maximum/minimum probability of reachability. However, it is challenging to compute a precise probability of reachability. Hence, we have developed methods for the safety analysis of probabilistic/stochastic hybrid systems. In this thesis, we first focus on polyhedral dynamical systems to model the physical systems, and develop a method to compute the maximum/minimum probabilistic reachability, and counterexample guided abstraction refinement algorithm for the safety analysis of polyhedral probabilistic hybrid systems. Then, we consider linear dynamical systems to model the physical systems, and develop a hierarchical abstraction for the safety analysis of linear probabilistic hybrid systems. Finally, we consider linear discrete-time stochastic systems for modeling the physical systems, and develop a method for its parametric property verification.

1.5 Thesis outline

In this section, we outline the thesis. In Chapter 2, we provide an overview of related works. We present all the preliminary details in Chapter 3. In Chapter 4, we present the details of computing the maximum/minimum probabilistic reachability of polyhedral probabilistic hybrid systems(PHS). Then, we present a counterexample guided abstraction refinement algorithm for the probabilistic safety of the polyhedral PHS in Chapter 5. Next, we present a hierarchical abstraction for the probabilistic safety of linear PHS in Chapter 6. Then, we present a bounded model checking and an abstraction based method for bounded and unbounded parametric verification of stochastic systems, respectively, in Chapter 7. Finally, we present the conclusion of the thesis in Chapter 8.

Chapter 2

Background

In this chapter, we provide a literature background of different classes of finite and infinite-state probabilistic systems. Finite state probabilistic systems, such as Markov chains, Markov decision processes are often used to model behaviors of discrete systems, such as biological and power generator systems. Infinite-state probabilistic systems, such as stochastic systems, stochastic hybrid systems, and probabilistic hybrid systems are used to model software-controlled physical systems. The most popular problems for these classes of systems are reachability and safety analysis. Although efficient methods have been discovered for the finite state systems, it is a challenge for infinite-state systems. One promising direction to deal with the infinite-state systems is an abstraction, where an infinite-state system is abstracted into a finite state system. Next, we will discuss methods for different classes of probabilistic systems for different related problems.

2.1 Markov Chains

Markov Chains[53;54] are stochastic models. The model is a sequence of random events, where the probability of the next event depends only on the current event and does not depend on the past events. This property is referred to as Markovian property[55;56], and stochastic information is stored in the form of transition probability. Markov Chains are widely used for

modeling the dynamics of biological systems[57;58], power generator[59], and financial systems[60], such as risk management, inventory control. Modeling of transition probabilities depends on the type of systems, such as discrete/continuous-time and their parameters. This leads to the investigation of different variants of Markov chains, such as discrete-time Markov chains (DTMC)[61], where events occur in discrete time, continuous-time Markov chains[62], where events occur in continuous time; interval (Int.) Markov chains[63], where transition probabilities are modeled as intervals, parametric (Par.) Markov chains[64], where transition probabilities are expressed in the form of parameters. Different properties of the stochastic models are often expressed in the form of probabilistic computation tree logic (PCTL)[65], continuous stochastic logic (CSL)[66;67], linear temporal logic (LTL)[68], and temporal computation tree logic (TCTL)[69]. For each variation of Markov Chains, several techniques, have been developed for the verification of the above properties.

For the verification of PCTL and CSL, three value abstraction method[70] has been developed for DTMC, where an abstract state-space model is generated over the values true, false, and unknown. The properties can then be evaluated on such three-valued models. If the evaluation is unknown, then the abstract model is refined until the properties of interest can be either proven or refuted. Traditionally, probabilistic model checkers, such as PRISM[35] is time-consuming for large DTMC due to state-space exploration. Hence, bi-simulation minimization[71] is developed that reduces the state space (up to logarithmic saving). Another approach is Statistical Model Checking (SMC) that infers the correctness of the probabilistic systems by evaluating certain test statistics on random samples (paths) generated from the probabilistic models. Different methods based on frequentist and Bayesian approach have been developed for SMC. A frequentist based algorithm for SMC of non-nested CSL on Discrete-time Markov chain (DTMC) has been presented in the paper[72] by conducting the sequential Wald Probability Ratio Test[73]. However, for nested CSL, a Bayesian statistical model checking algorithm[74] is developed that a correct answer with a certain confidence. Also, a method[75] based on run time model checking has been

Systems	Problems	Approaches
D(C)TMC	PCTL[65], CSL[66;67]	Three-valued abstraction[70]
-	-	Bisimulation minimization[71]
-	TCTL[69], PCTL[65], CSL[66;67]	Statistical Model Checking[78]
CTMC	Synthesizing parameters for PTBR	Discretization with refinement[79]
-	Counterexample for CSL[66;67]	Heuristic guided search based[80;81]
DTMC	Reliability checking	Run time model checking[75]
-	Counterexample for PCTL[65]	Reducing into graph problem[76]
-	ω-regular event under condition ω-regular	Reducing into unconditional DTMC[77]
Int. MC	max. prob. of ω-regular	Exp.-max. and LTL to unamb. automata[82]
Par. MC	Rational fun. expressing prob. reachability	Tightly interwining Reg. exp.[64]

Table 2.1: Background works on Markov chains

developed for the reliability checking on DTMC. Other problems, such as counterexample of PCTL[65] and the conditional probability of ω-regular events under ω-regular condition, have been solved by reducing PCTL property over DTMC into weighted graph problem[76] and reducing conditional probability problem over DTMC into unconditional probability problem[77], respectively. Also, Statistical model checking[78] has been developed for the verification of temporal computation tree logic (TCTL).

For CTMC, synthesizing parameters for probabilistic time bounded reachability and counterexample for CSL[66;67] problems have been solved by discretization of parameter ranges with refinement[79] and explicit state model checking based heuristic guided search[80;81], respectively. For IMC, maximizing the probability of satisfying ω-regular expressed as LTL[68] has been solved by expectation-maximization procedure and reducing LTL[68] into unambiguous automata[82]. For PMC, computing rational function expressing probabilistic reachability has been solved by reducing PMC into tightly interwining regular expression by its evaluation[64]. The above works are summarized in Table 2.1.

2.2 Markov decision processes

Markov Decision Processes (MDP) is an extended version of Markov chains where multiple probability distributions may be associated with each state. Different techniques have been

developed for the verification of different properties on MDP.

For safety property, the maximum/minimum probability of reaching a desired set of states is required to compute. For this, policy resolving non-determinism methods, such as value iteration, have been used to compute the bound on maximum/minimum probability of reachability with some precision, but they do not have stopping criteria. However, stopping criteria is straightforward in interval iteration algorithm[83]. Also, abstraction and refinement based method[84] has been developed for computing the upper and lower bound on the probabilistic reachability. In addition, counterexample generation for bounded probabilistic LTL property over MDP has been produced by reducing MDP into Acyclic Markov Chains and generating counterexamples for reachability property over Acyclic Markov Chains[85]. Reachability property over infinite-state MDP has been proposed based on abstraction[86] and implemented in tool PASS[87]. Bounded MDP (BMDP) is a generalization of MDP where the probability of each transition is uncertain. Reachability property over BMDP has been solved using the Interval value iteration method[88]. Reachability property for partially information probabilistic systems (PIPS) is undecidable[89], so an algorithm introduced in the paper[90] checks whether a total information scheduler complies with the partial information assumptions. If they do not comply with the assumptions, the model is modified using refinement.

For PCTL property, different methods, namely, game based abstraction[91;92], predicate abstraction with SMT solver[93], counterexample guided abstraction refinement (CEGAR) with predicate abstraction and interpolation based refinement[94], linear programming[95], successive refinement[96], value iteration[97], decomposition of MDP into strongly connected components (SCC) and application of policy iteration on each SCC[90], probabilistic model checker[98], have been developed. Also, conditional probability on CTL property over MDP has been solved via reducing MDP into acyclic MDP[99].

For qualitative property, a counterexample guided abstraction refinement (CEGAR)[100] has been developed over multiple MDPs, where simulation relation is first obtained for MDP

Systems	Problems	Approaches
MDP	Reachability	Interval iteration algorithm[83]
-	-	Abstraction-refinement based[84]
-	PCTL	Mutl-valued abs., CEGAR, and game based abs.[91;92]
-	-	predicate abstraction and SMT[93]
-	-	CEGAR (Pred. abst. and iterpolation based refine.)[94]
-	-	Linear programming[95]
-	-	Decomposing into SCC and apply Policy iteration[90]
-	reachability of PIPS	CEGAR[90]
-	PCTL, LTL, pSafe	Model checker, Automata based techniques[102]
-	cpCTL	Reducing into Acyclic MDP[99]
-	Counter. gen. for bounded pLTL	Reducing into Acyclic MDP[85]
-	PCTL, CSL, PTCTL	Sampling and monte carlo approximation[95]
-	CE for safety, liveness frag. of PCTL	based on simulation relation[101]
-	Counterexample for ω regular	Critical subsubsystem based on MILP[86]
MDPs	Qualitative property	CEGAR for simulation relation[100]
Infinite MDP	Reachability	Tool PASS[86]
BMDP	Reachability	Interval value iteration method[88]

Table 2.2: Background works on Markov decision processes

with respect to the qualitative property, and then CEGAR is used to obtain the combined simulation. A notion of a counterexample for the rich classes of specifications, such as safety, liveness fragment of PCTL over MDP have been introduced as a lexicographic ordering of a pair of MDP M and canonical simulation relation between M and M (original MDP)[101]. A method based on Mixed Integer Linear Programming (MILP)[86] has been developed for finding a counter-example of ω regular expression expressed in the form of a sub-DTMC of a model. The above works are summarized in Table 2.2.

2.3 Stochastic systems

Stochastic systems[103;104] are continuous systems, where randomness is involved in their behaviors. These systems are broadly classified into discrete/continuous-time stochastic systems. In the discrete-time stochastic systems, randomness appears at each discrete-time, and they are often in the form of Gaussian noise. While in continuous-time stochastic systems, they evolve over time.

Different subclasses of discrete-time stochastic systems, namely, discrete-time Max-plus

linear stochastic systems (DtMPLSS), Linear stochastic systems (DtLSS), non-linear stochastic systems (DtNLSS), and Interconnected discrete-time linear stochastic systems (IDtLSS), have been studied for time-difference and PCTL specifications, where the time-difference specification is the difference between states with/without delay at some iteration k.

For the Max-plus linear stochastic systems, a bounded model checking algorithm[105] has been developed based on predicate abstraction for the time difference specifications. For Discrete-time linear stochastic systems (DTLSS), where continuous dynamics is monotone, a model checking algorithm[106] has been developed for the PCTL specification, where the system is abstracted into a finite state interval Markov chain that over-approximates system's behavior, and state-space heuristic has been discussed when the specification is false on the interval Markov chain. While a counterexample guided abstraction refinement algorithm[107;108] for the verification and synthesis of DTLSS for the PCTL specification have been presented, where the system is abstracted into an interval Markov chain and a bounded Markov decision process, respectively, and refinement schemes for both verification and synthesis are discussed. An abstraction and aggregation method[109] of DTLSS is developed, where the system is abstracted as a finite state Markov chain via a finite partitioning of the continuous state space. In the above methods, the error between DTLSS and its abstraction has not been quantified. However, an approximate Markovian abstraction procedure[110] is developed by combining an existing approximate abstraction procedure with a bi-simulation like a refinement algorithm, where adaptive refinement algorithm is employed to achieve a given desired error bound. A set-theoretic approach[111] for the verification of robust obstacle avoidance is discussed based on inner and outer convex approximation of the exact non-convex capture set.

For Discrete-time non-linear stochastic systems, a discrete abstraction method[112] is presented based on bi-simulation function and convex optimization. For interconnected discrete-time linear systems, a composition approach[113] based on a stochastic simulation function, which makes the relationship between the original and abstract system, is discussed. The

Systems	Problems	Approaches
DtMPLSS	Bounded model checking	Predicate abstraction[105]
DtLSS	Model checking	Approximate as IMC[106]
-	PCTL	Counterexample guided abstraction refinement[107;108], abstraction[109]
-	Abstract system	Approximate abstraction[110]
-	Obstacle avoidance	Set-theoretic approach[111]
DtNLSS	Abstract system	Bi-simulation[112]
IDtLSS	Model checking	Stochastic simulation[113]
CtSCSS	Synthesis	Discrete time abstraction[114]

Table 2.3: Background works on stochastic systems

stochastic simulation function can be used to quantify the error between the original and abstract systems. The above works are summarized in Table 2.3.

However, there are limited works for continuous-time stochastic systems. A discrete abstraction based approach[114] has been investigated for the synthesis of continuous-time stochastic control stochastic systems (CtSCS), where the system is abstracted into a finite Markov decision process, and their probabilistic distance is quantified based on stochastic simulation function.

2.4 Stochastic hybrid systems

Stochastic hybrid systems (SHS)[115–118] exhibit a combination of continuous and discrete behaviors where randomness is involved in both the behaviors. Different methods have been investigated, specifically for LTL and safety/reachability specification for different subclasses of SHS, namely, discrete-time stochastic hybrid systems (DtSHS) where continuous behaviors are evaluated at discrete time instant, controlled DtSHS where stochastic behaviors are affected by control inputs.

For LTL verification on DtSHS, a product-construction based method[119] has been developed, where the system is abstracted via discrete abstraction, and product between the abstract system and Buchi automata for LTL is construction. Safety verification problem on controlled DtSHS is dealt via optimal control problem[120] of a certain controlled Markov

Systems	Problems	Approaches
DtSHS	LTL	Product construction and discrete time abstraction[119]
Controlled DTSHS	Reachability	Stochastic optimization prob.[120]
-	safety	Optimal control problem[121]
SHS	safety/reachability	Testing based methods[122]
-	safety	finite abstraction[123]
-	reachability	Numerical approximation[124], Barrier certificate[125]
-	Control	Discrete abstraction[126]

Table 2.4: Background works on stochastic hybrid systems

process[121], where safety verification problem is formulated as an optimal control problem. Next, a testing based method[122] has been developed for the safety/reachability of SHS using the notion of robustness for test runs. Safety property of SHS has been proposed based on abstraction which is computed by a solver for reachability in a non-probabilistic version of the hybrid systems in[123]. In addition, reachability analysis of large-scale SHS as a problem of rare event estimation has been developed in[127] based on an aggregation of the discrete mode process and important sampling approaches for the system. Reachability analysis of SHS[124] has been presented based on numerical approximation by the discretization of the state space and using an interpolated Markov chain to approximate the switching diffusion weakly. Also, reachability analysis of SHS has been presented based on barrier certificate[125], which is super-martingale (i.e, its expected value is non-increasing along time) under the given system dynamics. They put the condition on the barrier certificate function that its value at the initial state should be lower that its value at any point in the unsafe region. The probability of reaching the unsafe region is then bounded from above using a Chebyshev-like inequality for super- martingales. A discrete abstraction based method[126] has been presented in based on the notion of bounded bi-simulation, where the original system is transformed into a finite state system by abstracting state space into finite states and continuous probability distributions into discrete probability distributions. The above works are summarized in Table 2.4.

2.5 Probabilistic hybrid systems

Probabilistic hybrid systems (PHS) are a special subclass of SHS, where randomness occurs either in the initial state or on the discrete transitions. Safety/reachability analysis problems have been studied via different methods.

A δ-reachability based method[128] has been discovered for PHS, where randomness appears in the initial state. In the method, a weaker notion of δ-reachability is used in which the unsafe set is over-approximated by a user defined parameter δ. A statistical testing based method[?] has been developed for the reachability analysis. Also, a stochastic satisfiability modulo theory (SSMT)[129] based method[130] has been discovered for probabilistic bounded reachability problems of concurrent PHS. Safety verification of PHS[131] has been solved by lifting probabilities from PHS, abstraction on non-probabilistic systems, and introducing probabilities in the abstracted system.

In this thesis, we explore probabilistic hybrid systems, where continuous dynamics are expressed in the form of linear/polyhedral dynamics, and randomness appear on discrete transitions which we capture via non-deterministic probability distributions over the set of discrete locations. We focus on developing efficient methods for the reachability/safety analysis problems of PHS.

First, we consider bounded probabilistic reachability analysis of polyhedral PHS. Although the paper[132] has discovered to solve the bounded probabilistic reachability problem using bounded reachability of non-probabilistic hybrid systems[133;134], it is only applicable for deterministic probabilistic choices. However, we are interested in bounded reachability analysis of polyhedral PHS consisting of non-deterministic probabilistic choices. Since different probabilistic choices will lead to different probability of reachability, we consider computing exact maximum/minimum probabilistic reachability, which is an optimization problem.

Next, we consider probabilistic safety analysis of polyhedral PHS. The problem for the linear probabilistic hybrid systems has been solved using abstraction techniques[49;50], which have been applied in papers[51;52], that give the bound on the maximum/minimum probability

of reachability, and the safety is concluded by comparing the probabilistic reachability with the given probability threshold. However, this does not provide precise probabilistic safety analysis.

Our broad approach for precise probabilistic safety analysis of polyhedral PHS is based on counterexample guided abstraction refinement (CEGAR) framework. CEGARbased methods have been developed in the context of finite state probabilistic systems[101;135] and non-probabilistic hybrid systems[136–138], that have shown promising results. However, to the best of our knowledge, CEGARbased method has not been discovered for stochastic/probabilistic hybrid systems. Hence, we have discovered first CEGAR algorithm for models that have both probabilities and polyhedral continuous dynamics.

Next, we consider the probabilistic safety analysis of linear PHS. Developing an algorithm like CEGAR is complex for linear PHS due to two reasons. First, constructing an abstract system for the linear PHS is difficult because the solution of a linear dynamical system consists of an exponential expression, and there is no known efficient solver for a formula consisting of exponential expressions. Second, validating an abstract counterexample require exact reachability analysis of linear dynamical systems, which is not known. Hence, we have developed a hierarchical abstraction and compared it with the tool ProHVer[52].

Finally, several probabilistic properties have been investigated including probabilistic safety, as well as those expressed using probabilistic computation tree logic (PCTL)[65], and continuous stochastic logic (CSL)[66;67]. Here, we investigate the parametric property verification of discrete-time linear stochastic systems, which is different from traditional predicate abstract techniques for stochastic systems. We present a novel predicate abstraction based method. In traditional predicate abstraction, a finite-state system is constructed based on the partition of the continuous state-space; however, in our approach, we partition the set of random states, which are random vectors defined over the continuous state-space. Our approach provides a promising direction for the verification of parametric properties of stochastic systems.

Chapter 3

Probabilistic hybrid systems

3.1 Preliminaries

In this section, we present basic notations and definitions that we use in the rest of this thesis.

Numbers and sets: Let $\mathbb{R}, \mathbb{R}_{\geq 0}$, $\mathbb{Z}$, and $\mathbb{N}$ denote the set of real numbers, non-negative real numbers, integer numbers, and natural numbers, respectively. Given a countable set $\mathcal{S}$ of real numbers, $\sum \mathcal{S}$ denotes the sum of all elements in the set, that is, $\sum \mathcal{S} = \sum_{s \in \mathcal{S}} s$. We use $[n]$ and $\zeta(n)$ to denote the set $\{1, \ldots, n\}$ and the set of all subsets of $\mathbb{R}^n$, respectively. Given an element q and a set $\mathcal{S}$, we use $(q, \mathcal{S})$ for $\{q\} \times \mathcal{S}$ and $(\mathcal{S}, q)$ for $\mathcal{S} \times \{q\}$. Given a set $\mathcal{S}$, we use $\mathcal{S}_{\mathbb{N}}$ to denote the set $\mathcal{S} \times \mathbb{N}$. Furthermore, relation $Id \subseteq \mathbb{R}^n \times \mathbb{R}^n$ expresses the identity relation. $\Im(f)$ denotes the set of restricted reverse onto functions of f as $\Im(f) = \{f_1 : B \to A \mid B \subseteq \mathcal{S}_2,\ A \subseteq \mathcal{S}_1,\ f_1 \text{ is an onto function and } fof_1 = Id\}$. Given two sets $\mathcal{S}_1$, $\mathcal{S}_2$, $\mathcal{C}(\mathcal{S}_1, \mathcal{S}_2)$ denotes the Cartesian product of the sets $\mathcal{S}_1$ and $\mathcal{S}_2$, that is, $\mathcal{C}(\mathcal{S}_1, \mathcal{S}_2) = \{(s_1, s_2) \mid s_1 \in \mathcal{S}_1,\ s_2 \in \mathcal{S}_2\}$. In addition, given a set $\mathcal{S} = \mathcal{C}(\mathcal{S}_1, \mathcal{S}_2)$, we use $\mathcal{S}\lfloor_1$, and $\mathcal{S}\lfloor_2$ to denote the components $\mathcal{S}_1$, and $\mathcal{S}_2$, respectively.

Vectors, tuples and matrices: Let $\mathbb{R}^n$ denote the n-dimensional Euclidean space. Let $\mathbf{x} = [x_1, x_2, \ldots, x_n] \in \mathbb{R}^n$ denote the n-dimensional vector. Also, we use $(_t, j)$ to express the j^{th} projection of a tuple $t = (x_1, x_2, \ldots, x_n)$, that is, $(_t, j) = x_j$. We use $\mathbf{x}_i$ to denote the i^{th} element of $\mathbf{x}$, that is, $\mathbf{x}_i = x_i$. Given a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{A}[i, j]$ denotes $(i, j)^{th}$ entry of the matrix $\mathbf{A}$. A square matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is called a symmetric matrix if $\mathbf{A} = \mathbf{A}^{\mathrm{T}}$, where $\mathbf{A}^{\mathrm{T}}$ denotes the transpose of the matrix $\mathbf{A}$. In addition, a square matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is called a positive semi-definite matrix if $\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x} \geq 0$ for all $\mathbf{x} \in \mathbb{R}^n$. We use $\mathbb{S}_n^+$ to denote the set of all real symmetric and positive semi-definite matrices of size $n \times n$. Given an n^2 dimensional real vector $\mathbf{x}$, $\langle \mathbf{x} \rangle$ denotes the matrix of size $n \times n$ corresponding to the vector $\mathbf{x}$ obtained by splitting $\mathbf{x}$ into rows with n elements, that is,

$$\langle \mathbf{x} \rangle = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \ldots & \mathbf{x}_n \\ \vdots & \vdots & \vdots & \vdots \\ \mathbf{x}_{n^2-n+1} & \mathbf{x}_{n^2-n+2} & \ldots & \mathbf{x}_{n^2} \end{bmatrix}.$$

Also, given a set $\mathcal{S} \in \zeta(n^2)$, $\langle \mathcal{S} \rangle$ denotes the set of all matrices of size $n \times n$ corresponding to the set $\mathcal{S}$ that is, $\langle \mathcal{S} \rangle = \{\langle \mathbf{x} \rangle \mid \mathbf{x} \in \mathcal{S}\}$.

Polyhedra: A set $\mathcal{S} \in \zeta(n)$ is called an n-dimensional polyhedron if there exist a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ and a constant vector $\mathbf{b} \in \mathbb{R}^m$ for some $m \in \mathbb{N}$ such that $\mathcal{S} = \{\mathbf{x} \mid \mathbf{A}\mathbf{x} \leq \mathbf{b}\}$. We use $Poly(n)$ to denote the set of all n-dimensional polyhedra.

Partition of a Set: A partition of a set $\mathcal{S}$ is a set of subsets $\{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_k\}$ such that $\bigcup_{i=1}^{k} \mathcal{S}_i = \mathcal{S}$ and $\overset{0}{\mathcal{S}}_i \cap \overset{0}{\mathcal{S}}_j = \emptyset$, $1 \leq i, j \leq k$, $i \neq j$, where $\overset{0}{\mathcal{S}}$ denotes the interior region of $\mathcal{S}$. In addition, a partition $\mathcal{P}$ is called a polyhedral partition if each set $\mathcal{S} \in \mathcal{P}$ is a polyhedron. A partition element $\mathcal{S}_i$ is adjacent to $\mathcal{S}_j$ denoted as $\mathcal{S}_i \rhd \mathcal{S}_j$ if $\mathcal{S}_i \cap \mathcal{S}_j$ is non empty and $i \neq j$.

Probability distributions: We use $Dist(\mathcal{S})$ to denote the set of all probability distributions over the set $\mathcal{S}$, that is, $Dist(\mathcal{S}) = \{\pi : \mathcal{S} \to [0, 1] \mid \sum_{s \in \mathcal{S}} \pi(s) = 1\}$. We use $support(\pi)$ to

denote the set of elements $s \in \mathcal{S}$ for which $\pi(s) \neq 0$.

3.2 Timed Markov decision processes

In this section, we define timed Markov decision processes and its certain subclass.

Let $Dist(\mathcal{S})$ denote the set of all probability distributions over the set $\mathcal{S}$, that is, $Dist(\mathcal{S}) = \{\rho : \mathcal{S} \to [0,1] \mid \sum_{s \in \mathcal{S}} \rho(s) = 1\}$. We use $support(\rho)$ to denote the set of all elements $s \in \mathcal{S}$ for which $\rho(s) \neq 0$. We assume that for any probability distribution $\rho \in Dist(\mathcal{S})$, its support is finite. Next, we formally define syntax and semantics of the timed Markov decision processes.

Definition 1 (Timed Markov Decision Process (TMDP))**.** *A* TMDP *is a structure* $\mathcal{T} = (\mathcal{S}, \longrightarrow)$*, where:*

- $\mathcal{S}$ *is a set of states;*
- $\longrightarrow \subseteq \mathcal{S} \times \mathbb{R}_{\geq 0} \times Dist(\mathcal{S})$ *is a transition relation capturing a set of timed probabilistic edges.*

We denote a timed probabilistic edge $(s, t, \rho) \in \longrightarrow$ by $s \xrightarrow{t} \rho$. Note that a state may have multiple timed probabilistic edges associated with it, that is, distinct edges $(s, t_1, \rho_1), (s, t_2, \rho_2) \in \longrightarrow$, where $t_1 \neq t_2$ *or* $\rho_1 \neq \rho_2$. Thus, the transition relation allows non-determinism.

A path of TMDP $\mathcal{T} = (\mathcal{S}, \longrightarrow)$ is a finite sequence of states and times, $\sigma = s_0 t_1 s_1 t_2 s_2 \ldots t_n s_n$ such that there exists a sequence of probability distributions $\rho_1 \rho_2 \rho_3 \ldots \rho_n$ which satisfy $s_i \xrightarrow{t_{i+1}} \rho_{i+1}$ and $\rho_{i+1}(s_{i+1}) > 0$ for $0 \leq i \leq n-1$. $\sigma[i]$ denotes the i^{th} state of the path σ, that is, $\sigma[i] = s_i$; $len(\sigma)$ represents length of the path σ, that is, $len(\sigma) = n$, and $L(\sigma)$ represents the last state of the path σ, that is, $L(\sigma) = \sigma[len(\sigma)]$.

Also, $\mathcal{D}(\sigma)$ denotes the duration of the path σ, that is, $\mathcal{D}(\sigma) = \sum_{i=1}^{len(\sigma)} t_i$. $Paths(\mathcal{T})$ denotes the set of all paths of $\mathcal{T}$, and $Paths_k(\mathcal{T}, s, \mathbb{F}) = \{\sigma \in Paths(\mathcal{T}) \mid len(\sigma) = k,\ \sigma[0] = s,\ \sigma[k] \in \mathbb{F},\ \text{for } 0 \leq i < k,\ \sigma[i] \notin \mathbb{F}\}$. $Paths(\mathcal{T}, s, \mathbb{F})$ represents the set of all paths that start at

state s and end at some state t in $\mathbb{F}$ and no states in between s and t are in $\mathbb{F}$, that is, $Paths(\mathcal{T}, s, \mathbb{F}) = \bigcup_k Paths_k(\mathcal{T}, s, \mathbb{F})$.

Furthermore, we define a special subclass of TMDP, where there will be a unique probability distribution associated with each state.

Definition 2 (Timed Markov Chain (TMC)). *A* TMC *is a* TMDP $\mathcal{T} = (\mathcal{S}, \longrightarrow)$ *where the following condition holds:*

- *For each state* $s \in \mathcal{S}$ *if there exist* $s \xrightarrow{t_1} \rho_1$, $s \xrightarrow{t_2} \rho_2$, *then* $t_1 = t_2$ *and* $\rho_1 = \rho_2$.

Since TMC allows only one timed probabilistic edge for each state, we can define a probabilistic transition function $Pr : \mathcal{S} \times \mathcal{S} \to [0, 1]$, where $Pr(s, s') = \rho(s')$ if there exists $s \xrightarrow{t} \rho$.

Next, we explain how to resolve the non-determinism in a TMDP $\mathcal{T} = (\mathcal{S}, \longrightarrow)$ and obtain a TMC. First, we define a *scheduling function* to be a partial function $\gamma : Paths(\mathcal{T}) \to \mathbb{R}_{\geq 0} \times Dist(\mathcal{S})$ such that if $\gamma(\sigma) = (t, \rho)$, then $L(\sigma) \xrightarrow{t} \rho$. Let $\Gamma(\mathcal{T})$ denote the set of all scheduling functions for a TMDP $\mathcal{T}$.

Definition 3. *Given a* TMDP $\mathcal{T} = (\mathcal{S}, \longrightarrow)$ *and a scheduling function* γ, *we obtain a* TMC $\mathcal{T}_\gamma = (\mathcal{S}_\gamma, \longrightarrow_\gamma)$, *where* $\mathcal{S}_\gamma = Paths(\mathcal{T})$; *for each path* $s_\gamma \in Paths(\mathcal{T})$, $s_\gamma \xrightarrow{t}_\gamma \rho_\gamma$ *if* $\exists\ \rho$ *such that* $L(s_\gamma) \xrightarrow{t} \rho$ *and for all* $s \in \mathcal{S}$, $\rho_\gamma(s_\gamma t s) = \rho(s)$.

Markov decision processes

Markov decision processes are a special class of timed Markov decision processes, where time is unconstrained on the probabilistic edges.

Definition 4 (Markov Decision Process (MDP)). *An* MDP *is a structure* $\mathcal{M} = (\mathcal{S}, \longrightarrow)$, *where*

- $\mathcal{S}$ *is a finite (infinite) set of states;*
- $\longrightarrow \subseteq \mathcal{S} \times Dist(\mathcal{S})$ *is a transition relation.*

Next, we define discrete time Markov chains, which are MDPs, with at most one probabilistic edge associated with each state.

Definition 5 (Discrete-Time Markov Chain (DTMC))**.** *A* DTMC *is an* MDP $\mathcal{T} = (\mathcal{S}, \longrightarrow)$ *such that for each state* $s \in \mathcal{S}$*, if there exist* $s \longrightarrow \rho_1$ *and* $s \longrightarrow \rho_2$*, then* $\rho_1 = \rho_2$*.*

An execution of an MDP can be interpreted as a DTMC that is obtained by resolving the non-determinism in each step with a particular state distribution. We use $Exec(\mathcal{M})$ to denote the set of all DTMCs obtained by resolving the non-determinism.

3.2.1 Probabilistic reachability

Let us consider a TMC/DTMC $\mathcal{T} = (\mathcal{S}, \longrightarrow)$. Let $\mathcal{P}_{\mathcal{T}}(\sigma)$ denote the probability associated with a path σ in $\mathcal{T}$; we will drop the subscript $\mathcal{T}$ when it is clear from the context. We define $\mathcal{P}(\sigma)$ inductively as follows. If $len(\sigma) = 0$, then $\mathcal{P}(\sigma) = 1$.

$$\mathcal{P}(\sigma) = \prod_{i=1}^{len(\sigma)} Pr(\sigma[i-1], \sigma[i]).$$

Bounded probabilistic reachability

We define bounded probabilistic reachability of both TMC and TMDP.

- For TMC $\mathcal{T} = (\mathcal{S}, \longrightarrow)$, the probability of reaching a target set $\mathbb{F}$ from a state s with path length exactly k and at most k within time T, respectively, are defined as,

$$\mathcal{P}_{(T,=k,\mathbb{F})}(\mathcal{T}, s) = \sum \{\mathcal{P}(\sigma) \mid \sigma \in Paths_k(\mathcal{T}, s, \mathbb{F}),\ \mathcal{D}(\sigma) \leq T\},$$

$$\mathcal{P}_{(T,\leq k,\mathbb{F})}(\mathcal{T}, s) = \sum_{i=0}^{k} \mathcal{P}_{(T,=i,\mathbb{F})}(\mathcal{T}, s).$$

- For TMDP $\mathcal{T} = (\mathcal{S}, \longrightarrow)$, we define the maximum and minimum probability of reaching a target set $\mathbb{F}$ from a state s with path length at most k within time T, respectively,

to be:

$$\mathcal{P}^{sup}_{(T,\leq k,\mathbb{F})}(\mathcal{T}, s) = \sup_{\gamma \in \Gamma(\mathcal{T})} \mathcal{P}_{(T,\leq k,\mathbb{F})}(\mathcal{T}_\gamma, s),$$

$$\mathcal{P}^{inf}_{(T,\leq k,\mathbb{F})}(\mathcal{T}, s) = \inf_{\gamma \in \Gamma(\mathcal{T})} \mathcal{P}_{(T,\leq k,\mathbb{F})}(\mathcal{T}_\gamma, s).$$

Inductive definition of bounded probabilistic reachability

Here, we provide an alternate inductive definition for both $\mathcal{P}^{inf}_{(T,\leq k,\mathbb{F})}(\mathcal{T}, s)$ and $\mathcal{P}^{sup}_{(T,\leq k,\mathbb{F})}(\mathcal{T}, s)$.

- Base case:

$$\mathcal{P}^{inf}_{(T,\leq 0,\mathbb{F})}(\mathcal{T}, s) = \mathcal{P}^{sup}_{(T,\leq 0,\mathbb{F})}(\mathcal{T}, s) = \begin{cases} 1, \text{ if } s \in \mathbb{F} \\ 0, \text{ otherwise} \end{cases}$$

- Induction step:

  If $s \in \mathbb{F}$, then

$$\mathcal{P}^{inf}_{(T,\leq k,\mathbb{F})}(\mathcal{T}, s) = \mathcal{P}^{sup}_{(T,\leq k,\mathbb{F})}(\mathcal{T}, s) = 1.$$

  Otherwise,

$$\mathcal{P}^{inf}_{(T,\leq k,\mathbb{F})}(\mathcal{T}, s) = \inf_{s \xrightarrow{t} \rho} \left(\sum_{s' \in \mathcal{S}} \rho(s') \mathcal{P}^{inf}_{(T-t,\leq k-1,\mathbb{F})}(\mathcal{T}, s') \right);$$

$$\mathcal{P}^{sup}_{(T,\leq k,\mathbb{F})}(\mathcal{T}, s) = \sup_{s \xrightarrow{t} \rho} \left(\sum_{s' \in \mathcal{S}} \rho(s') \mathcal{P}^{sup}_{(T-t,\leq k-1,\mathbb{F})}(\mathcal{T}, s') \right).$$

Unbounded probabilistic reachability

We define unbounded probabilistic reachability of both DTMC and MDP.

- For DTMC $\mathcal{T} = (\mathcal{S}, \longrightarrow)$, the probability of reaching a set of states $\mathbb{F}$ from a state s is defined as,

$$\mathcal{P}(\mathcal{T}, s, \mathbb{F}) = \sum_{\sigma \in Paths(\mathcal{T}, s, \mathbb{F})} \mathcal{P}(\sigma).$$

- For MDP $\mathcal{M} = (\mathcal{S}, \longrightarrow)$, we define the minimum and maximum probability of reaching a set of states $\mathbb{F}$ from a state s, denoted as $\mathcal{P}_{inf}(\mathcal{M}, s, \mathbb{F})$ and $\mathcal{P}_{sup}(\mathcal{M}, s, \mathbb{F})$, respectively. Let $\bowtie \in \{inf, sup\}$.

$$\mathcal{P}_{\bowtie}(\mathcal{M}, s, \mathbb{F}) = \underset{\mathcal{T} \in Exec(\mathcal{M})}{\bowtie} \mathcal{P}(\mathcal{T}, s, \mathbb{F}).$$

3.3 Case study: vehicle navigation

In this section, we present a case study involving vehicle moving on a specific navigation scenario given in Figure 3.1. Here, we consider dubbin's dynamics for the vehicle. The continuous dynamics of the vehicle is given below:

$$\begin{aligned} \dot{x}(t) &= v\ cos(\theta(t)) \\ \dot{y}(t) &= v\ sin(\theta(t)) \\ \dot{\theta}(t) &= u(t) \end{aligned} \tag{3.1}$$

where $(x(t), y(t))$ denotes the position at time t; $\theta(t)$ denotes the rotational angle; v denotes the linear velocity; $u(t)$ denotes the control input at time t which is angular velocity ω. Note that Equation 3.1 is a non-linear system, and angular velocity is constant in each of the control modes (horizontal, left turn, and vertical). However, Equation 3.1 can be expressed into a linear system as given below:

$$\dot{z} = A(c_1, c_2, \omega)z,\ \ c_1, c_2 \in \{0, 1\},\ \text{where } c_1 + c_2 \geq 1 \tag{3.2}$$

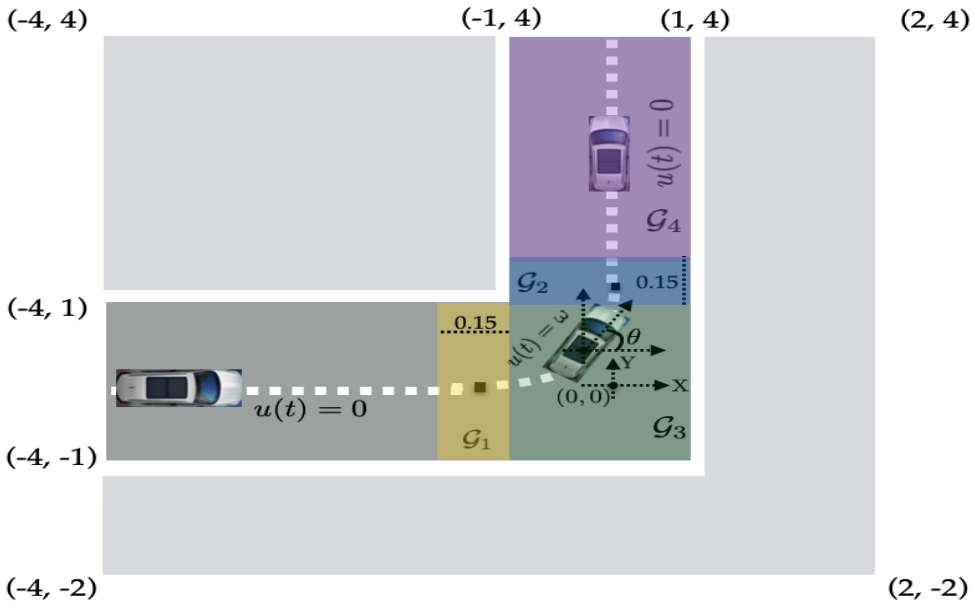

Figure 3.1: Vehicle navigation

where $z = (x, y, v_x, v_y)$ denotes the state variable consisting of horizontal and vertical position (x, y), and horizontal and vertical velocity (v_x, v_y), and

$$A(c_1, c_2, \omega) = \begin{bmatrix} 0 & 0 & c_1 & 0 \\ 0 & 0 & 0 & c_2 \\ 0 & 0 & 0 & -\omega \\ 0 & 0 & \omega & 0 \end{bmatrix}.$$

The values of c_1, c_2, and ω vary depending on the control modes. For example, for horizontal motion, $c_1 = 1$, $c_2 = 0$, and $\omega = 0$; for vertical motion, $c_1 = 0$, $c_2 = 1$, and $\omega = 0$; for left turn with $\pi/4$ angular velocity, $c_1 = 1$, $c_2 = 0$, and $\omega = \pi/4$.

We assume that there are errors in sensors. The vehicle may not detect guard regions G_1 or G_2 when it is actually there due to sensor errors. Hence, the vehicle switches probabilistically from one mode to another mode as described below. In Figure 3.1, initially, the vehicle is moving horizontally. When it reaches the yellow region ($\mathcal{G}_1$), there are two possibilities, namely, (a) it could take left turn; (b) it could still move horizontally. Since, the chances of taking a left turn are high, we consider 9/10 as its probability of taking left turn, and

1/10 as its probability of moving horizontally. Next, if the vehicle crosses the yellow region and misses the left turn, then the chances of moving it in a horizontal direction are high. Hence, we consider 9/10 as its probability of moving horizontally, and 1/10 as its probability of taking left turn. Further, if the vehicle takes a left turn, and reaches the blue region ($\mathcal{G}_2$), the vehicle could either continue turning or switch to move vertically. Since the chances of switching to move vertically are high, we consider 9/10 as its probability of moving vertically, and 1/10 as the probability of continuing turning. Next, if the vehicle crosses the blue region and misses switching to move vertically, then the chances of continuing turning are high. Hence, we consider 9/10 as its probability of continuing turning, and 1/10 as the probability of switching to move vertically.

Our objective is to check whether the maximum probability of the vehicle reaching beyond the lane is less than or equal to a desired probability threshold $p > 0$.

3.4 Probabilistic hybrid systems and its subclasses

In this section, we introduce the class of probabilistic hybrid systems and provide their formal syntax and semantics. In addition, we introduce a certain subclass of probabilistic hybrid systems that we study in this thesis.

Syntax

Probabilistic hybrid systems capture the discrete, continuous and probabilistic behaviors. The continuous behaviors are captured using differential equations, while the discrete and probabilistic behaviors are captured using probabilistic edges, guards, and resets. Next, we introduce the syntax of probabilistic hybrid systems.

Definition 6 (Probabilistic Hybrid Systems)**.** *A probabilistic hybrid system (*PHS*) is a tuple* $\mathcal{H} == (\mathcal{Q}, \mathcal{X}, \mathcal{Q}_0, \mathcal{X}_0, \mathit{Inv}, \mathit{Flow}, \mathit{Edges}, \mathit{Guard}, \mathit{Reset})$*, where:*

- $\mathcal{Q}$ *is a set of locations;*

- $\mathcal{X} \subseteq \mathbb{R}^n$ *is a continuous state space;*
- $\mathcal{Q}_0 \subseteq \mathcal{Q}$ *is a set of initial locations;*
- $\mathcal{X}_0 \subseteq \mathcal{X}$ *is a countable set of initial continuous states;*
- $Inv : \mathcal{Q} \to 2^{\mathcal{X}}$ *is an invariant function;*
- $Flow : \mathcal{Q} \times \mathcal{X} \to 2^{\mathcal{X}}$ *is a flow function which assigns a vector to each state* $(q, x) \in \mathcal{Q} \times \mathcal{X}$*;*
- $Edges \subseteq \mathcal{Q} \times Dist(\mathcal{Q})$ *is a finite set of probabilistic edges;*
- $Guard : Edges \to 2^{\mathcal{X}}$ *is a guard function;*
- $Reset : Edges \times \mathcal{Q} \times \mathcal{X} \to \mathcal{X}$ *is a reset function.*

Notation: Given a PHS $\mathcal{H}$, we will represent its elements using $\mathcal{H}$ as a subscript. For instance, the invariant and flow functions of $\mathcal{H}$, are represented as $Inv_{\mathcal{H}}$ and $Flow_{\mathcal{H}}$, respectively.

Next, we describe two different semantics for a given polyhedral PHS, namely, in terms of timed Markov decision processes and Markov decision processes.

Semantics

We describe the semantics of PHS in terms of an infinite state TMDP. A state of the PHS is a pair (q, x), where $q \in \mathcal{Q}$ is a discrete location, and $x \in \mathcal{X}$ is a continuous state. A timed probabilistic edge associated with a state (q, x) consists of a time T elapse in which the state x evolves according to the dynamics to some state x' and then x' transitions instantaneously to other states governed by guards and resets. The timed probabilistic edges correspond to a continuous evolution using the flow function which remains within the corresponding invariant, followed by an instantaneous probabilistic transition governed by the guards and the resets. More precisely, a continuous transition from a state (q, x) to a state (q, x') is

possible in time T if there exists a function $\phi : [0, T] \to \mathcal{X}$ such that $\phi(0) = x$, $\phi(T) = x'$, $\frac{d\phi(t)}{dt} \in \mathit{Flow}((q, \phi(t)))$ and $\phi(t) \in \mathit{Inv}(q)$ for $0 \leq t \leq T$. A probabilistic transition from a state (q, x) to a state (q', x') with probability p is possible if there exists an edge $(q, \rho) \in \mathit{Edges}$ such that $x \in \mathit{Guard}((q, \rho))$, $x' = \mathit{Reset}(((q, \rho), q', x))$ and $\rho(q') = p$.

Definition 7. *Given* PHS $\mathcal{H} = (\mathcal{Q}, \mathcal{X}, \mathcal{Q}_0, \mathcal{X}_0, \mathit{Inv}, \mathit{Flow}, \mathit{Edges}, \mathit{Guard}, \mathit{Reset})$*, the semantics of* $\mathcal{H}$ *is defined as* TMDP $[\![\mathcal{H}]\!] = (\mathcal{S}, \longrightarrow_{[\![\mathcal{H}]\!]})$*, where:*

- $\mathcal{S} = \mathcal{Q} \times \mathcal{X}$*;*
- $((q, x), T, \pi) \in \longrightarrow_{[\![\mathcal{H}]\!]}$ *if* $\exists\ (q, \rho) \in \mathit{Edges}$ *and* $\exists\ \phi : [0, T] \to \mathcal{X}$ *such that*
  1. $\phi(0) = x$ *and* $\phi(T) \in \mathit{Guard}((q, \rho))$*;*
  2. $\phi(t) \in \mathit{Inv}(q)$ *and* $\frac{d\phi(t)}{dt} \in \mathit{Flow}((q, \phi(t)))$ *for all* $t \in [0, T]$*;*
  3. *For each* $(q', x') \in \mathcal{Q} \times \mathcal{X}$, $\pi((q', x')) = \rho(q')$ *if* $x' = \mathit{Reset}(((q, \rho), q', \phi(T)))$ *else* $\pi((q', x')) = 0$.

The TMDP has an infinite number of states because each state is a pair of discrete location and a continuous value, where the number of values of the continuous variables is infinite. Note that although we have an infinite number of states, for every timed probabilistic edge $(q, x) \xrightarrow{T}_{[\![\mathcal{H}]\!]} \pi$, π has finite support.

Remark 1. *The semantics of* PHS *in terms of Markov decision process is the same except the probabilistic edges, where the edges will be in the form of* $((q, x), \pi)$ *instead of* $((q, x), T, \pi)$ *in Definition 29, that is, time* T *will also be the existential variable.*

Subclasses of probabilistic hybrid systems

We define subclasses of probabilistic hybrid systems. Next, we define the syntax of linear probabilistic hybrid systems, where the flow function is restricted to linear, and invariant and guard functions are restricted to polyhedral sets.

Definition 8 (Linear PHS)**.** *A linear probabilistic hybrid system is a* PHS $\mathcal{H} = (\mathcal{Q}, \mathcal{X}, Inv, Flow, Edges, Guard, Reset)$ *where:*

- $Flow(q, x) = \{A_q x + B_q\}$ *for some square matrix* A_q *and constant vector* B_q*;*
- $Inv : Q \to Poly(n)$*;*
- $Guard : Edges \to Poly(n)$*;*
- $Reset : Edges \times \mathcal{Q} \to Poly(2n)$*;*

The rest of the tuple entities are the same as defined in Definition 6.

Example 1. *Consider the vehicle navigation case-study given in Figure 3.1, which can be modeled as a linear probabilistic hybrid system shown in Figure 3.2.*

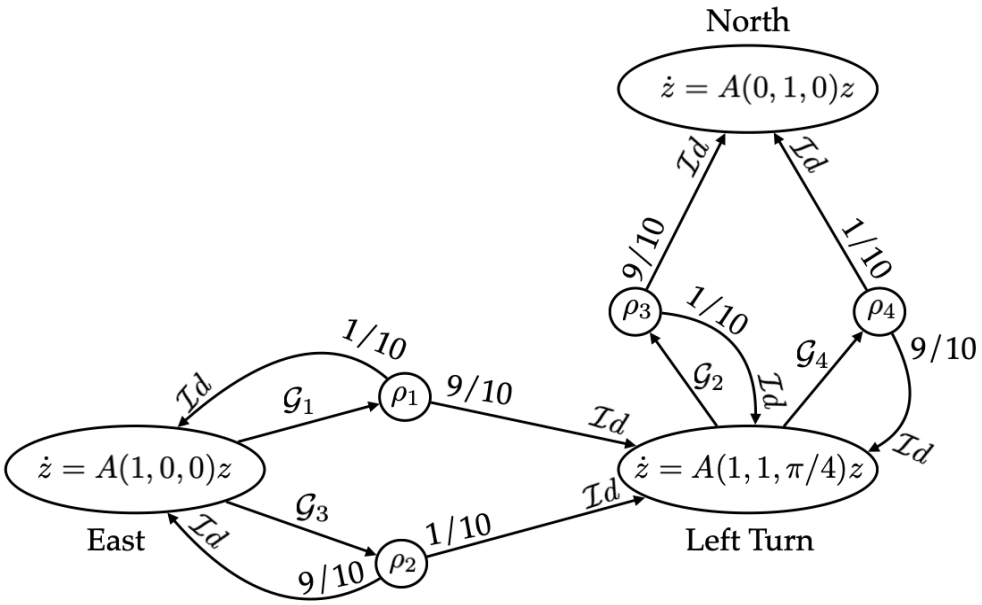

Figure 3.2: Linear PHS

It has four continuous state variables, namely position and velocity in horizontal and vertical direction represented by $z = (x, y, v_x, v_y)$. Formally, it can be modeled as a linear PHS $\mathcal{H} = (\mathcal{Q}, \mathcal{X}, Inv, Flow, Edges, Guard, Reset)$, where

- $\mathcal{Q} = \{\text{East}, \text{Left Turn}, \text{North}\}$;

- $\mathcal{X} = \{(x, y, v_x, v_y) \mid -4 \leq x \leq 2,\ -2 \leq y \leq 4,\ 0 \leq v_x, v_y \leq 2\}$;
- $\mathit{Inv}(q) = \mathcal{X}$ for all $q \in \mathcal{Q}$;
- The flow function which is given by $\mathit{Flow}(\text{East}, z) = A(1, 0, 0)z$, $\mathit{Flow}(\text{Left Turn}, z) = A(1, 1, \pi/4)z$, and $\mathit{Flow}(\text{North}, z) = A(0, 1, 0)z$, where $A(c_1, c_2, \omega)$ is defined in Subsection 3.3 of Chapter 3;
- $\mathit{Edges} = \{(\text{East}, \rho_1), (\text{East}, \rho_2), (\text{Left Turn}, \rho_3), (\text{Left Turn}, \rho_3)\}$, where:
  - $\rho_1(\text{East}) = \frac{1}{10}$, $\rho_1(\text{Left Turn}) = \frac{9}{10}$, $\rho_2(\text{East}) = \frac{9}{10}$, $\rho_2(\text{Left Turn}) = \frac{1}{10}$;
  - $\rho_3(\text{Left Turn}) = \frac{1}{10}, \rho_3(\text{North}) = \frac{9}{10}$, $\rho_4(\text{Left Turn}) = \frac{9}{10}, \rho_4(\text{North}) = \frac{1}{10}$.
- $\mathit{Guard}(\text{East}, \rho_1) = \{(x, y, v_x, v_y) \mid -1.15 \leq x \leq -1,\ -1 \leq y \leq 1,\ 0 \leq v_x, v_y \leq 2\}$,
  $\mathit{Guard}(\text{East}, \rho_2) = \{(x, y, v_x, v_y) \mid -1 \leq x, y \leq 1,\ 0 \leq v_x, v_y \leq 2\}$,
  $\mathit{Guard}(\text{Left Turn}, \rho_3) = \{(x, y, v_x, v_y) \mid -1 \leq x \leq 1,\ 1 \leq y \leq 1.15,\ 0 \leq v_x, v_y \leq 2\}$,
  $\mathit{Guard}(\text{Left Turn}, \rho_4) = \{(x, y, v_x, v_y) \mid -1 \leq x \leq 1,\ 1 \leq y \leq 4,\ 0 \leq v_x, v_y \leq 2\}$.
- All resets are identity functions, that is, $\mathit{Reset}(\rho, q) = \mathit{Id}$ for $\rho \in \{\rho_1, \rho_2\}$, $q \in \{\text{East}, \text{Left Turn}\}$, and $\mathit{Reset}(\rho, q) = \mathit{Id}$ for $\rho \in \{\rho_3, \rho_4\}$, $q \in \{\text{North}, \text{Left Turn}\}$.

Furthermore, we introduce the syntax of polyhedral probabilistic hybrid systems, where invariant, flow, guard, and reset functions are restricted to polyhedron.

Definition 9 ((Polyhedral PHS)**.** *A polyhedral probabilistic hybrid system is a linear* PHS $\mathcal{H} = (\mathcal{Q}, \mathcal{X}, \mathit{Inv}, \mathit{Flow}, \mathit{Edges}, \mathit{Guard}, \mathit{Reset})$ *where* $\mathit{Flow} : \mathcal{Q} \times \mathcal{X} \to \mathit{Poly}(n)$ *is a flow function which assigns a polyhedral set to each state* $(q, x) \in \mathcal{Q} \times \mathcal{X}$*. The rest of the tuple entities are the same as defined in Definition 8.*

Example 2. *Consider the vehicle navigation case-study given in Figure 3.1, and its linear* PHS *shown in Figure 3.2. We can reduce linear* PHS *into a polyhedral* PHS *by over-approximating the linear dynamics into a polyhedral dynamics. It can be obtained by collecting all the vector fields from the linear dynamics over the invariant set at respective location.*

The polyhedral PHS *corresponding to the linear* PHS *is shown in Figure 3.3 which can be*

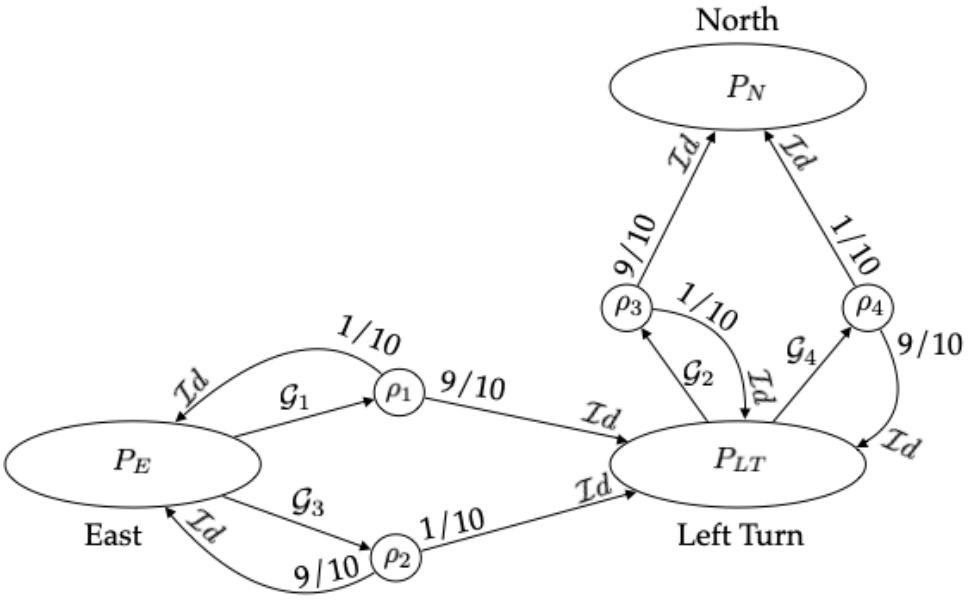

Figure 3.3: Polyhedral PHS

formally specified as $\mathcal{H} = (\mathcal{Q}, \mathcal{X}, Inv, Flow, Edges, Guard, Reset)$*, where all the components are the same as given in Example 1 except the flow function. Here, the flow function is given by* $Flow(East, z) = P_E$*,* $Flow(Left\ Turn, z) = P_{LT}$*, and* $Flow(North, z) = P_N$*, where:*

- $P_E = \{(\dot{x}, \dot{y}, \dot{v}_x, \dot{v}_y) \mid 0 \leq \dot{x} \leq 2,\ \dot{y} = \dot{v}_x = \dot{v}_y = 0\}$;
- $P_{LT} = \{(\dot{x}, \dot{y}, \dot{v}_x, \dot{v}_y) \mid 0 \leq \dot{x}, \dot{y} \leq 2,\ \dot{v}_x + \frac{157}{200} v_y = 0,\ \dot{v}_y - \frac{157}{200} v_x = 0\}$;
- $P_N = \{(\dot{x}, \dot{y}, \dot{v}_x, \dot{v}_y) \mid 0 \leq \dot{y} \leq 2,\ \dot{x} = \dot{v}_x = \dot{v}_y = 0\}$.

Chapter 4

Probabilistic reachability analysis

In this chapter, we consider probabilistic hybrid systems[139] (PHS) to capture discrete, continuous, and probabilistic behaviors. A PHS consists of a finite number of modes and a finite number of continuous variables. Each mode is associated with continuous dynamics that specify the evolution of the continuous variables using differential equations or inclusions. This is similar to that of a hybrid automaton. In addition, a PHS consists of transitions that are non-deterministic as well as probabilistic. Hence, our underlying model is a Markov Decision Process (MDP), which is then extended with continuous dynamics in each mode. We are interested in analyzing the probability of reaching a target set of states $\mathbb{F}$ within a given amount of time T and a bound on the number of discrete/probabilistic transitions k. Note that since our model encompasses both non-deterministic as well as probabilistic transitions, depending on how a scheduler resolves the non-determinism, there are different probabilities associated with reaching the target set $\mathbb{F}$. Hence, we consider the problem of computing the maximum and minimum probability of reaching the target set among all schedulers. The bounded verification problem, in general, captures the behavior of an under-approximation of the system where the number of transitions and the time of execution is constrained. However, it provides conservative bounds on the actual probabilities of an actual system. Also, in many cases, there is a practical upper bound on the total time, and a lower bound

on the dwell time (how fast the system can switch), which justifies the problem of bounded verification as a complete method for verification of the full system.

Here, we restrict ourselves to polyhedral dynamics that are an important and widely prevalent class of dynamics for modeling physical processes. The main challenge towards the probabilistic analysis of reachability is the computation of maximum and minimum probability of reachability which involves solving a complex optimization problem. To address the problem, we consider an encoding into a problem that involves optimization over SMT formulas with linear constraints, that can be solved efficiently using recent tools such as Z3opt[140] and SYMBA[141]. More precisely, our broad approach consists of computing exact bounds on the probability of reachability in the system. Next, we encode the computation trees of polyhedral PHS using SMT formulae. We use Z3opt and SYMBA solvers to find the maximum and minimum probabilities of reachability by optimizing the probability over constraints encoded in SMT. We have implemented our approach in a Python toolbox, and conducted experimental evaluation on a vehicle navigation case study for the probability of reaching beyond the lane/road and maintaining a safe distance among the vehicles. The experimental results demonstrate the feasibility of the proposed approach. The main challenge towards scaling the approach will be the exponential growth of the variables and the size of the encoding with the number of discrete transitions. In many cases, certain timing constraints can enforce a practical bound on the number of transitions. However, this bound could be large and the SMT solver might fail to return. Our future work will focus on efficient encodings and heuristics for reducing/pruning the computation tree.

Remark 2. *We use the timed Markov Decision Process for the semantics of polyhedral probabilistic hybrid systems in the rest of this chapter.*

4.1 Problem definition

Problem 1. *[Probabilistic bounded reachability problem] Given a polyhedral* PHS $\mathcal{H} =$

$(\mathcal{Q}, \mathcal{X}, \mathit{Inv}, \mathit{Flow}, \mathit{Edges}, \mathit{Guard}, \mathit{Reset})$, *a set of initial states* $\mathbb{I}$, *a set of final states* $\mathbb{F}$, *find the maximum/minimum probability of reaching* $\mathbb{F}$ *from* $\mathbb{I}$ *within a given time horizon* $T > 0$ *with at most* $k \geq 0$ *discrete transitions, that is,* $\underset{s \in \mathbb{I}}{\bowtie} \left(\mathcal{P}^{\bowtie}_{(T, \leq k, \mathbb{F})}([\![\mathcal{H}]\!], s) \right)$, *where* $\bowtie \in \{inf, sup\}$.

Next, we illustrate the problem through the case-study shown in Figure 3.1 and its corresponding polyhedral PHS shown in Figure 3.3. Here, we want to compute the maximum probability of the vehicle starting from $\mathbb{I} = \{\text{East}\} \times \{(x, y, v_x, v_y) \mid -4 \leq x \leq -3.5,\ -0.2 \leq y \leq 0.2,\ 1 \leq v_x, v_y \leq 2\}$ reaching beyond the lane within $T = 5$ units of time with at most $k = 2$ discrete transitions. Here, $\mathbb{F}$ is the space beyond the lane, that is, $\mathbb{F} = \{\text{East}, \text{Left Turn}, \text{North}\} \times (\{(x, y, v_x, v_y) \mid -4 \leq x \leq -1,\ 1 \leq y \leq 4,\ 0 \leq v_x, v_y \leq 2\} \cup \{(x, y, v_x, v_y) \mid -4 \leq x \leq 2,\ -2 \leq y \leq -1,\ 0 \leq v_x, v_y \leq 2\} \cup \{(x, y, v_x, v_y) \mid -4 \leq x \leq 1,\ -1 \leq y \leq 4,\ 0 \leq v_x, v_y \leq 2\})$.

4.2 Computing probability of reachability

Our main problem is to compute the probability of reaching a target set $\mathbb{F}$ within k discrete transitions and time T in a polyhedral probabilistic hybrid system. Our broad approach consists of reducing the problem of computing the minimum and maximum probability of reachability in a polyhedral PHS into two optimization problems with constraints expressed using a satisfiability modulo theory formula, which encodes the computation trees of the polyhedral PHS. From the inductive definition of the probability of reachability, we are required to unroll the polyhedral PHS for k steps to construct a tree with k levels (height k). The minimum/maximum probability of reachability at each node is expressed iteratively as a solution of an optimization problem over constraints which themselves recursively contain other optimization problems. Note that all the entities of a polyhedral PHS such as invariants, dynamics, guards, and resets, can be expressed as linear constraints. However, we have non-deterministic probabilistic edges, hence, each recursive call to the optimization problems is in the form of a linear optimization problem subject to constraints consisting

of conjunctions and disjunctions of linear constraints. Though theoretically this problem can be solved by encoding it in first-order logic, we do not know of any tool that efficiently solves optimization problems of this kind. Our broad idea is to lift the recursive optimization problem at each node of the computation tree up to the root level, that is, develop an encoding that solves a single optimization problem at the root of the computation tree. Linear optimization problems over conjunctions and disjunctions of linear constraints can be efficiently solved using the tools Z3opt SMT- solver[140] and SYMBA[141], which add optimization capabilities to SMT solving. Next, we explain the construction of the encoding of the computation, which will be the important part of the optimization problem we formulate.

Encoding

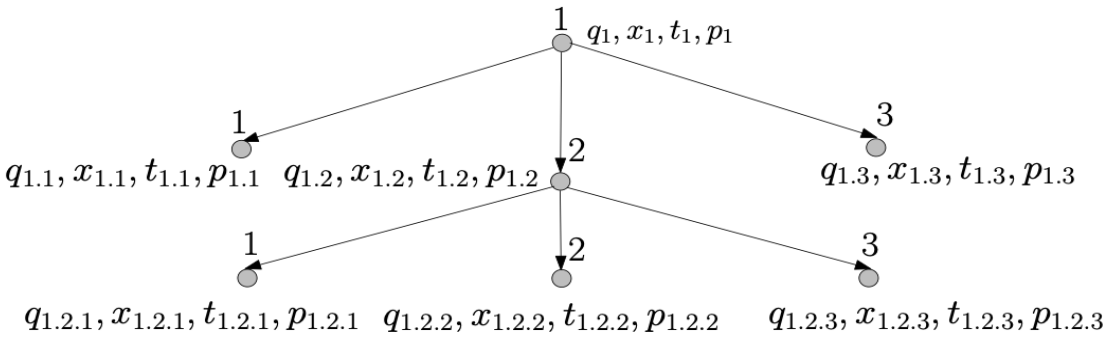

Figure 4.1: Illustration of computation tree for $k = 2$

We fix the following ordering on the locations of $\mathcal{H}$, namely, $q_1, \ldots, q_n$ and we assume that q_1 is the initial state we are interested in. Let us fix a polyhedral PHS $\mathcal{H}$. Let n be the number of locations in $\mathcal{H}$. A computation tree of level k is essentially an n-ary tree of height k as shown in Figure 4.1. We define the names of the node in the tree as follows. The name of the root node is 1. Inductively, the names of the i-th child of a node named α are $\alpha.i$, where i ranges over 1 to n. Hence, the node 1.2.1 refers to the first child of the second child of the root node. We annotate the tree with states reached along an unrolling of k

steps of $[\![\mathcal{H}]\!]$. We annotate the root with an initial state of $[\![\mathcal{H}]\!]$. The children of a node α are annotated by the states reached by taking a timed probabilistic edge of $[\![\mathcal{H}]\!]$. Note that in each timed probabilistic edge $((q,x),t,\pi)$ of $[\![\mathcal{H}]\!]$, π has finite support; more importantly, for each q', $\pi(q',x') \neq 0$ for at most one x'. Hence, we will fix an ordering of the locations $\mathcal{Q}_{\mathcal{H}}$ and assume that the i-th child is annotated by a state whose location is the i-location in the ordering, and the i-th continuous state is given by the reset function corresponding to the edge taken and the i-th location (target). (Recall $\mathit{Reset} : \mathit{Edges} \times \mathcal{Q} \times \mathcal{X} \to \mathcal{X}$, where $\mathcal{Q}$ captures the target location.) Next, we describe the SMT formula that encodes the computation tree.

First, we fix some notations that will be used in the rest of the section. Let $\mathcal{I}_q(x)$ be a predicate corresponding to $\mathit{Inv}(q)$, that is, $\mathcal{I}_q(x)$ evaluates to true if and only if $x \in \mathit{Inv}(q)$. Similarly, $\mathcal{F}low_q(x,r), \mathcal{G}_{(q,\rho)}(x)$, and $\mathcal{R}_{(q,\rho,q')}(x,x')$ be the predicates that capture $r \in \mathit{Flow}(q,x)$, $x \in \mathit{Guard}(q,\rho)$, and $x' = \mathit{Reset}((q,\rho),q',x')$, respectively. Also, let $\mathbb{F}_q(x)$ be a predicate for $(q,x) \in \mathbb{F}$. First, we explain how to capture the discrete, continuous and timed probabilistic edges of $[\![\mathcal{H}]\!]$ using first-order formulas with only existential quantification, conjunctions and disjunctions, that is, as a satisfiability modulo theory (SMT) formula.

Encoding of transition relation

We provide the details of the construction of transition relations in terms of continuous and discrete transitions given below.

Continuous transitions. We construct a formula $\mathit{Cont}_q(x,t,x')$ that encodes whether there exists an execution that starts from the state (q,x) and reaches the state (q,x') at time t demonstrate in Figure 4.2.

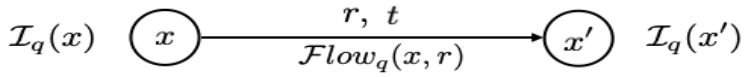

Figure 4.2: Continuous transition

$$\mathit{Cont}_q(x,t,x') = \exists\, r, \mathcal{F}low_q(x,r) \wedge x' = x + rt \wedge \mathcal{I}_q(x) \wedge \mathcal{I}_q(x')$$

Note that if $\mathcal{H}$ is a polyhedral PHS, then all the constraints in $Cont_q(x, t, x')$ will be linear expressions except for the multiplication of *rate* and *time*, that is, rt. We can eliminate the nonlinear expression rt by converting it into an equivalent linear expression. From the definition of polyhedral PHS, the predicates $\mathcal{F}low_q(x, r)$ and $\mathcal{I}_q(x)$ can be expressed as the conjunctions of linear constraints of the form $a \cdot r \leq b$, where a is a constant n dimensional vector and b is a constant number. We introduce a new variable (vector) $y = rt$ (note r is a vector and t is a scalar), and then replace all linear constraints $a \cdot r \leq b$ in $\mathcal{F}low_q(x, r)$ by the linear constraints $a \cdot y \leq b.t$, and the constraint $x' = x + rt$ by $x' = x + y$. The two constraints are equivalent, since we will have assumed that the domain of t is the non-negative real numbers.

Discrete probabilistic transitions. We construct a formula $Disc_q(x, \mathbf{x}, \mathbf{p})$ that encodes the distribution over the states reached by taking some probabilistic edge ρ from the state (q, x) demonstrated in Figure 4.3. If no such edge exists, we allow a dummy transition with 0 as the probabilities. The (q, ρ)'s range over *Edges*.

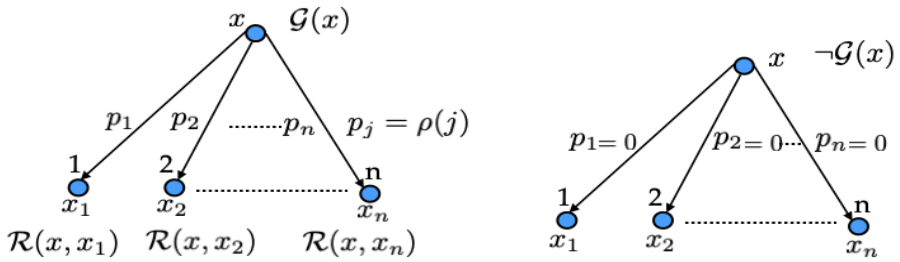

Figure 4.3: Discrete transition

$$Disc_q(x, \mathbf{x}, \mathbf{p}) = [(\bigwedge_{(q,\rho)} \neg\mathcal{G}_{(q,\rho)}(x) \wedge \bigwedge_{j=1}^{n} p_j = 0) \vee$$

$$\bigvee_{(q,\rho)} [\mathcal{G}_{(q,\rho)}(x) \wedge \bigwedge_{j=1}^{n} (\mathcal{R}_{(q,\rho,q_j)}(x, x_j) \wedge p_j = \rho(q_j))]$$

Transition relation. The formula $Trans_q(x, t, x', \mathbf{x}, \mathbf{p})$ encodes a timed probabilistic edge, that is, a combination of a continuous transition followed by a discrete probabilistic

transition.

$$Trans_q(x, t, \mathbf{x}, \mathbf{p}) = [\exists\ x', Cont_q(x, t, x') \wedge Disc_q(x', \mathbf{x}, \mathbf{p})]$$

We will use *Trans* as a primitive to encode computation tree of $\mathcal{H}$. We need variables to capture different entities at each of the nodes of the computation tree. Hence, we introduce some notation. Let $I_k = \{\alpha \mid \alpha = 1.i_1.i_2.\ldots.i_m,\ 0 \leq m \leq k,\ i_j \in [n] \text{ for } 1 \leq j \leq m\}$ denote the set of all paths starting from root node in the computation tree. Note that the location at node α is $L(\alpha)$. Given a variable z, $\mathcal{V}_z^k$ will denote a set of variables for each node in the tree corresponding to z, that is, $\mathcal{V}_z^k = \{z_\alpha \mid \alpha \in I_k\}$. The free variables of our formula will include $\mathcal{V}_x^k$, $\mathcal{V}_t^k$, and $\mathcal{V}_p^k$, where x_α denotes the continuous state in the computation tree at α; similarly, t_α denotes the time spent at location $q_{L(\alpha)}$, and $p_{\alpha.i}$ denotes the probability of the transition from $(q_{L(\alpha)}, x_\alpha)$ to $(q_i, x_{\alpha.i})$ in the computation tree. We will assume that $p_1 = Init(q_1, x_1)$.

Encoding of execution tree

We need to ensure that the total time spent on the executions is bounded by some value T. Hence, we track the total time spent along any path from the root in the computation tree using the variables in $\mathcal{V}_T$. Our goal is to optimize the probability of reaching a final state. Hence, we use the variables in $\mathcal{V}_P$ to capture the probabilities along the paths of the computation tree, that is, P_α captures the probability associated with execution corresponding to α. Finally, we need to add the probabilities of all those paths that end in a final state and don't reach a final state before that. Hence, we have boolean variable sets $\mathcal{V}_F$ and $\mathcal{V}_B$, where F_α is a boolean variable that is true when α corresponds to an execution that ends in a final state without having visited a final state before, and B_α is a boolean variable that is true if a final state has been visited along α.

Next, we construct a formula $Exec^{\leq k,T,\mathbb{F}}$ that captures the sum of the probabilities of the executions of computation tree with level k that reaches the target set $\mathbb{F}$ (for the first time) within time T.

Validation of tree edges. The formula *Tree* validates all the edges in the computation tree for given values of $\mathcal{V}_x, \mathcal{V}_t, \mathcal{V}_p$. It also checks that the initial state and probability (at the root) are valid. Here $\mathbb{I}_q(x, p)$ is a predicate such that $Init(q, x) = p$. We use $z_{\bar{\alpha}}$ to denote $(z_{\alpha.1}, \ldots, z_{\alpha.n})$.

$$Tree(\mathcal{V}_x, \mathcal{V}_t, \mathcal{V}_p) = \mathbb{I}_{q_1}(x_1, p_1) \wedge \bigwedge_{\alpha \in I_{k-1}} [Trans_{q_{L(\alpha)}}(x_\alpha, t_\alpha, x_\alpha, , p_\alpha)]$$

Timing constraints. The formula $Time_T$ checks the relation between global times in $\mathcal{V}_T$ and local times in $\mathcal{V}_t$, and ensures that the total times are less than T along any executions. The local and global time variables for a probabilistic edge are demonstrated in Figure 4.4.

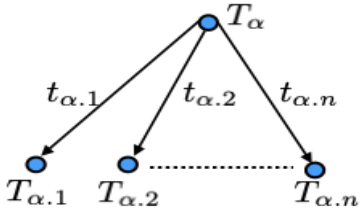

Figure 4.4: Time variables

$$Time_T(\mathcal{V}_T, \mathcal{V}_t) = [(T_1 = 0) \wedge \bigwedge_{\alpha \in I_{k-1}} \big(\bigwedge_{j=1}^{n} T_{\alpha.j} = T_\alpha + t_\alpha \big) \wedge \bigwedge_{\alpha \in I_k} (0 \leq T_\alpha \leq T)]$$

Probability checking. The formula *Prob* checks the relation between the total probability along a path captured using $\mathcal{V}_P$ and the local probabilities along the individual transitions captured using $\mathcal{V}_p$. The local and global probability variables for a probabilistic edge are demonstrated in Figure 4.5.

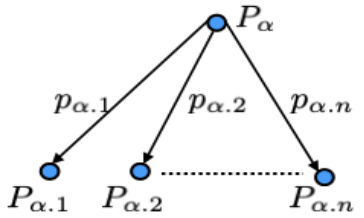

Figure 4.5: Probability variables

$$Prob(\mathcal{V}_P, \mathcal{V}_p) = [P_1 = p_1 \wedge \bigwedge_{\alpha \in I_{k-1}} \big(\bigwedge_{j=1}^{n} P_{\alpha.j} = P_\alpha * p_{\alpha.j} \big)]$$

Checking if final state has been reached. The formula *Before* captures using $\mathcal{V}_B$ whether a state from the target set has been seen at any previous node in the path from root node to that node in the execution of computation tree using the values in $\mathcal{V}_x$.

$$Before(\mathcal{V}_B, \mathcal{V}_x) = [(B_1 = \mathbb{F}_{q_1}(x_1)) \wedge \bigwedge_{\alpha \in I_{k-1}} \big(\bigwedge_{j=1}^{n} B_{\alpha.j} = (B_\alpha \vee \mathbb{F}_{q_{L(\alpha.j)}}(x_{\alpha.j})) \big)]$$

Final target state checking. The formula *Final* captures using $\mathcal{V}_F$ whether a target set is visited for the first time at a node in a path from the root node to that node in the execution of the computation tree using the values in $\mathcal{V}_B$.

$$Final(\mathcal{V}_F, \mathcal{V}_B) = [(F_1 = B_1) \wedge \bigwedge_{\alpha \in I_{k-1}} \big(\bigwedge_{j=1}^{n} \big(F_{\alpha.j} = (\neg B_\alpha \wedge B_{\alpha.j}) \big) \big)]$$

Sum of probabilities. The formula *Sum* checks whether given $\mathcal{V}_P, \mathcal{V}_F$ and p_s, if p_s is the sum of all the probabilities associated with nodes in the computation tree which correspond to executions that reach the final state only in the last state.

$$Sum(\mathcal{V}_P, \mathcal{V}_F, p_s) = [p_s = (\sum_{\alpha \in I_k} F_\alpha P_\alpha)]$$

Finally, we can put all the formulas defined above to construct the formula $Exec^{\leq k,T,\mathbb{F}}(\mathcal{V}_x, \mathcal{V}_t, \mathcal{V}_p, \mathcal{V}_T, \mathcal{V}_P, \mathcal{V}_B, \mathcal{V}_F, p_s)$ that captures the values of the variable $\mathcal{V}_x$, $\mathcal{V}_t$, $\mathcal{V}_p$, $\mathcal{V}_T$, $\mathcal{V}_P$, $\mathcal{V}_B$ and $\mathcal{V}_F$ for a computation tree, along with the total probability of reaching the target set $\mathbb{F}$ for this computation tree in p_s.

$$\begin{aligned} Exec^{\leq k,T,\mathbb{F}}(\mathcal{V}_x, \mathcal{V}_t, \mathcal{V}_p, \mathcal{V}_T, \mathcal{V}_P, \mathcal{V}_B, \mathcal{V}_F, p_s) = {} & Tree(\mathcal{V}_x, \mathcal{V}_t, \mathcal{V}_p) \wedge Time(\mathcal{V}_T, \mathcal{V}_t) \wedge \\ & Prob(\mathcal{V}_P, \mathcal{V}_p) \wedge Before(\mathcal{V}_B, \mathcal{V}_x) \wedge Final(\mathcal{V}_F, \mathcal{V}_B) \wedge Sum(\mathcal{V}_P, \mathcal{V}_F, p_s) \end{aligned}$$

Note that the formula $Exec^{\leq k,T,\mathbb{F}}$ does not check for the minimum/maximum probabilistic reachability. Therefore, we need to solve an optimization problem with p_s as the objective function over $Exec^{\leq k,T,\mathbb{F}}$ as the constraints. Note that the formula $Exec^{\leq k,T,\mathbb{F}}$ has conjunctions and disjunctions of linear expressions which can be solved by existing SMT optimization tools Z3opt SMT-solver[140] and SYMBA[141].

4.3 Experimental analysis

In this section, we present an implementation details of the SMT based approach for computing the bound on the minimum/maximum probability of reachability for a polyhedral probabilistic hybrid system. Our implementation has probabilistic reachability analysis module that takes as input a polyhedral PHS, number of discrete transitions k, total time T, an initial set $\mathbb{I}$, and a target set $\mathbb{F}$, and outputs an SMT formula that captures all the computation trees corresponding to the polyhedral PHS. The latter module calls either Z3opt to solve the optimization problems over the SMT formula that returns the minimum/maximum probability of reachability. Next, we explain a systematic way of constructing polyhedral PHS of different number of modes and dimensions associated with vehicle navigation. All the computation times are measured in seconds. The evaluation of the experiment has been conducted on Ubuntu 18.04 OS, 2.5 GHz Quad-Core Processor, 8GB RAM.

Polyhedral PHS for vehicle navigation

We consider a 2D field, where different number of horizontal and vertical roads intersect with each other. For example, two horizontal and vertical roads on a 2D field is shown in Figure 4.6. Here, dark grey regions are part of the road, and light grey regions are the blocks beyond the road. To construct a polyhedral PHS, we randomly generate a simple route, where the same cell does not appear again, as shown by black solid line in Figure 4.6, and consider uncertainties at each turn involved in the route. For example, when the vehicle is

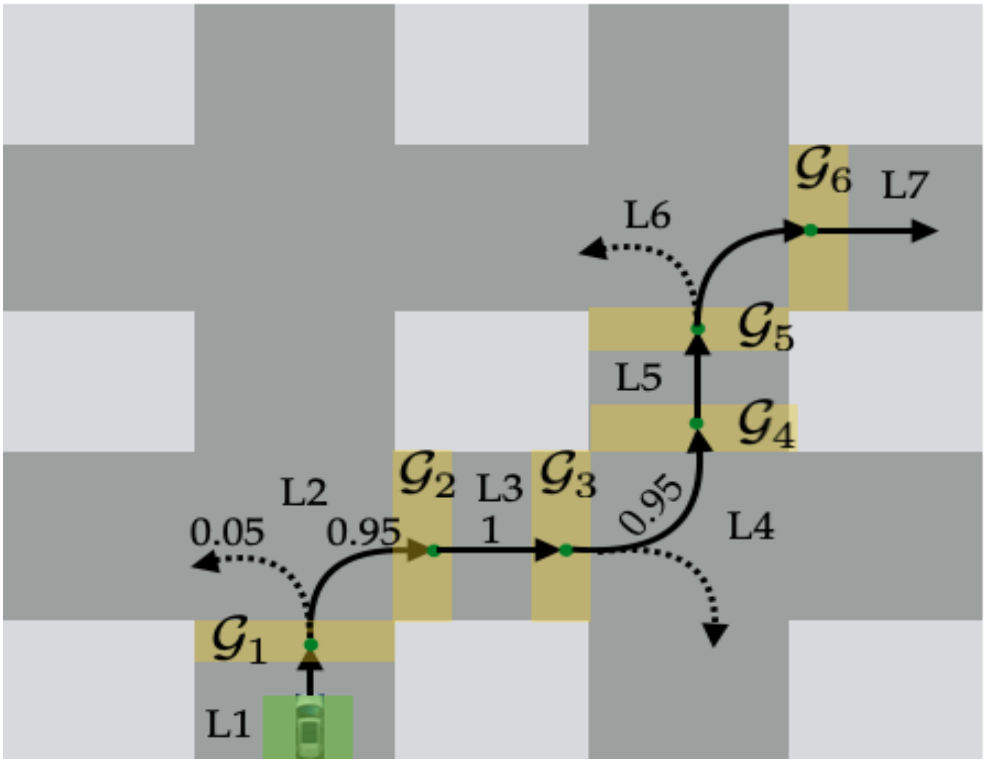

Figure 4.6: Vehicle navigation

in the region $\mathcal{G}_1$, the vehicle could either take left/right turn. Here, we assign probability value 0.95 to the desired direction and probability value 0.05 to the undesired direction as shown in Figure 4.6. Finally, we generate a polyhedral PHS corresponding to the route as shown in Figure 4.7. In Figure 4.7, we create a location for each cell involved in the route. Then, for each location, we assign appropriate dynamics based on the route. For example, in the location L_1, the vehicle moves in upward direction. Hence, we assign the dynamics of vertical motion, that is, $\dot{z} = A(0, 1, 0)z$. Here, the motion in anticlockwise and clockwise direction are annotated by AW and CW respectively.

We first evaluate our method for the case study demonstrated in Section 3.3 of Chapter 3. Let $\mathbb{U}$ be the region beyond the lane. Here, we want to compute the probability of the vehicle reaching $\mathbb{U}$ and $\neg\mathbb{U}$ in the North direction starting from $\mathbb{I} = [-4, -3.9] \times [-0.1, 0.1]$ towards the East direction, respectively. Further, for measuring the scalability of our method, we consider a set of examples. We generate different polyhedral PHS of different number of

locations and probabilistic edges. We consider a scenario consisting of 20 horizontal and 20 vertical roads intersection with each other in a 2D plane. Then, we randomly generate a simple route of different length and its corresponding polyhedral probabilistic hybrid system as described in the previous paragraph. Here, we want to compute the probability of the vehicle reaching the end cell($\mathbb{F}$) from the initial cell.

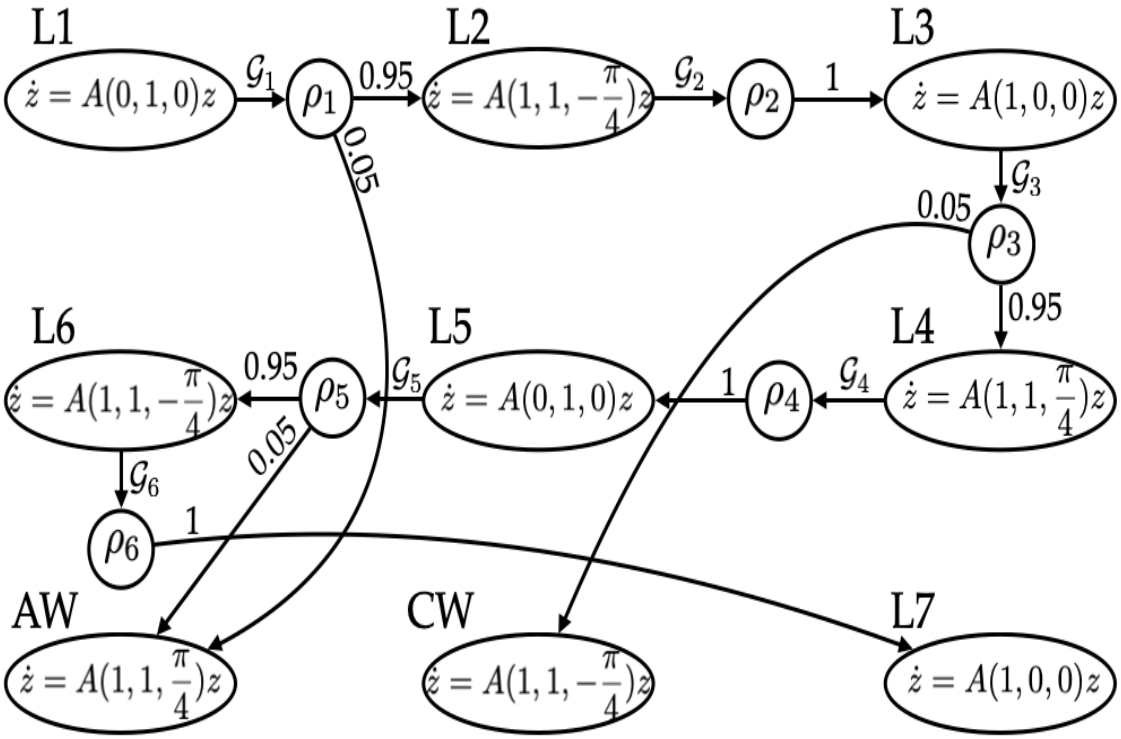

Figure 4.7: Polyhedral PHS for the Navigation shown in Figure 4.6

Experimental results

The experimental results are summarized in Tables 4.1, 4.2, and 4.3. In the tables, *Prob* denotes the maximum probability of reachability; K and T denote the number of discrete transitions and time horizon, respectively. E_{time} and C_{time}denote the time taken for encoding the bounded probabilistic behaviors, time for computing the maximum/minimum probability by the tool Z3opt, respectively. In Table 4.3, *Nodes* and *Edges* denote the number of locations and probabilistic edges, respectively.

Row	K	$Prob(\mathbb{U})$	$Prob(\neg\mathbb{U})$	E_{time}(sec.)	C_{time} (sec.)
1	2	0	0.81	0.063	0.049
2	4	0	0.9963	0.107	0.082
3	6	0	0.999945	0.189	0.195
4	8	0	0.99999927	0.190	0.580

Table 4.1: Probabilistic reachability of the case-study for $T = 5$

Next, in Tables 4.1 and 4.2, we report the results of analyzing the case study for different number of discrete transitions K and time horizon T, respectively. In Table 4.1, we have fixed the value of T, and varied the number of discrete transitions K. We observe that the probability of the vehicle reaching beyond the lane in the North direction is zero for all values of discrete transitions. However, the probability of the vehicle being in the lane in the North direction increases as we increase the value of K. This is expected because the number of paths reaching $\neg\mathbb{U}$ in the North direction increases when the value of K increases; thus increasing the probability. In adition, both the encoding and computation time grow linearly as we increase the number of discrete transitions as it can be seen in the Table 4.1.

Row	T	$Prob(\mathbb{U})$	$Prob(\neg\mathbb{U})$	E_{time}(sec.)	C_{time} (sec.)
1	2	0	0	0.118	0.080
2	3	0	0	0.78	0.075
3	4	0	0.9963	0.105	0.095
4	5	0	0.9963	0.117	0.070

Table 4.2: Probabilistic reachability of the case-study for $K = 4$

In Table 4.2, we have fixed the number of discrete transitions K, and varied time horizon T. We observe that the probability of the vehicle reaching beyond and on the lane in the North direction is zero for $T = 2, 3$. This is because 3 units of time are not sufficient for the vehicle to move from East direction to North direction. For $T = 4, 5$, the probability of $\mathbb{U}$ in the North direction is zero; however, probability of $\neg\mathbb{U}$ in the North direction is 0.9963 because all paths up to length 4 may have been explored for $T = 4$. In addition, both the encoding and computation time remain constant because size of the encoding does not grow

when we increase the time horizon T.

Row	*Length*	Nodes	Edges	(K,T)	$Prob(\mathbb{F})$	E_{time}(sec.)	C_{time} (sec.)
1	5	7	4	(5,5)	0.9025	0.11	0.062
2	10	12	9	(10,10)	0.8145	0.19	0.057
3	15	17	14	(15,15)	0.73556	0.27	0.06
4	20	22	19	(20,20)	0.63024	0.35	0.105

Table 4.3: Probability of reaching $\mathbb{F}$

In Table 4.3, we report the results of analyzing polyhedral PHS of different number of locations and edges generated from different lengths of route. Here, we choose number of transitions K and time horizon T to be the same as the length of the respective route. In Table 4.3, we observe that the probability of reaching end cell ($\mathbb{F}$) of the respective route decreases, when we increase the length of the route. This is because there is only one path reaching the location corresponding to the end cell $\mathbb{F}$ as can be seen in Figure 4.7. Hence, the probability of a shorter path is high, and the probability of a longer path is less.

Remark 3. *Methods similar to Bounded model checking require solvers to solve the encoding capturing all potential bounded executions. For solvers like Z3, scalable is limited due to the computational complexity which increases with the number of quantified variables. An alternative approach to deal with bounded probabilistic reachability analysis is sampling based approach, such as statistical model checking (SMC)[142]. Different methods, such as frequetist sequential probability ratio test (SPRT)[72], Bayesian SMC[74] have been explored. However, there is trade-off between accuracy and computational effort. While SMC based approaches provide approximate results, it is beneficial for bounded reachability analysis of probabilistic hybrid systems with complex dynamics. Note that such an approach can not be directly employed for unbounded probabilistic reachability analysis because paths of infinite length need to be sampled.*

Chapter 5

Safety analysis of polyhedral probabilistic hybrid systems

In this chapter, we consider safety verification problems of polyhedral probabilistic hybrid systems. We present a counterexample guided abstraction refinement (CEGAR) based algorithm. The broad approach consists of starting with abstraction and iteratively refining the abstraction based on a counterexample in the the abstract system that points to a potential violation of the property. To the best of our knowledge, this is the first CEGAR algorithm for probabilistic hybrid systems.

Developing a CEGAR algorithm for PHSs is challenging because it needs to address the branching behavior due to probabilities and the infinite state space due to continuous variables simultaneously. We view a given concrete PHS $\mathcal{H}$ as an infinite state MDP and start by constructing an abstract finite state MDP $\hat{\mathcal{H}}$ that simulates $\mathcal{H}$. We use the model-checker PRISM to obtain a counterexample of the abstract MDP, which is a DTMC in the form of a layered directed acyclic graph (LDAG), that is obtained by a finite unrolling of the DTMC returned by PRISM. Unlike DTMC, the LDAG has bounded paths and is more succinct than a tree unrolling. While the validation problem is a bounded model-checking problem, an SMT based approach for validation typically requires an exponential number of variables in the

length of the counterexample[143]. We present an efficient validation procedure that performs a bottom-up backward reachability analysis and either finds a concrete counterexample or points to a node in the LDAG that needs to be refined. The refinement procedure is more involved as compared to the non-probabilistic setting due to the absence of clear disjoint sets that can be separated in the refinement step. Let us say that the validation fails at the layer k from the top at a node v, which corresponds to a set of concrete states S_v. Let $R_1, R_2, \cdots, R_n$ be the nodes reached using the backward reachable set computation at layer $k+1$. We know that the set of predecessors of the states $R_1, R_2, \cdots, R_n$ which are in S_v is empty (because the validation failed). This implies that the successors of the states S_v which correspond to a set of n tuples, say P_v, is disjoint from $R_1 \times R_2 \times \cdots \times R_n$. However, this does not immediately imply that the projection of P_v to the i-th component, namely, P_v^i is disjoint from R_i. Refinement in the purely hybrid setting is based on finding two disjoint sets, and splitting the corresponding node in the abstract system to separate a post set and a reach set. We observe that in general, we do not find two disjoint sets P_v^i and R_i for every i, nevertheless, we can split in a manner that ensures progress, that is, guarantees the elimination of the abstract counterexample in a finite number of refinements.

We implemented our algorithm in a Python toolbox called Procegar, and we compared our method with the tool ProHVer[131]. Our method concludes safety in many more instances than ProHVer. The method can return concrete counterexamples and it scales to higher dimensions than ProHVer.

Remark 4. *We use the Markov decision process for the semantics of polyhedral probabilistic hybrid systems in the rest of this chapter.*

5.1 Preliminaries

Notations and Definitions

Definition 10 (Layered Directed Acyclic Graph (LDAG))**.** *An* LDAG $\mathcal{D} = (\mathcal{S}', \longrightarrow')$ *is a special case of* DTMC *where* $\mathcal{S}' \subseteq \mathcal{S} \times \mathbb{N}$ *for some set* $\mathcal{S}$ *such that*

- $\star$ *There is a unique state* $s \in \mathcal{S}$ *such that* $(s, 0) \in \mathcal{S}'$*;*
- $\star$ *If* $((s, i), \rho) \in \longrightarrow'$*, then for all* $t \in \mathcal{S}$*,* $\rho(t, j) = 0$ *for* $j \neq i + 1$*;*
- $\star$ *If* $(s, i) \in \mathcal{S}'$*, then there exists* $(s', i-1) \longrightarrow' \rho$ *for some* $(s', i-1) \in \mathcal{S}'$ *such that* $\rho(s, i) \neq 0$*.*

The LDAG begins with a unique root state at layer 0. All state transitions with positive probability correspond to jumps to states in the next layer. The last condition ensures that each state in the LDAG can be reached from the root. The depth d of a finite LDAG $\mathcal{D} = (\mathcal{S}', \longrightarrow')$ is the maximum length of any path in $\mathcal{D}$, that is, $d = \max\{i \mid (s, i) \in \mathcal{S}'\}$. Next, we define a sub-LDAG as a substructure of an LDAG rooted at a certain state/node.

Definition 11 (Sub-LDAG)**.** *Given an* LDAG $\mathcal{D} = (\mathcal{S}, \longrightarrow)$*, a sub-*LDAG $\mathcal{D}' = (\mathcal{S}', \longrightarrow')$ *rooted at* $(s, j) \in \mathcal{S}$ *is defined as*

- $\star$ $\mathcal{S}' = \{(s', i - j) \mid (s', i)$ *is reachable from* (s, j) *in* $\mathcal{D}\}$*;*
- $\star$ $\longrightarrow' = \{((s', i), \rho') \mid (s', i) \in \mathcal{S}', \exists((s', i + j), \rho) \in \longrightarrow, \forall\ (s'', i + 1) \in \mathcal{S}', \rho'((s'', i + 1)) = \rho(s'', i + j + 1)\}$

We consider executions of $\mathcal{M}$ as LDAGs that arise due to unrolling of $\mathcal{M}$.

Definition 12. *Given an* MDP $(\mathcal{S}, \longrightarrow)$*, Exec*$(\mathcal{M})$ *is the set of all* LDAG*s* $\mathcal{D} = (\mathcal{S}', \longrightarrow')$ *where* $\mathcal{S}' \subseteq \mathcal{S} \times \mathbb{N}$*; and for every* $((s, i), \rho') \in \longrightarrow'$*, then there is* ρ *such that* $(s, \rho) \in \longrightarrow$ *and for every* $(t, i + 1) \in \mathcal{S}'$*,* $\rho'(t, i + 1) = \rho(t)$*.*

Probabilistic simulation

Probabilistic simulation captures a relation between states of two MDPs wherein if two states are related then any probabilistic transition from the simulated state can be mimicked by the simulating state. In this section, we consider a restricted version of the probabilistic simulation relation between two probabilistic systems as defined in[144], which is sufficient to capture the relation between the concrete system and its abstraction defined in this paper. In particular, we enforce a bijection between the probability distribution associated with the two states. We consider this restricted version because it simplifies our technical discussion and suffices for our purposes.

Definition 13. *Let* $\mathcal{M}_1 = (\mathcal{S}_1, \longrightarrow_1)$ *and* $\mathcal{M}_2 = (\mathcal{S}_2, \longrightarrow_2)$ *be two* MDP*s. An onto function* $\alpha : \mathcal{S}_1 \to \mathcal{S}_2$ *is a probabilistic simulation from* $\mathcal{S}_1$ *to* $\mathcal{S}_2$ *if for all* $s_1 \in \mathcal{S}_1$, $s_2 \in \mathcal{S}_2$ *such that* $\alpha(s_1) = s_2$ *and* $s_1 \longrightarrow \pi$, $\exists\ \rho \in Dist(\mathcal{S}_2)$ *such that* $s_2 \longrightarrow \rho$, *and*

- $\pi(s) = \rho(\alpha(s))$, *for all* $s \in support(\pi)$;
- *For* $s, s' \in support(\pi)$ *if* $s \neq s'$, *then* $\alpha(s) \neq \alpha(s')$.

We say that α is a probabilistic simulation from $\mathcal{M}_1$ to $\mathcal{M}_2$, denoted by $\mathcal{M}_1 \leq_\alpha \mathcal{M}_2$. In addition, given $\mathscr{F} \subseteq \mathcal{S}_1$, we say that α respects $\mathscr{F}$ if for $s \in \mathscr{F}$ and $t \in \mathcal{S}_1 \setminus \mathscr{F}$, $\alpha(s) \neq \alpha(t)$.

Next, the following theorem states the fact that if two MDPs $\mathcal{M}_1$ and $\mathcal{M}_2$ are related by a probabilistic simulation α, that is, $\mathcal{M}_1 \leq_\alpha \mathcal{M}_2$, then MDP $\mathcal{M}_2$ is an over-approximation of $\mathcal{M}_1$, that is, $\mathcal{M}_2$ has a larger number of executions than $\mathcal{M}_1$; thus the minimum and maximum probability of reaching a set of states in $\mathcal{M}_1$ is lower bounded by the minimum and upper bounded by the maximum probability of reaching the related set of states in $\mathcal{M}_2$. The following theorem is similar to the theorem given in the paper[96].

Theorem 1. *Let* $\mathcal{M}_1 = (\mathcal{S}_1, \longrightarrow_1)$ *and* $\mathcal{M}_2 = (\mathcal{S}_2, \longrightarrow_2)$ *be two* MDP*s and* $\mathscr{F}_1 \subseteq \mathcal{S}_1$. *If* α *is a probabilistic simulation from* $\mathcal{M}_1$ *to* $\mathcal{M}_2$ *which respects* $\mathscr{F}_1$ *then for any* $s_1 \in \mathcal{S}_1$, *we have*

(a) $\mathcal{P}_{sup}(\mathcal{M}_1, s_1, \mathscr{F}_1) \leq \mathcal{P}_{sup}(\mathcal{M}_2, \alpha(s_1), \alpha(\mathscr{F}_1))$;

(b) $Prob_{inf}(\mathcal{M}_2, \alpha(s_1), \alpha(\mathscr{F}_1)) \leq Prob_{inf}(\mathcal{M}_1, s_1, \mathscr{F}_1)$.

Next, we define a function $\alpha_{\mathbb{N}}$ that extends probabilistic simulation relation between states of MDPs to states of corresponding executions.

Definition 14. *Given a probabilistic simulation* $\alpha : \mathcal{S}_1 \to \mathcal{S}_2$ *between two* MDP*s* $\mathcal{M}_1 = (\mathcal{S}_1, \longrightarrow_1)$ *and* $\mathcal{M}_2 = (\mathcal{S}_2, \longrightarrow_2)$*, we define* $\alpha_{\mathbb{N}} : \mathcal{S}_1 \times \mathbb{N} \to \mathcal{S}_2 \times \mathbb{N}$ *such that* $\alpha_{\mathbb{N}}(s, i) = (\alpha(s), i)$ *for each* $s \in \mathcal{S}_1$*,* $i \in \mathbb{N}$*.*

Furthermore, we define the set of all executions $\mathcal{D}_1$ of an MDP $\mathcal{M}_1$ which simulate an execution $\mathcal{D}_2$ of an MDP $\mathcal{M}_2$ with respect to the probabilistic simulation α between $\mathcal{M}_1$ and $\mathcal{M}_2$.

Definition 15. *Given a probabilistic simulation* $\alpha : \mathcal{S}_1 \to \mathcal{S}_2$ *between two* MDP*s* $\mathcal{M}_1 = (\mathcal{S}_1, \longrightarrow_1)$ *and* $\mathcal{M}_2 = (\mathcal{S}_2, \longrightarrow_2)$*, an* LDAG $\mathcal{D}_2 \in Exec(\mathcal{M}_2)$*, we define* $\alpha^{-1}(\mathcal{D}_2)$ *as follow:*

$$\alpha^{-1}(\mathcal{D}_2) = \{\mathcal{D}_1 \in Exec(\mathcal{M}_1) \mid \mathcal{D}_2 \leq_\gamma \mathcal{D}_1 \text{ for some } \gamma \in \Im(\alpha_{\mathbb{N}})\}.$$

Additionally, we define an abstraction of an MDP $\mathcal{M} = (\mathcal{S}, \longrightarrow)$ denoted $\alpha(\mathcal{M})$ with respect to an abstraction function $\alpha : \mathcal{S} \to \mathcal{S}'$ for some $\mathcal{S}'$. The abstraction function is formally defined as follows.

Definition 16. *Given an* MDP $\mathcal{M} = (\mathcal{S}, \longrightarrow)$*, an abstraction function* $\alpha : \mathcal{S} \to \mathcal{S}'$ *for some set* $\mathcal{S}'$ *is a function such that for any* $(s, \rho) \in \longrightarrow$*, if* $s_1, s_2 \in support(\rho)$*, then* $\alpha(s_1) \neq \alpha(s_2)$*, where* $s_1 \neq s_2$*.*

The above definition appears to be a restricted notion of abstraction on the MDPs, however, the particular abstractions we consider on probabilistic hybrid systems in the sequel will correspond to abstraction functions of this kind on the underlying MDPs. Furthermore, the formal definition of $\alpha(\mathcal{M})$ is given as below.

Definition 17. *Given an* MDP $\mathcal{M} = (\mathcal{S}, \longrightarrow)$ *and an abstraction function* $\alpha : \mathcal{S} \to \mathcal{S}'$*,* $\alpha(\mathcal{M}) = (\mathcal{S}', \longrightarrow')$*, where* $\longrightarrow' = \{(s', \rho') \mid \exists (s, \rho) \in \longrightarrow, \alpha(s) = s', \forall t, \rho(t) = \rho'(\alpha(t))\}$.

The theorem below states that $\alpha(\mathcal{M})$ is an over-approximation of $\mathcal{M}$.

Theorem 2. *Given an* MDP $\mathcal{M} = (\mathcal{S}, \longrightarrow)$*, an abstraction function* $\alpha : \mathcal{S} \to \mathcal{S}'$ *for some set* $\mathcal{S}'$*, we have* $\mathcal{M} \leq_\alpha \alpha(\mathcal{M})$*.*

5.2 Problem definition

Problem 2 (The Maximum Probabilistic Safety Problem)**.** *Given a polyhedral* PHS $\mathcal{H}$*, an initial location* q_0*, a set of final locations* $\mathbb{F}$ *and probability bound* $\theta \in [0, 1]$*, check whether the maximum probability of reaching* $\mathbb{F} \times \mathcal{X}$ *from* $\{q_0\} \times Inv(q_0)$ *is less than or equal to* θ*, that is,* $\max_{s \in \{q_0\} \times Inv(q_0)} \mathcal{P}_{sup}([\![\mathcal{H}]\!], s, \mathbb{F} \times \mathcal{X}) \leq \theta$*.*

Similarly, the minimum probabilistic reachability problem can be defined. Here, we focus on the maximum probabilistic safety problem. However, our results extend in a straightforward manner to verify lower bound on the minimum probability of reachability.

5.3 Running example

We consider a one dimensional example shown in Figure 5.1 as a running example throughout this chapter to illustrate each step of our CEGAR algorithm.

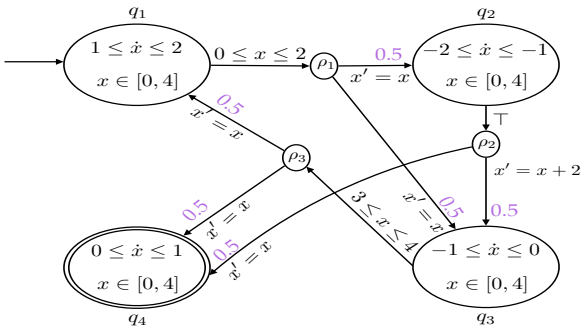

Figure 5.1: Polyhedral Probabilistic Hybrid System

Example 3. *Figure 5.1 illustrates a polyhedral* PHS*, where the locations* q_1, q_2, q_3 *and* q_4 *are represented using circle nodes, where the initial location is indicated with an incoming arrow and the final location is indicated with double circles. The variable* x *represents the continuous variable which takes real values, that is, the continuous statespace is* $\mathbb{R}$*. Each location is annotated with a polyhedral inclusion dynamics and an invariant which are written within the corresponding circle. A polyhedral inclusion dynamics is represented using a constraint over the rates of the continuous state variables, and the invariant is a constraint over the continuous state variables, which constrains the system evolution at a particular location. For example, in location* q_1*, the polyhedral inclusion dynamics is* $1 \leq \dot{x} \leq 2$*, where* $\dot{x}$ *is the rate of continuous state evolution, and the invariant is* $x \in [0, 4]$*. Next, the switching from one location to another location is expressed by a probabilistic edge which has mainly three components, namely, the guard, the probability distribution and the reset. The guard is a constraint over the continuous state variables that represents the condition that enables the system to switch between locations; the probability distribution provides the probability of switching to a target location, and the reset expressed as a constraint over the continuous state variables and its primed versions, provides the values of the continuous state before and after a transition, respectively. For example, in location* q_1*, the system has a probabilistic edge with guard* $0 \leq x \leq 2$*, probability distribution* $\rho_1(q_2) = \rho_1(q_3) = 0.5$ *and reset for locations* q_2 *and* q_3 *as* $x' = x$*. Hence, when the system is in location* q_1 *and the continuous state* x *satisfies the guard* $0 \leq x \leq 2$*, the system switches to locations* q_2 *and* q_3 *with probability* 0.5 *each, and the continuous state remains unchanged (*$x_p = x$*).*

5.4 Counterexample guided abstraction refinement

In this section, we provide the details of our Counter-Example Guided Abstraction Refinement (CEGAR)[145] algorithm for the probabilistic safety analysis of polyhedral PHS. The maximum probabilistic safety problem for polyhedral PHS is challenging due to the infinite

statespace. An abstraction based analysis consists of *abstracting* a given (concrete) system to a simpler abstract system, such that the satisfaction of a certain property, such as, the maximum probability of reaching an unsafe state being less than θ, on the abstract system, implies the satisfaction of the property on the concrete system. In our case, the concrete system is a polyhedral PHS and the abstract system is a finite state MDP, on which the maximum probabilistic safety analysis can be carried out efficiently. One of the challenges towards abstraction based analysis is the choice of the abstraction, on which the success of the verification crucially relies. In other words, the abstract system is often an "over-approximation" and hence, the violation of the property by the abstract system does not imply a violation in the concrete system. However, a violation in the abstract system provides a potential reason for the violation of the property in the concrete system, in the form of an *abstract counterexample* (ACE). Analysis of the abstract counterexample using a *validation* algorithm, results in either a concrete counterexample corresponding to it (a bug) or in concluding that the abstract counterexample is spurious, that is, does not correspond to a real counterexample. In the latter case, the spurious counterexample analysis can be used to *refine* the abstract system so as to eliminate the same. This process is continued until either some abstraction of the concrete system satisfies the property or the validation algorithm determines that the system violates the property. This iterative refinement of the abstract system based on counterexamples is referred to as the counterexample guided abstraction refinement. While CEGAR algorithms have been proposed for (non-probabilistic) hybrid systems[138] and finite state probabilistic systems[94], there are several challenges due to the combination of infinite statespace and the branching behavior in a probabilistic hybrid system. In particular, we need a succinct representation of the abstract counterexample, efficient validation and refinement algorithms. We provide the details of each of the steps of the CEGAR loop in the sequel.

5.4.1 Abstraction

In this section, we develop a partition based predicate abstraction technique that converts a polyhedral PHS into a finite state MDP. The broad idea is to divide the statespace of each mode into a finite number of regions, and construct an MDP which consists of these regions as abstract nodes. Let us fix a polyhedral PHS $\mathcal{H} = (\mathcal{Q}, \mathcal{X}, \mathit{Inv}, \mathit{Flow}, \mathit{Edges}, \mathit{Guard}, \mathit{Reset})$ and a finite partition $\mathcal{R}$ of $\mathcal{Q} \times \mathcal{X}$ such that for any partition element $\mathcal{P} \in \mathcal{R}$ if $(q, x), (q', x') \in \mathcal{P}$ then $q = q'$. So, $\mathcal{R}$ consists of a partitioning of the statespace where states from different modes are kept separate. Predicate abstraction creates a set of locations, where each location is a partition element $\mathcal{P} \in \mathcal{R}$. For each probabilistic edge (q, ρ), where $\rho(q_1) = p_1, \rho(q_2) = p_2, \ldots, \rho(q_c) = p_c$, for each tuple $((q_0, R_0), (q_1, R_1), \ldots, (q_c, R_c)) \in \mathcal{R}^{c+1}$, we check whether there exists a probabilistic execution starting from $x \in \mathit{Inv}(q) \cap R_0$ and reaching $\mathit{Inv}(q_i) \cap R_i$, $i = 1, 2, \ldots, c$. More precisely, there exists a vector field r in flow function associated with the location q, $T \geq 0$ such that $x + rt \in \mathit{Inv}(q)$ for $0 \leq t \leq T$ and guard condition corresponding to the probabilistic edge (q, ρ) is satisfied by $x + rT$; there exist $x_1, x_2, \ldots, x_c$ are reset by reset function for $((q, \rho), q_1, x + rT), ((q, \rho), q_2, x + rT), \ldots, ((q, \rho), q_c, x + rT)$, respectively. Then, there exist vector fields $r_1, r_2, \ldots, r_c$ in the flow function associated with the locations $q_1, q_2, \ldots, q_c$, respectively, $T_1 \geq 0, T_2 \geq 0, \ldots, T_c \geq 0$ such that for $0 \leq t \leq T_i$, $x_i + r_i t \in \mathit{Inv}(q_i)$ and $x_i + r_i T_i \in R_i$, $i = 1, 2, \ldots, c$. If the execution exists, then we create a probabilistic edge $((q, R_0), \rho')$ such that $\rho'((q_i, R_i)) = \rho(q_i)$, $i = 1, 2, \ldots, c$. Next, we present the formal definition of predicate abstraction.

Definition 18. *Given a polyhedral* PHS $\mathcal{H} = (\mathcal{Q}, \mathcal{X}, \mathit{Inv}, \mathit{Flow}, \mathit{Edges}, \mathit{Guard}, \mathit{Reset})$ *and a finite partition* $\mathcal{R}$ *of the state space* $\mathcal{Q} \times \mathcal{X}$*, where* $\mathcal{R} = \mathcal{Q} \times \{R_i \mid 1 \leq i \leq k,\ k \in \mathbb{N}\}$*, we construct a finite state* MDP $\mathit{Abs}(\mathcal{H}, \mathcal{R}) = (\mathcal{S}, \longrightarrow)$*, where*

- $\star$ $\mathcal{S} = \mathcal{R}$;
- $\star$ $((q, R), \pi) \in \longrightarrow$ *if* $\exists$ $(q, \rho) \in \mathit{Edges}$, $x \in R$, $I = |\{q' \mid \rho(q') \neq 0\}|$, $((q_1, R_1), (q_2, R_2), \ldots, (q_I, R_I)) \in \mathcal{R}^I$, *where* $\rho(q_i) \neq 0$ *for* $i \in [I]$, $\phi : [0, T] \to \mathcal{X}$, $x_i, \phi_i : [0, T_i] \to \mathcal{X}$, $i \in [I]$

such that

- $\star$ $\phi(0) = x$ *and* $\phi(T) \in \mathit{Guard}((q, \rho))$;
- $\star$ *For* $0 \leq t \leq T$, $\frac{d\phi(t)}{dt} \in \mathit{Flow}((q, \phi(t)))$, $\phi(t) \in \mathit{Inv}(q)$;
- $\star$ $x_i = \mathit{Reset}(((q, \rho), q_i, \phi(T)))$, $\phi_i(0) = x_i$, $i \in [I]$;
- $\star$ *for* $0 \leq t \leq T_i$, $\frac{d\phi_i(t)}{dt} \in \mathit{Flow}((q_i, \phi_i(t)))$, $\phi_i(t) \in \mathit{Inv}(q_i)$, $i \in [I]$;
- $\star$ $\phi_i(T_i) \in R_i$, $i \in [I]$; $\forall\ \pi((q_i, R_i)) > 0$, $\rho(q_i) = \pi((q_i, R_i))$.

From the definition, for each state (q, R), there could be more than one probabilistic distributions over $\mathcal{R}$ associated with the state, thus allowing non-determinism. Therefore, $\mathit{Abs}(\mathcal{H}, \mathcal{R})$ need not be DTMC in general even if $\mathcal{H}$ has deterministic probabilistic transitions. Therefore, $\mathit{Abs}(\mathcal{H}, \mathcal{R})$ is a finite state MDP. In addition, $\mathit{Abs}(\mathcal{H}, \mathcal{R})$ captures all valid probabilistic edges of $\mathcal{H}$. Next, we present the correctness of the predicate abstraction.

Consider a function $\alpha_{\mathcal{R}} : \mathcal{Q} \times \mathcal{X} \to \mathcal{R}$ such that $\alpha_{\mathcal{R}}(q, x) = (q, R)$, where $x \in R$. That is, $\alpha_{\mathcal{R}}$ maps a concrete state to the corresponding region in the partition it belongs to. Note that $\alpha_{\mathcal{R}}$ is an abstraction function, because, for any $((q, x), \rho) \in \longrightarrow_{[\![\mathcal{H}]\!]}$, if $(q_1, x_1), (q_2, x_2) \in \mathit{support}(\rho)$ and $(q_1, x_1) \neq (q_2, x_2)$, then $q_1 \neq q_2$ (from the semantics, there is a unique successor state corresponding to every target location), and hence, $\alpha_{\mathcal{R}}(q_1, x_1) \neq \alpha_{\mathcal{R}}(q_2, x_2)$.

The abstraction of $\mathcal{H}$ with respect to the finite partition $\mathcal{R}$ denoted as $\mathit{Abs}(\mathcal{H}, \mathcal{R})$ is $\alpha_{\mathcal{R}}([\![\mathcal{H}]\!])$. From Theorem 4, $\mathit{Abs}(\mathcal{H}, \mathcal{R})$ is an over-approximation of polyhedral PHS $\mathcal{H}$. From Theorem 6, if the abstract system $\mathit{Abs}(\mathcal{H}, \mathcal{R})$ is probabilistically safe, that is, $\mathcal{P}_{sup}(\mathit{Abs}(\mathcal{H}, \mathcal{R}), (q_0, R), \alpha_{\mathcal{R}}(\mathbb{F} \times \mathcal{X})) \leq \theta$ for every $(q_0, R) \in \alpha_{\mathcal{R}}(\{q_0\} \times \mathit{Inv}(q_0))$, then the concrete system $\mathcal{H}$ is also probabilistically safe, that is, $\mathcal{P}_{sup}([\![\mathcal{H}]\!], (q_0, x), \mathbb{F} \times \mathcal{X}) \leq \theta$ for every $(q_0, x) \in \{q_0\} \times \mathit{Inv}(q_0)$. If $\mathcal{R}$ is a polyhedral partition, then $\mathit{Abs}(\mathcal{H}, \mathcal{R})$ can be constructed. Note that from Definition 30, to check whether a probabilistic edge exists in the abstract system, we need to check if a corresponding concrete edge exists. This can be encoded as a satisfiability problem of a conjunction of linear constraints. Alternately, one could use polyhedral manipulations to compute the predecessor states corresponding to the abstract probabilistic edge in the

concrete system, and check for its emptiness. The next example illustrates the computation of the abstract MDP.

Example 4. *Consider the polyhedral* PHS $\mathcal{H}$ *shown in Figure 5.1 with its state space* $\mathcal{Q}\times\mathcal{X} = \{q_1, q_2, q_3, q_4\} \times [0, 4]$. *Let* $\mathcal{R} = \{(q_1, R), (q_2, R), (q_3, R), (q_4, R)\}$ *be a finite partition of the state space* $\mathcal{Q} \times \mathcal{X}$, *where* $R = [0, 4]$. *The abstraction of* $\mathcal{H}$ *with respect to the partition* $\mathcal{R}$ *is* $Abs(\mathcal{H}, R)$ *which we construct according to Definition 30. Each partition element of* $\mathcal{R}$ *is an abstract state, for instance,* (q_1, R) *is a state of the abstract system as shown in Figure 5.2. Next, we construct a probabilistic edge from one abstract state to a tuple of abstract states with certain probabilities if there is a corresponding concrete edge in* $\mathcal{H}$. *For example, we construct an abstract edge from* (q_1, R) *to* (q_2, R) *and* (q_3, R) *with probabilities* 0.5 *and* 0.5, *respectively, because a corresponding concrete edge exists in* $\mathcal{H}$, *as given next. The system starts from* $x = 0.5 \in R$ *at location* q_1 *and evolves for time* $t = 2$ *with flow* $\dot{x} = 0.5$ *and reaches the state* $x_1 = x + \dot{x}t = 0.5 + 0.5 * 2 = 1.5$ *which satisfies the guard constraints, that is,* $1.5 \in [0, 2]$. *Then, the system switches to* q_2 *and* q_3 *by resetting the continuous states to* $x_2 = x_1 = 1.5$ *and* $x_3 = x_1 = 1.5$, *respectively. Since* $x_2, x_3 \in R$, *we have a concretization of the edge from* (q_1, R) *to* (q_2, R) *and* (q_3, R) *with probability* 0.5 *each.*

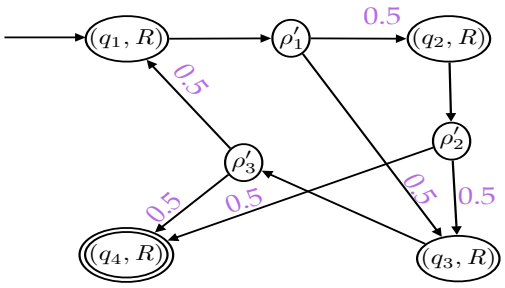

Figure 5.2: Markov Decision Process $Abs(\mathcal{H}, \mathcal{R})$

5.4.2 Model checking and counterexample

In this section, we verify the maximum probabilistic safety problem for the abstract system $Abs(\mathcal{H}, \mathcal{R})$. Since the abstract system is a finite MDP, we adopt probabilistic model checker

tool PRISM[35] for the purpose of verification. PRISM[35] takes as input a finite MDP, a set of initial states and a PCTL formula[65], and returns either *true* to indicate that the MDP satisfies the formula, or returns a counterexample $\widehat{\mathcal{T}}$ in the form of a DTMC. Maximum probabilistic safety specification can be encoded as a PCTL-formula. While $\widehat{\mathcal{T}}$ is a counterexample, it contains cycles and hence, analyzing it for spuriousness is complex. Hence, we unroll $\widehat{\mathcal{T}}$ and construct a finite LDAG $\widehat{\mathcal{D}}$ as a succinct counterexample. Next, we provide the formal definition of a counterexample.

Definition 19. *Given an* MDP $\mathcal{M}$*, an initial state* q_0*, a set of final states* $\mathbb{F}$ *and* $\theta \in [0,1]$*, an* LDAG $\widehat{\mathcal{D}} \in \mathit{Exec}(\mathcal{M})$ *is a counterexample of* $\mathcal{M}$ *with respect to* q_0*,* $\mathbb{F}$ *and* θ*, if* $\mathcal{P}(\widehat{\mathcal{D}}, (q_0, 0), \mathbb{F} \times \mathbb{N}) > \theta$*.*

Algorithm 1 provides the details of extracting a succinct LDAG counterexample $\widehat{\mathcal{D}}$ from a given DTMC counterexample $\widehat{\mathcal{T}}$ that is returned by PRISM which indicates that the probability of reaching $\mathbb{F}$ in $\widehat{\mathcal{T}}$ from q_0 is greater than θ. The crux of the algorithm consist of unrolling the DTMC and collecting enough paths reaching $\mathbb{F}$ from q_0 whose sum total exceeds θ. The algorithm iteratively computes p_q^i, which is the probability of reaching a final state in $\mathbb{F}$ from q within i steps. It can be observed that there is always a d for which $p_{q_0}^d$ exceeds θ. To see this, note that in the DTMC $\widehat{\mathcal{T}}$, the probability of reaching $\mathbb{F}$ from q is $> \theta$, which is the sum of probabilities of paths reaching $\mathbb{F}$. Since, the DTMC is finite, we can enumerate the paths, and the corresponding probabilities. Let $p_1, p_2, \cdots$ be the probabilities of the paths reaching $\mathbb{F}$, where $\sum_i p_i > \theta$. Our objective is to show that there is finite number of these probabilities (correspond to finitely many paths) who sum exceeds θ. Note that if the sequence is finite, we can take all the paths that reach $\mathbb{F}$. Otherwise, $\sum_i p_i > \theta$ implies that $\lim_{i\to\infty} \sum_{j=1}^{i} p_i > \theta$ by definition of infinite sum. From the definition of limit, there is a k such that $\sum_{j=1}^{i} p_i > \theta$ for all $i \geq k$. In particular, $\sum_{j=1}^{k} p_i > \theta$. Hence, in at most k iterations, p_q^d will exceed θ and exit the loop. Once the length d for the sufficient unrolling is computed, the function ForwardReach performs a forward reachability analysis to only keep those nodes in the unrolling of $\mathcal{M}$ up to length d that can be reached from q_0

using transitions in the support (that is, edges with non-zero probability).

Algorithm 1: **Get_LDAG:** Extract a succinct LDAG counterexample from a given DTMC counterexample

Input: $\widehat{\mathcal{T}}$ - a DTMC, $\mathbb{F}$ - a set of final states, θ - a probability bound, q_0 - a state of $\widehat{\mathcal{T}}$

Output: a finite LDAG $\widehat{\mathcal{D}}$

```
1 SetKwKwGoTogo to begin
2     For each state q of T̂ if q ∈ F then p_q^0 = 1 else p_q^0 = 0   d=0
3     while p_{q0}^d ≤ θ do
4         d =d+1
5         for each edge (q, ρ) of T̂ do
6             if q ∈ F then p_q^d = 1 else p_q^d = Σ_{state s' of T̂} ρ(s') * p_{s'}^{d-1}
7     D̂ = ForwardReach(M, q0, d) return D̂
8 end
```

Example 5. *Consider the* MDP $Abs(\mathcal{H}, \mathcal{R})$ *shown in Figure 5.2, an initial state* (q_1, R), *a set of final states* $\{(q_4, R)\}$ *and* $\theta = 0.5$. *The* LDAG $\widehat{\mathcal{D}} \in Exec(Abs(\mathcal{H}, \mathcal{R}))$ *shown in Figure 5.3 is a counterexample, because* $Prob(\widehat{\mathcal{D}}, ((q_1, R), 0), \{(q_4, R)\} \times \mathbb{N}) = Prob(\sigma_1) + Prob(\sigma_2) + Prob(\sigma_3) = 0.25 + 0.25 + 0.125 = 0.625$, *which is greater than* 0.50, *where* σ_1, σ_2 *and* σ_3 *are paths given by* $\sigma_1 = ((q_1, R), 0) \to ((q_2, R), 1) \to ((q_4, R), 2)$, $\sigma_2 = ((q_1, R), 0) \to ((q_3, R), 1) \to ((q_4, R), 2)$ *and* $\sigma_3 = ((q_1, R), 0) \to ((q_2, R), 1) \to ((q_3, R), 2) \to ((q_4, R), 3)$ *in* $\widehat{\mathcal{D}}$.

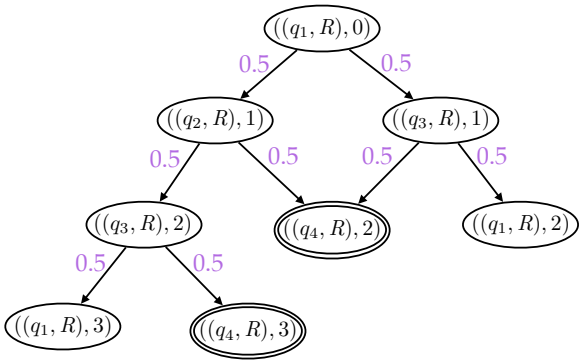

Figure 5.3: LDAG $\widehat{\mathcal{D}} \in Exec(Abs(\mathcal{H}, \mathcal{R}))$ violating $Prob_{sup}$

If $Abs(\mathcal{H}, \mathcal{R})$ is not probabilistically safe, then an abstract counterexample $\widehat{D}$ of $Abs(\mathcal{H}, \mathcal{R})$ can be computed. The abstract counterexample indicates only a potential violation of safety in the concrete system. Hence, we need to validate the counterexample $\widehat{D}$ to check if it is realizable in the concrete system $\mathcal{H}$. Next, we provide the details about the validation of a counterexample.

5.4.3 Validation

In this section, we provide a method to validate whether a counterexample expressed as an LDAG for the abstract system $Abs(\mathcal{H}, \mathcal{R})$ is realizable in the concrete system $\mathcal{H}$. Let us fix a finite LDAG $\widehat{\mathcal{D}} = (\mathcal{S}_{\widehat{\mathcal{D}}}, \longrightarrow_{\widehat{\mathcal{D}}})$ as a counter example of $Abs(\mathcal{H}, \mathcal{R})$ whose depth is d. Our validation problem is to check whether $\widehat{\mathcal{D}}$ is realizable in the concrete system $\mathcal{H}$. This is formally defined as below.

Problem 3 (Validation Problem)**.** *Verify whether there exists* $\mathcal{D} \in Exec([\![\mathcal{H}]\!])$ *such that* $\mathcal{D} \in \alpha_{\mathcal{R}}^{-1}(\widehat{\mathcal{D}})$.

Our broad approach for the validation of $\widehat{\mathcal{D}}$ is to compute all those concrete set of states associated with the abstract states from which sub-LDAG of $\widehat{\mathcal{D}}$ rooted at the abstract state is realizable in $\mathcal{H}$. We compute such sets from bottom to top layer, which is based on backward reachability analysis for which we define a function *Pre*.

Definition 20. *Given a set* S*, a finite set of sets* $\{S_i\}_{i \in A}$ *and* $\{p_i\}_{i \in A}$ *for some index set* A*, we define a function* $Pre_{[\![\mathcal{H}]\!]}(S, \{S_i\}_{i \in A}, \{p_i\}_{i \in A})$ *as*

$$\{s \in S \mid \exists\ \rho,\ s \longrightarrow_{[\![\mathcal{H}]\!]} \rho,\ \forall\ i \in\ A,\ \exists\ s_i \in S_i \text{ such that } \rho(s_i) = p_i\}.$$

We are going to compute a set of concrete states $\mathfrak{R}^k_{(q,R)}$ for each abstract state $((q, R), k)$ of $\widehat{\mathcal{D}}$, where $\mathfrak{R}^k_{(q,R)}$ denotes the set of all concrete states $(q, x) \in \{q\} \times R$ from which there exists an LDAG $\mathcal{D} \in Exec([\![\mathcal{H}]\!])$ such that the sub-LDAG $\mathcal{D}'$ of $\widehat{\mathcal{D}}$ rooted at $((q, R), k)$ is realizable by $\mathcal{D}$ in the concrete system $\mathcal{H}$. This is formally defined as follows.

Definition 21. *Given an abstract state* $((q,R),k) \in \mathcal{S}_{\widehat{\mathcal{D}}}$, *we have* $\mathfrak{R}^k_{(q,R)}$ *as* $\{(q,x) \in \{q\} \times R \mid \exists\ \mathcal{D} \in Exec([\![\mathcal{H}]\!])$ rooted at $((q,x),0)$ for which there exists a sub − LDAG $\mathcal{D}'$ of $\widehat{\mathcal{D}}$ rooted at $((\mathrm{q},\mathrm{R}),\mathrm{k})$ such that $\mathcal{D} \in \alpha^{-1}_{\mathcal{R}}(\mathcal{D}')\}$.

Alternatively, $\mathfrak{R}^k_{(q,R)}$ can be inductively defined as follows. Let $\mathcal{S}^k_{(q,R)}$ be the set of all concrete states associated with the abstract state $((q,R),k)$, that is, $\mathcal{S}^k_{(q,R)} = \alpha^{-1}_{\mathcal{R}}(q,R)$, and $\mathcal{Q}_k$ be the set of all those abstract states (q,R) at layer k in $\widehat{\mathcal{D}}$, that is, $((q,R),k) \in \mathcal{S}_{\widehat{\mathcal{D}}}$.

- ⋆ If $k = d$ or there is no abstract edge $((q,R),k) \longrightarrow_{\widehat{\mathcal{D}}} \rho$, then $\mathfrak{R}^k_{(q,R)}$ is $\mathcal{S}^k_{(q,R)}$;
- ⋆ If there is an abstract edge $((q,R),k) \longrightarrow_{\widehat{\mathcal{D}}} \rho$, then $\mathfrak{R}^k_{(q,R)}$ is

$$Pre_{[\![\mathcal{H}]\!]}(\mathcal{S}^k_{(q,R)}, \{\mathfrak{R}^{k+1}_{(q',R')}\}_{(q',R')\in\mathcal{Q}_{k+1}}, \{\rho((q',R'),k+1)\}_{(q',R')\in\mathcal{Q}_{k+1}}).$$

Example 6. *Consider the* LDAG $\widehat{\mathcal{D}}$ *shown in Figure 5.3. We compute the concrete set of states* $\mathcal{S}^k_{(q,R)}$ *and backward reach sets* $\mathfrak{R}^k_{(q,R)}$ *for each abstract state* (q,R) *at layer* k, *from the bottom to the top layer. The continuous part of* $\mathcal{S}^k_{(q,R)}$ *and* $\mathfrak{R}^k_{(q,R)}$ *for each abstract state of* $\widehat{\mathcal{D}}$ *are expressed in the lower and upper part of the circle, respectively, in Figure 5.4. The concrete sets of states are computed as* $\mathcal{S}^k_{(q,R)} = \alpha^{-1}_{\mathcal{R}}(q,R) = (q,R)$ *from* $\alpha_{\mathcal{R}}$ *given in Example 4. For example, for the abstract state* (q_3,R) *at layer* 1, $\mathcal{S}^1_{(q_3,R)} = (q_3,[0,4])$ *and its continuous part is* $[0,4]$ *which is written in lower part of its circle. Backward reach sets are computed according to its inductive definition. There are three cases. (a) The abstract state is located at the last layer of* $\widehat{\mathcal{D}}$. *For example, abstract state* (q_1,R) *is located at layer* 3 *which is the depth of* $\widehat{\mathcal{D}}$. *Hence* $\mathfrak{R}^3_{(q_1,R)} = \mathcal{S}^3_{(q_1,R)} = (q_1,[0,4])$. *(b) There is no probabilistic edge from the abstract state and it is not located at the last layer. For example, there is no probabilistic edge from* (q_4,R) *at layer* 2. *Hence,* $\mathfrak{R}^2_{(q_4,R)} = \mathcal{S}^2_{(q_4,R)} = (q_4,[0,4])$. *(c) All those abstract states from which there is a probablistic edge. For example, abstract state* (q_3,R) *at layer* 2 *has a probabilistic edge to* (q_1,R) *and* (q_4,R) *at layer* 3 *with probabilities* 0.5 *and* 0.5, *respectively. The backward reach set* $\mathfrak{R}^2_{(q_3,R)}$ *is* $Pre_{[\![\mathcal{H}]\!]}(\mathcal{S}^2_{(q_3,R)}, \{\mathcal{S}^3_{(q_1,R)}, \mathcal{S}^3_{(q_4,R)}\}, \{0.5,\ 0.5\})$.

From Definition 29, $Pre_{[\![\mathcal{H}]\!]}(\mathcal{S}^2_{(q_3,R)}, \{\mathcal{S}^3_{(q_1,R)}, \mathcal{S}^3_{(q_4,R)}\}, \{0.5, 0.5\})$ is computed as follows. First, for the edge from q_3 to q_1, we compute the set G_1 of continuous states which satisfy the guard and transition to (q_1, R). Since the guard constraint is $x \in [3, 4]$, $R = [0, 4]$ and the reset is identity, we obtain $G_1 = [3, 4]$. Similarly, the continuous states which can transition to q_4 is given by $G_2 = [3, 4]$. Hence, the continuous states from which the probabilistic edge can be taken are given by the set $G = G_1 \cap G_2 = [3, 4]$. The set of all continuous states in $[0, 4]$ which can reach G with rate $-1 \leq \dot{x} \leq 0$ is $[3, 4]$, since, all states in $[3, 4]$ can reach G by following the rate 0, and no states in $[0, 3)$ can reach G because there are no positive rates. Hence, $Pre_{[\![\mathcal{H}]\!]}(\mathcal{S}^2_{(q_3,R)}, \{\mathcal{S}^3_{(q_1,R)}, \mathcal{S}^3_{(q_4,R)}\}, \{0.5, 0.5\})$ is $(q_3, [3, 4])$ which is the set of states in $\mathcal{S}^2_{(q_3,R)}$ from which there is a concrete probabilistic edge to $\mathcal{S}^3_{(q_1,R)}$ and $\mathcal{S}^3_{(q_4,R)}$ with probabilities 0.5 each.

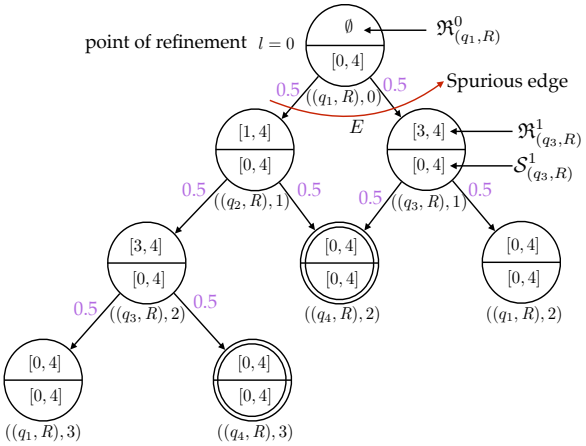

Figure 5.4: Point of refinement and spurious edge in $\widehat{\mathcal{D}}$ shown in Figure 5.3

The next proposition states that if any of the backward reach sets becomes empty there there is no concrete counterexample corresponding to the abstract counterexample.

Proposition 1. *$\alpha^{-1}_{\mathcal{R}}(\widehat{\mathcal{D}})$ is an empty set if and only if there exists k such that $\mathfrak{R}^k_{(q,R)} = \emptyset$ for some $((q, R), k) \in \mathcal{S}_{\widehat{\mathcal{D}}}$.*

Proof. If $\alpha^{-1}_{\mathcal{R}}(\widehat{\mathcal{D}})$ is an empty set, then $\mathfrak{R}^k_{(q,R)}$ is empty for $k = 0$. Suppose there exists k such that $\mathfrak{R}^k_{(q,R)} = \emptyset$ for some $((q, R), k) \in \mathcal{S}_{\widehat{\mathcal{D}}}$. This implies that $\mathfrak{R}^{k'}_{(q',R')}$ is also an empty

set for all those states $((q', R'), k')$ which are on the path from the root state of $\widehat{\mathcal{D}}$ to the state $((q, R), k)$. In particular, we have $\mathfrak{R}^0_{(q_0,R_0)} = \emptyset$, which by definition implies that $\alpha^{-1}_{\mathcal{R}}(\widehat{\mathcal{D}})$ is an empty set. □

If $\alpha^{-1}_{\mathcal{R}}(\widehat{\mathcal{D}})$ is not empty, then there is a valid counterexample $\mathcal{D} \in \mathit{Exec}([\![\mathcal{H}]\!])$ for the concrete system $\mathcal{H}$, that is, $\mathcal{H}$ is not probabilistically safe. However, if $\alpha^{-1}_{\mathcal{R}}(\widehat{\mathcal{D}})$ is empty, then $\widehat{\mathcal{D}}$ is a spurious counterexample. Hence, we use the information from the analysis of the counterexample $\widehat{\mathcal{D}}$ to refine the concrete system. We define the following which will be useful in defining the refinement.

Definition 22. *A point of refinement denoted as* $por_{Abs(\mathcal{H},\mathcal{R})}(\widehat{\mathcal{D}})$*, is defined as the largest layer* l *of* $\widehat{\mathcal{D}}$ *such that there is a state* $((q, R), l) \in \mathcal{S}_{\widehat{\mathcal{D}}}$ *where* $\mathfrak{R}^l_{(q,R)}$ *is an empty set. A probabilistic edge* $(((q, R), l'), \rho) \in \longrightarrow_{\widehat{\mathcal{D}}}$ *is spurious if* $\mathfrak{R}^{l'}_{(q,R)} = \emptyset$ *and* $l' = l$*.*

In Figure 5.4, the largest layer at which the backward reach set is empty in $l = 0$. Hence, the point of refinement is 0 and the spurious edge is $(((q_1, R), 0), \rho)$, where $\rho((q_2, R), 1) = \rho((q_3, R), 1) = 0.5$.

5.4.4 Refinement

In this section, we provide a method to refine the abstract system $\mathit{Abs}(\mathcal{H}, \mathcal{R})$ to eliminate the spurious counterexample $\widehat{\mathcal{D}}$. Let us fix a point of refinement l for $\widehat{\mathcal{D}}$ and a refinement of partition $\mathcal{R}$ denoted as $\mathcal{R}_1$. Consider a function mapping the refinement to the current abstraction, given by $\beta_{(\mathcal{R}_1,\mathcal{R})} : \mathcal{R}_1 \to \mathcal{R}$ where for all $(q, R) \in \mathcal{R}_1$, $(q, R) \subseteq \beta_{(\mathcal{R}_1,\mathcal{R})}(q, R)$. We need to eliminate the spurious counterexample from the abstract system $\mathit{Abs}(\mathcal{H}, \mathcal{R}_1)$. Hence, we define a progressive refinement which ensures that the refinement makes progress towards eliminating the counterexample $\widehat{\mathcal{D}}$. It essentially captures the intuition that every abstract counterexample in the refinement that corresponds to $\widehat{\mathcal{D}}$ has a point of refinement which is lower in the LDAG.

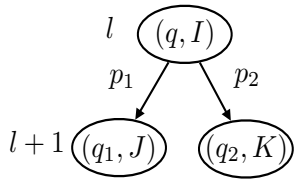

Figure 5.5: Spurious Edge

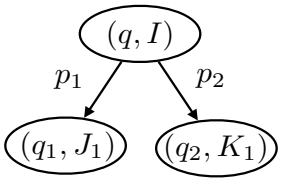

Figure 5.6: Edge 1

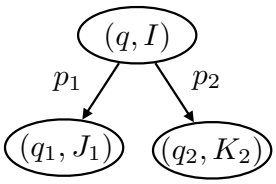

Figure 5.7: Edge 2

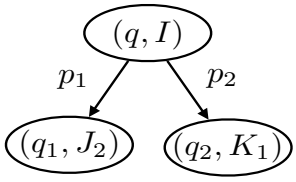

Figure 5.8: Edge 3

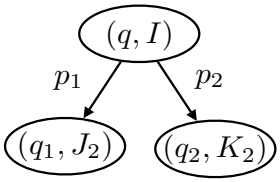

Figure 5.9: Edge 4

Definition 23 (Progressive Refinement)**.** *A refinement $\mathcal{R}_1$ of $\mathcal{R}$ is a progressive refinement if we have*

$$por_{Abs(\mathcal{H},\mathcal{R}_1)}(\mathcal{D}) > por_{Abs(\mathcal{H},\mathcal{R})}(\widehat{D}), \text{ for all } \mathcal{D} \in \beta^{-1}_{(\mathcal{R}_1,\mathcal{R})}(\widehat{\mathcal{D}}).$$

Our objective is to find a progressive refinement $\mathcal{R}_1$ of $\mathcal{R}$. Since the main reason for a probabilistic edge $((q, R), l), \rho)$ to be spurious is the emptiness of the backward reach set $\mathcal{R}^l_{(q,R)}$, our broad approach is to find a splitting of the nodes in the $l+1$ layer that eliminate the spuriousness at layer $l+1$. This is achieved by splitting the nodes in layer l to separate the post of $\mathcal{S}^l_{(q,R)}$ from the backward reach sets of layer $l+1$. Hence, we define a function *Post*, which is along the lines of *Pre*.

Definition 24. *Given a set $\mathcal{S}$, a tuple $(p_1, p_2, \ldots, p_n)$, we define a function $Post_{[\![\mathcal{H}]\!]}(\mathcal{S}, (p_1, \ldots, p_n))$ as $\{(s_1, s_2, \ldots, s_n) \mid \exists\ s \in \mathcal{S},\ (s, \rho) \in \longrightarrow_{[\![\mathcal{H}]\!]}$ such that $\rho(s_i) = p_i$ for all $i \in [n]\}$.*

For any spurious edge $(((q, R), l), \rho)$, backward reach set $\mathfrak{R}^l_{(q,R)} = \emptyset$, that is, $Pre_{[\![\mathcal{H}]\!]}(\mathcal{S}^l_{(q,R)}, \{\mathfrak{R}^{l+1}_{(q',R')}\}_{(q',R')\in\mathcal{Q}_{l+1}}, (p_1, \ldots, p_n)) = \emptyset$, where $(p_1, \ldots, p_n) \in \underset{(q',R')\in\mathcal{Q}_{l+1}}{\times} \{\rho((q', R'), l+1)\}$ for some $n \in \mathbb{N}$. From the definition of *Post* function as given in Definition 24, $Post_{[\![\mathcal{H}]\!]}(\mathcal{S}^l_{(q,R)}, (p_1, \ldots, p_n))$ intersected with the tuple of concrete states corresponding to the abstract states in layer $l+1$ is not empty, because there exists an abstract probabilistic edge corresponding to the spurious edge. Also, we obtain that no common element in the post of the concrete

set of states of the abstract state (q, R) at layer l with respect to the spurious edge and Cartesian product of backward reach sets of all target abstract states (q', R') at layer $l+1$ of the spurious edge. This is formally stated in the following proposition.

Proposition 2. *Given a probabilistic edge* $(((q, R), l), \rho) \in \longrightarrow_{\widehat{\mathcal{D}}}$ *(spurious) and* $(p_1, \ldots, p_n) \in \underset{(q',R')\in\mathcal{Q}_{l+1}}{\times} \{\rho((q', R'), l+1)\}$ *for some* $n \in \mathbb{N}$*, we have*

$$Post_{[\![\mathcal{H}]\!]}(\mathcal{S}^l_{(q,R)}, (p_1, \ldots, p_n)) \cap \underset{(q',R')\in\mathcal{Q}_{l+1}}{\times} \mathfrak{R}^{l+1}_{(q',R')} = \emptyset. \tag{5.1}$$

It might be possible that intersection between the projection of the post for an abstract state and backward reach set for the same abstract state is not empty. This is formally written as a remark.

Remark 5. *Given a probabilistic edge* $(((q, R), l), \rho) \in \longrightarrow_{\widehat{\mathcal{D}}}$ *(spurious), it is not necessary that the following statement is true. Let* $(p_1, \ldots, p_n) \in \underset{(q',R')\in\mathcal{Q}_{l+1}}{\times} \{\rho((q', R'), l+1)\}$ *for some* $n \in \mathbb{N}$*.*

$$Proj(Post_{[\![\mathcal{H}]\!]}(\mathcal{S}^l_{(q,R)}, (p_1, \ldots, p_n)), ((q'', R''), l+1)) \cap \mathfrak{R}^{l+1}_{(q'',R'')}$$

need not be empty for all $(q'', R'') \in \mathcal{Q}_{l+1}$*.*

Example 7. *We illustrate Remark 5 using a counterexample. Consider the spurious edge* E *shown in Figure 5.4. The post of the abstract state* $(q_1, [0, 4])$ *is* $\mathcal{B} = (q_2, [0, 2]) \times (q_3, [0, 2])$*. From Figure 5.4, we have* $\mathfrak{R}^1_{(q_2,R)} = (q_2, [1, 4])$ *and* $\mathfrak{R}^1_{(q_3,R)} = (q_3, [3, 4])$*. This implies that* $\mathcal{B} \cap (\mathfrak{R}^1_{(q_2,R)} \times \mathfrak{R}^1_{(q_3,R)}) = \emptyset$*. Next, the projection of* $\mathcal{B}$ *for the abstract state* (q_2, R) *and* (q_3, R) *at layer* 1 *are* $Proj(\mathcal{B}, ((q_2, R), 1)) = (q_2, [0, 2])$ *and* $Proj(\mathcal{B}, ((q_3, R), 1)) = (q_3, [0, 2])$*, respectively. Note that although* $Proj(\mathcal{B}, ((q_3, R), 1)) \cap \mathfrak{R}^1_{(q_3,R)} = \emptyset$*,* $Proj(\mathcal{B}, ((q_2, R), 1)) \cap \mathfrak{R}^1_{(q_2,R)} \neq \emptyset$*.*

Next, we provide the details about the separation. Let us consider a spurious edge E with the point of refinement l shown in Figure 5.5 where the probability of transitions to

abstract states (q_1, J) and (q_2, K) are p_1 and p_2, respectively. Let $\mathcal{S}^{l}_{(q,I)}$, $\mathcal{S}^{l+1}_{(q_1,J)}$ and $\mathcal{S}^{l+1}_{(q_2,K)}$ be concrete set of states for the abstract states (q, I), (q_1, J) and (q_2, K), respectively, and $\mathfrak{R}^{l}_{(q,I)}$, $\mathfrak{R}^{l+1}_{(q_1,J)}$ and $\mathfrak{R}^{l+1}_{(q_2,K)}$ be the backward reach set for the abstract states (q, I), (q_1, J) and (q_2, K), respectively. Let X be the post of $\mathcal{S}^{l}_{(q,I)}$, and $Y = \mathfrak{R}^{l+1}_{(q_1,J)} \times \mathfrak{R}^{l+1}_{(q_2,K)}$. Let X_1 and X_2 be the projection of X for the abstract states (q_1, J) and (q_2, K), respectively, and Y_1 and Y_2 be the projection of Y for the abstract states (q_1, J) and (q_2, K) at layer $l+1$, respectively, that is, $Y_1 = \mathfrak{R}^{l+1}_{(q_1,J)}$ and $Y_2 = \mathfrak{R}^{l+1}_{(q_2,K)}$.

Case (a) When $X_i \cap Y_i = \emptyset$ for $i = 1, 2$, X_1, Y_1 and X_2, Y_2 need to be separated by the partition of $\mathcal{S}^{l+1}_{(q_1,J)}$ and $\mathcal{S}^{l+1}_{(q_2,J)}$, respectively. Next, we show that such partition eliminates the spurious edge. Let $\mathcal{S}^{l+1}_{(q_1,J)}$ and $\mathcal{S}^{l+1}_{(q_2,K)}$ be partitioned into $\mathcal{S}^{l+1}_{(q_1,J_1)}$, $\mathcal{S}^{l+1}_{(q_1,J_2)}$ and $\mathcal{S}^{l+1}_{(q_2,K_1)}$, $\mathcal{S}^{l+1}_{(q_2,K_2)}$, respectively, such that $X_1 \subseteq \mathcal{S}^{l+1}_{(q_1,J_1)}$, $Y_1 \subseteq \mathcal{S}^{l+1}_{(q_1,J_2)}$ and $X_2 \subseteq \mathcal{S}^{l+1}_{(q_2,K_1)}$ and $Y_2 \subseteq \mathcal{S}^{l+1}_{(q_2,K_2)}$. There are four potential probabilistic edges corresponds to the spurious edge shown in Figures 5.6, 5.7, 5.8 and 5.9.

(1) Edge shown in Figure 5.6 exists in the refined system because $X \cap (\mathcal{S}^{l+1}_{(q_1,J_1)} \times \mathcal{S}^{l+1}_{(q_2,K_1)}) \neq \emptyset$. However, the point of refinement must be $l+1$ because backward reach set $\mathfrak{R}^{l+1}_{(q_1,J_1)} = \emptyset$ because $\mathfrak{R}^{l+1}_{(q_1,J_1)} \subseteq \mathcal{S}^{l+1}_{(q_1,J_1)}$, $\mathfrak{R}^{l+1}_{(q_1,J_1)} \subseteq Y_1$ and $\mathcal{S}^{l+1}_{(q_1,J_1)} \cap Y_1 = \emptyset$.

(2) Edge shown in Figure 5.7 does not exist in the refined system because $X_2 \cap \mathcal{S}^{l+1}_{(q_2,K_2)} = \emptyset$.

(3) Edge shown in Figure 5.8 does not exist in the refined system because $X_1 \cap \mathcal{S}^{l+1}_{(q_1,J_2)} = \emptyset$.

(4) Edge shown in Figure 5.9 does not exist in the refined system because $X_1 \cap \mathcal{S}^{l+1}_{(q_1,J_1)} = \emptyset$ and $X_2 \cap \mathcal{S}^{l+1}_{(q_2,K_2)} = \emptyset$.

Hence, such separation guarantees that the spurious edge gets eliminated.

Case (b) When either $X_1 \cap Y_1 \neq \emptyset$ or $X_2 \cap Y_2 \neq \emptyset$ is true, $X_1 \setminus Y_1$, Y_1 and $X_2 \setminus Y_2$, Y_2 need to be separated by the partition of $\mathcal{S}^{l+1}_{(q_1,J)}$ and $\mathcal{S}^{l+1}_{(q_2,J)}$, respectively. Next, we show that spurious edge gets eliminated by such partition. Assume that $X_1 \cap Y_1 \neq \emptyset$. Note that for a spurious edge, there must be at least one target abstract state such that intersection

between the projection of post with respect to the target abstract state and its backward reach set is empty. This implies that $X_2 \cap Y_2 = \emptyset$.

(a) Edge shown in Figure 5.6 exists in the refined system, but the point of refinement must be $l+1$ due to the same reason given in case (a)(1).

(b) Edges shown in Figures 5.7, 5.9 do not exist in the refined system due to the same reason given for case (a)(2).

(c) Edge shown in Figure 5.8 exist in the refined system but the point of refinement must be $l+1$ due to the same reason given in case (a)(1).

From case (a) and case (b), we have identified that $X_i \setminus Y_i$ and Y_i, $i = 1, 2$ needs to be separated. Next, we provide strategies for partitioning the concrete states associated with the abstract states such that there identified sets must be separated by the partition.

Definition 25. *Let $(((q, R), l), \rho)$ be a probabilistic edge (spurious) and Q_{l+1} be a set of all abstract states which are at layer $l+1$. A refinement $\mathcal{R}_1$ eliminates a spurious edge $(((q, R), l), \rho)$ if for each $(q', R') \in \mathcal{Q}_{l+1}$, there is no partition element $(q', R_1) \in \mathcal{R}_1$ such that the following two conditions hold: let $(p_1, \ldots, p_n) \in \underset{(q'', R'') \in \mathcal{Q}_{l+1}}{\times} \{\rho((q'', R''), l+1)\}$ for some $n \in \mathbb{N}$.*

C1: $R_1 \cap (Proj(Post_{[\![\mathcal{H}]\!]}(\mathcal{S}^l_{(q,R)}, (p_1, \ldots, p_n)), ((q', R'), l+1)) \setminus \mathfrak{R}^{l+1}_{(q',R')}) \neq \emptyset$;

C2: $R_1 \cap \mathfrak{R}^{l+1}_{(q',R')} \neq \emptyset$.

The strategies in Definition 25 are basically restrictions on the partition of the concrete set of states corresponding to the abstract states which make sure that the identified sets are not shared by the same partition element. Since the spurious edge gets eliminated by such a partitioning, the progress of the refinement is guaranteed. We formally state it in the following theorem.

Theorem 3. *Given a refinement $\mathcal{R}_1$ of $\mathcal{R}$, if $\mathcal{R}_1$ eliminates the spurious edge, then $\mathcal{R}_1$ is a progressive refinement.*

Any such splitting as given by Definition 25 will eliminate the counterexample. Note that Definition 25 requires that there is no partition element which for any location q intersects with both the projection of *Post* to q and the reach set $\mathfrak{R}$ corresponding to q. Hence, for every partition element which intersects with both, we need a strategy to separate the two sets, say, S_1 and S_2. One possible way to have such splitting can be obtained by Algorithm 2. Algorithm 2 aims to partition a polyhedral set $\mathcal{S}$ for given two polyhedral sets $\mathcal{S}_1 \subseteq \mathcal{S}$ and $\mathcal{S}_2 \subseteq \mathcal{S}$ such that $\mathcal{S}_1 \setminus \mathcal{S}_2$ and $\mathcal{S}_2$ does not share the same partition element of the partition of $\mathcal{S}$.

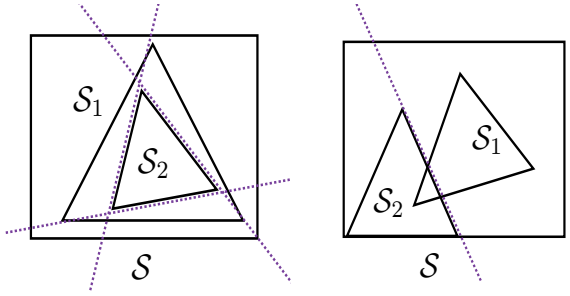

Figure 5.10: Illustration of Algorithm 2

Algorithm 2: Refine a partition $\mathcal{P}$ to separate $\mathcal{S}_1 \setminus \mathcal{S}_2$ and $\mathcal{S}_2$

Input: $\mathcal{P}$ - a partition, $\mathcal{S}_1 \subseteq \mathcal{S}$ - a polyhedral set, $\mathcal{S}_2 \subseteq \mathcal{S}$ - a polyhedral set

Output: $\mathcal{P}$ - a partition of $\mathcal{S}$ that separates $\mathcal{S}_1 \setminus \mathcal{S}_2$ and $\mathcal{S}_2$

```
1 SetKwKwGoTogo to  begin
2     Let S'_1 = S_1 \ S_2
3     while ∃ P ∈ P such that P ∩ S_1 ≠ ∅ and P ∩ S'_1 ≠ ∅ do
4         if ∃ c ∈ Cons(S_2) such that [[c]] ∩ S'_1 ≠ ∅ then
5             P = P \ {P}
6             P = P ∪ {P ∩ [[c]]}
7             P = P ∪ {P ∩ complement([[c]])}
8     return P
9 end
```

Algorithm 2 checks whether there is a partition element P in the current partition $\mathcal{P}$ which overlaps with both $\mathcal{S}_1$ and $\mathcal{S}_1 \setminus \mathcal{S}_2$ at line 3. If so, then we check if there exists a

constraint $c \in Cons(\mathcal{S}_2)$ such that $\mathcal{S}_1 \setminus \mathcal{S}_2$ overlaps with the complement of the set $[\![c]\!]$, then we split P into two polyhedral sets $P \cap [\![c]\!]$ and $P \cap \overline{[\![c]\!]}$ and add them into $\mathcal{P}$ at lines 5-7. We repeat 4-7 until we come up $\mathcal{P}$ such that no partition element of $\mathcal{P}$ overlaps with both $\mathcal{S}_1$ and $\mathcal{S}_1 \setminus \mathcal{S}_2$. We have illustrated Algorithm 2 in Figure 5.10. In the left picture of Figure 5.10, all the constraints of $\mathcal{S}_2$ are required to separate $\mathcal{S}_2$ and $\mathcal{S}_1 \setminus \mathcal{S}_2$ and it results in 7 partition elements of $\mathcal{S}$. However, in the right picture of Figure 5.10, one constraint is enough to separate $\mathcal{S}_2$ and $\mathcal{S}_1 \setminus \mathcal{S}_2$, and it results in two partition elements.

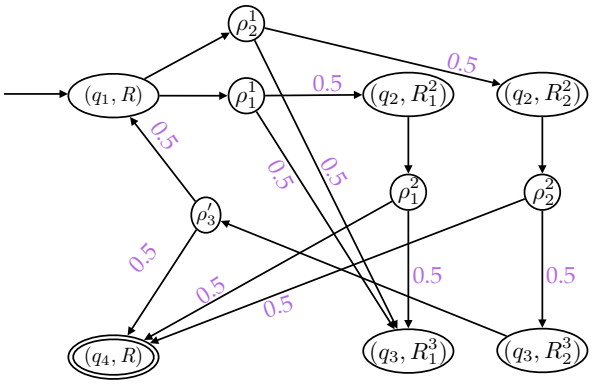

Figure 5.11: Markov Decision Process after Refinement

Example 8. *We illustrate the separation based on the spurious edge E shown in Figure 5.4. From Examples 6, 7, we have* $\mathcal{S}^1_{(q_2,R)} = (q_2, [0,4])$, $\mathcal{S}^1_{(q_3,R)} = (q_3, [0,4])$, $\mathfrak{R}^1_{(q_2,R)} = (q_2, [1,4])$, $\mathfrak{R}^1_{(q_3,R)} = (q_2, [3,4])$, *and the post of the abstract state* $(q_1, [0,4])$ *is* $\mathcal{B} = (q_2, [0,2]) \times (q_3, [0,2])$. *Let* $\mathcal{B}_{(q_2,R)}$, $\mathcal{B}_{(q_3,R)}$ *be the projection of* $\mathcal{B}$ *for the abstract states* (q_2, R), (q_3, R), *respectively, at layer* 1. *Then, we use Algorithm 2 to partition* $\mathcal{S}_{(q_2,R)}$ *and* $\mathcal{S}_{(q_3,R)}$. *We obtain* $\{(q_2, R_1^2), (q_2, R_2^2)\}$ *as a partition of* $\mathcal{S}_{(q_2,R)}$, *where* $R_1^2 = [0,1]$, $R_2^2 = [1,4]$. *Similarly, we get* $\{(q_3, R_1^3), (q_3, R_2^3)\}$ *as a partition of* $\mathcal{S}_{(q_3,R)}$, *where* $R_1^3 = [0,3)$, $R_2^3 = [3,4]$. *Thus, we obtain a refinement of the partition* $\mathcal{R}$, *which is* $\mathcal{R}_1 = \{(q_1, R), (q_2, R_1^2), (q_2, R_2^2), (q_3, R_1^3), (q_3, R_2^3), (q_4, R)\}$. *Next, we use* $\mathcal{R}_1$ *for refining the abstraction and we obtain a Markov Decision Process as shown in Figure 5.11. The probability of reaching final state* (q_4, R) *from initial state* (q_1, R) *is less than* 0.50. *Hence, polyhedral* PHS *shown in Figure 5.1 is probabilistically safe with*

respect to the initial location q_0*, a set of final locations* $\mathbb{F} = \{q_4\}$*, and probability bound* $\theta = 0.50$*.*

5.5 Computability

In this section, we summarize the CEGAR algorithm and discuss computability and complexity issues.

CEGAR Algorithm

Algorithm 3 presents the CEGAR framework to check whether a given polyhedral PHS $\mathcal{H}$ is the maximum probabilistically safe with respect to an initial location q_0, a set of final locations $\mathbb{F}$, and a probability bound θ.

The function Abstraction takes as input the system $\mathcal{H}$, partition $\mathcal{R}$, the initial state q_0 and the set of final locations $\mathbb{F}$ and returns the MDP abstraction $\mathcal{M} = Abs(\mathcal{H}, \mathcal{R})$ along with the set of abstract states S corresponding to the concrete states $\{q_0\} \times Inv(q_0)$ and the set of abstract states $\mathbb{F}'$ corresponding to the concrete states $\mathbb{F} \times \mathcal{X}$. The function Model-Checking takes the abstract MDP $\mathcal{M}$, abstract initial states S and final state $\mathbb{F}'$ as input and either returns a $Status = \top$, if the abstract system satisfies the property and $\bot$ otherwise. In the latter case, it also returns a counterexample $\widehat{\mathcal{T}}$ in the form of a DTMC, which is then passed on to the function Get_LDAG to extract the LDAG $\widehat{\mathcal{D}}$. The function Validation function validates the counterexample $\widehat{\mathcal{D}}$ using the backward reachability algorithm and checking if $\mathfrak{R}^k_{(q,R)} = \emptyset$ for some $((q, R), k) \in \mathcal{S}_{\widehat{\mathcal{D}}}$. $valid = \top$ if the above condition is satisfied in which case a concrete counterexample exists, otherwise, $\widehat{\mathcal{D}}$ is a spurious abstract counterexample, in which case some spurious information SI is returned. It first computes the spurious edge $SE = (((q, R), l), \rho) \in \longrightarrow_{\widehat{\mathcal{D}}}$ where l is the largest number satisfying $\mathfrak{R}^l_{(q,R)} = \emptyset$ for some (q, R) and ρ. Then, SI consists of three sets S_1, S_2 and S_3, where $S_1 = \underset{((q',R'),l+1) \in support(\rho)}{\times} \mathcal{S}^{l+1}_{(q',R')}$, $S_2 = Post(S^l_{(q,R)}, p_1, p_2, \ldots, p_n)$, where $p_i = \rho((q_i, R_i), l+1)$, and $S_3 = \underset{((q',R'),l+1) \in support(\rho)}{\times} \mathfrak{R}^{l+1}_{(q',R')}$.

Algorithm 3: CEGAR Algorithm

Input: $\mathcal{H}$ - polyhedral PHS, q_0 - initial location, $\mathbb{F}$ - a set of final locations, θ - probability bound, $\mathcal{R}$ - partition of the state space

Output: $\top/\bot$ based on satisfaction/rejection of the specification with respect to q_0, $\mathbb{F}$, θ

```
1  begin
2      Sat = None
3      while Sat = None do
4          M, S, F' = Abstraction(H, R, q_0, F)
5          T̂, Status = Modelchecking(M, s, F', θ)
6          if Status = ⊤ then
7              Sat = ⊤
8              break
9          else
10             D̂ = Get_LDAG(T̂, F',S, θ)
11             SI, valid = Validation(H, D̂, R)
12             if valid = ⊤ then
13                 Sat = ⊥
14                 break
15             else
16                 R_1 = Refinement(R, SI)
17                 R = R_1
18 end
```

Refinement($\mathcal{R}$, SI) runs Algorithm 2 for partitioning $Proj(S_1, ((q, R), l{+}1))$ for each abstract state $((q, R), l{+}1) \in \mathit{support}(\rho)$ with respect to $Proj(S_2, ((q, R), l{+}1)$, $Proj(S_3, ((q, R), l{+}1)$. Then, for each for each abstract state $((q, R), l + 1) \in \mathit{support}(\rho)$, the partition element $Proj(S_1, ((q, R), l{+}1)$ of $\mathcal{R}$ is replaced by all its partition elements obtained from Algorithm 2. The CEGAR loop is repeated until the specification is refuted or satisfied, in general, it is not guaranteed to terminate. Hence, it is a semi-decision procedure.

Complexity Analysis

In this section, we discuss the computability and complexity of the different steps in the CEGAR algorithm. The crux of Algorithm 3 is the computation of *Pre* and *Post* functions,

corresponding to computing the one step backward and forward reach sets. When the dynamics is polyhedral inclusion $\dot{x} \in P$ for a polyhedral set P, the *Pre* and *Post* with respect to a polyhedral set of states S, results in a polyhedral set. This can be computed by writing a finite set of linear constraints that the states before and after a transition need to satisfy, and eliminating one of them (depending on whether we are computing *Pre* or *Post*). This corresponds to computing intersections, checking satisfiability and variable elimination in a set of linear constraints. The first two operations can be performed efficiently (polynomial in the size of the constraints), while variable elimination is more expensive and can be performed using Fourier−Motzkin elimination whose complexity is $O(4(\frac{n_c}{4})^{2^{n_v}})$, where n_c is the number of linear constraints in a polyhedral set and n_v is the number of variables that need to be eliminated. It also leads to a exponential blow-up in the number of constraints with respect to the number of eliminated variables. In the rest of the section, we present the complexity analysis assuming a cost of C for each *Pre* and *Post* computation. We realize that C depends on the size of the representation of the intermediate reach sets which can themselves grow exponentially as the algorithm proceeds, however, we would like to characterize the behavior of the rest of the algorithm.

We note that for more general dynamics, the reach set is not a polyhedral set and could require exponential functions for representation, for instance, for linear dynamics such as $\dot{x} = Ax$ or other non-linear dynamics. Hence, though the CEGAR framework in Algorithm 3 can be applied, one will need to resort to over-approximate computations of *Pre* and *Post* for abstraction construction and deducing spuriousness of the counterexample.

Next, we discuss the complexity of each step presented in Algorithm 3. Let us consider an $n-$dimensional polyhedral PHS $\mathcal{H}$, and a partition $\mathcal{R}$ of its state space $Q \times \mathcal{X}$. Let $|\mathcal{R}| = k_1 \times k_2$ where $k_1 = |\mathcal{Q}|$ and k_2 is the number of distinct partition elements of $\mathcal{X}$.

In the abstraction, each abstract edge has a source of the form (q, R) and a set of target states of the form $(q_1, R_1), \cdots, (q_{k_1}, R_{k_1})$. We need to check if each such potential abstract edge corresponds to a concrete edge, which can be computed by one *Pre* operation followed

by checking emptiness. Hence, the cost of abstraction is given by $O(Ck_2^{k_1+1})$, and size (representation) of the output MDP $\mathcal{M}$ is $O(k_2^{k_1+1})$. The complexity of Model-Checking function is polynomial in the size of the MDP $\mathcal{M}$. The GET_LDAG algorithm runs until enough paths are found. There is a priori no bound on the length of paths explored, however, it is guaranteed to terminate and the time complexity is polynomial in d, which is the length of the unrolling. Then the complexity of validation and computation of SI is $O(DC)$, since the validation requires one *Pre* computation and one emptiness checking operation (which is cheaper than *Pre* computation) corresponding to every node in $\widehat{\mathcal{D}}$. The Refinement function calls the partitioning algorithm at most once for every location. The complexity of the partitioning algorithms is $O(n_c n_p)$, where n_c number of constraints in the representation of $\mathfrak{R}$ sets, and n_p is the number of partition elements. However, the number of elements of the refined partition could be exponential in n_c.

5.6 Experimental analysis

In this section, we provide the details of the implementation and experimental analysis. We have implemented the CEGAR framework for polyhedral PHS in a Python toolbox called Procegar. We use Parma Polyhedra Library (PPL)[146] to compute the abstract MDP, and PRISM[35] model checker to verify the probabilistic safety specification on the abstract system. PRISM generates a counterexample in the form of a DTMC. We unroll the DTMC and convert it into an LDAG which we validate with respect to the polyhedral PHS using PPL[146]. The refinement module generates a new partition of the state space that eliminates the counterexample, and it is implemented using PPL[146]. We compare our experimental evaluation of probabilistic safety analysis with tool ProHVer[131], which has been performed on Ubuntu 12.04 OS, Intel R©Pentium(R) CPU B960 with 2.20GHz× 2 Processor and 2GB RAM.

We consider two examples, one corresponding to a grid world, and the other an oscillator-

filter. For the grid world example, we construct grids with $n \times n$ cells, where we have polyhedral dynamics $\dot{x} - \dot{y} <= 0$, $2\dot{x} - \dot{y} >= 0$ in each cell. A probabilistic edge is defined from a cell to two of its adjacent cells with equal probability. here, we consider the specification, where we choose cell $(1, 1)$ as an initial location with continuous states $[0, 0.5] \times [0, 0.5]$, and cell $(2, 2)$ as a final location with continuous states $[1.5, 2] \times [1, 1.5]$. Next, for the 2-dimensional oscillator and m-filter benchmark, we modify the deterministic version of the oscillator-filter benchmark[147] by introducing probabilities and additional reset constraints. Note that the deterministic version of the benchmark is a linear PHS. So, we first convert the benchmark into polyhedral PHS by transforming the linear dynamics into polyhedral dynamics by hybridization over the invariant set. In addition, we consider the state space for each variable as the interval $[-3, 3]$. For the specification, we consider $(l_1, \{-0.5 \leq x \leq 0,\ 0 \leq y \leq 0.35\})$ as the initial set of states and $(l_4, \{0 \leq x \leq 0.5, -0.35 \leq y \leq 0\})$ as the final set of states. For all the experiments, we are interested in verifying $\max_{s \in \mathbb{I}} \mathcal{P}_{sup}([\![\mathcal{H}]\!], s, \mathbb{F}) \leq \theta$ for different values of probability bound θ, where $\mathbb{I}$ is the set of initial states and $\mathbb{F}$ is the set of final states.

We perform experiments using tools Procegar and ProHVer[131] and compare the experimental results. For the probabilistic safety analysis using tool ProHVer[131], we first compute the maximum probability of reaching a set of final states from a set of initial states and check whether the maximum probability of reachability is less than or equal to θ.

			Procegar						ProHVer		
Rows	*Grid Size*	θ	*Size(init)*	*Size(final)*	*Status*	T_{A+CE+V}(*Sec.*)	T_{ver}(*Sec.*)	P_T(*Sec.*)	*Pr*	*Status*	P_T(*Sec.*)
1	2×2	0.25	(4,3)	(9,6)	$\bot$	6.56	3.13	9.69	1	*U*	0.03
2		0.50	(4,3)	(9,6)	$\top$	3.31	3.08	6.39	1	*U*	0.03
3	4×4	0.10	(16,15)	(23,19)	$\bot$	6.20	3.18	9.38	0.25	*U*	0.057
4		0.25	(16,15)	(23,19)	$\top$	3.17	3.08	6.25	0.25	$\top$	0.057
5	6×6	0.10	(36,35)	(43,39)	$\bot$	6.32	3.07	9.39	0.25	*U*	0.085
6		0.25	(36,35)	(43,39)	$\top$	3.20	2.96	6.16	0.25	$\top$	0.085
7	8×8	0.10	(64,63)	(71,67)	$\bot$	6.30	3.17	9.47	0.25	*U*	0.122
8		0.25	(64,63)	(71,67)	$\top$	3.26	2.94	6.2	0.25	$\top$	0.122

Table 5.1: Verification results for the Grid World ($n = 2$, $K = 2$)

In Table 5.1 and Table 5.2, the results of analysis of the grid world and the oscillator-filter are summarized, respectively. In the tables, θ shows the probability bound; *Size(init)*,

Size(final) show the size of initial and final abstract system which are expressed as a pair ($|V|$, $|E|$), where $|V|$, $|E|$ are number of vertices and edges, respectively. *Status* represents whether a given specification is satisfied ($\top$), non-satisfied ($\bot$) or the verification result is inconclusive(U). T_{A+CE+V} *shows the total time (among all iterations) taken for the abstraction, counterexample simplification, and validation steps;* T_{ver} *represents the total time (among all iterations) for checking the property by PRISM and* P_T *represents the total time taken for the safety verification. All times are measured in seconds.* *Pr* represents the value of the maximum probability of reachability for a given specification in ProHVer[131]. *dim*, K represent dimension of the system and number of iterations in CEGAR, respectively. m shows the number of filters for 2-dimensional oscillator and m-filter benchmark.

			Procegar						ProHVer		
m	*dim*	θ	*Size(init)*	*Size(final)*	*Status*	$T_{A+CE+V}(Sec.)$	$T_{ver}(Sec.)$	$P_T(Sec.)$	*Pr*	*Status*	$P_T(Sec.)$
1	3	0.10	(4,3)	(4,3)	$\bot$	3.25	1.51	4.76	0.73	*U*	11.66
		0.20	(4,3)	(4,3)	$\top$	0.01	1.45	1.46	0.73	*U*	11.66
2	4	0.10	(4,3)	(4,3)	$\bot$	4.70	1.46	6.16	0.73	*U*	711.96
		0.20	(4,3)	(4,3)	$\top$	0.01	1.48	1.49	0.73	*U*	711.96
3	5	0.10	(4,3)	(4,3)	$\bot$	36.11	1.65	37.76	0.73	*U*	4385.63
		0.20	(4,3)	(4,3)	$\top$	0.01	1.55	1.56	0.73	*U*	4385.63
4	6	0.10	(4,3)	(4,3)	$\bot$	1273.30	1.55	1274.85	—	—	*TO*
		0.20	(4,3)	(4,3)	$\top$	0.01	1.88	1.89	—	—	*TO*

Table 5.2: Verification results for the oscillator-filter (K=1)

We observe that in Table 5.1, Procegar concludes safety of the specification for different values of θ for all grid sizes. Procegar takes little more time in comparison to ProHVer; however, ProHVer[131] is unable to conclude safety in Rows $1, 2, 3, 5$, and 7.

In Table 5.2, Procegar takes less time in comparison to ProHver[131]. ProHVer[131] is unable to conclude safety of the specification for $m = 1, 2, 3$. ProHVer[131] did not terminate for $m = 4$ within a timeout of 75 minutes. Also, for Procegar, the total verification time grows with respect to the dimension of the system when the specification is false. It is expected because a finite number of validation operation needs to be performed, which is based on backward reach sets, which are obtained using computationally expensive quantifier elimination procedure, specifically, for large dimensional systems.

Overall, Procegar is able to find proofs in many more instances and with less time as compared to ProHVer for three or more dimensional systems, which does not use any clever refinement techniques. There is no extensive set of benchmarks for probabilistic hybrid systems so we have modified some of the existing benchmarks in the non-probabilistic setting.

Chapter 6

Safety analysis of linear probabilistic hybrid systems

In this chapter, we consider the problem of unbounded safety analysis of linear probabilistic hybrid systems (PHS) which model discrete, continuous and probabilistic behaviors. The discrete and probabilistic dynamics are captured using finite state Markov decision processes (MDP), and the continuous dynamics are modeled by annotating the states of the MDP with linear differential equations. Finite state abstractions of discrete-time and continuous time PHS have been investigated[51;52]. One of the challenges with constructing finite-state abstractions of continuous dynamics is efficiently determining the existence of a probabilistic edge in the abstract system. It requires performing a continuous post-computation as in the non-probabilistic case, but in addition requires, computing for multiple discrete posts simultaneously corresponding to a probabilistic edge. Post computation is challenging even for linear dynamics, and hence, instead of directly computing post, we first apply hybridization to simplify the dynamics to polyhedral inclusion dynamics. To address the computation of multiple discrete posts, we encode the existence of a probabilistic edge as a set of linear constraints in GNU Linear Programming Kit (GLPK). This is feasible because of the dynamics in polyhedral inclusion. Hence, our abstraction consists of two steps: hybridization to con-

struct a PHS with polyhedral inclusion dynamics and predicate abstraction to construct a finite state MDP. We show that the abstractions are sound, and by adding more predicates, we can obtain more precise abstractions.

We have implemented the abstraction procedure in Python. We show that the abstractions are sound, and by adding more predicates, we can obtain more precise abstractions. We refer to this two step abstraction procedure as the hierarchical abstraction. We perform experimental comparison with ProHVer and perform experimental evaluation of the computational complexity of the abstraction procedures.

6.1 Abstractions

In this section, we develop an abstraction procedure that abstracts linear PHS to finite state MDP. The abstraction procedure consists of hybridization and predicate abstraction. Hybridization is used to simplify linear PHS to polyhedral PHS based on a partition of its state space, and predicate abstraction is used for abstracting polyhedral PHS into a finite state MDP. Next, we formally define hybridization procedure.

6.1.1 Hybridization

Hybridization takes as input linear probabilistic hybrid system and a partition $\mathcal{R}$ of its continuous state space, and constructs a polyhedral probabilistic hybrid system. For each location q in linear PHS, hybridization creates a set of locations for polyhedral PHS, where each location is a pair of q and R, where R is the intersection of a partition element of $\mathcal{R}$ and invariant set associated with q. The flow function associated with each location (q, R) in polyhedral PHS is the union of the set of all vector fields $\{A_q x + B_q\}$ over R, $\{A_q x + B_q\}$ is the flow function associate with the location q in respective linear PHS. Thus, the flow function in polyhedral PHS over-approximates the flow function in respective linear PHS. The flow function associated with the location (q, R) can be obtained by performing linear

transformation on $A_q x + B_q$ over R. The invariant for each location (q, R) is R. Next, for each probabilistic edge (q, ρ), where $\rho(q_1) = p_1$, $\rho(q_2) = p_2$, ..., $\rho(q_c) = p_c$, hybridization creates a set of probabilistic edges associated with the location (q, R), where each probabilistic edge is $((q, R), \rho')$, where $\rho'((q_1, R_1)) = p_1$, $\rho'((q_2, R_2)) = p_2$, ... $\rho'((q_c, R_c)) = p_c$. Also, guard, reset associated with the probabilistic edge $((q, R), \rho')$ is the same as the guard, reset associated with the probabilistic edge (q, ρ), respectively. In addition, it also constructs an additional set of probabilistic edges for each location (q, R), where each probabilistic edge is $((q, R), \rho')$, and $\rho'(q, R') = 1$, $R \rhd R'$ or $R = R'$. In this case, guard is $\mathbb{R}^n$ and reset is *Id*. Next, we present the formal definition of the hybridization.

Definition 26. *Given two sets $\mathcal{Q}$, $\mathcal{R}$, $\rho' \in Dist(\mathcal{Q} \times \mathcal{R})$, and $\rho \in Dist(\mathcal{Q})$, we say ρ' and ρ are consistent denoted as $cst_{\mathcal{Q}\times\mathcal{R}}(\rho', \rho)$ if $\sum_{R \in \mathcal{R}} \rho'(q, R) = \rho(q)$ and $0 \leq |\{R \mid \rho'(q, R) > 0\}| \leq 1$, for each $q \in \mathcal{Q}$.*

Definition 27. *Given n-dimensional linear* PHS $\mathcal{H} = (\mathcal{Q}, \mathcal{X}, Inv, Flow, Edges, Guard, Reset)$ *and a partition set $\mathcal{R} = \{R_1, R_2, \ldots, R_k\}$, where $R_i \in Poly(n)$, $1 \leq i \leq k$. we define hybridization of $\mathcal{H}$ with respect to $\mathcal{R}$ to be a polyhedral* PHS $Abs_1(\mathcal{H}, \mathcal{R}) = (\mathcal{Q}', \mathcal{X}', Inv', Flow', Edges', Guard', Reset')$, *where*

- $\mathcal{Q}' = \{(q, R) \mid q \in \mathcal{Q}, R' \in \mathcal{R}$ and $R = R' \cap Inv(q) \neq \emptyset\}$;
- $\mathcal{X}' = \mathcal{X}$;
- $Inv'((q, R)) = R$;
- $Flow'(((q, R), x)) = \{Flow((q, x')) \mid x, x' \in R\}$;
- $Edges' = E_1 \cup E_2$ where
  - $E_1 = \{((q, R), \rho') \mid \exists\ (q, \rho) \in Edges$ such that $\quad cst_{\mathcal{Q}'}(\rho', \rho)\}$.
  - $E_2 = \{((q, R), \rho') \mid \rho' \in Dist(\mathcal{Q}')$ and $\exists\ R', \ (q, R') \in \mathcal{Q}', \ \rho'(q, R') = 1, \ (R \rhd R'$ *or* $R = R')\}$;

- ⋆ $Guard'(e') = Guard(e)$ if $e' \in E_1$, $e' = ((q, R), \rho')$, $e = (q, \rho)$, and $cst(\rho', \rho)$; $Guard'(e') = \mathbb{R}^n$ if $e' \in E_2$.

- ⋆ $Reset'((e', (q', R'), x)) = Reset((e, q', x))$ if $e' \in E_1$, $e' = ((q, R), \rho')$, $e = (q, \rho)$, $\rho'((q', R')) > 0$, and $cst_{\mathcal{Q}'}(\rho', \rho)$; $Reset'((e', (q', R'), x)) = Id$ if $e' \in E_2$, $e' = ((q, R), \rho')$, and $\rho'((q', R')) = 1$.

Next, we show that $Abs_1(\mathcal{H}, \mathcal{R})$ is an over-approximation of $[\![\mathcal{H}]\!]$.

Theorem 4. *Given a linear* PHS $\mathcal{H}$ *and a partition of its state space* $\mathcal{R}$*, we have*

$$[\![\mathcal{H}]\!] \leq_{\alpha_1} [\![Abs_1(\mathcal{H}, \mathcal{R})]\!],$$

where $\alpha_1 = \{((q, x), ((q, R), x)) \in \mathcal{S}_1 \times \mathcal{S}_2 \mid x \in R\}$.

Theorem 4 provides the correctness of hybridization construction that over-approximates the linear PHS $\mathcal{H}$. However, $Abs_1(\mathcal{H}, \mathcal{R})$ still has infinite number of states, thus resulting infinite state MDP.

Computability In the hybridization construction, we are needed to compute the intersection of two polyhedral sets for constructing the locations in polyhedral PHS. In addition, we are required to perform the linear transformation over a polyhedral set for obtaining the flow function. For probabilistic edges, we are needed to check the adjacency between two polyhedral sets. All these operations are supported by Parma Polyhedra Library [146].

Furthermore, we use a partition based predicate abstraction technique that converts a polyhedral PHS into a finite state MDP as described in Subsection 5.4.1 of Chapter 5. Thus, the probabilistic safety analysis of linear PHS can be done automatically.

6.2 Experimental analysis

In this section, we present the analysis of some examples using our hierarchical abstraction procedure. The abstraction has been implemented in a Python toolbox. Before applying the hybridization, we parse a probabilistic linear hybrid system expressed in an input file into a graph data structure which is supported by the NetworkX package within the sagemath platform. Hybridization procedure converts the system into a polyhedral probabilistic hybrid system with respect to a partition of the state space using linear transformation supported by Parma Polyhedra Library[146]. Predicate abstraction procedure transforms a polyhedral probabilistic hybrid system into a finite state Markov decision process with respect to a partition of the state space using the GLPK solver available in sagemath. Then, the toolbox automatically computes the minimum/maximum probability of reachability for a given initial and unsafe specification over the Markov decision process using the PRISM model checker[35]. The evaluation of the experiment is performed with Ubuntu 14.04 OS, Intel ® Pentium(R) CPU B960 2.20GHz × 2 Processor, 4GB RAM.

We compare our toolbox with the tool ProHVer for the following examples. First, we consider an $m \times m$ grid and generate a linear dynamics for each cell given as $\dot{v} = \begin{bmatrix} -1.2 & 0.1 \\ 0.1 & -1.5 \end{bmatrix} (v - v_d)$, where v_d is the desired velocity, which we generate randomly. For each cell, we randomly create probabilistic edge for moving into its adjacent cells. For the experiment, we fix the initial region to be $[-0.9, -0.6] \times [-0.9, -0.6]$ in $(\frac{m}{2}, \frac{m}{2})$ cell and unsafe region to be $[0.6, 0.9] \times [0.6, 0.9]$ in $(\frac{m}{2} + 1, \frac{m}{2} + 1)$ cell, where $m \times m$ is the size of grid. We compute the maximum probability of reaching the unsafe region. We choose predicates based on uniform grid for hybridization and predicate abstraction. Next, we consider the bouncing ball and thermostat benchmarks.

In Table 6.1, *Loc*, *Pred*, *PE*, show the number of locations, number of predicates, and number of probabilistic edges, respectively. T_H, T_A, T_{MDP}, T_{total}, T_V show the time for hybridization, predication abstraction, probabilistic reachability computation in MDP, total

	System		Hierarchical abstraction based analysis							ProHVer	
Grid	*Loc*	*PE*	$Pred_H$	T_H	$Pred_A$	T_A	T_{MDP}	T_{total}	*Prob*	*Prob*	T_V
2×2	4	4	0	0.002	0	0.14	1.68	1.82	1	1	0.038
					1	0.36	1.70	2.06			
					2	0.98	1.69	2.67			
			1	0.004	0	1.02	1.69	2.71	0	0	0.068
					1	1.56	1.66	3.22			
					2	6.48	1.68	8.16			
4×4	16	16	0	0.006	0	0.46	1.66	2.13	1	1	0.07
					1	1.50	1.67	3.18			
					2	5.69	1.67	7.37			
			1	0.010	0	11.68	1.66	13.44	1	*TO*	*TO*
					1	13.04	1.65	14.79			
					2	49.65	1.69	51.44			
6×6	36	36	0	0.012	0	0.76	1.64	2.41	1	1	0.16
					1	5.17	1.67	6.85			
					2	14.63	1.63	16.27			
			1	0.030	0	34.05	1.66	35.74	0	0	2.40
					1	34.71	1.67	36.41			
					2	130.24	1.64	131.91			

Table 6.1: Probabilistic Reachability Analysis of Grid Navigation

			Hierarchical abstraction based analysis				ProHVer	
Benchmarks	*Rows*	*Time bounds*	T_{H+A}	T_{MDP}	T_{total}	*Prob*	*Prob*	T_V
Bouncing ball	1	1	1.14	1.63	2.77	1	1	25.69
	2	2	1.24	1.92	3.18	1	1	33.47
	3	4	1.19	1.76	2.96	1	*TO*	*TO*
Thermostat	1	1	0.03	1.64	1.67	0	0	0.07
	2	2	0.49	1.64	2.13	0	0	0.08

Table 6.2: Probabilistic Reachability Analysis of Benchmarks

time, and probabilistic reachability computation in ProHVer, respectively; $Pred_H$ and $Pred_A$ show the number of predicates for hybridization and predicate abstraction, respectively. In Table 6.2, *Time_bound* shows the time bound.

For the grid navigation, we use the same predicate for hybridization in our tool and tool ProHVer. In Table 6.1, we observe that the time for hybridization increases slowly when we increase the number of locations, however, the time for predicate abstraction rapidly increases when we increase the number of locations. This is reasonable because the construction of predicate abstraction requires solving a set of linear constraints for each probabilistic edge.

Also, we have obtained the same results in our tool and ProHVer for 2×2 and 6×6 grid examples, however, ProHVer did not terminate for 4×4 grid example within 5 minutes.

For bouncing ball, we choose the initial specification as $x = 2, v = 0$ and unsafe specification $x = 0, v < 0$. In addition, we choose the same predicate for our tool and tool ProHVer. We observe that the tool ProHVer does not terminate when we increase the time bound to 4. Also, the tool ProHVer takes more time than our tool for time-bound $1, 2$, respectively. Similarly, we compare our tool with ProHVer for the thermostat. In Table 6.2, we observe that we have obtained the same probability from both the tools.

Chapter 7

Parametric verification of linear discrete-time stochastic systems

In this chapter, we consider parametric verification problem of a class of stochastic systems, namely, linear discrete-time stochastic systems DTSSs. Linear DTSSs start in a given set of normal random vectors and evolve according to a linear function $\mathbf{X}' = \mathbf{A}\mathbf{X} + \mathbf{W}$, where $\mathbf{X}$ and $\mathbf{X}'$ are current and next-step random vectors, respectively, $\mathbf{A}$ is a linear transformation matrix, and $\mathbf{W}$ is an additive noise from a given set of normal random vectors. The behavior of the system is a sequence of normal random vectors, which can be identified with the sequence of mean and covariance matrices associated with the random vectors, namely, $(\mu_1, \Sigma_1)(\mu_2, \Sigma_2)\cdots$. We investigate the verification of parametric properties of linear DTSSs. Since all the states of the system are normal random vectors, we express parametric properties as a subset of mean and covariance space of a normal random vector. Given a parametric property $\mathbb{S}$, we are interested in checking whether all the random vectors reachable from an initial set within bounded or unbounded steps satisfy $\mathbb{S}$. The problem consists of universal quantifiers over an initial set of random vectors and time steps, which is expensive. Hence, we reduce the universally quantified problem into an equivalent existentially quantified problem. Let $\mathbb{U}$ be the complement of $\mathbb{S}$, which we call undesirable property. Then,

the existentially quantified problem corresponds to checking whether all the states reachable from an initial set in K steps, avoid undesirable property $\mathbb{U}$. In the rest of the paper, we use bounded/unbounded parametric properties verification (PPV) problem for the existentially quantified problem of bounded/unbounded steps, respectively.

For bounded PPV problem, we consider the approach bounded model-checking in[148–150], wherein, we encode the K-step executions of the system as a finite set of constraints, such that the satisfying assignments provide an execution that meets the undesirable properties $\mathbb{U}$. The variables in our encoding capture the parameters (μ, Σ) rather than random vectors, and constraints capture the evolution of the parameters along with the execution, which uniquely specify the sequence of the normal random vectors. The constraints contain linear as well as semidefinite constraints, which we solve using a semi-definite programming solver CVXPY[151].

While the bounded PPV problem is decidable, in fact, efficiently solvable, the unbounded PPV problem for linear discrete-time stochastic systems is undecidable[152]. In particular, the bounded model-checking approach cannot be extended, since there is no a priori bound K on the length of executions, and hence, require a potentially unbounded number of constraints in the encoding. One way to tackle this issue is based on abstraction, which simplifies the system for the purpose of analysis. A widely studied class of abstractions for stochastic systems consists of finite abstractions, either probabilistic or non-deterministic, that are obtained by partitioning the state-space.

In contrast to existing techniques that partition the state-space, we propose an abstraction based on the random vector space. We exploit the elegant properties of normal random vectors, to develop an efficient method for abstraction construction based on parameters.

7.1 Motivation

In this section, we present an example of parametric verification in the context of an autonomous vehicle. Let us consider an autonomous vehicle as shown in Figure 7.1, where the position sensors are noisy. We consider the following linear discrete-time dynamics,

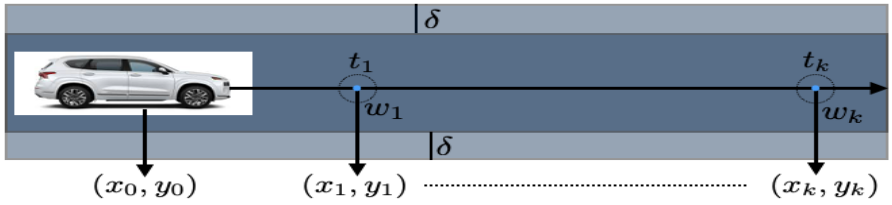

Figure 7.1: Expectation and Covariance Verification Problem

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} = \begin{bmatrix} a_{11} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix} \begin{bmatrix} x_{i-1} \\ y_{i-1} \end{bmatrix} + \begin{bmatrix} w_i^x \\ w_i^y \end{bmatrix} \tag{7.1}$$

where (x_i, y_i) denotes the position of the vehicle at time i and (w_i^x, w_i^y) denotes the corresponding sensor noise. (w_i^x, w_i^y) is assumed to be a zero-mean normal random vector, and $a_{i,j}$, $1 \leq i, j \leq 2$ are constants. An initial state (x_0, y_0) is considered to be a normal random vector specified as a pair of mean μ_0 and covariance Σ_0 (note that the linear dynamics has the property that all the states (x_i, y_i) are normal random vectors from the linear transformation of normal random vectors). For the autonomous vehicles, it is desirable to have expected positions in each lane with a limited deviation. We can capture it as a parametric property on expectation μ and covariance Σ, where μ corresponds to all the points within the lane and Σ lies in a δ-ball. To verify such properties, we need to check whether all random states reachable from a given initial random state of the vehicle satisfy the parametric property at all time steps.

7.2 Preliminaries

Multivariate Normal Distributions. Let $\mathbf{X} \sim \mathcal{N}(\mu, \Sigma)$ denote a multivariate normal vector (NV) with mean μ and covariance matrix Σ. It is known that Σ is always a symmetric and positive semi-definite matrix. We use $\mathcal{P}(n)$ to denote the set of all parameters of n-dimensional NVs, that is, $\mathcal{P}(n) = \{(\mu, \Sigma) \mid \mu \in \mathbb{R}^n,\ \Sigma \in \mathbb{S}_n^+\}$. Note that $\mathcal{P}(n)$ is an uncountable set, and is not continuous. Hence, we use $\widehat{\mathcal{P}}(n) = \{(\mu,\ \mathbf{A}) \mid \mu \in \mathbb{R}^n,\ \mathbf{A} \in \mathbb{R}^{n\times n}\}$ as a continuous over-approximation of $\mathcal{P}(n)$, which is used for obtaining a partition of $\mathcal{P}(n)$. Given $\mathcal{P}' \subseteq \mathcal{P}(n)$, $\mathbf{X} \in \mathcal{P}'$ states that $\mathbf{X} \sim \mathcal{N}(\mu, \Sigma)$ for some $(\mu, \Sigma) \in \mathcal{P}'$. Next, we state a well-known result about NVs, that is, a linear transformation of a multivariate normal vector is normal, whose parameters can be easily computed as in the following property.

Property 1. *Given two n-dimensional independent NVs $\mathbf{X} \sim \mathcal{N}(\mu_{\mathbf{x}}, \Sigma_{\mathbf{x}})$ and $\mathbf{W} \sim \mathcal{N}(\mu_d, \Sigma_d)$, $\mathbf{Y} = \mathbf{A}\mathbf{X} + \mathbf{W}$ for some square matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ is given by $\mathbf{Y} \sim \mathcal{N}(\mu_y, \Sigma_y)$ where $\mu_{\mathbf{y}} = \mathbf{A}\mu_{\mathbf{x}} + \mu_d$, and $\Sigma_{\mathbf{y}} = \mathbf{A}\Sigma_{\mathbf{x}}\mathbf{A}^{\mathrm{T}} + \Sigma_d$.*

Discrete Time Stochastic Processes. An n-dimensional discrete time stochastic process is a finite sequence of NVs denoted as $\{\mathbf{X}_i\}_{i\in[K]}$, $K \in \mathbb{Z}^+$, where $\mathbf{X}_i$ is a NV.

Graphs. A directed graph is a tuple $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where

- $\mathcal{V}$ is a set of nodes;
- $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a set of edges that captures a relation between any two nodes.

We define a path of a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ as a sequence of nodes $\sigma = s_0, s_1, s_2 ... s_n$ such that $(s_i, s_{i+1}) \in \mathcal{E}$. We use $\sigma[i]$ to denote the i^{th} node, that is, $\sigma[i] = s_i$ and $\mathit{len}(\sigma)$ to denote the length of path σ, that is, $\mathit{len}(\sigma) = n$. Let $\mathcal{P}\mathit{ath}(\mathcal{G})$ denote the set of all paths of $\mathcal{G}$. Given two nodes s, t of $\mathcal{G}$, we use $\mathit{Paths}(\mathcal{G}, s, t)$ to denote the set of all paths starting from s that visit t, that is, $\mathit{Paths}(\mathcal{G}, s, t) = \{\sigma \in \mathcal{P}\mathit{ath}(\mathcal{G}) \mid \sigma[0] = s,\ \text{for some } 0 \leq i \leq$

$len(\sigma),\ \sigma[i] = t\}$. In addition, we use $\mathcal{R}each(\mathcal{G}, s, t)$ to denote whether node t is reachable from node s in $\mathcal{G}$, which is defined as follows:

$$\mathcal{R}each(\mathcal{G}, s, t) = \begin{cases} true & \text{if } Paths(\mathcal{G}, s, t) \neq \emptyset \\ false & \text{otherwise} \end{cases}$$

7.3 Stochastic systems

In this paper, we study the class of linear discrete time stochastic systems (DTSSs), where stochastic dynamics is expressed in the form of a linear stochastic difference equation. Although the dynamics is linear, analyzing DTSS is complex due to the stochastic nature. First, we provide the syntax and semantics of linear discrete time stochastic systems.

Definition 28 (linear DTSS)**.** *An n-dimensional linear discrete time stochastic system is a tuple $\mathcal{H} = (\mathbb{X}_0, \mathbb{W}, \mathbf{A})$, where*

- $\star$ *$\mathbb{X}_0 \subseteq \mathcal{P}(n)$ is a set of parameters representing an initial set of multivariate normal vectors;*
- $\star$ *$\mathbb{W} \subseteq \mathcal{P}(n)$ is a set of of parameters representing a set of multivariate normal vectors for noise;*
- $\star$ *$\mathbf{A} \in \mathbb{R}^{n \times n}$ is a square matrix of dimension $n \times n$ representing a linear transition relation.*

The instantaneous description of the system is captured using an NV $\mathbf{X}$. The system evolves discretely from an NV $\mathbf{X}_i$ at time i to another NV $\mathbf{X}_{i+1}$ at time $i+1$ under some noise NV $\mathbf{W} \in \mathbb{W}$, that is, $\mathbf{X}_{i+1} = \mathbf{A}\mathbf{X}_i + \mathbf{W}$, where $\mathbf{X}_0 \in \mathbb{X}_0$ which is independent of the noise vector $\mathbf{W}$. In addition, the noises at different times are independent.

Next, we illustrate a linear DTSS with an example.

Example 9. *Consider the following system:* $\begin{bmatrix} x_i \\ y_i \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_{i-1} \\ y_{i-1} \end{bmatrix} + \begin{bmatrix} w_x \\ w_y \end{bmatrix}$, where $[x_i, y_i]^{\mathrm{T}} \in \mathcal{P}(2)$, $\begin{bmatrix} w_x \\ w_y \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right)$, *and* $[x_0, y_0]^{\mathrm{T}} \in \mathcal{C}(\mathcal{S}_\mu, \mathcal{S}_\Sigma)$, *where* $S_\mu = \left\{ \begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix} \mid \mu_x, \mu_y \in [0,1] \right\}$, *and* $\mathcal{S}_\Sigma = \{\Sigma \in \langle([0,1])^4\rangle \mid \Sigma \text{ is a } SPSD\}$.

This can be formally expressed as a linear DTSS $\mathcal{H} = (\mathbb{X}_0, \mathbb{W}, \mathbf{A})$, *where* $\mathbb{X}_0 = \mathcal{C}(\mathcal{S}_\mu, \mathcal{S}_\Sigma)$, $\mathbb{W} = \left\{ \left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right) \right\}$, *and* $\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$.

We define a behavior of a linear DTSS as a stochastic process where the i-th NV X_i and the $i+1$-st NV X_{i+1} are related by a linear transformation corresponding to the matrix $\mathbf{A}$ along with an additive noise from $\mathbb{W}$, that is, $\mathbf{X}_{i+1} = \mathbf{A}\mathbf{X}_i + \mathbf{W}$, where $\mathbf{W} \in \mathbb{W}$. We capture the semantics of a linear DTSS as a set of discrete time stochastic processes. The formal definition of the semantics is as follows.

Definition 29. *Given an n-dimensional linear* DTSS $\mathcal{H} = (\mathbb{X}_0, \mathbb{W}, \mathbf{A})$, *the semantics of $\mathcal{H}$ is defined as a set of discrete time stochastic processes* $[\![\mathcal{H}]\!] = \{\{\mathbf{X}_i\}_{i\in[K]} | K \in \mathbb{Z}^+, \mathbf{X}_0 \in \mathbb{X}_0,\ \forall\, j \in [K-1]\ \mathbf{X}_{j+1} = \mathbf{A}\mathbf{X}_j + \mathbf{W}_{j+1}, \text{for some}\ \ \mathbf{W}_{j+1} \in \mathbb{W}\}$.

The semantics $[\![\mathcal{H}]\!]$ consists of infinitely many discrete time stochastic processes because of the unbounded length of these processes. Further, there could be many behaviors starting from a NV due to a non-deterministic choice of noise NV at each point of time. We refer to Example 9 to illustrate the semantics.

Example 10. *We demonstrate a behavior of $\mathcal{H}$ presented in Example 9 starting from an initial NV* $\begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \right)$. *From Property 1, we have* $\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 2 \\ 2 \end{bmatrix}, \begin{bmatrix} 5 & 4 \\ 4 & 5 \end{bmatrix} \right)$ *because* $\mu_{\mathbf{y}} = \mathbf{A}\mu_{\mathbf{x}} + \mu_d = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$ *and*

$\Sigma_{\mathbf{y}} = \mathbf{A}\Sigma_{\mathbf{x}}\mathbf{A}^{\mathrm{T}} + \Sigma_d = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 5 & 4 \\ 4 & 5 \end{bmatrix}$. *Similarly, we obtain* $\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 4 \\ 4 \end{bmatrix}, \begin{bmatrix} 19 & 18 \\ 18 & 19 \end{bmatrix} \right)$, $\begin{bmatrix} x_3 \\ y_3 \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 8 \\ 8 \end{bmatrix}, \begin{bmatrix} 75 & 74 \\ 74 & 75 \end{bmatrix} \right)$, *and so on. Thus, we obtain a discrete time stochastic process* $\left\{ \begin{bmatrix} x_i \\ y_i \end{bmatrix} \right\}_{i \in [K]} \in [\![\mathcal{H}]\!]$ *for some* $K \in \mathbb{Z}^+$ *starting from* $\begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$, *where for each* $1 \leq i \leq K$, *we have*

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 2^i \\ 2^i \end{bmatrix}, \begin{bmatrix} 2\left(\sum_{j=0}^{i-2} 4^j\right) + 4^i + 1 & 2\left(\sum_{j=0}^{i-2} 4^j\right) + 4^i \\ 2\left(\sum_{j=0}^{i-2} 4^j\right) + 4^i & 2\left(\sum_{j=0}^{i-2} 4^j\right) + 4^i + 1 \end{bmatrix} \right).$$

We are interested in checking whether the model satisfies a desirable parametric property $\mathbb{S}$ of NVs, that is, whether all the stochastic behaviors of the model starting from the initial NV $[x_0, y_0]^{\mathrm{T}}$ always satisfy the parametric property $\mathbb{S}$. In other-words, whether all the stochastic behaviors of the model starting from the initial NV $[x_0, y_0]^{\mathrm{T}}$ avoid the undesirable set $\mathbb{U}$. Next, we illustrate with an example.

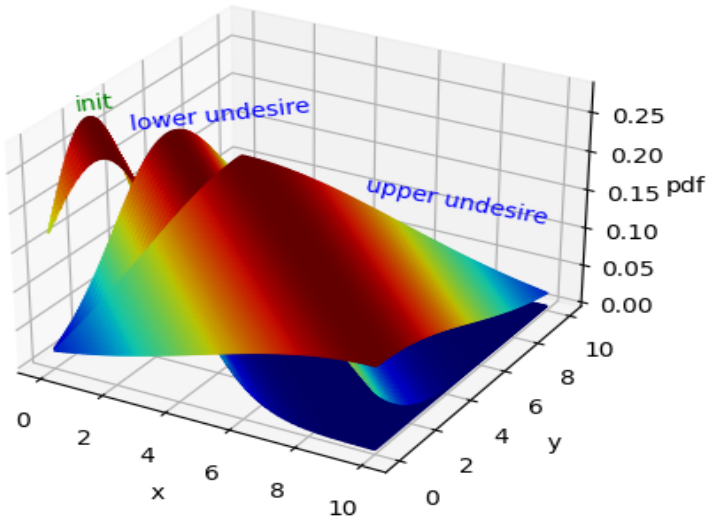

Figure 7.2: Specification Illustration for the Model shown in Example 9

Let us consider undesirable set $\mathbb{U} = \{(\mu, \Sigma) \mid \mu \in ([3,5])^2,\ \Sigma \in \langle([2,7])^4\rangle \text{ is a SPSD matrix}\}$.

The probability density function (*pdf*) of initial and undesirable set $\mathbb{U}$ of NVs have been plotted in Figure 7.2, where *init* indicates *pdf* of $[x_0, y_0]^{\mathrm{T}}$, and *lower undesire* and *upper undesire* indicate *pdf*s of $\mathbb{U}$ corresponding to the lowest and highest NV $\mathbf{X}_l \sim \mathcal{N}(\mu_l, \Sigma_l)$, $\mathbf{X}_u \sim \mathcal{N}(\mu_u, \Sigma_u)$, respectively, where

$$\mu_l = \begin{bmatrix} 3 \\ 3 \end{bmatrix}, \Sigma_l = \begin{bmatrix} 2 & 2 \\ 5 & 2 \end{bmatrix}, \mu_u = \begin{bmatrix} 5 \\ 5 \end{bmatrix}, \Sigma_l = \begin{bmatrix} 7 & 7 \\ 7 & 7 \end{bmatrix}.$$

There is only one deterministic stochastic behavior starting from $[x_0, y_0]^{\mathrm{T}}$ for model shown in Example 9 because of a singleton noise set. The behavior demonstrated in Example 10 avoids $\mathbb{U}$. However, there could be behaviors starting at other NVs which could reach $\mathbb{U}$. that might meet the undesirable properties $\mathbb{U}$.

Next, we formally define two versions of the PPV problem, the bounded and the unbounded PPV problems. In the bounded PPV problem, we are interested in checking if the undesirable property $\mathbb{U}$ of NVs can be satisfied by a NV reachable from an initial NV at some time step less than or equal to a *given* time bound K; whereas in the unbounded version we check if the the undesirable property $\mathbb{U}$ of NVs can be satisfied by a NV reached at the K-th step for *some* K. That is, in the unbounded version, we check if the undesirable property $\mathbb{U}$ can be satisfied by a NV reached in some finite number of steps.

Problem 4. *[Bounded* PPV *Problem] Given an n-dimensional linear* DTSS $\mathcal{H} = (\mathbb{X}_0, \mathbb{W}, \mathbf{A})$*, an undesirable property $\mathbb{U}$ of NVs, and an integer $K \in \mathbb{Z}^+$, we want to check whether there exists a behavior $\{\mathbf{X}_i\}_{i\in[l]} \in [\![\mathcal{H}]\!]$ for some $l \in [K]$ starting from initial NV $\mathbf{X}_0 \in \mathbb{X}_0$ such that $\mathbf{X}_l \in \mathbb{U}$. If no such process exists then we say that $\mathcal{H}$ satisfies the bounded parametric property with respect to $\mathbb{U}$.*

Problem 5. *[Unbounded* PPV *Problem] Given an n-dimensional linear* DTSS $\mathcal{H} = (\mathbb{X}_0, \mathbb{W}, \mathbf{A})$*, and an unsafe set $\mathbb{U}$ of NVs, we want to check whether there exists a behavior $\{\mathbf{X}_i\}_{i\in[K]} \in [\![\mathcal{H}]\!]$, for some $K \in \mathbb{Z}^+$ starting from initial NV $\mathbf{X}_0 \in \mathbb{X}_0$ such that $\mathbf{X}_K \in \mathbb{U}$. If no such*

process exists then we say that $\mathcal{H}$ satisfies the unbounded parametric property with respect to $\mathbb{U}$.

Note that $\mathbb{X}_0$ and $\mathbb{W}$ could have infinitely many elements; thus, $[\![\mathcal{H}]\!]$ could contain infinitely many discrete time stochastic processes. We address the bounded PPV problem by encoding processes of length $\leq K$ and checking if they satisfy the undesirable property. For the unbounded PPV, we need to check the existence of a K and a corresponding process $\{\mathbf{X}_i\}_{i\in[K]} \in [\![\mathcal{H}]\!]$ for which NV $[x_K, y_K]^{\mathrm{T}} \in \mathbb{U}$. Since, the set of unbounded length processes cannot be encoded by a finite number of constraints, we resort to an abstraction based approach.

7.4 Bounded PPV using semi-definite programming

In this section, we develop a method similar to bounded model checking for the bounded PPV of a linear DTSS $\mathcal{H} = (\mathbb{X}_0, \mathbb{W}, \mathbf{A})$. Our broad approach is to encode all stochastic processes up to a certain length in conjunction with an undesirable property of NVs into a semidefinite programming problem on the parameter space. Then, we use the semidefinite program solver CVXPY[151] for checking the feasibility of the encoding. Next, we provide the formal construction of the encoding.

Let $init(\mu, \Sigma)$, $undesire(\mu, \Sigma)$, and $noise(\mu, \Sigma)$ be specified as part of the problem description denoting the set of linear constraints over (μ, Σ) that capture the initial set, undesirable property and noise normal random vectors, respectively. Also, the transition matrix A is provided. We use $\longrightarrow (\mu_1, \Sigma_1, \mu_d, \Sigma_d,\ \mu_2, \Sigma_2)$ to capture one step transition between two NVs (μ_1, Σ_1), and (μ_2, Σ_2) under noise NV (μ_d, Σ_d). According to Property 1, it is given by,

$$\longrightarrow (\mu_1, \Sigma_1, \mu_d, \Sigma_d, \mu_2, \Sigma_2) = [\mu_2 = \mathbf{A}\mu_1 + \mu_d,\ \Sigma_2 = \mathbf{A}\Sigma_1\mathbf{A}^{\mathrm{T}} + \Sigma_d],$$

for the linear relation $\mathbf{A}$. Furthermore, we encode the set of all finite stochastic processes of $\mathcal{H}$ up to length l as a formula $\varphi^l_{\mathcal{H}}$ using $l+1$ copies of vector and symmetric positive

semidefinite matrix variables defined as follows:

$$\varphi^{l}_{\mathcal{H}}(\mu_0, \Sigma_0, \mu_1, \Sigma_1, \ldots, \mu_l, \Sigma_l) = \exists\ (\mu^d_i, \Sigma^d_i),\ 1 \leq i \leq l,\ init(\mu_0, \Sigma_0) \wedge$$
$$\bigwedge_{i=1}^{l} noise(\mu^d_i, \Sigma^d_i) \wedge \bigwedge_{i=0}^{l-1} \longrightarrow (\mu_i, \Sigma_i, \mu^d_{i+1}, \Sigma^d_{i+1}, \mu_{i+1}, \Sigma_{i+1}) \wedge undesire(\mu_l, \Sigma_l)$$

Finally, we construct a formula $\phi^K_{\mathcal{H}}$ as a disjunction of $\varphi^0_{\mathcal{H}}, \varphi^1_{\mathcal{H}}, \ldots, \varphi^K_{\mathcal{H}}$, that is,

$$\phi^K_{\mathcal{H}} = \bigvee_{l=0}^{K} \varphi^l_{\mathcal{H}}.$$

Unsatisfiability of the formula $\phi^K_{\mathcal{H}}$ implies that the system satisfies bounded parametric property with respect to $\mathbb{U}$. Otherwise, the system meets undesirable properties $\mathbb{U}$, that is, one of the formulas $\varphi^0_{\mathcal{H}}, \varphi^1_{\mathcal{H}}, \ldots, \varphi^K_{\mathcal{H}}$ is true, and its satisfying assignment provides a counterexample. We formally state this in the following theorem.

Theorem 5. *Given an n-dimensional linear* DTSS $\mathcal{H} = (\mathbb{X}_0, \mathbb{W}, \mathbf{A})$*, an undesirable property* $\mathbb{U}$ *of NVs, and an integer* K*, we have*

$$\phi^K_{\mathcal{H}} \text{ is unsatisfiable} \Leftrightarrow \mathcal{H} \text{ satisfies bounded parametric property w.r.t. } \mathbb{U}.$$

Next, we illustrate with an example.

Example 11. *We again consider Example 9 to demonstrate bounded* PPV*. Let us consider the undesirable set* $\mathbb{U} = \{(\mu, \Sigma) \mid \mu \in [10, 20],\ \Sigma \in \langle([250, 300])^4\rangle\}$ *of NVs and an integer* $K = 3$*. We construct the formula* $\phi^K_{\mathcal{H}}$ *for* $\mathbb{U}$ *as* $\varphi^0_{\mathcal{H}} \vee \varphi^1_{\mathcal{H}} \vee \varphi^2_{\mathcal{H}} \vee \varphi^3_{\mathcal{H}}$*, where*

$$\varphi^0_{\mathcal{H}}(\mu_0, \Sigma_0) = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \leq \mu_0 \leq \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \leq \Sigma_0 \leq \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix},$$
$$\begin{bmatrix} 10 \\ 10 \end{bmatrix} \leq \mu_0 \leq \begin{bmatrix} 20 \\ 20 \end{bmatrix}, \begin{bmatrix} 250 & 250 \\ 250 & 250 \end{bmatrix} \leq \Sigma_0 \leq \begin{bmatrix} 300 & 300 \\ 300 & 300 \end{bmatrix};$$

$$\varphi^1_{\mathcal{H}}(\mu_0,\Sigma_0,\mu_1,\Sigma_1) = \exists\ (\mu_1^d,\Sigma_1^d) \in \left\{\left(\begin{bmatrix}0\\0\end{bmatrix},\begin{bmatrix}1&0\\0&1\end{bmatrix}\right)\right\},\ \begin{bmatrix}0\\0\end{bmatrix} \le \mu_0 \le \begin{bmatrix}1\\1\end{bmatrix},$$
$$\begin{bmatrix}0&0\\0&0\end{bmatrix} \le \Sigma_0 \le \begin{bmatrix}1&1\\1&1\end{bmatrix}, \mu_1 = \mathbf{A}\mu_0 + \mu_1^d,\ \Sigma_1 = \mathbf{A}\Sigma_0\mathbf{A}^{\mathrm{T}} + \Sigma_1^d,$$
$$\begin{bmatrix}10\\10\end{bmatrix} \le \mu_1 \le \begin{bmatrix}20\\20\end{bmatrix},\ \begin{bmatrix}250&250\\250&250\end{bmatrix} \le \Sigma_1 \le \begin{bmatrix}300&300\\300&300\end{bmatrix};$$

For $l = 2, 3$, $\varphi^l_{\mathcal{H}}(\mu_0,\Sigma_0,\mu_1,\Sigma_1,\ldots,\mu_l,\Sigma_l) =\ \exists\ (\mu_i^d,\Sigma_i^d) \in$

$$\left\{\left(\begin{bmatrix}0\\0\end{bmatrix},\begin{bmatrix}1&0\\0&1\end{bmatrix}\right)\right\}\ \text{for } i = 1,\ldots,l,\ \begin{bmatrix}0\\0\end{bmatrix} \le \mu_0 \le \begin{bmatrix}1\\1\end{bmatrix},\ \begin{bmatrix}0&0\\0&0\end{bmatrix} \le \Sigma_0 \le \begin{bmatrix}1&1\\1&1\end{bmatrix}$$
$$\mu_{i+1} = \mathbf{A}\mu_i + \mu_{i+1}^d,\ \Sigma_{i+1} = \mathbf{A}\Sigma_i\mathbf{A}^{\mathrm{T}} + \Sigma_{i+1}^d,\ \text{for } 0 \le i \le l-1$$
$$\begin{bmatrix}10\\10\end{bmatrix} \le \mu_l \le \begin{bmatrix}20\\20\end{bmatrix},\ \begin{bmatrix}250&250\\250&250\end{bmatrix} \le \Sigma_l \le \begin{bmatrix}300&300\\300&300\end{bmatrix}.$$

The formula $\phi^3_{\mathcal{H}}$ *is unsatisfiable because none of the formulas* $\varphi^0_{\mathcal{H}},\varphi^1_{\mathcal{H}},\varphi^2_{\mathcal{H}},\varphi^3_{\mathcal{H}}$ *are true. Hence the system satisfies bounded parametric properties with respect to* $\mathbb{U}$. *However, for* $K = 4$, $\phi^4_{\mathcal{H}}$ *is satisfiable because* $\varphi^4_{\mathcal{H}}$ *is true and its satisfying instance is the following:*

$$(\mu_0,\Sigma_0) = \left(\begin{bmatrix}1\\1\end{bmatrix},\begin{bmatrix}1&1\\1&1\end{bmatrix}\right), (\mu_1,\Sigma_1) = \left(\begin{bmatrix}2\\2\end{bmatrix},\begin{bmatrix}5&4\\4&5\end{bmatrix}\right), (\mu_2,\Sigma_2) = \left(\begin{bmatrix}4\\4\end{bmatrix},\begin{bmatrix}19&18\\18&19\end{bmatrix}\right),$$

$$(\mu_3,\Sigma_3) = \left(\begin{bmatrix}8\\8\end{bmatrix},\begin{bmatrix}75&74\\74&75\end{bmatrix}\right), (\mu_4,\Sigma_4) = \left(\begin{bmatrix}16\\16\end{bmatrix},\begin{bmatrix}299&298\\298&299\end{bmatrix}\right),\ \mathit{undesire}(\mu_4,\Sigma_4) = \top$$

Computability. In the encoding, the formula $\varphi_{\mathcal{H}}^{K}$ consists of a $2K+1$ symmetric and positive semidefinite matrix variables and $K(n^2+n)+3m$ linear constraints, where K is the number of transitions and each transition consists of n^2+n linear constraints, and *Init*, *undesire*, and *noise* are expressed using at most m linear constraints. The worst case complexity of checking feasibility of the semidefinite constraints is $O((\max(m,n))^4 n^{\frac{1}{2}} log(\frac{1}{\epsilon}))$[153] for a given solution accuracy $\epsilon > 0$, where m is the number of linear constraints and $n \times n$ is the dimension of positive semidefinite matrix. We use the semidefinite program solver CVXPY[151] to check the satisfiability of the encoding.

7.5 Unbounded parametric properties verification

In this section, we present a novel predicate abstraction procedure for unbounded PPV. Our algorithm takes as input a linear DTSS $\mathcal{H}$ and a partition of the multivariate normal vector space, provided as a partition of the parameter space $\mathcal{P}(n)$, and outputs a finite abstract graph, which captures all the behaviors of the DTSS. Hence, unbounded PPV of the DTSS can be inferred by the absence of paths in the graph that reach undesired nodes.

Predicate abstraction and soundness

Our broad approach for constructing a finite abstract graph is based on a polyhedral partition of the parameter space $\mathcal{P}(n)$. Let $\mathcal{R} = \{R_1, R_2, \ldots, R_m\}$ be a polyhedral partition of $\mathcal{P}(n)$. First, we construct a finite set of nodes $\mathcal{V}$ by considering each partition element $R \in \mathcal{R}$ as a node, that is, $\mathcal{V} = \{R_1, R_2, \ldots, R_m\}$. Then, an abstract edge between two abstract nodes V_i and V_j is constructed if there exist NVs $\mathbf{X} \in V_i$, and $\mathbf{Y} \in V_j$, such that $\mathbf{Y} = \mathbf{A}\mathbf{X} + \mathbf{W}$ for some $\mathbf{W} \in \mathbb{W}$. In the rest of the section, we use $\widehat{S}$ to denote the set of abstract nodes corresponding to a given set S of NV parameters, that is, $\widehat{S} = \{V \in \mathcal{V} \mid V \cap S \neq \emptyset\}$. Next, the formal construction of the abstract graph is defined as follows.

Definition 30. *Given an n-dimensional linear* DTSS $\mathcal{H} = (\mathbb{X}_0, \mathbb{W}, \mathbf{A})$*, and a polyhedral*

partition $\mathcal{R}$ *of the parameter space* $\mathcal{P}(n)$*, we construct a finite abstract graph* $\mathcal{G}_{\mathcal{H}}^{\mathcal{R}} = (\mathcal{V}, \mathcal{E})$*, where* $\mathcal{V} = \mathcal{R}$*, and* $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$*, where* $(V_1, V_2) \in \mathcal{E}$ *if there exist* $(\mu_1, \Sigma_1) \in V_1$*,* $(\mu_2, \Sigma_2) \in V_2$ *such that* $\mu_2 = \mathbf{A}\mu_1 + \mu_d$*,* $\Sigma_2 = \mathbf{A}\Sigma_1\mathbf{A}^{\mathrm{T}} + \Sigma_d$ *for some* $(\mu_d, \Sigma_d) \in \mathbb{W}$*.*

From the definition, $\mathcal{G}_{\mathcal{H}}^{\mathcal{R}}$ captures all behaviors of $\mathcal{H}$, which is stated in the following lemma.

Lemma 1. *Given an n-dimensional linear* DTSS $\mathcal{H} = (\mathbb{X}_0, \mathbb{W}, \mathbf{A})$*, a polyhedral partition* $\mathcal{R}$ *of the parameter space* $\mathcal{P}(n)$ *and the finite abstract graph* $\mathcal{G}_{\mathcal{H}}^{\mathcal{R}} = (\mathcal{V}, \mathcal{E})$*, for each stochastic behavior* $\{\mathbf{X}_i\}_{i\in[K]} \in [\![\mathcal{H}]\!]$*, there exists a path* $\sigma \in Paths\left(\mathcal{G}_{\mathcal{H}}^{\mathcal{R}}\right)$ *of length* K*, such that* $\mathbf{X}_i \in \sigma[i]$ *for all* $i \in [K]$*.*

Proof. Let us consider a discrete time stochastic process $\{\mathbf{X}_i\}_{i\in[K]} \in [\![\mathcal{H}]\!]$, where $\mathcal{X}_i \sim \mathcal{N}(\mu_i, \Sigma_i)$. Next, we show the existence of a path $\sigma \in Paths\left(\mathcal{G}_{\mathcal{H}}^{\mathcal{R}}\right)$ corresponding to the process. Since abstract nodes forms a partition of $\mathcal{P}(n)$, each (μ_i, Σ_i) should be in some abstract node $V_i \in \mathcal{V}$. We show that $\sigma = V_0, V_1, V_2, \ldots, V_K$ is a path in the graph. Note that $\mathcal{X}_i \in \sigma[i]$ by choice of V_i. From Definition 29, for $0 \leq i < K$, we have $\mathbf{X}_{i+1} = \mathbf{A}\mathbf{X}_i + \mathbf{W}_{i+1}$, for some $\mathbf{W}_{i+1} \in \mathbb{W}$. From Property 1, we have $\mu_{i+1} = \mathbf{A}\mu_i + \mu_d$, $\Sigma_{i+1} = \mathbf{A}\Sigma_i\mathbf{A}^{\mathrm{T}} + \Sigma_d$ for some $(\mu_d, \Sigma_d) \in \mathbb{W}$. Along with Definition 30, this implies that for each $0 \leq i < K$, $(V_i, V_{i+1}) \in \mathcal{E}$. □

From Lemma 1, the finite abstract graph $\mathcal{G}_{\mathcal{H}}^{\mathcal{R}}$ is an over-approximation of $\mathcal{H}$, hence, the safety of $\mathcal{H}$ can be inferred by checking that no nodes corresponding to the undesirable set of NVs is reachable in $\mathcal{G}_{\mathcal{H}}^{\mathcal{R}}$, which is stated in Theorem 6.

Theorem 6 (Soundness)**.** *Given an n-dimensional linear* DTSS $\mathcal{H} = (\mathbb{X}_0, \mathbb{W}, \mathbf{A})$*, a polyhedral partition* $\mathcal{R}$ *of the parameter space* $\mathcal{P}(n)$*, the finite abstract graph* $\mathcal{G}_{\mathcal{H}}^{\mathcal{R}}$*, and undesirable set* $\mathbb{U} \subseteq \mathcal{P}(n)$*, if* $\widehat{\mathbb{U}}$ *is not reachable in* $\mathcal{G}_{\mathcal{H}}^{\mathcal{R}}$ *from the abstract initial set* $\widehat{\mathbb{X}}_0$*, then* $\mathcal{H}$ *satisfies the unbounded parametric property with respect to* $\mathbb{U}$*.*

Proof. Let the undesired set $\widehat{\mathbb{U}}$ be not reachable from the set $\widehat{\mathbb{X}}_0$, that is, there is no path starting from a node in $\widehat{\mathbb{X}}_0$ reaching one of the nodes in $\widehat{\mathbb{U}}$. Assume that $\mathcal{H}$ does not satisfy

the unbounded parametric property with respect to $\mathbb{U}$, that is, there exists $\{\mathbf{X}_i\}_{i\in[K]} \in [\![\mathcal{H}]\!]$ for some $K \in \mathbb{Z}^+$ such that $\mathbf{X}_0 \in \mathbb{X}_0$, and $\mathbf{X}_K \in \mathbb{U}$. From the partition of parameter space, for each $\mathbf{X}_i$ there exists a node $V_i \in \mathcal{V}$ such that $\mathbf{X}_i \in V_i$. From Definition 29, we have $\mathbf{X}_0 \in \mathbb{X}_0$, and $\mathbf{X}_{j+1} = \mathbf{A}\mathbf{X}_j + \mathbf{W}_{j+1}$, for some $\mathbf{W}_{j+1} \in \mathbb{W}$, $j \in [K-1]$. This implies that $(V_i, V_{i+1}) \in \mathcal{E}$ from Definition 30. So, there is a path $\sigma = V_0, V_1, \ldots, V_k$ in the graph $\mathcal{G}^{\mathcal{R}}_{\mathcal{H}}$. Since $\mathbf{X}_0 \in \mathbb{X}_0$, we have $V_0 \in \widehat{\mathbb{X}}_0$. Similarly, $\mathbf{X}_K \in \mathbb{U}$ implies $V_K \in \widehat{\mathbb{U}}$. This implies that there is a path starting from $V_0 \in \widehat{\mathbb{X}}_0$ reaching $V_K \in \widehat{\mathbb{U}}$, which is contradicts the fact that $\widehat{\mathbb{U}}$ is not reachable from the set $\widehat{\mathbb{X}}_0$. Hence, $\mathcal{H}$ satisfies the unbounded parametric property with respect to $\mathbb{U}$. □

Example

We illustrate the abstraction procedure by constructing a finite abstract graph for the linear DTSS shown in Example 9. In Example 9, an over-approximation of $\mathcal{P}(n)$ is $\mathcal{C}(\mathcal{S}_\mu, \mathcal{S}_\Sigma)$, where $\mathcal{S}_\mu = \mathbb{R}^2$, $\mathcal{S}_\Sigma = \mathbb{R}^4$. Let us consider a polyhedral partition of the set $\mathcal{C}(\mathcal{S}_\mu, \mathcal{S}_\Sigma)$ as the following set: $\mathcal{R} = \{\mathcal{C}(P^i_\mu, P^j_\Sigma) \mid 1 \le i \le 2,\ 1 \le j \le 4\}$, where P^i_μs are partition elements for $\mathcal{S}_\mu$, which are shown via regions in Figure 7.3, and given as below:

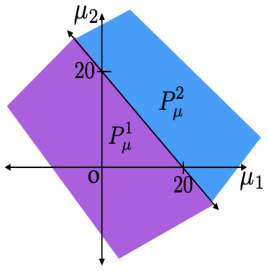

Figure 7.3: Partition of $\mathcal{S}_\mu$

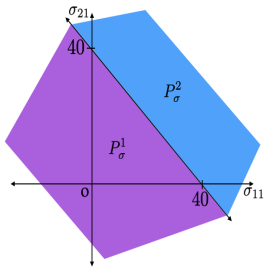

Figure 7.4: Partition of $\mathcal{S}_{\Sigma_1}$

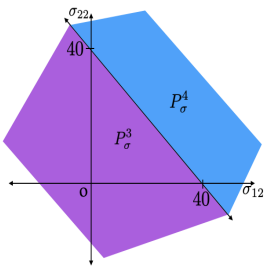

Figure 7.5: Partition of $\mathcal{S}_{\Sigma_2}$

$P^1_\mu = \left\{ \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix} \;\middle|\; \mu_1 + \mu_2 \le 20 \right\}$, $P^2_\mu = \left\{ \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix} \;\middle|\; \mu_1 + \mu_2 \ge 20 \right\}$. Partition of $\mathcal{S}_\Sigma$ is expressed via cartesian product of regions of $\mathcal{S}_{\Sigma_1}$, and $\mathcal{S}_{\Sigma_2}$ shown in Figures 7.4, and 7.5, respectively, that is, $P^1_\Sigma = \langle \mathcal{C}(P^1_\sigma, P^3_\sigma) \rangle$, $P^2_\Sigma = \langle \mathcal{C}(P^1_\sigma, P^4_\sigma) \rangle$, $P^3_\Sigma = \langle \mathcal{C}(P^2_\sigma, P^3_\sigma) \rangle$, $P^4_\Sigma = \langle \mathcal{C}(P^2_\sigma, P^4_\sigma) \rangle$, where P^i_σs are as below:

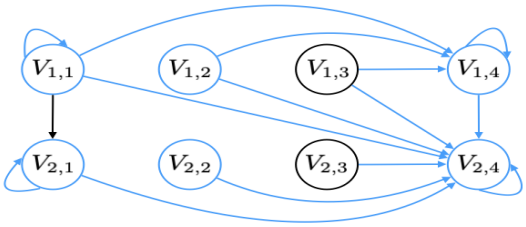

Figure 7.6: Abstract Graph $\mathcal{G}^{\mathcal{R}}_{\mathcal{H}}$

- $P^1_\sigma = \{(\sigma_{11}, \sigma_{21}) \mid \sigma_{11} + \sigma_{21} \leq 40\}$, $P^2_\sigma = \{(\sigma_{11}, \sigma_{21}) \mid \sigma_{11} + \sigma_{21} \geq 40\}$,
- $P^3_\sigma = \{(\sigma_{12}, \sigma_{22}) \mid \sigma_{12} + \sigma_{22} \leq 40\}$, $P^4_\sigma = \{(\sigma_{12}, \sigma_{22}) \mid \sigma_{12} + \sigma_{22} \geq 40\}$.

From Definition 30, we construct a finite state graph $\mathcal{G}^{\mathcal{R}}_{\mathcal{H}} = (\mathcal{V}, \mathcal{E})$, where

- $\mathcal{V} = \{V_{i,j} \mid 1 \leq i \leq 2,\ 1 \leq j \leq 4\}$, where $V_{i,j} = \mathcal{C}(P^i_\mu, P^j_\Sigma)$;
- $\mathcal{E} = \{(V_{1,1}, V_{1,1}), (V_{1,1}, V_{1,4}), (V_{1,1}, V_{2,1}), (V_{1,1}, V_{2,4}), (V_{1,2}, V_{1,4}), (V_{1,2}, V_{2,4}), (V_{1,3}, V_{1,4}), (V_{1,3}, V_{2,4}), (V_{1,4}, V_{1,4}), (V_{1,4}, V_{2,4}), (V_{2,1}, V_{2,1}), (V_{2,1}, V_{2,4}), (V_{2,2}, V_{2,4}), (V_{2,3}, V_{2,4}), (V_{2,4}, V_{2,4})\}$.

The abstract graph is shown in Figure 7.6. Note that an abstract edge (V, V') belongs to $\mathcal{E}$

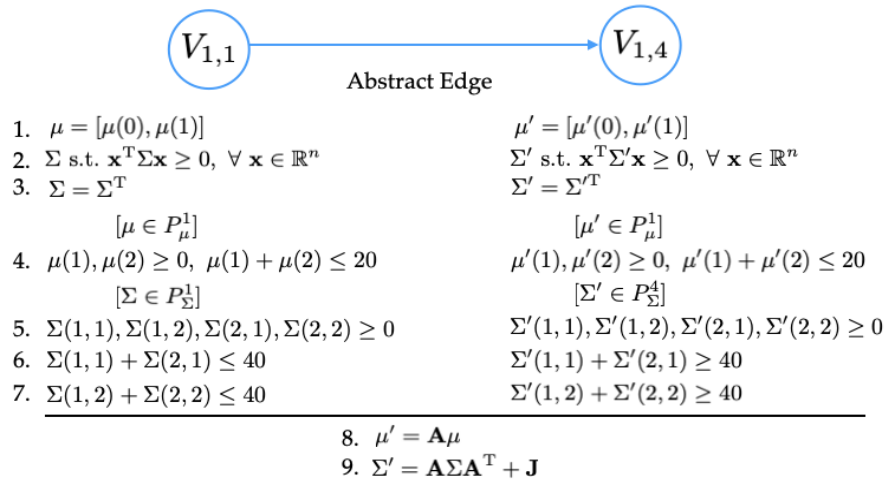

Figure 7.7: Validation of the Abstract Edge $(V_{1,1}, V_{1,4})$

only if it is valid, that is, it corresponds to a concrete transition in the DTSS, wherein, there are NVs $X \in V$ and $X' \in V'$, that are related by the transition matrix with some additive noise. We encode the transition relation, that is, the set of all possible concrete transitions,

corresponding to the abstract edge into semi-definite programming. We use semi-definite program solver CVXPY[151] to check the satisfiability of the encoding. An illustration of the encoding for the abstract edge $(V, V') = (V_{1,1}, V_{1,4}) \in \mathcal{E}$ is shown in Figure 7.7, where Lines 1, 2 and 3 encode the constraints of the parameter space, corresponding to semi-definiteness and symmetricity of the covariance matrices. Here, (μ, Σ) and (μ', Σ') are the parameters corresponding to V and V', respectively. Line 4 encodes the polyhedral partition elements $\mathcal{P}^1_\mu$ and $\mathcal{P}^2_\mu$ corresponding to the mean variables. Lines 5, 6, and 7 encode the polyhedral matrix partition elements $\mathcal{P}^1_\Sigma$ and $\mathcal{P}^4_\Sigma$ in the matrix variables Σ, Σ', respectively. Lines 7 and 8 encode the linear relation between any two multivariate normal parameter values in $V_{1,1}$ and $V_{1,4}$, respectively, where $\mathbf{J}$ is the identity matrix of size 2×2 that we considered for the covariance matrix of noise NV $[w_x, w_y]^{\mathrm{T}}$.

Computability

In the predicate abstraction, there are two main computations, namely, polyhedral partitioning of the parameter space $\mathcal{P}(n)$ and validation of the abstract edges. Note that $\mathcal{P}(n)$ is the Cartesian product of mean space $\mathbb{R}^n$ and covariance matrix space, that the set of all symmetric and positive semi-definite matrices of dimension $n \times n$. $\mathcal{P}(n)$ is an uncountable set, and is not continuous. Hence, to represent a region of $\mathcal{P}(n)$, we consider a continuous set $\widehat{\mathcal{P}}$ as an over-approximation of $\mathcal{P}(n)$, and construct the partition of $\widehat{\mathcal{P}}$ instead of $\mathcal{P}(n)$ based on a given set of linear expressions of size p. This require us to perform at most 2^p operations corresponding to computing intersections of polyhedral sets and checking emptiness of polyhedral sets. Both intersection and emptiness can be performed in polynomial in the number of constraints (say $O(C)$). The worst case complexity of constructing polyhedral partition of $\widehat{\mathcal{P}}$ is $O(2^p C)$. Note that a partition of $\widehat{\mathcal{P}}$ will represent a partition of $\mathcal{P}(n)$ via by adding the constraint that the matrix corresponding to the Σ parameter is symmetric and positive semidefinite matrix. Next, we need to solve positive semidefinite constraints for the validation of an abstract edge. For an abstract edge, the semidefinite constraints consist

of $2p + n^2 + n$ linear constraints, where $2p$ linear constraints are for encoding membership in the two abstract nodes, since a polyhedral partition element can have at most p linear constraints, and $n^2 + n$ linear constraints are required to encode the linear relation. Hence, the complexity of the validation of an abstract edge is $O((\max(2p + n^2 + n, n))^4 n^{\frac{1}{2}} log(\frac{1}{\epsilon}))$ from the worst-case complexity of semidefinite programming[153] for a given solution precision $\epsilon > 0$. We use Parma Polyhedra Library (PPL)[146] for polyhedral partitioning of $\widehat{\mathcal{P}}$, and the semi-definite program solver CVXPY[151] for checking the existence of positive semidefinite matrices satisfying a set of linear constraints.

7.6 Experimental analysis

In this section, we present details of the implementation and the experimental analysis of a fixed 2-dimensional model shown in Figure 9 and some random linear systems of different dimensions. We have implemented our bounded model checking based method for bounded PPV and abstraction procedure for unbounded PPV in a Python toolbox. We have implemented bounded encoding into semidefinite programming and used semidefinite program solver CVXPY[151] for bounded PPV. For the unbounded PPV, the abstraction procedure consists of mainly three modules: (a) Polyhedral Partitioning; (b) Abstraction; (c) Safety Checking. We have used Parma Polyhedra Library (PPL)[146] for the module (a). The abstraction module outputs a finite abstract graph from a linear DTSS based on a polyhedral partition of the parameter space, and we have used CVXPY[151] for checking the validity of an abstract edge. NetworkX[154] has been used for checking parametric properties on the finite abstract graph with respect to given undesirable properties. We have performed our experimental evaluation with Ubuntu 18.04 OS, Intel ® Pentium(R) CPU B960 2.5GHz Quad-Core Intel Core i7, 8GB RAM.

We consider Example 9 for the bounded PPV. Let the undesirable set $\mathbb{U}$ be $\{(\mu_l, \Sigma) \mid \mu \in ([10, 20])^2,\ \Sigma \in \langle([250, 300])^4\rangle \cap \mathbb{S}_2^+\}$. We randomly generate models with different dimensions to measure the computational performance of the method, where $\mathbf{A}$ is chosen randomly with

entries in between 1 and 10. For all random models, the disturbance set is chosen as a pair of the interval vector $([0,1])^2$ for mean vectors and the interval set $([0,1])^4$ for covariance matrices $\langle([0,1])^4\rangle \cap \mathbb{S}_2^+$. For the specification, an initial set is randomly generated as a pair of mean vectors $([\mu_l, \mu_u])^n$, and covariance matrices $\langle([\sigma_l, \sigma_u])^{n^2}\rangle \cap \mathbb{S}_n^+$, where $\mu_l, \sigma_l \in [0,5]$, $\mu_u, \sigma_u \in [5,10]$. Similarly, undesirable properties are randomly generated as a pair of mean vectors $([\mu_l, \mu_u])^n$, and covariance matrices $\langle([\sigma_l, \sigma_u])^{n^2}\rangle \cap \mathbb{S}_n^+$ where $\mu_l \in [30,300]$, $\mu_u \in [500,600]$, $\sigma_l \in [100,300]$, $\sigma_u \in [500,600]$.

In Tables 7.1 and 7.2, *Rows*, *Dim* and *result* show row number, dimension of the system and the result of the parametric properties verification, respectively. K, T_{enc}, T_{safe}, and T_{total} represent bound on the number of transitions in the stochastic processes, time for encoding of bounded PPV (into CVXPY), time for bounded PPV checking (time taken by CVXPY solver), and total time, respectively. The experimental results for different values of K and *Dim* are presented in Tables 7.1 and 7.2.

In Table 7.2, we observe from rows $1, 2, 3$ that the encoding time increases linearly as we increase the value of K while retaining the same dimension. Also, it is almost constant as we increase the dimension while keeping the value of K the same as observed from rows $3, 6, 12, 15$. The bounded PPV checking time is constant for a fixed value of K regardless of the dimension, and it increases linearly as we increase the value of K for a fixed dimension. The experimental results show that our method is reasonably scalable.

K	T_{enc}(sec.)	T_{safe}(sec)	T_{total}(sec)	*result*
2	0.003	0.062	0.06	*safe*
3	0.004	0.082	0.08	*safe*
4	0.006	0.10	0.10	*unsafe*

Table 7.1: Bounded PPV of the Model shown in Example 9

Furthermore, we consider the following 2-dimensional model for the unbounded paramet-

Rows	*Dim*	K	T_{enc}(sec.)	T_{safe}(sec.)	T_{total}(sec.)
1	2	1	0.002	0.041	0.044
2		5	0.006	0.113	0.12
3		25	0.021	0.47	0.49
4	4	1	0.002	0.042	0.048
5		5	0.006	0.12	0.131
6		25	0.022	0.54	0.56
7	6	1	0.002	0.049	0.05
8		5	0.006	0.127	0.135
9		25	0.022	0.71	0.73
10	8	1	0.002	0.054	0.058
11		5	0.007	0.136	0.145
12		25	0.022	0.624	0.648
13	10	1	0.002	0.059	0.063
14		5	0.006	0.16	0.169
15		25	0.022	0.70	0.729

Table 7.2: Computational Analysis for Bounded PPV of Random Models

ric properties verification.

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} x_{i-1} \\ y_{i-1} \end{bmatrix} + \begin{bmatrix} w_x \\ w_y \end{bmatrix}, \tag{7.2}$$

where $[x_i, y_i]^{\mathrm{T}} \in \mathcal{P}(2)$, $\begin{bmatrix} w_x \\ w_y \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right)$. For the specification, an initial and undesirable set are considered as follows: $\mathbb{X}_0 = \{(\mu, \Sigma) \mid \mu \in ([4,5])^2,\ \Sigma \in \langle([4,5])^4\rangle \cap \mathbb{S}_2^+\}$, $\mathbb{U} = \{(\mu, \Sigma) \mid \mu \in ([0,1])^2,\ \Sigma \in \langle([1,2])^4\rangle \cap \mathbb{S}_2^+\}$. We partition the mean and covariance matrix space with respect to linear expressions $[x-2, y-2]$ and $[\sigma_{xx}-2, \sigma_{xy}-2, \sigma_{yx}-2, \sigma_{yy}-2]$, respectively. Also, we randomly generate models with different dimensions and undesirable sets to evaluate the computational performance of the abstraction method. We randomly generate a linear matrix $\mathbf{A}$ of size $n \times n$ where each entry lies in $[0, 1]$. Similarly, we randomly generate n-dimensional interval vectors and interval matrices of size $n \times n$, where initial interval mean vector and covariance matrices are $([\mu_l, \mu_u])^n$ and $\langle([\sigma_l, \sigma_u])^{n^2}\rangle \cap \mathbb{S}_n^+$, where $\mu_l = 0,\ \mu_u, \sigma_l \in [0, 5],\ \sigma_u \in [5, 10]$, and interval mean vector and covariance matrices for

	Predicate		Abstract Graph		Time(seconds)				
Rows	*#mean*	*#variance*	*#nodes*	*#edges*	T_P	T_{abs}	T_{safe}	T_{total}	*result*
1	2	4	32	112	0.036	34.86	23.50	58.37	safe
2	2	4	32	112	0.039	34.61	24.14	58.75	Unknown

Table 7.3: Unbounded PPV of the Model shown in Equation 7.2

the undesirable set are $([\mu_l, \mu_u])^n$ and $\langle([\sigma_l, \sigma_u])^{n^2}\rangle \cap \mathbb{S}_n^+$, where $\mu_l, \sigma_l \in [5000, 5050]$, $\mu_u, \sigma_u \in [5100, 5150]$. For all random models, we consider zero vector for noise mean and identity matrix for noise covariance matrix. In addition, we randomly generate linear expressions for polyhedral partitioning of mean space $\mathbb{R}^n$ and matrix space $\mathbb{R}^{n\times n}$ via randomly choosing coefficients of the linear expressions to be either 1 or -1.

In Tables 7.3 and 7.4, *Rows* and *Dim* show row number and dimension of the random model, respectively. *#mean* and *#variance* represent the number of linear expressions used for polyhedral partitioning of mean and matrix space, respectively. In addition, T_P, T_{abs}, T_{safe}, and T_{total} show the computation time for polyhedral partitioning, predicate abstraction, parametric properties checking on the abstract graph, and total time, respectively. All computation times are measured in seconds. The experimental results are reported in Tables 7.3 and 7.4.

In Table 7.3, we report the experimental analysis in row 1 for the model shown in Equation 7.2 with respect to the initial set $\mathbb{X}_0$ and undesirable properties $\mathbb{U}$, respectively. We found that the model satisfies unbounded PPV, which is expected for the model. However, when we switch the initial and undesirable set with each other, the result is *unknown* due to the abstraction.

In Table 7.4, we observe that the computation time for the polyhedral partitioning for the same number of linear expressions is almost constant for all dimensions as we can see in rows $1, 7, 13, 19$,and 25 for 3 linear expressions for mean space and 1 linear expression for matrix space. In addition, abstraction time for the same number of abstract nodes increases linearly as we increase the dimension, which can be seen in rows $1, 10, 16, 22$, and 28. Also, time for

		Predicate		Abstract Graph		Time(seconds)			
Rows	*Dim*	*#mean*	*#variance*	*#nodes*	*#edges*	T_P	T_{abs}	T_{safe}	T_{total}
1	2	3	1	8	64	1.34	2.81	6.06	10.22
2			2	12	112	1.84	6.95	9.46	18.25
3			4	24	168	4.59	38.13	18.62	61.36
4		1	2	6	24	1.25	1.71	4.58	7.55
5		2		3	3	1.44	0.41	2.27	4.13
6		4		16	192	2.66	13.85	12.74	29.25
7	4	3	1	12	144	1.71	9.29	9.21	20.22
8			2	20	375	2.51	35.91	15.29	53.72
9			4	64	2960	4.31	508.94	52.98	566.23
10		1	2	8	52	1.29	4.97	6.18	12.45
11		2		12	144	11.52	11.37	9.35	22.25
12		4		18	216	4.32	29.59	14.28	48.20
13	6	3	1	10	100	2.14	9.23	7.99	19.37
14			2	16	256	1.84	30.47	12.64	44.95
15			4	92	9216	5.58	2133.55	120.75	2259.49
16		1	2	8	64	1.30	7.29	6.29	14.81
17		2		16	256	1.84	30.47	12.64	44.95
18		4		32	1024	2.42	131.59	25.46	159.51
19	8	3	1	16	256	2.08	30.04	13.04	45.17
20			2	24	576	2.56	105.81	19.96	128.34
21			4	96	9216	5.76	3420.54	115.78	3542.09
22		1	2	8	64	1.37	11.65	6.80	19.85
23		2		16	256	1.78	45.55	13.20	60.53
24		4		48	2304	4.17	435.52	42.11	481.82
25	10	3	1	12	144	2.19	23.43	10.93	36.55
26			2	32	1024	2.75	272.19	29.72	304.67
27			4	128	16384	6.61	10088.34	143.07	10234.03
28		1	2	8	64	1.57	15.86	7.35	24.79
29		2		16	256	1.96	66.37	14.88	83.23
30		4		64	4096	4.44	1188.03	66.56	1253.60

Table 7.4: Computational Analysis for Unbounded Safety of random Models

unbounded PPV checking is constant for the same number of abstract nodes regardless of the number of edges for all dimensions as it can be seen in rows $1, 10, 16, 22, 28$. The total time increases quadratically as we increase the dimension for the same number of linear expressions used for mean and matrix space. This can be observed in rows $1, 10, 16, 22, 28$. The experimental observations show that our method is reasonably scalable.

Chapter 8

Conclusions and future work

In thesis, first we have developed a method for computing the bound on the minimum/maximum probability of reachability in a polyhedral probabilistic hybrid system, where non-deterministic probabilistic transitions are allowed. This issue is not only a probabilistic reachability problem, but also an optimization problem. Our method exploits the recent advances of existing tool Z3opt and Symba to solve the bounded reachability problem. We have found that Z3opt performs faster than Symba. Hence, we have used Z3opt for computing the maximum/minimum probabilistic reachability. We have run our method for the probabilistic reachability analysis of the vehicle navigation. The experimental results show that the method is efficient for solving the bounded reachability problem. In future, we will extend the bounded reachability analysis to systems with complex non-linear dynamics, wherein in addition to non-determinism, we consider constraints involving rewards and gains. In addition, we will investigate existing tools that support linear objective function with conjunction and disjunction of linear/non-linear constraints and select appropriate tool for the efficient computation of probabilistic reachability. Also, we will explore sampling based approaches, such as statistical model checking.

Next, we have developed a CEGAR based method for probabilistic safety analysis of polyhedral probabilistic hybrid systems. We have implemented the method in a Python

toolbox and compared our method with the tool ProHVer[52]. Our experimental analysis demonstrates the advantages of our technique in terms of both being able to verify many more systems as well as being able to validate counterexamples. However, there are several strategies for splitting to achieve progressive refinements, and we will explore some of those in the future work. Some of the suggested ideas to speed up the algorithm are the following. To speed up the construction of the refined abstract graph in each iteration, if we store non-modified vertices and edges of the abstract graph, then we will only need to check branching behaviors corresponding to the new potential abstract edges. The verification time for the abstract graph can be improved in two ways. First, we may need to check other probabilistic safety verification tools, such as statistical model checker. Second, the refined abstract graph could be expressed as a pair of modified and non-modified parts with respect to the previous abstract system. Then, the analysis on the refined system can be achieved via performing the analysis on only its modified part and combining its analysis with the analysis of the non-modified part. We could also take an advantage of the tool ProHVer via running ProHVer and our CEGAR algorithm in parallel, specifically for those systems, for which ProHVer returns true earlier than our method. To speed up the validation of an LDAG, other quantifier elimination tools may be useful.

Then, we have developed a hierarchical abstraction for safety analysis of linear probabilistic hybrid systems. We have implemented the hierarchical abstraction in a Python toolbox and compared our method with the tool ProHVer. The performance of our tool relies on choosing the right predicates for hybridization and predicate abstraction procedures. Hence, in the future, we will develop a counter-example guided abstraction refinement scheme for choosing predicates carefully to improve the precision of the probabilistic reachability and the performance of our method.

Finally, we have developed methods for verification of parametric properties of linear discrete time stochastic systems for both bounded and unbounded time horizons. We have presented a bounded model checking based method for the bounded PPV that reduces the

problem to that of the satisfiability of a semidefinite program, which can be efficiently solved using solvers such as CVXPY[151]. For the unbounded safety analysis, we have presented a novel predicate abstraction method based on partitioning the space of random vectors that we have implemented in a Python toolbox using Parma Polyhedral Library[146] and semidefinite program solver CVXPY[151]. We have used our methods for the the verification of parametric properties of randomly generated models up to 10 dimensions. Our experimental analysis shows that our methods are reasonably scalable. However, the performance of the abstraction method relies on the selection of the right predicates. However, the performance of the abstraction method relies on the selection of the right predicates. Hence, in the future, we will develop a CEGAR based method for the careful selection of predicates to achieve precise result for the PPV. Finally, we will explore the benefits and limitations of our methods for relevant industrial applications, such as simplified model of autonomous systems, power trains, and robotics.

Bibliography

[1] Toshiaki Kakinami, Mitsuyoshi Saiki, and Jun Sato. Vehicle cruise control system, 1993. US Patent 5,230,400.

[2] Rajesh Rajamani. *Vehicle dynamics and control*. Springer Science & Business Media, 2011.

[3] Manana S Netto, Salim Chaib, and Said Mammar. Lateral adaptive control for vehicle lane keeping. In *Proceedings of the American Control Conference*, 2004.

[4] Jochen Kaller and Dieter Hoetzer. Lane-change assistant for motor vehicles, 2011. US Patent 8,040,253.

[5] Xiangru Xu, Jessy W Grizzle, Paulo Tabuada, and Aaron D Ames. Correctness guarantees for the composition of lane keeping and adaptive cruise control. *IEEE Transactions on Automation Science and Engineering*, 2017.

[6] Mike Daily, Swarup Medasani, Reinhold Behringer, and Mohan Trivedi. Self-driving cars. *Computer*, 2017.

[7] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. On a formal model of safe and scalable self-driving cars. *arXiv preprint arXiv:1708.06374*, 2017.

[8] Claudine Badue, Rânik Guidolini, Raphael Vivacqua Carneiro, Pedro Azevedo, Vinicius Brito Cardoso, Avelino Forechi, Luan Jesus, Rodrigo Berriel, Thiago Paixao, Filipe Mutz, et al. Self-driving cars: A survey. *arXiv preprint arXiv:1901.04407*, 2019.

[9] Luu Anh Tuan, Man Chun Zheng, and Quan Thanh Tho. Modeling and verification

of safety critical systems: A case study on pacemaker. In *International Conference on Secure Software Integration and Reliability Improvement*, 2010.

[10] Zhihao Jiang, Miroslav Pajic, Salar Moarref, Rajeev Alur, and Rahul Mangharam. Modeling and verification of a dual chamber implantable pacemaker. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2012.

[11] Timothy W Vanderveen. System and method for verifying connection of correct fluid supply to an infusion pump, 2006. US Patent 7,092,796.

[12] Mark C Estes, Mitchell Wenger, Morten Mernoe, and James Causey. Method and system for manual and autonomous control of an infusion pump, 2012. US Patent 8,192,394.

[13] Jodi Forlizzi and Carl DiSalvo. Service robots in the domestic environment: a study of the roomba vacuum in the home. In *Proceedings of the conference on Human-robot interaction*, 2006.

[14] Ja-Young Sung, Lan Guo, Rebecca E Grinter, and Henrik I Christensen. my roomba is rambo: intimate home appliances. In *International conference on ubiquitous computing*, 2007.

[15] Robert D Howe and Yoky Matsuoka. Robotics for surgery. *Annual review of biomedical engineering*, 1999.

[16] Santiago Horgan and Daniel Vanuno. Robots in laparoscopic surgery. *Journal of Laparoendoscopic & Advanced Surgical Techniques*, 2001.

[17] Haiyang Chao, Yongcan Cao, and YangQuan Chen. Autopilots for small fixed-wing unmanned air vehicles: A survey. In *International Conference on Mechatronics and Automation*, 2007.

[18] HaiYang Chao, YongCan Cao, and YangQuan Chen. Autopilots for small unmanned aerial vehicles: a survey. *International Journal of Control, Automation and Systems*, 2010.

[19] CDR HR Everett. Survey of collision avoidance and ranging sensors for mobile robots. *Robotics and Autonomous Systems*, 1989.

[20] BM Albaker and NA Rahim. A survey of collision avoidance approaches for unmanned aerial vehicles. In *international conference for technical postgraduates (TECHPOS)*, 2009.

[21] Stamatis Karnouskos. Cyber-physical systems in the smartgrid. In *international conference on industrial informatics*. IEEE, 2011.

[22] Xi Fang, Satyajayant Misra, Guoliang Xue, and Dejun Yang. Smart gridthe new and improved power grid: A survey. *IEEE communications surveys & tutorials*, 2011.

[23] Gregor V Bochmann and Carl A Sunshine. A survey of formal methods. In *Computer Network Architectures and Protocols*. 1982.

[24] Dan Craigen, Susan Gerhart, and Ted Ralston. An international survey of industrial applications of formal methods. In *Z User Workshop, London*, 1993.

[25] Kenneth L McMillan. Symbolic model checking. In *Symbolic Model Checking*. 1993.

[26] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT press, 2008.

[27] Edmund M Clarke Jr, Orna Grumberg, Daniel Kroening, Doron Peled, and Helmut Veith. *Model checking*. MIT press, 2018.

[28] Stephen A Cook. The complexity of theorem-proving procedures. In *Proceedings of the symposium on Theory of computing*, 1971.

[29] Melvin Fitting. *First-order logic and automated theorem proving.* Springer Science & Business Media, 2012.

[30] Chin-Liang Chang and Richard Char-Tung Lee. *Symbolic logic and mechanical theorem proving.* Academic press, 2014.

[31] Donald W Loveland. *Automated Theorem Proving: a logical basis.* Elsevier, 2016.

[32] Kim G Larsen, Paul Pettersson, and Wang Yi. Uppaal in a nutshell. *International journal on software tools for technology transfer*, 1997.

[33] Alessandro Cimatti, Edmund Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. Nusmv 2: An opensource tool for symbolic model checking. In *International Conference on Computer Aided Verification*, 2002.

[34] Gerard J Holzmann. *The SPIN model checker: Primer and reference manual*, volume 1003. Addison-Wesley Reading, 2004.

[35] Marta Kwiatkowska, Gethin Norman, and David Parker. Prism 4.0: Verification of probabilistic real-time systems. In *International Conference on Computer Aided Verification*, 2011.

[36] Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. A storm is coming: A modern probabilistic model checker. In *International Conference on Computer Aided Verification*, 2017.

[37] Harsh Raju Chamarthi, Peter Dillinger, Panagiotis Manolios, and Daron Vroon. The acl2 sedan theorem proving system. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2011.

[38] Yves Bertot and Pierre Castéran. *Interactive theorem proving and program development: CoqArt: the calculus of inductive constructions*. Springer Science & Business Media, 2013.

[39] Jasmin Christian Blanchette, Lukas Bulwahn, and Tobias Nipkow. Automatic proof and disproof in isabelle/hol. In *International Symposium on Frontiers of Combining Systems*, 2011.

[40] Zohar Manna, Nikolaj Bjørner, Anca Browne, Edward Chang, Michael Colón, Luca de Alfaro, Harish Devarajan, Arjun Kapur, Jaejin Lee, Henny Sipma, et al. Step: The stanford temporal prover. In *Colloquium on Trees in Algebra and Programming*, 1995.

[41] Susanne Graf and Hassen Saïdi. Construction of abstract state graphs with pvs. In *International Conference on Computer Aided Verification*, 1997.

[42] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2008.

[43] Christos G Cassandras and John Lygeros. *Stochastic hybrid systems*. CRC Press, 2018.

[44] Jianghai Hu, John Lygeros, and Shankar Sastry. Towards a theory of stochastic hybrid systems. In *International Workshop on Hybrid Systems: Computation and Control*. Springer, 2000.

[45] John Lygeros and Maria Prandini. Stochastic hybrid systems: a powerful framework for complex, large scale applications. *European Journal of Control*, 2010.

[46] Jan JMM Rutten, Marta Kwiatkowska, Gethin Norman, and David Parker. Mathematical techniques for analyzing concurrent and probabilistic systems. In *Amer- ican Mathematical Soc.*, 2004.

[47] Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata? *J. Comput. Syst. Sci.*, 1998.

[48] Gerardo Lafferriere, George J. Pappas, and Sergio Yovine. A new class of decidable hybrid systems. In *International Conference on Hybrid Systems: Computation and Control*, 1999.

[49] Goran Frehse. Phaver algorithmic verification of hybrid systems past hytech. In *Proceedings of the Hybrid Systems: Computation and Control, International Workshop, HSCC*, 2005.

[50] Stefan Ratschan and Zhikun She. Safety verification of hybrid systems by constraint propagation based abstraction refinement. In *Proceedings of the Hybrid Systems: Computation and Control, International Workshop, HSCC*, 2005.

[51] Wenji Zhang, Pavithra Prabhakar, and Bala Natarajan. Abstraction based reachability analysis for finite branching stochastic hybrid systems. In *International Conference on Cyber-Physical Systems*, 2017.

[52] Lijun Zhang, Zhikun She, Stefan Ratschan, Holger Hermanns, and Ernst Moritz Hahn. Safety verification for probabilistic hybrid systems. *Eur. J. Control*, 2012.

[53] John G Kemeny and J Laurie Snell. *Markov chains*. Springer-Verlag, New York, 1976.

[54] James R Norris, John Robert Norris, and James Robert Norris. *Markov chains*. Cambridge university press, 1998.

[55] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[56] Charles Miller Grinstead and James Laurie Snell. *Introduction to probability*. American Mathematical Soc., 2012.

[57] A Agung Julius, Ádám Halász, M Selman Sakar, Harvey Rubin, Vijay Kumar, and George J Pappas. Stochastic modeling and control of biological systems: the lactose regulation system of escherichia coli. *IEEE Transactions on Automatic Control*, (Special Issue), 2008.

[58] Jaime Gomez-Ramirez and Ricardo Sanz. On the limitations of standard statistical modeling in biological systems: a full bayesian approach for biology. *Progress in biophysics and molecular biology*, 2013.

[59] Ying-zi Li and Jin-cang Niu. Forecast of power generation for grid-connected photovoltaic system based on markov chain. In *Asia-Pacific Power and Energy Engineering Conference*. IEEE, 2009.

[60] Wai-Ki Ching and Michael K Ng. Markov chains. *Models, algorithms and applications*, 2006.

[61] Giorgio Alfredo Spedicato. Discrete time markov chains with r. *R J.*, 2017.

[62] Ward Whitt. Continuous-time markov chains. *Dept. of Industrial Engineering and Operations Research, Columbia University, New York*, 2006.

[63] Damjan Škulj. Discrete time markov chains with interval probabilities. *International journal of approximate reasoning*, 2009.

[64] Ernst Moritz Hahn, Holger Hermanns, and Lijun Zhang. Probabilistic reachability for parametric markov models. *International Journal on Software Tools for Technology Transfer*, 2011.

[65] Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. 1994.

[66] Adnan Aziz, Kumud Sanwal, Vigyan Singhal, and Robert Brayton. Model-checking

continuous-time markov chains. *ACM Transactions on Computational Logic (TOCL)*, 2000.

[67] Christel Baier, Boudewijn Haverkort, Holger Hermanns, and Joost-Pieter Katoen. Model-checking algorithms for continuous-time markov chains. *IEEE Transactions on software engineering*, 2003.

[68] Amir Pnueli. The temporal logic of programs. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*, 1977.

[69] Rajeev Alur, Costas Courcoubetis, and David Dill. Model-checking for probabilistic real-time systems. In *International Colloquium on Automata, Languages, and Programming*, 1991.

[70] Joost-Pieter Katoen, Daniel Klink, Martin Leucker, and Verena Wolf. Three-valued abstraction for probabilistic systems. *The Journal of Logic and Algebraic Programming*, 2012.

[71] Joost-Pieter Katoen, Tim Kemna, Ivan Zapreev, and David N Jansen. Bisimulation minimisation mostly speeds up probabilistic model checking. In *International Conference on tools and algorithms for the construction and analysis of systems*, 2007.

[72] Håkan LS Younes and Reid G Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In *International Conference on Computer Aided Verification*. Springer, 2002.

[73] Abraham Wald. Sequential tests of statistical hypotheses. *The annals of mathematical statistics*, 1945.

[74] Ratan Lal, Weikang Duan, and Pavithra Prabhakar. Bayesian statistical model checking for continuous stochastic logic. In *ACM-IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE)*. IEEE, 2020.

[75] Antonio Filieri, Carlo Ghezzi, and Giordano Tamburrelli. Run-time efficient probabilistic model checking. In *Proceedings of the 33rd international conference on software engineering*, 2011.

[76] Tingting Han, Joost-Pieter Katoen, and Damman Berteun. Counterexample generation in probabilistic model checking. *IEEE transactions on software engineering*, 2009.

[77] Christel Baier, Joachim Klein, Sascha Klüppelholz, and Steffen Märcker. Computing conditional probabilities in markovian models efficiently. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2014.

[78] Håkan LS Younes and Reid G Simmons. Statistical probabilistic model checking with a focus on time-bounded properties. *Information and Computation*, 2006.

[79] Tingting Han, Joost-Pieter Katoen, and Alexandru Mereacre. Approximate parameter synthesis for probabilistic time-bounded reachability. In *Real-Time Systems Symposium, 2008*, 2008.

[80] Husain Aljazzar, Holger Hermanns, and Stefan Leue. Counterexamples for timed probabilistic reachability. In *International Conference on Formal Modeling and Analysis of Timed Systems*, 2005.

[81] Husain Aljazzar and Stefan Leue. Extended directed search for probabilistic timed reachability. In *International Conference on Formal Modeling and Analysis of Timed Systems*, 2006.

[82] Michael Benedikt, Rastislav Lenhardt, and James Worrell. Ltl model checking of interval markov chains. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2013.

[83] Serge Haddad and Benjamin Monmege. Reachability in mdps: Refining convergence of value iteration. In *International Workshop on Reachability Problems*, 2014.

[84] Björn Wachter and Lijun Zhang. Best probabilistic transformers. 2010.

[85] Miguel E Andrés, Pedro DArgenio, and Peter van Rossum. Significant diagnostic counterexamples in probabilistic model checking. In *Haifa Verification Conference*, 2008.

[86] Ralf Wimmer, Nils Jansen, Erika Ábrahám, Joost-Pieter Katoen, and Bernd Becker. Minimal counterexamples for linear-time probabilistic verification. *Theoretical Computer Science*, 2014.

[87] Ernst Moritz Hahn, Holger Hermanns, Björn Wachter, and Lijun Zhang. Pass: Abstraction refinement for infinite probabilistic models. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2010.

[88] Di Wu and Xenofon Koutsoukos. Reachability analysis of uncertain systems using bounded-parameter markov decision processes. *Artificial Intelligence*, 2008.

[89] Sergio Giro and Pedro R Dargenio. Quantitative model checking revisited: neither decidable nor approximable. In *International Conference on Formal Modeling and Analysis of Timed Systems*, 2007.

[90] Vojtěch Forejt, Marta Kwiatkowska, David Parker, Hongyang Qu, and Mateusz Ujma. Incremental runtime verification of probabilistic systems. In *International Conference on Runtime Verification*, 2012.

[91] Christian Dehnert, Daniel Gebler, Michele Volpato, and David N Jansen. On abstraction of probabilistic systems. In *Stochastic Model Checking. Rigorous Dependability Analysis Using Model Checking Techniques for Stochastic Systems*. 2014.

[92] Mark Kattenbelt and Michael Huth. Verification and refutation of probabilistic specifications via games. In *LIPIcs-Leibniz International Proceedings in Informatics*, 2009.

[93] Mark Kattenbelt, Marta Kwiatkowska, Gethin Norman, and David Parker. Game-based probabilistic predicate abstraction in prism. *Electronic Notes in Theoretical Computer Science*, 2008.

[94] Holger Hermanns, Björn Wachter, and Lijun Zhang. Probabilistic cegar. In *International Conference on Computer Aided Verification*, 2008.

[95] Marta Kwiatkowska. Model checking for probability and time: from theory to practice. In *Logic in Computer Science, 2003. Proceedings. 18th Annual IEEE Symposium on*, 2003.

[96] Pedro R. D'Argenio, Bertrand Jeannet, Henrik Ejersbo Jensen, and Kim Guldstrand Larsen. Reachability analysis of probabilistic systems by successive refinements. In *Process Algebra and Probabilistic Methods, Performance Modeling and Verification: Joint International Workshop, PAPM-PROBMIV*, 2001.

[97] Peng Dai and Judy Goldsmith. Topological value iteration algorithm for markov decision processes. In *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, 2007.

[98] Andrew Hinton, Marta Kwiatkowska, Gethin Norman, and David Parker. Prism: A tool for automatic verification of probabilistic systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2006.

[99] Miguel E Andrés and Peter Van Rossum. Conditional probabilities over probabilistic and nondeterministic systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2008.

[100] Krishnendu Chatterjee, Martin Chmelík, and Przemysław Daca. Cegar for qualitative analysis of probabilistic systems. In *International Conference on Computer Aided Verification*, 2014.

[101] Rohit Chadha and Mahesh Viswanathan. A counterexample-guided abstraction-refinement framework for markov decision processes. 2010.

[102] Vojtěch Forejt, Marta Kwiatkowska, Gethin Norman, and David Parker. Automated verification techniques for probabilistic systems. In *International School on Formal Methods for the Design of Computer, Communication and Software Systems*, 2011.

[103] Vidyadhar G Kulkarni. *Modeling and analysis of stochastic systems*. Crc Press, 2016.

[104] Peter E Caines. *Linear stochastic systems*, volume 77. SIAM, 2018.

[105] Muhammad Syifaul Mufid, Dieky Adzkiya, and Alessandro Abate. Bounded model checking of max-plus linear systems via predicate abstractions. In *International Conference on Formal Modeling and Analysis of Timed Systems*, 2019.

[106] Maxence Dutreix and Samuel Coogan. Efficient verification for stochastic mixed monotone systems. In *Proceedings of the ACM/IEEE International Conference on Cyber-Physical Systems*, 2018.

[107] Morteza Lahijanian, Sean B Andersson, and Calin Belta. Formal verification and synthesis for discrete-time stochastic systems. *IEEE Transactions on Automatic Control*, 2015.

[108] Mária Svoreňová, Jan Křetínskỳ, Martin Chmelík, Krishnendu Chatterjee, Ivana Černá, and Calin Belta. Temporal logic control for stochastic linear systems using abstraction refinement of probabilistic games. *Nonlinear Analysis: Hybrid Systems*, 2017.

[109] Sadegh Esmaeil Zadeh Soudjani and Alessandro Abate. Aggregation and control of populations of thermostatically controlled loads by formal abstractions. *IEEE Transactions on Control Systems Technology*, 2014.

[110] Morteza Lahijanian, Sean B Andersson, and Calin Belta. Approximate markovian abstractions for linear stochastic systems. In *IEEE Conference on Decision and Control (CDC)*, 2012.

[111] Sasa V Rakovic, Franco Blanchini, Eva Cruck, and Manfred Morari. Robust obstacle avoidance for constrained linear discrete time systems: A set-theoretic approach. In *IEEE Conference on Decision and Control*, 2007.

[112] Shun-ichi Azuma and George J Pappas. Discrete abstraction of stochastic nonlinear systems. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 2014.

[113] Abolfazl Lavaei, Sadegh Esmaeil Zadeh Soudjani, Rupak Majumdar, and Majid Zamani. Compositional abstractions of interconnected discrete-time stochastic control systems. In *IEEE 5Annual Conference on Decision and Control (CDC)*, 2017.

[114] Ameneh Nejati, Sadegh Soudjani, and Majid Zamani. Abstraction-based synthesis of continuous-time stochastic control systems. In *European Control Conference (ECC)*, 2019.

[115] Jianghai Hu, John Lygeros, and Shankar Sastry. Towards a theory of stochastic hybrid systems. In *International Conference on Hybrid Systems: Computation and Control*, 2000.

[116] John Lygeros and Maria Prandini. Stochastic hybrid systems: A powerful framework for complex, large scale applications. 2010.

[117] Christos G Cassandras and John Lygeros. *Stochastic hybrid systems*. CRC, 2006.

[118] Manuela L Bujorianu and John Lygeros. Toward a general theory of stochastic hybrid systems. In *Stochastic hybrid systems*. 2006.

[119] Alessandro Abate, Joost-Pieter Katoen, and Alexandru Mereacre. Quantitative automata model checking of autonomous stochastic hybrid systems. In *Proceedings of the 14th international conference on Hybrid systems: computation and control*, 2011.

[120] Alessandro Abate, Maria Prandini, John Lygeros, and Shankar Sastry. Probabilistic reachability and safety for controlled discrete time stochastic hybrid systems. 2008.

[121] Saurabh Amin, Alessandro Abate, Maria Prandini, John Lygeros, and Shankar Sastry. Reachability analysis for controlled discrete time stochastic hybrid systems. In *International Workshop on Hybrid Systems: Computation and Control*, 2006.

[122] A Agung Julius and George J Pappas. Probabilistic testing for stochastic hybrid systems. In *IEEE Conference on Decision and Control*, 2008.

[123] Martin Fränzle, Ernst Moritz Hahn, Holger Hermanns, Nicolás Wolovick, and Lijun Zhang. Measurability and safety verification for stochastic hybrid systems. In *International conference on Hybrid systems: computation and control*, 2011.

[124] Maria Prandini and Jianghai Hu. A numerical approximation scheme for reachability analysis of stochastic hybrid systems with state-dependent switchings. In *Decision and Control, 2007 46th IEEE Conference on*, 2007.

[125] Stephen Prajna, Ali Jadbabaie, and George J Pappas. Stochastic safety verification using barrier certificates. In *IEEE Conference on Decision and Control*, 2004.

[126] Koichi Kobayashi, Yasuhito Fukui, and Kunihiko Hiraishi. Discrete abstraction for a class of stochastic hybrid systems based on bounded bisimulation. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 2014.

[127] Henk AP Blom, GJ Bakker, and Jaroslav Krystul. Probabilistic reachability analysis for large scale stochastic hybrid systems. In *Decision and Control, 2007 46th IEEE Conference on*, 2007.

[128] Fedor Shmarov and Paolo Zuliani. Probreach: verified probabilistic delta-reachability for stochastic hybrid systems. In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*, 2015.

[129] Tino Teige and Martin Fränzle. Stochastic satisfiability modulo theories for non-linear arithmetic. In *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*, 2008.

[130] Tino Teige, Andreas Eggers, and Martin Fränzle. Constraint-based analysis of concurrent probabilistic hybrid systems: An application to networked automation systems. *Nonlinear Analysis: Hybrid Systems*, 2011.

[131] Lijun Zhang, Zhikun She, Stefan Ratschan, Holger Hermanns, and Ernst Moritz Hahn. Safety verification for probabilistic hybrid systems. In *International Conference on Computer Aided Verification*, 2010.

[132] Martin Fränzle, Holger Hermanns, and Tino Teige. Stochastic satisfiability modulo theory: A novel technique for the analysis of probabilistic hybrid systems. In *Proceedings of the Hybrid Systems: Computation and Control, 11th International Workshop, HSCC*, 2008.

[133] Audemard Gilles, Bozzano Marco, Cimatti Alessandro, and Sebastiani Roberto. Verifying industrial hybrid systems with mathsat. *Electron. Notes Theor. Comput. Sci.*, 2005.

[134] Martin Frnzle and Christian Herde. Efficient proof engines for bounded model checking of hybrid systems. *Electronic Notes in Theoretical Computer Science*, 2005.

[135] Husain Aljazzar, Matthias Kuntz, Florian Leitner-Fischer, and Stefan Leue. Directed and heuristic counterexample generation for probabilistic model checking: a comparative evaluation. In *ICSE Workshop on Quantitative Stochastic Models in the Verification and Design of Software Systems*, 2010.

[136] Rajeev Alur, Thao Dang, and Franjo Ivančić. Counter-example guided predicate abstraction of hybrid systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2003.

[137] Nima Roohi, Pavithra Prabhakar, and Mahesh Viswanathan. Hare: A hybrid abstraction refinement engine for verifying non-linear hybrid automata. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2017.

[138] Nima Roohi, Pavithra Prabhakar, and Mahesh Viswanathan. Hybridization based CEGAR for hybrid automata with affine dynamics. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2016.

[139] Jeremy Sproston. Decidable model checking of probabilistic hybrid automata. In *Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, 2000.

[140] Nikolaj Bjørner, Anh-Dung Phan, and Lars Fleckenstein. νz - an optimizing SMT solver. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2015.

[141] Yi Li, Aws Albarghouthi, Zachary Kincaid, Arie Gurfinkel, and Marsha Chechik. Symbolic optimization with SMT solvers. In *Symposium on Principles of Programming Languages, POPL*, 2014.

[142] Christian Ellen, Sebastian Gerwinn, and Martin Fränzle. Confidence bounds for statistical model checking of probabilistic hybrid systems. In *International Conference on Formal Modeling and Analysis of Timed Systems*. Springer, 2012.

[143] Ratan Lal and Pavithra Prabhakar. Bounded verification of reachability of probabilistic hybrid systems. In *Quantitative Evaluation of Systems QEST*, 2018.

[144] Roberto Segala and Nancy A. Lynch. Probabilistic simulations for probabilistic processes. In *International Conference on Concurrency Theory*, 1994.

[145] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In *International Conference on Computer Aided Verification*, 2000.

[146] Roberto Bagnara, Patricia M. Hill, and Enea Zaffanella. The parma polyhedra library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Sci. Comput. Program.*, 2008.

[147] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. Spaceex: Scalable verification of hybrid systems. In *International Conference on Computer Aided Verification*, 2011.

[148] Armin Biere, Alessandro Cimatti, Edmund M Clarke, Ofer Strichman, Yunshan Zhu, et al. Bounded model checking. *Advances in computers*, 2003.

[149] Edmund Clarke, Armin Biere, Richard Raimi, and Yunshan Zhu. Bounded model checking using satisfiability solving. *Formal methods in system design*, 2001.

[150] Armin Biere, Alessandro Cimatti, Edmund M Clarke, Ofer Strichman, and Yunshan Zhu. Bounded model checking. *Handbook of satisfiability*, 2009.

[151] Steven Diamond and Stephen Boyd. Cvxpy: A python-embedded modeling language for convex optimization. *The Journal of Machine Learning Research*, 2016.

[152] Alessandro Abate, Joost-Pieter Katoen, John Lygeros, and Maria Prandini. Approximate model checking of stochastic hybrid systems. *European Journal of Control*, 2010.

[153] Zhi-Quan Luo, Wing-Kin Ma, Anthony Man-Cho So, Yinyu Ye, and Shuzhong Zhang.

Semidefinite relaxation of quadratic optimization problems. *IEEE Signal Processing Magazine*, 2010.

[154] Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.