FORMAL METHODS FOR SAFE AUTONOMY: DATA-DRIVEN VERIFICATION, SYNTHESIS, AND APPLICATIONS

BY

CHUCHU FAN

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2019

Urbana, Illinois

Abstract

Autonomous systems are often safety-critical and are expected to work in uncertain environments. Ensuring design correctness and safety of autonomous systems has significant financial and legal implications. Existing design and test methodologies are inadequate for providing the needed level of safety assurances. Can formal methods provide certifiable trust or assurance for products with the vagaries of real-world autonomous systems? In this dissertation, we try to answer this question in the affirmative by developing new verification and synthesis algorithms, implementing them in software tools, and studying their performance on realistic applications.

Verification and synthesis for typical models of real-world autonomous systems are well known to be theoretically undecidable, and approximate solutions are challenging due to their high dimensionality, nonlinearities, and their nondeterministic and hybrid nature. In addressing these challenges, we present (a) data-driven algorithmic verification via reachability analysis of nonlinear hybrid systems and (b) controller synthesis for high-dimensional linear systems under disturbances. The key technical developments within this theme include: (1) The first algorithm for over-approximating reach sets of general nonlinear models with locally optimal tightness guarantees. (2) A novel verification framework that can tackle systems with incomplete models, by treating them as combinations of a "white-box" control graph and "black-box" simulators. We also provide a learning-based verification algorithm to provide probabilistic guarantees. (3) An approximate partial order reduction method to exponentially reduce the number of executions to be explored for reachability analysis of nondeterministic models of distributed systems. (4) An algorithm to find a combined open-loop controller and tracking controller for high-dimensional linear systems to meet reach-avoid specifications. On the theoretical front, the techniques are armed with soundness, precision, and relative completeness guarantees. On the experimental side, we show that the techniques can be successfully applied on a sequence of challenging problems, including a suite of Toyota engine control models verified for the first time, satellite control systems, and autonomous driving and ADAS-based maneuvers.

To my son Jayden for brightening up my days with his smile.

Acknowledgments

In retrospect, I had never expected my six-year Ph.D. life to be such a wonderful experience. I am grateful for receiving a lot of help from my colleagues, friends, and family. First of all, I am truly lucky to have had Prof. Sayan Mitra as my advisor. His enthusiasm, dedication, creativity, and perfectionism about research have shaped my personality as a researcher. Sayan not only gives me a lot of guidance and inspiration as an advisor, but also understands and supports me as a friend. None of my achievements during graduate school, including this dissertation, would be possible without Sayan's help.

I am grateful to have the rest of my doctoral committee: Prof. Mahesh Viswanathan, Prof. Richard Murray, Prof. Daniel Liberzon, and Prof. William Sanders. They have provided invaluable insights, feedback, and suggestions for my dissertation. More importantly, through collaboration and communication with my doctoral committee, I got to learn how extraordinary researchers think and solve problems.

This dissertation benefited a lot from my research collaborations with Dr. Zhenqi Huang, Dr. Xiaoqing Jin, Dr. James Kapinski, Dr. Parasara Sridhar Duggirala, Mr. Bolun Qi, Mr. Yangge Li, and Mr. Umang Mathur, who will become a doctor soon. I would also like to thank my other reach collaborators during my Ph.D., including Prof. Marta Kwiatkowska, Prof. Ulrich Schmid, Prof. Ezio Bartocci, Prof. Scott Smolka, Prof. Jyotirmoy Deshmukh, and Mr. Yu Meng, who will also become a doctor in the near future.

Many thanks to Ritwika Ghosh and Hussein Sibaie, for being awesome labmates who share with me numerous research ideas, give me constant feedback and encourage me whenever I lose confidence. I am also very honored to spend a great time with my colleagues in the Coordinate Science Laboratory (CSL): Chiao Hsieh, Tianqi Liu, Dawei Sun, Navid Mokhlesi, Kristina Miller, Rongzhou Li, Sheng Shen, Nirupam Roy, Kaiqing Zhang, just to name a few.

I am very thankful to my close research friends, who are also alumni of UIUC: Dr. Stanley Bak, Prof. Taylor Johnson, and Dr. Yu Wang. These research friends gave me continuous instructions on moving forward on my research road. I am grateful to the ECE Editorial Services, particularly James Hutchinson, for carefully proofreading this dissertation. I would also like to express my appreciation to the National Science Foundation and Siebel Energy Institute for funding my position as a graduate research assistant.

I would like to take this opportunity to thank my long-time friends: Yichi Zhang, Zhenming Xiong, Fumeng Wang, and Nan Sun. They knew me back in the day when I had nothing but dreams. We lived on different continents and barely saw each other over the past six years. But I knew when I wanted to talk, especially in hard times, they were always there.

I am greatly indebted to my husband Qiang. Qiang can become anyone he wants to be, but he chooses to be a great husband and always gives up a lot to support my dreams. I am particularly grateful to my parents for their unconditional love. My parents have literally taken care of every aspect of my and my son's life during the past few years, and especially when I was working on my dissertation. Thanks to my son Jayden for being a constant source of happiness. The world is full of uncertainties, and that is why I work on verification. Luckily, I know your love to me is beyond verification.

Table of Contents

List of Abbreviations and Math Notations

ADAS	Advanced Driving Assist System
AEB	Automatic Emergency Brake
ASIL	Automotive Safety Integrity Level
CPS	Cyber-physical System
DAG	Directed Acyclic Graph (DAG)
LP	Linear Programming
LQR	Linear Quadratic Regulator
MILP	Mixed Integer Linear Programming
ML	Machine Learning
MPC	Model Predictive Control
ODE	Ordinary Differential Equation
PAC	Probably Approximately Correct
POR	Partial Order Reduction
QP	Quadratic Programming
SDP	Semi-definite Programming
SMT	Satisfiability Modulo Theory
$\mathbb{B}$	The set of Boolean values
$\mathbb{N}$	The set of natural numbers
$\mathbb{R}$	The set of all real numbers
$\mathbb{R}^{\geq 0}$	The set of all nonnegative real numbers

$\|x\|_p$	The norm of a vector x
$\|A\|_p$	The induced p-norm of a matrix A
$\lambda_{\max}(A)$	The maximum eigenvalue of a symmetric matrix A
$\lambda_{\min}(A)$	The minimum eigenvalue of a symmetric matrix A
$\mu(A)$	The matrix measure of a matrix A
$\rho(A)$	The spectral radius of a matrix A
$\mathbb{A}$	An interval matrix $\mathbb{A}$
$\mathtt{VT}(\mathbb{A})$	Vertex matrices of an interval matrix $\mathbb{A}$
$S_1 \oplus S_2$	The Minkowski sum of two sets S_1 and S_2
$M \otimes S$	The coordinate transformation of a set S using a matrix M
$\mathtt{hull}(S_1, S_2)$	The convex hull of two sets S_1 and S_2
$S \lceil k$	Restrict a set S to the k^{th} component of each element
$\mathcal{A}$	Automaton
$\mathcal{H}$	Hybrid system

Chapter 1

Introduction

Safe autonomous systems are entering many aspect of our lives: robots and drones are helping with rescuing and delivering; medical devices are being implanted for health monitoring and drug delivery; self-driving cars are expected to improve road safety and efficiency. Designers of these complex systems cannot foresee all corner-cases that may arise in the field, and a single design defect can wreak havoc across thousands of deployed instances. Existing design and test methodologies are inadequate for providing the needed level of safety assurances.

Current approaches to ensure safety mainly rely on large-scale simulations and field tests, in an iterative bug-fixing process. There are two problems with that brute-force methodology: cost and coverage. Both the environment and the autonomous systems have many uncertainties. Simply sampling from those uncertainties will result in a set of combinatorial choices and simulating every scenario in the set is expensive; field tests are even more so. For example, it is estimated that billions of miles of test-driving is necessary in order to reduce the catastrophic failure rates to less than one per hour [1]. This figure is prohibitive even for large corporations. Second, it is inevitable that certain scenarios will remain untested, particularly those outside the so-called *operational design domains* [2]. Such uncertainties grow as systems get equipped with machine learning (ML) algorithms, and the test coverage problem gets exacerbated.

Rigorous approaches based on formal methods and control theory, deployed properly, can in principle be the first line of defense against design bugs making their way into unsafe products [3]. They could transform the conventional trial-and-error paradigm and improve safety in autonomous systems. Rigorous approaches can, in principle, generate provably correct decision systems, provide safety guarantees, and perform root-cause analyses. The common claim that formal techniques do not scale beyond academic problems has been countered by recent results [4, 5, 6], including works [4, 7] that are part of this dissertation. In this dissertation, we present several computationally effective formal techniques that can provide useful coverage at an acceptable cost for designing and analyzing complex real-world autonomous systems.

Terminology The notion of autonomous systems is used in two different ways in this dissertation. Mathematically, an autonomous system is defined as a general (possibly nonlinear) dynamical system system with no external inputs [8]. Alternatively, an autonomous system is a system without human assistance, such as an autonomous car or robot where hardware and software work together to solve problems by performing actions. It should be clear from the context which one is referred to. The integration of computation with physical processes can also be captured by *cyber-physical systems* (CPS), which tightly couple physical processes with software, networks, and sensing. *Requirements*, also called *specifications* or *properties* of an autonomous system, say something about the desired behaviors or performance of the system. The mathematical model for autonomous systems may be a dynamical [8], switched [9], or a hybrid system [10]. The former one, *dynamical systems*, are defined as a system of *ordinary differential equations* (ODE) without explicit independent variable [11]. The later ones, *switched systems* and *hybrid systems*, exhibit both the continuous evolution of the states that describe the dynamics of physical processes and the discrete changes of the states that describe the possible states of a computation [12, 13, 14]. The requirement of an autonomous system may be a safety property, a stability property, or a temporal logic property [15, 9].

Within the broad area of safe autonomy and CPS, the techniques presented in this dissertation span two high-level themes: *formal verification* and *control synthesis*.

Verification. A formal verification algorithm takes as input an autonomous system's model and a requirement, and decides whether or not all the behaviors of the system meet the requirement. If decision is 'yes', the algorithm provides a supporting proof of this fact, which can then be used for certification, documentation, and for future testing and maintenance. If the decision is 'no', the algorithm produces a supporting counter-example or a 'bug trace'. This is a particular behavior of the systems resulting from specific initial states and inputs, that violates the requirement.

Synthesis. The controller synthesis question asks whether an input can be generated for a given system (or a plant) so that it achieves a given specification. Algorithms for answering this question hold the promise of automating controller design. They have the potential to yield high-assurance systems that are *correct-by-construction*, and even negative answers to the question can convey insights about unrealizability of specifications.

Challenges. Verification and synthesis problems are challenging and known to be theoretically undecidable for typical real-world models [16, 17]. Significant progress has been made in the last decade and many powerful tools have been developed to solve approximate versions of these problems for specific model classes [18, 19, 20, 21, 22, 23]. However, these existing techniques do not scale to nonlinear and hybrid models that arise in practice due to intractable peculiarities of CPS, including nonlinear dynamics, networked structures, the non-deterministic and hybrid nature of models, and sometimes even the absence of complete models.

Dissertation theme. Unlike testing that cannot prove absence of bugs, verification provides a mathematical proof of the safety of all possible (often infinitely many) behaviors of a system. However, purely model-based verification techniques cannot directly handle real-world autonomous systems that have nonlinear and hybrid models, or systems for which we lack complete or precise models. In this dissertation, we aim to address the above challenges using a *data-driven verification* method. Data-driven algorithms use executions (or numerical simulations) of the model in addition to statically analyzing the model itself. Thus the verification algorithm can use powerful numerical simulators as subroutines, which is particularly relevant for nonlinear models that do not permit a closed form analytical solution. This also opens the door to verifying autonomous systems without complete and precise models.

The basic approach of data-driven verification is to combine executions of the model with model-based *reachability analysis*, by performing effective *sensitivity analysis* for the complex or unknown components of the system. Such a sensitivity analysis gives probabilistic or worst-case bounds on how much the states or outputs of a module will change, with small changes in the input. In data-driven verification, we will use sensitivity analysis to generalize an individual execution to a set of reachable states, and then verify the requirement on this generalized set. Sensitivity analysis is the key principle underlying the rigorous soundness (i.e., the result returned by the algorithm is correct) and completeness (i.e., the algorithm always terminates and returns the correct result) guarantees of the data-driven verification approach. In this dissertation, we introduce sensitivity analysis techniques that can advance the verification or synthesis of

(1) autonomous systems with nonlinear dynamics [24] (Chapter 5),

(2) hybrid systems with incomplete models [25] (Chapter 6),

(3) distributed autonomous systems whose components take actions concurrently [26] (Chapter 7), and

(4) correct-by-construction controllers for high-dimensional linear dynamical systems to meet reach-avoid specifications [27] (Chapter 8).

Next, we give a more detailed introduction of each of the four parts, followed by a summary of contributions and impacts.

1.1 Locally Optimal Guaranteed Reachability for Nonlinear Dynamics

Obtaining an exact solution for autonomous systems with nonlinear dynamics is often impossible. Nevertheless, approximate solutions for such dynamics may be too conservative or computationally expensive to be useful. Therefore, it is crucial for verification methods to perform tight yet computationally efficient over-approximations of the set of reachable states.

In this dissertation, we show that the sensitivity of a nonlinear dynamical system can be bounded by *matrix measures* of the ODEs. Such sensitivity can then be used to generalize finite simulation executions of the system to get the over-approximations of the set of reachable states. Moreover, we prove that using matrix measures we can compute over-approximations whose sizes change over time with the minimum exponential rate on local compact sets.

Based on this, we introduce the first algorithm that can provide locally optimal guarantees for computing tight reachable sets over-approximations for nonlinear models. Furthermore, we show that the matrix measures of a nonlinear system can be computed via semi-definite programming (SDP). This allows us to develop effective algorithms for automatic construction of the tightest reachable sets.

In the first technical part of this dissertation, we develop sensitivity analysis algorithms for nonlinear and hybrid systems with known models. These techniques are implemented in the C2E2 tool [28], which has been effectively used to verify an engine control system [4], a NASA-developed collision alerting protocol [29], and satellite controllers [30, 31]. The approach can also be extended to conduct compositional analysis for large-scale autonomous systems in which multiple components communicate with each other. Using the compositional analysis, we can compute the reachable sets of a suite of challenging pacemaker-heart models that have as many as 20 continuous variables and 29^5 discrete modes [32]. The compositional analysis is beyond the scope of this dissertation and we refer the interested readers to [33].

1.2 Verification of Black-box Components with Probabilistic Guarantees

Many autonomous systems are a heterogeneous mix of simulation code, differential equations, block diagrams, and handcrafted look-up tables, with the increasing presence of machine learning modules. It is sometimes impossible even to model a system completely and precisely in the first place. Therefore, a barrier for applicability of verification tools is the unavailability of precise and complete models.

To overcome this issue, in this dissertation, we introduce a novel verification framework DryVR [25, 34] that treats the system as a combination of a "white-box" control graph and "black-box" simulators. For systems with unknown models, the deterministic sensitivity analysis algorithms have to be replaced with methods that only rely on execution data. Using the *probably approximately correct* (PAC) learning principle [35], we show that sensitivity analysis could be formulated as the well-known problem of learning a linear separator, and could thus be solved with probabilistic correctness guarantees. To achieve an error of ε with probability $1 - \delta$, the number of samples the algorithm needs from the black-box simulator is only $\frac{1}{\varepsilon} \log \frac{1}{\delta}$. That approach can achieve the same level of probabilistic guarantees as testing, in significantly less time.

We have used DryVR to verify a wide range of applications, including autonomous driving maneuvers, powetrain control systems, and automatic transmission control systems [25]. We also use it to conduct risk analysis of an ADAS system to determine its ASIL [36]. Moreover, DryVR has been incorporated to enhance other research works, for example, to verify spacecraft rendezvous [37] and perform on-line monitoring on autonomous vehicles.

1.3 Approximate Partial Order Reduction for Distributed Autonomous Systems

In distributed autonomous systems, components take actions concurrently. Therefore, in considering all possible behaviors of a system, there is a combinatorial explosion in the total number of action sequences due to the interleavings of each individual system's concurrent actions. Partial order reduction (POR) methods tackle this combinatorial explosion by eliminating executions that are *equivalent*, i.e., do not provide new information about reachable states. This equivalence is based on *independence* of actions: a pair of actions are independent if they commute, i.e., applying them in any order results in the same state. Thus, of all execution branches that start and end at the same state, but perform commuting actions in different order, only one has to be explored.

Existing (POR) methods are limited when it comes to computations with numerical data and physical quantities since reduction is allowed only when actions can commute exactly. In this dissertation, we develop an approximate POR method that allows actions to be *nearly commutative*. That is, we allow the reachable states resulting from applying action sequences in different orders to have some distance with each other. The resulting algorithm computes the reachable sets for such systems where nondeterminism arises from both the choice of the initial state and the choice of actions. It reduces the number of action sequences that must be explored in safety analysis by a factor of $O(t!)$ with t being the time steps [26].

We have implemented the approximate POR method and used it to analyze a set of benchmark models including an iterative consensus protocol, a simple multi-car platoon, and a distributed room heating system. Results show that the proposed reachability algorithm using approximate POR can achieve the $O(t!)$ reduction on the number of explored executions, compared with exhaustive enumeration. These preliminary results have shown promise for exponentially expediting the safety analysis of distributed systems.

1.4 Control Synthesis of Large-dimensional Systems

In the last technical part of this dissertation, we study the synthesis problem for linear time-varying plant models with bounded disturbance—a standard view of control systems [38, 39]. We will consider *reach-avoid* specifications which require that starting from any initial state Θ, the controller has to drive the system to a target set G, while avoiding certain unsafe states or obstacles O. *Reach-avoid* specifications arise naturally in many domains such as autonomous and assisted driving, multi-robot coordination, and spacecraft autonomy, and have been studied for linear, nonlinear, as well as stochastic models [40, 41, 42, 43].

Current control synthesis approaches for the above problem suffer from poor scalability: They normally end up solving a nonlinear or mixed-integer optimization problem, or facing the curse of dimensionality. In this dissertation, we will study a novel approach that finds a *state feedback controller* for time-varying linear systems under disturbances [27]. The instrumental idea is to use a combination of an *open-loop controller* and a *tracking controller* to reformulate the overall synthesis problem as a satisfiability problem over quantifier-free linear real arithmetic, which can be efficiently solved by off-the-shelf SMT solvers. The number of constraints for the satisfiability problem is only linear to the total number of surfaces in the obstacles as polytopes. Moreover, we prove that the proposed approach is sound and complete—a theoretical guarantee that is beyond most conventional synthesis methods.

We have implemented this synthesis approach tool in a tool RealSyn and used it to analyze a suite of case studies including multi-robot and multi-car motion planning and platooning. RealSyn shows encouraging results: it finds controllers within seconds for systems with up to 20 state variables to satisfy reach-avoid specification with more than 7 static or dynamic obstacles.

1.5 Summary of Contributions

Scalable verification and synthesis techniques are of fundamental importance to ensuring autonomous systems' safety. The objective of this dissertation is to push the theoretical limits of verification and synthesis techniques for complex autonomous systems and CPS, and develop algorithms and tools with theoretical guarantees like soundness, precision, and completeness. We give a brief overview of the key contributions of this dissertation below.

1. Advancement of the state-of-the-art on verification of CPS by developing a data-driven safety verification algorithm through reachability analysis for nonlinear hybrid systems and infinite transitions systems. This data-driven algorithm is locally optimal in data usage [24] for nonlinear systems, and exponentially reduces the number of executions that need to be explored for infinite transition systems. Therefore, it allows us to verify large models that were previously intractable [26, 44] (Chapter 5 and Chapter 7).

2. Development of the first framework for verifying real-world CPS for which certain parts may not have precise mathematical models [25]. The key idea is to see such systems as a white-box automaton with embedded black-box simulators. With this new view, our verification approach can bring together worst-case formal reasoning on the automaton with probabilistic reasoning on the black-boxes (Chapter 6).

3. Development of an algorithm that significantly improves the practical efficiency of control synthesis for large linear systems with disturbances [27]. The algorithm achieves scalability by reducing the synthesis problem to satisfiability over quantifier-free linear arithmetic and leveraging modern SMT solvers (Chapter 8).

On the practical side, we have developed software tools for these techniques: C2E2 [28] (for verification of hybrid systems), DryVR [34] (for verification of systems with black-box components), and RealSyn [27] (for synthesis). These tools have shown their ability to handle complex real-world systems. For instance, C2E2 was the first tool to successfully verify the Toyota powertrain control system and a spacecraft rendezvous problem (Section 5.5.1 and Section 6.6); currently, it is also the only tool that can handle highly nonlinear models such as mixed-signal circuits [7]. DryVR has been successfully used in verifying autonomous driving maneuvers (Section 6.4) and determining automotive safety integrity levels (ASIL) based on risk analysis of an advanced driver-assistance system (ADAS) feature (Section 6.5). Other successful applications of the tools range across medical devices [45, 46], air-traffic management [29], and energy systems [47].

1.6 Dissertation Structure

We introduce mathematical notations and preliminary results in Chapter 2. In Chapter 3, we discuss three different mathematical models for autonomous systems and CPS. We present a broad and unified overview of the data-driven verification approach and the related sub-problem of sensitivity analysis in Chapter 4. The existing techniques for sensitivity analysis are described in the context of dynamical systems and hybrid systems in Chapter 5. In Chapter 6, we discuss the black-box verification as in DryVR. In Chapter 7, we discuss the approximate partial order reduction method for distributed systems. In Chapter 8, we present details of the controller synthesis approach. Finally, in Chapter 9, we conclude with a short summary.

Chapter 2

Mathematical Preliminaries

In this chapter, we introduce definitions, notations, operations, and background results on vectors, matrices, sets, and functions, which will be used throughout the dissertation.

2.1 Vectors and Matrices

2.1.1 Vectors and Vector Norms

Let us denote the set of all real numbers by $\mathbb{R}$, the set of non-negative real numbers by $\mathbb{R}^{\geq 0}$, the set of Boolean values by $\mathbb{B}$, and the set of natural numbers by $\mathbb{N}$.

The n-dimensional *Euclidean space*, denoted by $\mathbb{R}^n$, is defined by the set of all n-dimensional vectors $x = [x^{(1)}, \cdots, x^{(n)}]^\top$, where $x^{(1)}, \cdots, x^{(n)} \in \mathbb{R}$, $\top$ means transpose, and $x^{(i)}$ is the i^{th} entry of x.

The *norm* $\|x\|$ of a vector x is a nonnegative-valued scalar function satisfying the following properties [8]:

1. Positive definite: $\|x\| \geq 0, \forall x \in \mathbb{R}^n$, $\|x\| = 0$ if and only if $x = 0$.
2. Absolutely scalable: $\|\alpha x\| = |\alpha| \|x\|, \forall c \in \mathbb{R}$ and $x \in \mathbb{R}^n$, where $|\cdot|$ means the absolute value.
3. Triangle inequality: $\|x + y\| \leq \|x\| + \|y\|, \forall x, y \in \mathbb{R}^n$.

The p-norm of a vector is defined as

$$\|x\|_p = \left(|x^{(1)}|^p + \cdots |x^{(n)}|^p\right)^{\frac{1}{p}}, 1 \leq p < \infty,$$

and $\|x\|_\infty = \max_i |x^{(i)}|$.

Frequently used norms are 1, 2 and ∞ norms. They are equivalent up to a constant scaling in the sense they satisfy the following inequalities [48]:

$$\|x\|_\infty \le \|x\|_2 \le |x\|_1 \le \sqrt{n}\|x\|_2 \le n\|x\|_\infty.$$

A *linear transformed* norm of a vector x is defined as: $\|x\|_M = \|Mx\|$, where $M \in \mathbb{R}^{n\times n}$ is a nonsingular matrix which defines the linear coordinate transformation. We call it the *M-norm* of the vector x. The norm of the vector x would be different under different coordinates. Hereafter, if not specifically stated otherwise, $\|x\|_M$ refers to the linear transformed 2-norm:

$$\|x\|_M = \sqrt{x^\top M^\top M x}.$$

2.1.2 Matrices and Matrix Norms

For any matrix $A \in \mathbb{R}^{n\times m}$, $A^\top$ is its *transpose*; $A^{(i)}$ is the i^{th} row of A and $A^{(i,j)}$ denotes the entry in the i^{th} row and j^{th} column.

For a square matrix $A \in \mathbb{R}^{n\times n}$, the *spectral radius* $\rho(A)$ is the largest absolute value of its eigenvalues. A square matrix A is *stable* if its spectral radius $\rho(A) < 1$.

We call a real square matrix $A \in \mathbb{R}^{n\times n}$ *symmetric* if $A = A^\top$. A real symmetric matrix A's eigenvalues are all real numbers, and we use $\lambda_{\max}(A)$ and $\lambda_{\min}(A)$ to denote its maximum and minimum eigenvalues respectively. A symmetric matrix $A \in \mathbb{R}^{n\times n}$ is said to be positive (negative) semi-definite if the scalar $x^\top Ax$ is nonnegative (nonpositive) for every non-zero column vector $x \in \mathbb{R}^n$. For symmetric matrices A and B, the inequality $A \preceq B$ ($A \succeq B$) means that $B - A$ is *positive (negative) semi-definite* and $A \prec B$ ($A \succ B$) means $B - A$ is positive (negative) definite.

The induced p-norm of the matrix A is defined as

$$\|A\|_p = \sup_{x\neq 0} \frac{\|Ax\|_p}{\|x\|_p}.$$

For $p = 1, 2, \infty$, the p-norm is given by

$$\|A\|_1 = \max_j \textstyle\sum_{i=1}^m |A^{(i,j)}|, \|A\|_2 = \sqrt{\lambda_{\max}(A^\top A)}, \|A\|_\infty = \max_i \textstyle\sum_{j=1}^n |A^{(i,j)}|.$$

In the rest of the dissertation, if not specifically claimed otherwise, $\|A\|$ also refers to the 2-norm of A. The matrix norm also obeys some inequalities. If matrices $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{n\times l}$ are real valued matrices, then

$$\begin{aligned}
&\frac{1}{\sqrt{n}}\|A\|_\infty \leq \|A\|_2 \leq \sqrt{m}\|A\|_\infty, && \frac{1}{\sqrt{m}}\|A\|_1 \leq \|A\|_2 \leq \sqrt{n}\|A\|_1,\\
&\|A\|_2 \leq \sqrt{\|A\|_1\|A\|_\infty}, && \|AB\|_p \leq \|A\|_p\|B\|_p.
\end{aligned}$$

2.1.3 Interval Matrices

Given any two real numbers $a < b$, $[a, b]$ is is defined as the interval $\{x \in \mathbb{R} \mid a \leq x \leq b\}$. When each entry is an interval instead of a constant scalar value, we call the matrix an *interval matrix*. The interval matrices are used to locally approximate a nonlinear model's behaviors.

Given two matrices $B, C \in \mathbb{R}^{n\times m}$, if $B^{(i,j)} \leq C^{(i,j)}$ for all $1 \leq i \leq n, 1 \leq j \leq m$, we call $[B, C]$ a *matrix pair*. For a matrix pair $[B, C]$, we define the *matrix interval*,

$$\texttt{IntV}([B, C]) \triangleq \{A \in \mathbb{R}^{n\times n} | B^{(i,j)} \leq A^{(i,j)} \leq C^{(i,j)}, 1 \leq i \leq n, 1 \leq j \leq m\}.$$

We call $\mathbb{A} = \texttt{IntV}([B, C])$ an interval matrix. When a constant matrix V's entries are either the upper bound or the lower bound of the corresponding entries of an inverval matrix $\mathbb{A}$, V is called a *vertex matrix* of $\mathbb{A}$. Formally, we define the set of vertex matrices of an interval matrix $\mathbb{A} = \texttt{IntV}[B, C]$ as

$$\begin{aligned}
&\texttt{VT}(\texttt{IntV}([B, C])) =\\
&\{V \in \mathbb{R}^{n\times n} | V^{(i,j)} = B^{(i,j)}, \text{or, } V^{(i,j)} = C^{(i,j)}, 1 \leq i \leq n, 1 \leq j \leq m\}.
\end{aligned}$$

The cardinality of $\texttt{VT}(\texttt{IntV}([B, C]))$ is 2^{nm}.

Given a sequence of matrices $A_i \in \mathbb{R}^{n\times m}$, $i = 1, 2, \ldots, N$, the convex hull of all A_is is defined as:

$$\begin{aligned}
\texttt{hull}(\{A_1, \ldots, A_N\}) \triangleq\ & \{A \in \mathbb{R}^{n\times m} | \exists \alpha_1, \ldots, \alpha_N \geq 0, \text{ and}\\
& \sum_{i=1}^{N} \alpha_i = 1, \text{ s.t. } A = \sum_{i=1}^{N} \alpha_i A_i\}.
\end{aligned}$$

It can be shown that the convex hull of the vertex matrices for an interval matrix $\mathbb{A}$ is the interval matrix itself.

Proposition 2.1. For any interval matrix $\mathbb{A}$, $\texttt{hull}(\texttt{VT}(\mathbb{A})) = \mathbb{A}$.

Proof. This proposition can be proved by constructing a bijection that maps an $n \times n$ square interval matrix to an n^2-dimensional hyper-rectangle. Vectorizing, or *flattening*, the vertex matrices in $\mathbb{A}$ to n^2-dimensional vectors, we obtain the vertices of this hyper-rectangle. Since the convex hull of the vertices of a rectangle is the rectangle itself, we have $\texttt{hull}(\texttt{VT}(\mathbb{A})) = \mathbb{A}$. $\square$

2.1.4 Matrix Measures

The *matrix measure*, also known as the *logarithmic norm*, is a real-valued function on square matrices. Matrix measure has been used to bound the growth rate of the solutions to differential equations. Letting $A \in \mathbb{R}^{n\times n}$ be a square matrix and $\|\cdot\|$ be any matrix induced norm, the matrix measure $\mu(A)$ can be seen as the one-sided derivative of $\|\cdot\|$ at the identity matrix $I \in \mathbb{R}^{n\times n}$ in the direction of A:

$$\mu_p(A) = \lim_{t\to 0^+} \frac{\|I + tA\|_p - \|I\|_p}{t}. \tag{2.1}$$

Some commonly seen induced matrix measures are [49]:

$$\begin{aligned} \mu_1(A) &= \max_j \left(A^{(j,j)} + \textstyle\sum_{i\neq j} |A^{(i,j)}|\right), \\ \mu_2(A) &= \max_j \tfrac{1}{2}\left(\lambda_j(A + A^T)\right), \\ \mu_\infty(A) &= \max_i \left(A^{(i,i)} + \textstyle\sum_{j\neq i} |A^{(i,j)}|\right). \end{aligned}$$

In the rest of the dissertation, if not specifically claimed otherwise, $\mu(\cdot)$ means the matrix measure can be correspond to any induced matrix norm. Some useful results about matrix measures that will be used in this dissertation are summarized in Lemma 2.1 and Lemma 2.2, which are results proved in [50] and [51].

Lemma 2.1. For any $A \in \mathbb{R}^{n\times n}$, $\mu(A)$ is well defined.

Lemma 2.2. Let $A \in \mathbb{R}^{n\times n}$; then

1. $-\|A\| \leq -\mu(-A) \leq \mu(A) \leq \|A\|$.
2. $\mu(cA) = c\mu(A), \forall c \geq 0$.
3. If $M \in \mathbb{R}^{n \times n}$ is nonsingular, then the measure μ_M of the norm $\|x\|_M = \|Mx\|$ is given in terms of μ by $\mu_M(A) = \mu(MAM^{-1})$.

2.2 Sets

A subset $S \subset \mathbb{R}$ is said to be *bounded* if there exists a constant $c > 0$ such that for all $x \in S$, we have $\|x\| \leq c$. A subset $S \subset \mathbb{R}$ is said to be *open* if for $\forall x \in S$, we can find an arbitrary small neighborhood of x, $B_\epsilon(x) = \{y \in \mathbb{R}^n \mid \|y - x\| < \epsilon\}$ such that $B_\epsilon \subset S$. A set S is said to be *closed* if its complement in $\mathbb{R}^n$ is open. A set S is said to be *compact* if it is closed and bounded. The radius of a compact set S is defined as $\mathsf{Rad}(S) = \sup_{x,y \in S} \|x - y\|/2$. $|S|$ denotes the cardinality of S.

We will use the following sets to represent the set of initial states, reachable states, bounded inputs, and disturbances in the later part of the dissertation.

- Ball: Given a $r \geq 0$, an r-ball around $x \in \mathbb{R}^n$ is defined as $B_r(x) = \{x' \in \mathbb{R}^n \mid ||x' - x||_2 \leq r\}$. We call r the radius of the ball.

- Ellipsoid: Given an invertible matrix $M \in \mathbb{R}^{n \times n}$ and $r \geq 0$, a (M, r)-ellipsoid around $x \in \mathbb{R}^n$ is defined as $E_r(x, M) = \{x' \in \mathbb{R}^n \mid ||x' - x||_M \leq r\}$. We call M the shape of the ellipsoid and r the radius of the ellipsoid. When $M = I$, $E_r(x, M) = B_r(x)$.

- Rectangle: Given two vectors $x, y \in \mathbb{R}^n$ with $x^{(i)} \leq y^{(i)}, \forall i = 1, \cdots, n$, a (x, y)-rectangle is defined as $R(x, y) = \{x' \in \mathbb{R}^n \mid x^{(i)} \leq x'^{(i)} \leq y^{(i)}, \forall i = 1, \cdots, n\}$. The vectors x and y define the lower and upper bounds of the rectangle in each dimension. When x and y are clear from the context, we will drop x, y and denote the rectangle by R.

- Polytope: Given a matrix $A \in \mathbb{R}^{k \times n}$ and a vector $b \in \mathbb{R}^k$, an (A, b)-polytope is defined as $P(A, b) = \{x \mid Ax \leq b\}$. A rectangle is also a convex polytope.

Next, we will introduce several operations on sets.

- Minkowski sum: The Minkowski sum of two sets S_1 and S_2 is defined to be the addition of each element in S_1 and each element in S_2. We use $\oplus$ to denote the Minkowski sum, so $S_1 \oplus S_2 = \{x + y \mid x \in S_1, y \in S_2\}$.

- Coordinate transformation: For a set $S \subseteq \mathbb{R}^n$ and a matrix $M \in \mathbb{R}^{n\times n}$, we define $M \otimes S \triangleq \{Mx \mid x \in S\}$.

- Convex hull: For sets $S_1, S_2 \subset \mathbb{R}^n$, the convex hull of S_1 and S_2 is the smallest convex set that contains S_1 and S_2, and we denote it by $\texttt{hull}(S_1, S_2)$.

- Bloating: For a set $S \subseteq \mathbb{R}^n$, the set S bloated by r is defined as $B_r(S) = \cup_{x\in S} B_r(x)$. It is easy to see that $B_r(S)$ is the Minkowski sum of S and the ball $B_r(0)$, so $B_r(S) = S \oplus B_r(0)$.

- Restrict: For a set of tuples $S = \{\langle s_{i1}, \ldots, s_{in}\rangle_i\}$, $S \lceil k$ denotes the set $\{s_{ik}\}$ which is the set obtained by taking the k^{th} component of each tuple in S.

2.3 Functions

Let $f : S_1 \to S_2$ denote a function f which maps a set S_1 to a set S_2. S_1 is called the domain of f and is denoted by $\mathsf{dom}(f)$. S_2 is called the range of f and is denoted by $\mathsf{range}(f)$.

Given two functions f, g such that $\mathsf{range}(g) \subseteq \mathsf{dom}(f)$, the composition of f and g is $f \circ g : \mathsf{dom}(g) \to \mathsf{range}(f)$ such that $f \circ g(x) = f(g(x))$. For any functions f such that $\mathsf{range}(f) \subseteq \mathsf{dom}(f)$, for any $n \in \mathbb{N}$, we define the nested form f^n as $f^n = f \circ f^{n-1}$, with f^0 being the identity mapping.

A function f is *uniformly continuous* if $\forall \epsilon > 0, \exists \delta > 0$ such that $\forall \|x - y\| < \delta \Rightarrow \|f(x) - f(y)\| < \epsilon$. The δ here is independent of x, but only depends on ϵ. A continuous function $f : \mathbb{R}^n \to \mathbb{R}$ is *smooth* if all its higher derivatives and partial derivatives exist and are also continuous. For example, any polynomial function is smooth.

A function is called *Lipschitz continuous* if it has a *Lipschitz constant* $L \geq 0$ for which every $x, y \in \mathbb{R}^n$, $\|f(x) - f(y)\| \leq L\|x - y\|$.

Chapter 3

Models of Autonomous Systems

When modeling autonomous systems, we need to consider joint dynamics of physical processes and cyber elements such as computers. In this dissertation, we will study different mathematical models with emphasis on different verification and synthesis techniques that can be exploited. In Chapter 4, Chapter 5, and Chapter 8, we study dynamical systems, where the states evolve continuously over time and follow ordinary differential equations (ODE), differential algebraic equations (DAE), or inclusions. In Chapter 7, we work on transition systems or discrete-time automata, where the states change according to discrete transitions instantaneously. In Section 5.5 and Chapter 6, we study models of hybrid systems, where we unify the continuous evolution of the states that describe the dynamics of physical processes and the discrete changes of the states that describe the possible states of a computation. A hybrid system is one of the most well-known formalisms for describing the interactions between physical plants and computing and control units [52, 53, 54, 55, 12, 56]. A more detailed literature review on different sub-classes and alternatives of hybrid systems is provided in Section 4.2.

In this chapter, we introduce the above three mathematical models used in dissertation for modeling different systems. For each model, we will define syntax, semantics—executions and reachable states—and give examples.

3.1 Discrete-time Transition Systems

The mathematical object we will use to model computation is a discrete-time transition system (also called an automaton or a state machine). The following notions are important elements that we will need to define such a transition system [57].

State Variables and Valuations State represents information that will be used to predict the future behaviors of a system, and it is modeled using state variables and valuations.

A *variable* is a name and an associated *type* over which it takes values. For a variable v, the type is denoted by $\mathsf{type}(v)$. In general a variable can have types used in mathematics and programming languages, such as Booleans, natural numbers, integers, real numbers, floats, doubles, and enumerated sets. Type constructors like arrays, lists, and tuples can be used to construct user-defined types from the basic types. Variables that remain constant along trajectories are called *discrete variables*; all other variables are called *continuous variables*, which will be introduced in Section 3.2.

A *valuation* for a set of variables V maps each variable $v \in V$ to a value in $\mathsf{type}(v)$. Given a set of variables V, $\mathsf{val}(V)$ denotes the set of all possible valuations of the variables in V. This is also called the *state space* of the model. In this dissertation, we will abuse the notation and use variables' names to denote their valuations as well.

Predicates and Transitions A *predicate* over $\mathsf{val}(V)$ is a computable function $\phi : \mathsf{val}(V) \to \mathbb{B}$ that maps each state in $\mathbb{R}^n$ to either True (true) or False (false).

Transitions specify the rules for state change relations. A transition relation over a set of variables V is a relation $R \subseteq \mathsf{val}(V) \times \mathsf{val}(V)$, and it relates a pair of states: a *prestate*, which is the state just before the change occurs, and a *poststate*, the state after the change.

Transition relations are specified as predicates on the pre- and post-states. The predicate following the *precondition* defines the set of states in which this transition rule can be applied. The precondition is also known as the *guard condition* or the *enabling condition* of the action. The *effect* specifies how the state changes when the action does occur, and the (possibly nondeterministic) command following the effect relates the prestate and the poststate. The variables that are not mentioned in the effect statements are assumed to remain unchanged by the transition.

With all these elements, we can define a discrete-time transition system as follows:

Definition 3.1. A *discrete-time transition system* $\mathcal{A}$ is a tuple $\langle V, \Theta, A, \mathcal{D} \rangle$, where

1. V is a set of variables called the *state variables*. The set $\mathsf{val}(V)$ of valuations of V is the set of *states*.

2. $\Theta \subseteq \mathsf{val}(V)$ is a nonempty set of *start states*.

3. A is a set of *actions* or transition labels.

4. $\mathcal{D} \in \mathsf{val}(V) \times A \times \mathsf{val}(V)$ is called the set of *transitions*.

```
automaton Consensus($n \in \mathbb{N}, N \in \mathbb{N}$)
  variables
    $x : \mathbb{R}^n$
    $d : \mathbb{B}^N$
  initially
    $x^{(i)} \in [-4, 4]$ for each $i \in \{0, \cdots, N-1\}$
    $d^{(i)} :=$ false for each $i \in \{0, \cdots, N-1\}$

transitions
  a_i for each $i \in \{0, \cdots, N-1\}$
    pre  $d^{(i)} =$ false
    eff  $x := A_i x \ \wedge \ d^{(i)} :=$ true
  $\mathsf{a}_\perp$
    pre $\wedge_{i \in \{0, \cdots, N-1\}]} d^{(i)}$
    eff $d^{(i)} :=$ false for each
          $i \in \{0, \cdots, N-1\}$
```

Figure 3.1: Discrete-time transition system model of iterative consensus.

Example 3.1 (Iterative consensus)**.** Consider an n-dimensional iterative consensus protocol with N processes as shown in Figure 3.1. The state space consists of two types of variables, which are represented using two vectors: the real-valued part of state $x \in \mathbb{R}^n$ and the Boolean-valued part of the state $d \in \mathbb{B}^N$. Each process i changes the real-valued part of state by the linear transformation $x \leftarrow A_i x$. The system evolves in rounds: in each round, each process i updates the state x exactly once but in arbitrary order. The Boolean vector d marks the processes that have acted in a round.

The set of actions is $A = \{a_i\}_{i \in \{0, \cdots, N-1\}} \cup \{a_\perp\}$. The guard and transition function of each action are defined by the precondition (**pre**) and effect (**eff**) statements. For each $i \in \{0, \cdots, N-1\}$, the action a_i is enabled when $d^{(i)}$ is false and when it occurs x is updated as $A_i x$, where A_i is an $n \times n$ matrix, and $d^{(i)}$ is updated to be true. The action $a_\perp$ can occur only when all $d^{(i)}$'s are set to true and it resets all the $d^{(i)}$'s to false.

It can be seen that the linear transformations can be applied in any order and one application of each constitutes a "round". This can be viewed as an abstraction of iterative consensus [58, 59], where N processes perform computations on a shared state.

Deterministic Labeled Transition System. We call a transition system a *labeled transition system* if the state $v \in V$ is a valuation of the real-valued and finite-valued variables. We denote by $v.X$ and $v.L$, respectively, the real-valued and discrete (finite-valued) parts of the state v. We will view the continuous part $v.X$ as a vector in $\mathbb{R}^{|X|}$ by fixing an arbitrary ordering of X.

For $\delta \geq 0$, the δ-*ball* of v is denoted by $B_\delta(v) \triangleq \{v' \in V : v'.L = v.L \wedge \|v'.X - v.X\| \leq \delta\}$. For any $(v, a, v') \in \mathcal{D}$, we write $q \xrightarrow{a} q'$. For any action $a \in A$, its guard is the set $\texttt{guard}(a) = \{v \in V \mid \exists v' \in V, v \xrightarrow{a} v'\}$. We assume that guards are closed sets.

An action a is *deterministic* if for any state $v \in V$, if there exists $v_1, v_2 \in V$ with $v \xrightarrow{a} v_1$ and $v \xrightarrow{a} v_2$, then $v_1 = v_2$. In this dissertation, we only consider deterministic labeled transition system satisfies the following the below assumption:

Assumption 3.1. (i) Actions are deterministic. For notational convenience, the name of an action a is identified with its transition function, i.e., for each $v \in \texttt{guard}(a)$, $v \xrightarrow{a} a(v)$. We extend this notation to all states, i.e., even those outside $\texttt{guard}(a)$. (ii) For any state pair v, v', if $v.L = v'.L$ then $a(v).L = a(v').L$.

Executions and Traces. For a deterministic transition system, a state $v_0 \in V$ and a finite action sequence (also called a *trace*) $\tau = a_0 a_1 \ldots a_{n-1}$ uniquely specifies a *potential execution* $\xi(v_0, \tau) = v_0, a_0, v_1, a_1, \ldots, a_{n-1}, v_n$ where for each $i \in \{0, \cdots, n-1\}$, $a_i(v_i) = v_{i+1}$.

A *valid execution* (also called execution for brevity) is a potential execution with (i) $v_0 \in \Theta$ and (ii) for each $i \in \{0, \cdots, n-1\}$, $v_i \in \texttt{guard}(a_i)$. That is, a valid execution is a potential execution starting from the initial set with each action a_i enabled at state v_i. For any potential execution $\xi_{v_0,\tau}$, its *trace* is the action sequence τ, i.e., $\texttt{trace}(\xi(q_0, \tau)) = \tau \in A^*$. We denote by $len(\tau)$ the length of τ. For any $i \in \{0, \cdots, len(\tau) - 1\}$, $\tau(i)$ is the i-th action in τ. The length of $\xi(v_0, \tau)$ is the length of its trace, and it is denoted by $len(\xi(v_0, \tau))$. When v_0 and τ are clear from the context, we write $\xi(v_0, \tau)$ as ξ for brevity and let $\xi(i) = v_i$ denote the state visited after the i-th transition. The first and last states of an execution ξ are denoted as $\xi.\mathsf{fstate} = \xi(0)$ and $\xi.\mathsf{lstate} = \xi(len(\xi))$.

For any $t \geq 0$, $\mathsf{Execs}(\Theta, t)$ is the set of length t executions starting from the initial set Θ. We denote the *set of reachable states at time t* by

$$\mathsf{Reach}(\Theta, t) \triangleq \{\xi.\mathsf{lstate} \mid \xi \in \mathsf{Execs}(S, t)\}.$$

Similarly, for a time bound $T \geq 0$, the set of all reachable states from the initial set Θ during time interval $[0, T]$ is denoted by

$$\mathsf{Reach}(\Theta, [0, T]) \triangleq \bigcup_{t=0}^{T} \mathsf{Reach}(\Theta, t).$$

In Example 3.1, if $N = 3$, a valid execution could have the trace $\tau = a_0 a_2 a_1 a_\perp a_1 a_0 a_2 a_\perp$. It can be checked that Assumption 3.1 holds. In fact, the assumption will continue to hold if $A_i x$ is replaced by a nonlinear transition function $a_i : \mathbb{R}^n \to \mathbb{R}^n$.

3.2 Trajectories and Closures

A *trajectory* for a set of continuous variables V defines how the valuations evolve over time. To be concrete, a trajectory ξ is defined as a function $\xi : \mathsf{dom} \to \mathsf{val}(V)$, where dom is the time domain of evolution, and it is either $[0, T]$ for some $T > 0$, or it is $[0, \infty)$. The domain of ξ is referred to as $\xi.\mathsf{dom}$.

The state of the system along the trajectory at time $t \in \xi.\mathsf{dom}$ is $\xi(t)$. For a bounded trajectory with $\xi.\mathsf{dom} = [0, T]$, the *duration* $\xi.\mathsf{dur} = T$. For unbounded trajectories, $\xi.\mathsf{dur}$ is defined as ∞. The *first state* $\xi(0)$ is denoted by $\xi.\mathsf{fstate}$, and for a bounded trajectory the *last state* $\xi.\mathsf{lstate} = \xi(T)$ and $\xi.\mathsf{ltime} = T$.

The *t-prefix* of ξ, for any $t \in \xi.\mathsf{dom}$, is the trajectory $\xi' : [0, t] \to \mathsf{val}(V)$, such that for all $t' \in [0, t]$, $\xi'(t') = \xi(t')$. A set of trajectories Ξ is *prefix-closed* if for any $\xi \in \Xi$, any prefix of ξ is also in Ξ. A set Ξ is *deterministic* if for any pair $\xi_1, \xi_2 \in \Xi$, if $\xi_1(0) = \xi_2(0)$ then one is a prefix of the other.

The *t-suffix* of ξ, for any $t \in \xi.\mathsf{dom}$, is the trajectory $\xi' : [0, \xi.\mathsf{ltime} - t] \to \mathsf{val}(V)$, such that for all $t' \in [0, t]$, $\xi'(t') = \xi(t' + t)$. So ξ' is a shorter trajectory that is identical to ξ from t onwards. A set of trajectories Ξ is *suffix-closed* if for any $\xi \in \Xi$, any suffix of ξ is also in Ξ.

Given a trajectory $\xi : [0, T] \to \mathsf{val}(V)$ for a set of variables V and a time

$t > 0$, the *time shift* $(\xi + t) : [t; t + T] \rightarrow \mathsf{val}(V)$ is defined as $\forall t' \in [t, t + T], (\xi + t)(t') = \xi(t' - t)$. Strictly speaking, for $t > 0, \xi + t$ is not a trajectory. The *concatenation* of two trajectories $\xi_1 \frown \xi_2$ is a new trajectory ξ in which ξ_1 is followed by ξ_2. That is, $(\xi_1 \frown \xi_2).\mathsf{dom} = \xi_1.\mathsf{dom} \cup (\xi_2 + \xi_1.\mathsf{ltime}).\mathsf{dom}$, and for each $t \in (\xi_1 \frown \xi_2).\mathsf{dom}$,

$$(\xi_1 \frown \xi_2)(t) = \begin{cases} \xi_1(t) & t \leq \xi_1.\mathsf{ltime} \\ \xi_2(t - \xi_1.\mathsf{ltime}) & t > \xi_1.\mathsf{ltime}. \end{cases}$$

A set of trajectories Ξ is *concatenation-closed* if for any $\xi_1, \xi_2 \in \Xi$ with $\xi_1.\mathsf{lstate} = \xi_2.\mathsf{fstate}$, $\xi_1 \frown \xi_2 \in \Xi$. See [60] for detailed explanation of trajectories closed under prefix, suffix and concatenation.

3.3 Dynamical Systems

The continuous evolution of an autonomous system or cyber-physical system can be mathematically modeled as a dynamical system. Consider an n-dimensional *dynamical system* defined as an ordinary differential equation (ODE):

$$\dot{x}(t) = f(x(t)), \tag{3.1}$$

where $t \in \mathbb{R}$ is the time, x is the state with $\mathsf{type}(x) = \mathbb{R}^n$, and $f : \mathsf{val}(x) \rightarrow \mathsf{val}(x)$ is a locally Lipschitz continuous function describing the continuous evolution of the physical variables of the autonomous system.

For any initial valuation $x_0 \in \mathsf{val}(x)$, a trajectory $\xi : [0, T] \rightarrow \mathsf{val}(x)$ is a solution or execution of (3.1) if $\xi(0) = x_0$, and $\forall t \in \xi.\mathsf{dom}, \frac{d}{dt}\xi(t) = f(\xi(t))$. The existence and uniqueness of the solution can be guaranteed by the Lipschitz continuity of f. With an initial state and a time bound, an ODE that locally Lipschitz continuous defines a unique trajectory. Therefore, we will abuse the notation and let $\xi(x_0, t)$ denote the solution $\xi(t)$ starting from $\xi(0) = x_0$ for dynamical systems.

Given an initial set Θ, we say a state x is *reachable* from Θ if there exist a state $\theta \in \Theta$ and a time $t \geq 0$ such that $\xi(\theta, t) = x$. We call the set of states that are reachable from the initial set Θ during time $[0, T]$ the *reach set*, and denote it as $\mathsf{Reach}(\Theta, [0,T])$. Formally,

$$\mathsf{Reach}(\Theta, [0,T]) = \{x \mid \exists \theta \in \Theta, \exists t \in [0,T], \xi(\theta, t) = x\}.$$

Similarly, we denote the set of reachable states at time t from initial set Θ as $\mathsf{Reach}(\Theta, t)$.

Assume that the function f is also continuously differentiable. The *Jacobian* of f, $J_f : \mathbb{R}^n \to \mathbb{R}^{n \times n}$, is a matrix-valued function of all the first-order partial derivatives of f with respect to x, that is

$$J_f^{(i,j)}(x) = \frac{\partial f^{(i)}(x)}{\partial x^{(j)}}.$$

The following lemma states a relationship between f and its Jacobian J_f which can be proved using the generalized mean value theorem [61].

Lemma 3.1. For any continuously differentiable vector-valued function $f : \mathbb{R}^n \to \mathbb{R}^n$, and $x, r \in \mathbb{R}^n$,

$$f(x+r) - f(x) = \left(\int_0^1 J_f(x + sr) ds \right) \cdot r, \tag{3.2}$$

where the integral is component-wise.

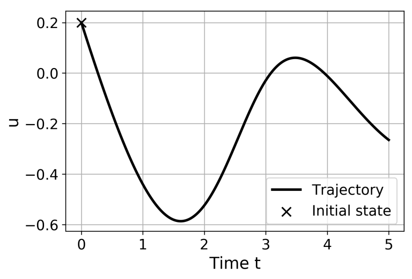
Example 3.2 (Jet engine control system)**.** The Moore-Greitzer model of a jet engine compression system is studied in [62] to understand and prevent two types of instabilities: rotating stall and surge. With a stabilizing feedback controller operating in the no-stall mode, it has the following dynamics:

$$\begin{cases} \dot{u} = -v - \frac{3}{2}u^2 - \frac{1}{2}u^3 \\ \dot{v} = 3u - v. \end{cases} \tag{3.3}$$

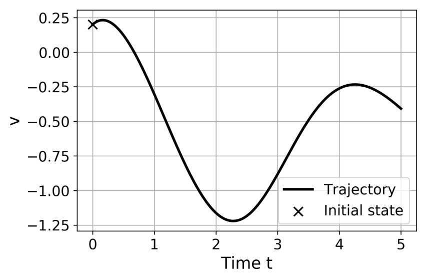
The Jacobian of the system is:

$$J_f(x) = \begin{bmatrix} -3u - \frac{3}{2}u^2 & -1 \\ 3 & -1 \end{bmatrix}. \tag{3.4}$$

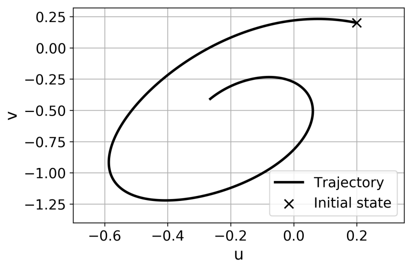
Figure 3.2 shows the trajectories of the engine system starting from an initial state $[0.2, 0.2]^\top$ and the set of reachable states starting from an initial set $B_{0.05}([0.2, 0.2]^\top)$.



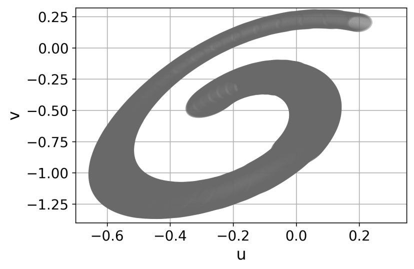
(a) Trajectory of t vs. u



(b) Trajectory of t vs. v



(c) Trajectory of u vs. v



(d) Reachable states of u vs. v

Figure 3.2: (a)-(c): Trajectories of Example 3.2 starting from the initial state $u_0 = 0.2, v_0 = 0.2$. (d): The set of reachable states of Example 3.2 starting from an initial set $B_{0.05}([0.2, 0.2]^\top)$.

3.4 Hybrid Systems

Hybrid systems or hybrid automata unify the discrete-time transition system of Section 3.1 and dynamical systems of Section 3.3. They are natural and popular models for representing cyber-physical systems [14, 13, 53, 57]. The state of a hybrid system can change through instantaneous transitions and over time intervals through trajectories. One can view a hybrid system as a collection of ODEs—one for each *mode*–and a set of discrete transition rules for switching between the ODEs or modes. Thus, the continuous behavior of a hybrid system is described by differential equations, and discrete behavior is described by a set of transition rules that can be defined in terms of a labeled transition system or automaton.

Definition 3.2. A hybrid system $\mathcal{H}$ is a tuple $\langle V = (X \cup L), \Theta, \mathsf{L}_{\mathsf{init}}, A, \mathcal{D}, \mathsf{TL} \rangle$, where

1. $V = (X \cup L)$ is a set of variables called the *state variables*, where X is the set of real-valued variables and L is the finite-valued variables. We will use L to denote $\mathsf{val}(L)$, which is a finite set of *modes* (a.k.a. *locations* or *discrete states*). We will use a subset of the Euclidean space

$\mathsf{X} \subseteq \mathbb{R}^n$ to denote $\mathsf{val}(X)$, which is the space for continuous states. The combined *hybrid* state space is $\mathsf{val}(V) = \mathsf{X} \times \mathsf{L}$.

2. $\Theta \subseteq \mathsf{X}$ is a compact set of initial states for the continuous variables X, and $\mathsf{L}_{\mathsf{init}} \subseteq \mathsf{L}$ is the set of initial modes for the discrete variables L.

3. A is a set of *actions* or transition labels.

4. $\mathcal{D} \subseteq \mathsf{val}(V) \times A \times \mathsf{val}(V)$ is called the set of *transitions.*

5. TL is a set of deterministic trajectories for the variables in V that is closed under prefix, suffix, and concatenation,

Each state of the hybrid system $\mathcal{H}$ is a pair $\langle x, \ell \rangle$, where $x \in \mathsf{X}$ and $\ell \in \mathsf{L}$ represent the continuous and discrete states, respectively. The evolution of the systems continuous state variables X is formally described by the continuous functions of initial states and time called trajectories (see Section 3.2). For a hybrid system with L modes, each trajectory is labeled by a mode in L. A *trajectory labeled by* L is a pair $\langle \xi(x_0, t), \ell \rangle$ where $\xi(x_0, t)$ is a trajectory starting from $x_0 \in \mathsf{X}$, and $\ell \in \mathsf{L}$. A deterministic, prefix-closed set of labeled trajectories TL describes the behavior of the continuous variables in modes L.

The set of trajectories TL is not necessarily specified by ODEs. Indeed, in Chapter 6 we will see that it can be provided by any simulators. For now, let us assume that for each $\ell \in \mathsf{L}$, a set of trajectories TL_ℓ is specified by ODEs $f_\ell : \mathbb{R}^n \to \mathbb{R}^n$ and an invariant $\mathsf{I}_\ell \subseteq \mathbb{R}^n$, where invariant is a predicate involving the state variables. For any trajectory $\langle \xi, \ell \rangle \in \mathsf{TL}_\ell$, ξ is a valid trajectory if it evolves according to $\frac{d}{dt}\xi = f_\ell(\xi)$ at each time in the domain of ξ, and ξ satisfies the invariant I_ℓ.

A transition $(v, a, v') \in \mathcal{D}$ is written in short as $v \xrightarrow{a} v'$, and it is defined similarly as in the care of discrete transitions systems (Section 3.1). For any action $a \in A$, its guard is the set $\mathtt{guard}(a) = \{v \in V \mid \exists v' \in V, v \xrightarrow{a} v'\}$. This is also the set of states that satisfy the **pre**conditions of the action a in the specification. We still assume that guards are closed sets.

Semantics of $\mathcal{H}$ are given in terms of executions. Formally, an execution is an alternating sequence of trajectories and actions that are consistent with the modes transitions defined in the hybrid system. A finite execution of $\mathcal{H}$ starting from $x_0 \in \Theta$ and $\ell_{init} \in \mathsf{L}_{\mathsf{init}}$ is a sequence of labeled trajectories and actions, $exec(x_0, \ell_{\mathsf{init}}) = \langle \xi_{\ell_0}, \ell_0 \rangle, a_0, \langle \xi_{\ell_1}, \ell_1 \rangle, a_1, \cdots, a_{k-1}, \langle \xi_{\ell_k}, \ell_k \rangle$ such that

1. $\xi_{\ell_0}.\mathsf{fstate} = x_0 \in \Theta$ and $\ell_0 = \ell_{\mathsf{init}} \in \mathsf{L}_{\mathsf{init}}$.

2. $\sum_{j=0}^{k} \xi_{\ell_j}.\mathsf{dur} = T$, which is also called the *duration* or *time horizon* of the execution.

3. For each $i = 0, \cdots, k-1$, $\langle \xi_{\ell_i}.\mathsf{lstate}, \ell_i \rangle \xrightarrow{a_i} \langle \xi_{\ell_{i+1}}.\mathsf{fstate}, \ell_{i+1} \rangle$. That is, the last state of $\langle \xi_{\ell_i}, \ell_i \rangle$ is in $\mathtt{guard}(a_i)$.

The set of all executions of $\mathcal{H}$ is denoted by $\mathtt{Execs}(\Theta, \mathsf{L}_{\mathsf{init}})$. A state $\langle x, \ell \rangle$ is called to be *reachable* if there exists an execution $\langle \xi_{\ell_0}, \ell_0 \rangle, a_0, \cdots, a_{k-1}, \langle \xi_{\ell_k}, \ell_k \rangle \in \mathtt{Execs}(\Theta, \mathsf{L}_{\mathsf{init}})$, $i \in \{0, \ldots k\}$, and $t' \in \xi_i.\mathsf{dom}$ such that $\ell = \ell_i$, $x = \xi_{\ell_i}(t')$. The set of reachable states is defined as

$$\mathsf{Reach}_{\mathcal{H}}(\langle \Theta, \mathsf{L}_{\mathsf{init}} \rangle, [0, T]) = \{\langle x, \ell \rangle \mid \langle x, \ell \rangle \text{ is reachable}\}.$$

We also write the reachable set as $\mathsf{Reach}(\langle \Theta, \mathsf{L}_{\mathsf{init}} \rangle, [0, T])$ if $\mathcal{H}$ is clear from the context.

Example 3.3 (Cardiac pacemaker)**.** A hybrid automaton that models the behavior of a cardiac pacemaker system (also discussed in [63]) is given in Figure 3.3. The hybrid system has two modes, namely, $\mathsf{Stim\_on}$ and $\mathsf{Stim\_off}$. The continuous variables u and v model the voltage and the current on the tissue membrane and the timer t measures the time spent in each location. The system stays in $\mathsf{Stim\_on}$ location when the pacemaker gives a stimulus to the cell and is in $\mathsf{Stim\_off}$ when the stimulus is absent. The discrete transition from $\mathsf{Stim\_on}$ to $\mathsf{Stim\_off}$ is enabled when $t = 5$; and t is reset to 0 after a transition; u and v are left unchanged. Transition from $\mathsf{Stim\_off}$ to $\mathsf{Stim\_on}$ is enabled when $t = 20$; and both these transitions are urgent. Thus, the pacemaker gives a stimulus every 25 time units for a duration of 5 time units.

Figure 3.4 also describes the cardiac pacemaker hybrid system in a way that is similar to MATLAB Stateflow. Each rounded rectangle represents a mode, including the mode name on the first line and the corresponding ODEs that specify the evolutions of trajectories for the continuous variables. The arrowed lines that connect the modes present transitions, where $[\cdot]$ encodes **pre**conditions and $\{\cdot\}$ encodes **eff**ects of the transitions.

The behavior of the continuous variables t, u, v within a time period is given in Figure 3.5.

```
automaton Cardiac
  actions
    On; Off
  variables
    $[t, u, v]^\top : \mathbb{R}^3$
    $\ell$ : enumeration [{Stim_on}, {Stim_off}]
  initially
    $[t, u, v]^\top := [0, 0.1, 0.1^\top]$
    $\ell :=$ On
  transitions
    On
      pre  $t = 20 \wedge \ell =$ Stim_off
      eff  $t := 0 \wedge \ell :=$ Stim_on
    Off
      pre $t = 5 \wedge \ell =$ Stim_on
      eff $t := 0 \wedge \ell :=$ Stim_off
  trajectories
```

```
Stim_on
evolve
  $\dot{t} = 1$
  $\dot{u} = -u(0.9(u+1) + u^2) - v + 1$
  $\dot{v} = u - 2v$
invariant $\ell =$ Stim_on $\wedge\, t \leq 5$

Stim_off
evolve
  $\dot{t} = 1$
  $\dot{u} = -u(0.9(u+1) + u^2) - v$
  $\dot{v} = u - 2v$
invariant $\ell =$ Stim_off $\wedge\, t \leq 20$
```

Figure 3.3: Hybrid automaton model of cardiac pacemaker system.

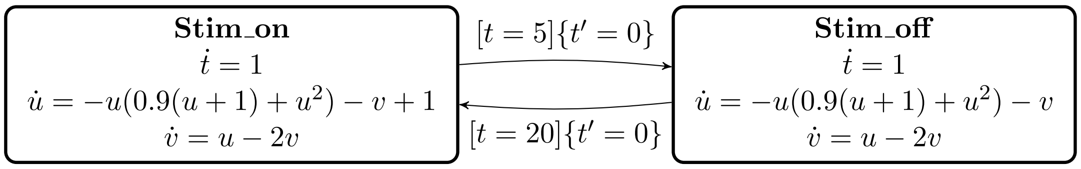
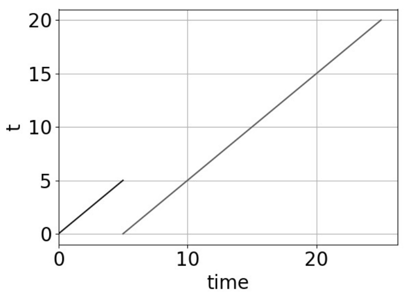
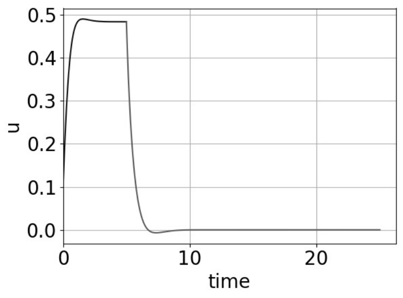


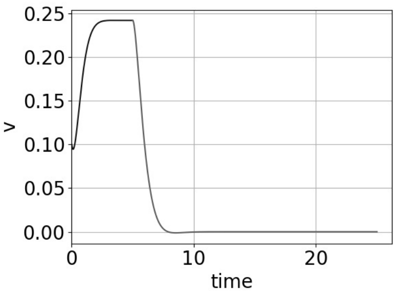
Figure 3.4: Hybrid system model of a cardiac cell with a pacemaker.



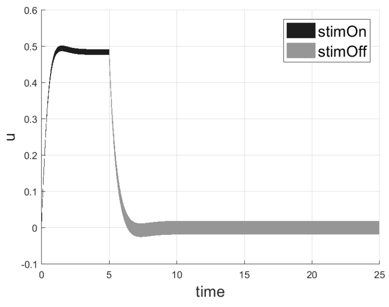
(a) Trajectory of time vs. t.



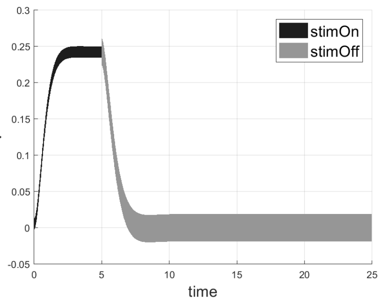
(b) Trajectory of time vs. u.



(c) Trajectory of time vs. v.



(d) Reachable states of time vs. u.



(e) Reachable states of time vs. v.

Figure 3.5: (a)-(c): Sample trajectories of the continuous variables of the cardiac cell-pacemaker system from the initial state $[0, 0.1, 0.1]$. Black and gray trajectories correspond to the Stim_on and Sim_off modes respectively. (d)-(e): The set of reachable states of the variables u and v starting from an initial set $t = 0, u \in [0, 0.2], v \in [0, 0.2]$. Black and gray regions correspond to the Stim_on and Sim_off modes respectively.

Chapter 4

Data-driven Verification

4.1 Introduction

A verification problem asks the question whether a system $\mathcal{A}$ meets the requirement Q. In this dissertation, we will mainly focus on bounded safety verification problems, which require an algorithm to decide whether any reachable state of the system violates some safety requirement within bounded time. In Chapter 3, we have introduced models for autonomous systems, including discrete-time transition systems, dynamical systems, and hybrid systems. We have also introduced the set of all behaviors of the system from a set of initial conditions, which is the set of reachable states $\mathsf{Reach}(\Theta, [0, T])$ from the initial set Θ within a time bound $T > 0$ (or for hybrid systems, the reachable sets are denoted by $\mathsf{Reach}(\langle\Theta, \mathsf{L_{init}}\rangle, [0, T])$ from the initial states Θ and initial modes $\mathsf{L_{init}}$. The violation of the safety requirements is abstracted as an unsafe or forbidden set F. Given a system $\mathcal{A}$ and the unsafe set F, the *safety verification* problem (also called the bounded invariant verification) is to decide whether

$$\mathsf{Reach}(\Theta, [0, T]) \cap F = \emptyset \tag{4.1}$$

for discrete transition systems or dynamical systems, or

$$\mathsf{Reach}(\langle\Theta, \mathsf{L_{init}}\rangle, [0, T]) \cap F = \emptyset$$

for hybrid systems.

A verification algorithm takes as input a system model $\mathcal{A}$ and the requirements Q, and returns either a mathematical proof that all possible behaviors of $\mathcal{A}$ meet the requirement Q (safe) or a single counterexample behavior of $\mathcal{A}$ that violates Q (unsafe). A verification algorithm for checking the safety of the system is said to be *sound* if the answers of safety/unsafety of the system given by the algorithm are correct. The algorithm is said to be *complete* if the algorithm is guaranteed to terminate when the system is either safe or unsafe.

Safety verification of autonomous systems and cyber-physical systems is a hard problem. There are several theoretical undecidability results that show there is no algorithm that can perform that check automatically, even for relatively simple models. For example, it has been shown that the unbounded-time safety verification of rectangular hybrid systems is undecidable [16]. Relaxing the verification question by asking the bounded-time horizon version of the question (e.g. solving (4.1)), and by tolerating some false positives, can lead to decidable problems. However, computing exact reachable sets for general nonlinear dynamical systems is also undecidable [64, 65]. Recent focus has been on methods to over-approximate the reachable set of the system over bounded time. Existing methods like Taylor models [66] use interval arithmetic [67] to bound the integration value. However, this method suffers from complexity that increases exponentially with both the dimension of the system and order of the Taylor model. One of the major contributions of this dissertation is the introduction of a method to compute reachable set over-approximations that are less conservative and less time consuming. The method is applicable to all of the systems in Chapter 3.

In this chapter, we give the background of a promising approach called *data-driven verification* that answers the safety verification problem for a broad class of nonlinear dynamical and hybrid systems. The basic principle of this approach is to combine numerical simulations with *sensitivity analysis* of the complex or unknown parts of the system. Sensitivity analysis gives bounds on how much the states or outputs of a module can change, with small changes in the input parameters. The data-driven verification approach uses sensitivity analysis to generalize an individual execution (or test) to a set of reachable states, and then verifies the property on this generalized set. The success of this approach hinges on good generalizations that can lead to coverage of all possible behaviors from only finitely many executions. The notion of sensitivity is formalized as *discrepancy functions*, which we will formally define in Section 4.4. In Chapter 5, Chapter 6, and Chapter 7, we will introduce methods that compute discrepancy functions for discrete transitions systems, dynamical systems, and even black-box systems with unknown models.

In the rest of the chapter, we discuss in detail the data-driven verification approach for dynamical systems. With appropriate sensitivity and reachability analysis, the same approach can be utilized for the verification of discrete-time transitions systems, hybrid systems, and systems with unknown models.

Consider a dynamical system described by the differential equation $\dot{x}(t) = f(x(t))$, where $f : \mathbb{R}^n \to \mathbb{R}^n$ is a Lipschitz continuous function. The initial set $\Theta \subset \mathbb{R}^n$ is a compact set in $\mathbb{R}^n$. Recall, the goal is to have an algorithm that answers bounded safety queries correctly: given system as in Equation (3.1), a compact initial set $\Theta \subset \mathbb{R}^n$, an unsafe set $F \subseteq \mathbb{R}^n$, and a time bound $T > 0$, it answers whether $\mathsf{Reach}(\Theta, [0, T]) \cap F = \emptyset$. Getting the closed-form solution of general nonlinear ODEs is still an open problem, not to say getting the set of reachable states which contains infinite many solutions. Therefore, we start by defining simulation and reachtube, which are over-approximations of a single execution and reachable sets, respectively.

4.2 Related Work on Verification

Verification of CPS and autonomous systems has been studied for more than three decades. Every year, the proceedings of conferences like Computer Aided Verification, Hybrid Systems: Computation and Control, and Tools and Algorithms for the Construction and Analysis of Systems contain several papers with results advancing the state of the art in verification of CPS. In this section, we briefly review the related work on formal verification on CPS and hybrid systems, including modeling, verification tools, and different approaches.

Modeling CPS and Autonomous Systems. The popular models for describing the interactions between physical plants and computing and control units are hybrid automata [52, 53, 54], hybrid input/output automata [55, 57], and hybrid programs [68, 69, 70, 12]. The major differences among these models are their syntax and the expressiveness of physical environment. Timed automata [14], which are finite automata extended with a finite set of real-valued clocks, were proposed as a sub-class of hybrid automata to model the timing behavior of real-time systems and networks. It has been shown that the verification and reachability problem for timed automata is decidable [14]. The continuous variables of a time automaton are clocks which increase always at rate 1. If we extend the definition of timed automata to allow the continuous variables to increase with different constant speeds, the resulting type of hybrid automaton is called a rectangular hybrid automaton. The reachability problem for a rectangular hybrid automaton is undecidable [16]. In fact, the class of initialized rectangular hybrid automata is a maximal class for which the reachability problem is decidable [71].

Hybrid automata provide a more expressive way to describe the behaviors of continuous variables. Compared with timed automata, in a hybrid automaton, the value of the continuous variables can be described by a set of general ordinary differential equations [13]. In this dissertation, we adopt the hybrid input/output automata framework proposed in [55, 57], which has been extensively used in modeling many CPS such as biological systems [72], robotic cars [73], helicopters [74], spacecraft [75], and mixed-signal circuits [76].

A switched system is another way of modeling CPS [9]. It generalizes ODEs by allowing the system to be described by a set of ODEs for each mode and a piecewise constant switching signal that defines the operating mode. A switched system model can be seen as a hybrid system in which the switching signal defines the transitions.

Verification Tools. Tools like HyTech [77], PHAVer [19], Coho [78], Checkmate [79], Hylaa [80], HyPro [81], SpaceEx [18], d/dt [82], and ET [83] have targeted and successfully verified linear dynamical and hybrid models. More recently, verification tools such as Flow* [22], NLTOOLBOX [84], iSAT [85], dReach [21], Isabelle/HOL-ODE-Numerics [86] and CORA [23], have demonstrated the feasibility of verifying nonlinear dynamic and hybrid models. These tools are still limited in terms of the complexity of the models and the type of external inputs they can handle, and they require quite often manual tuning of algorithmic parameters. Some of these tools' approaches for reach set estimation operate directly on the vector field involving higher-order Taylor expansions [22, 21]. However, this method suffers from complexity that increases exponentially with both the dimension of the system and the order of the model. In the following, we give a brief overview of several verification tools that are still under active development at time of the writing of this dissertation.

- C2E2: The tool Compare-Execute-Check-Engine (C2E2) [87, 88] is a simulation and verification tool for nonlinear hybrid systems models. It takes as input the model of a hybrid system and a safety property and verifies whether the safety property is satisfied or violated by the model. If satisfied, C2E2 can return reachable sets over-approximations. If violated, C2E2 also presents the counterexample that violates the safety property. The first version of C2E2 as developed in [87, 88] requires that sensitivity (i.e. discrepancy functions) for each mode of the hybrid systems are given by the user as model annotations. One of the contributions of this dissertation is the elimination of this manual step as we present algorithms for automatically computing discrepancy for a broad class of nonlinear models. We will discuss the details in Section 5.5.1.

- CORA: The tool COntinuous Reachability Analyzer (CORA) [23] is a MATLAB-based verification tool using zonotopes for reachability analysis. It can handle hybrid systems with nonlinear continuous dynamics and/or nonlinear differential-algebraic equations. Moreover, it can handle the analysis of uncertain parameters and system inputs.

- dReach: The tool dReach [21] is an SMT-based δ-reachability analysis tool for nonlinear hybrid systems. It checks whether a hybrid system can run into an unsafe region of its state space. dReach picks a single start state in the initial set and attempts to finds a counterexample that reaches the unsafe set. If it succeeds, a (spurious or real) counterexample has been found and the tool finishes. Otherwise, it picks a suitable starting state for the next execution to be processed.

- Flow*: The tool Flow* [22, 89] is a verification tool that computes Taylor model (TM) flowpipes as over-approximations for continuous and hybrid system reachable sets. It can handle nonlinear hybrid systems with uncertainties from the initial set and nondeterministic discrete transitions. It can also decompose large dimensional systems to a network of smaller dimensional system to make the reachability analysis more scalable.

- HyDRA: The tool Hybrid systems Dynamic Reachability Analysis (HyDRA) implements flow-pipe construction based reachability analysis for linear hybrid automata. The tool is built on top of HyPro [81], a C++ library for reachability analysis. HyPro provides different implementations of set representations tailored for reachability analysis such as boxes, convex polyhedra, support functions, or zonotopes.

- Hylaa: The tool Hylaa [80, 90] is a verification tool that computes the simulation-equivalent reachable set of states for a hybrid system with linear ODEs. For a given model, Hylaa can compute all the states reached by any fixed-step simulation. However, it does not reason between time steps. Furthermore, time-varying inputs are considered to be constant between time steps [91].

- Isabelle/HOL-ODE-Numerics: The tool Isabelle/HOL-ODE-Numerics [86] is a collection of rigorous numerical algorithms for continuous systems. It is based on Runge-Kutta methods implemented with affine arithmetic. All algorithms of Isabelle are formally verified in the interactive theorem prover Isabelle/HOL: from single roundoff errors to the global approximation scheme is proved correct with respect to a formalization of ODEs in Isabelle/HOL.

- JuliaReach: The tool JuliaReach is a software framework for reachability computations of dynamical systems. It is written in Julia, a high-level language for scientific computing. JuliaReach can handle continuous affine systems using a block decomposition technique presented in [92].

- KeYmaera X: The tool KeYmaera X is a theorem prover and verification tool for differential dynamic logic [68], a logic for specifying and verifying properties of hybrid systems with mixed discrete and continuous dynamics. KeYmaera X allows users to specify custom proof search techniques as tactics, execute tactics in parallel, and interface with partial proofs via an extensible user interface.

- SpaceEx: The tool SpaceEx is a C++ tool for computing reachability of linear hybrid systems whose continuous and jump dynamics are piecewise affine with nondeterministic inputs [18, 93]. SpaceEx comes with a web-based graphical user interface and a graphical model editor. Its input language facilitates the construction of complex models from automata components that can be combined to networks and parameterized to construct new components.

- XSpeed: The tool XSpeed [94] implements algorithms for reachability analysis for continuous and hybrid systems with linear dynamics. The focus of the tool is to exploit the modern multicore architectures and enhance the performance of reachability analysis through parallel computations.

Sensitivity Analysis. Several approaches have been proposed to obtain proofs about (bounded time) invariant or safety properties from simulations [95, 96]. A similar idea to discrepancy functions is the *sensitivity matrix*, a matrix that captures the sensitivity of the system to its initial condition x_0. This is then used to give an upper bound on the distance between two system trajectories. The sensitivity matrix approach was implemented in a falsification tool Breach, which can generate counter-examples that violate requirements given as signal temporal logic (STL) formulas. In [97], the authors provided sound simulation-driven methods to over-approximate the distance between trajectories, but these methods are mainly limited to affine and polynomial systems. For general nonlinear models, this approach may not be sound, as higher-order error terms are ignored when computing this upper bound. Analysis of sensitivity and the related notion of robustness analysis functions, automata, and executions also received significant attention [98, 99]. In [100] the authors presented an algorithm to compute the output deviation with bounded disturbance combining symbolic execution and optimization. In [98] and [99], the authors presented algorithms for robustness analysis of programs and networked systems.

The idea of computing the reach sets from trajectories is similar to the notion of incremental Lyapunov function [101], which describes the convergence of trajectories with respect to themselves, rather than with respect to an equilibrium point. Similar ideas have also been considered for control synthesis in [102]. Incremental Lyapunov function can also be used for the discrepancy computation [88]. However, we do not require systems to be incrementally stable.

The work closest to the discrepancy computation discussed in this dissertation involves reachability analysis using matrix measures [49], where the authors use the fact that the matrix measure of the Jacobian matrix can bound the distance between neighboring trajectories [103, 51]. Unlike the approach in this dissertation which automatically computes the bounds on matrix measures, the technique there relies on user-provided closed-form matrix measure functions, which are in general difficult to compute.

Simulation-driven Falsification. A noteworthy related approach is *simulation-driven falsification*, which addresses the problem of finding bugs, but does not aim to prove their absence [104]. The search for bugs is formulated as an optimization problem, and since this typically works out to be a nonlinear and non-convex problem, stochastic optimization tools are employed to guide the search. The preeminent tool implementing this approach is S-TaLiRo [105]; it has been effectively used to search for bugs in several practical applications [106, 107].

4.3 Simulation and Reachtube

Although it is generally difficult to get the closed-form solution of dynamic systems, validated simulation libraries, such as VNODE-LP [108] and CAPD [109], use numerical integration to generate a sequence of states with guaranteed error bounds. First we give the following definition of *simulation* as a sequence of time-stamped hyper-rectangles.

Definition 4.1. (Simulation) For any $x_0 \in \mathbb{R}, \tau > 0, \epsilon > 0, T > 0$, a (x_0, τ, ϵ, T)-*simulation* of the system described in Equation (3.1) is a sequence of time-stamped sets $\{(R_i, t_i)_{i=0}^{n}\}$ satisfying the following:

1. R_i is a compact set in $\mathbb{R}^n$ with a diameter smaller than ϵ, for $i = 0, 1, \dots, n$.

2. Let $\xi(x_0, t)$ be the trajectory of the system starting from x_0 along time. Then $\xi(x_0, t_i) \in R_i, i = 0, 1, \dots, n$, and $\forall t \in (t_{i-1}, t_i)$, $\xi(x_0, t) \in \texttt{hull}(R_{i-1}, R_i)$, for $i = 1, \dots, n$.

3. τ is called the *maximum sampling period*, which means that for each $i = 1, \dots, n$, $0 < t_i - t_{i-1} \leq \tau$. Note $t_0 = 0$ and $t_n = T$.

Next, we introduce the definition of *reachtubes*, which is also a sequence of time-stamped hyper-rectangles. However, instead of a single initial state, it contains all the trajectories starting from the initial set Θ.

Definition 4.2. (Reachtube) For any $\Theta \subseteq \mathbb{R}^n, T > 0$, a (Θ, T)-*reachtube* is a sequence of time-stamped compact sets $\{(O_i, t_i)_{i=0}^n\}$, such that each $\mathsf{Reach}(\Theta, [t_{i-1}, t_i]) \subseteq O_i$.

Data-driven verification algorithms generalize single simulations to reachtubes using sensitivity analysis. The most crucial and difficult step is to decide how to conduct such generalizations using sensitivity analysis. In next section, we introduce *discrepancy function*, which formalizes the idea of sensitivity. On the one hand, the discrepancy function should be large enough to give a strict over-approximation of the reachable set; on the other hand, it should be small enough so that the over-approximation is not too pessimistic. Moreover, the value of the function should converge to 0 as the initial set converges to a single point to have completeness guarantees.

4.4 Discrepancy Function

A discrepancy function bounds the distance between two neighboring trajectories as a function of the initial distance between states and the time [61, 87]. That is, given any two trajectories $\xi(x_1, t)$ and $\xi(x_2, t)$ of the system (3.1) starting from states x_1 and x_2 respectively, the discrepancy function β is a function of initial distance between x_1 and x_2, and time t. At any time t, the distance between $\xi(x_1, t)$ and $\xi(x_2, t)$ should be no greater than the value of discrepancy function at t. The distance between any two states can be measured using any norm defined in Section 2.1.1.

Definition 4.3. A continuous function $\beta : \mathbb{R}^{\geq 0} \times \mathbb{R}^{\geq 0} \to \mathbb{R}^{\geq 0}$ is a discrepancy function of the system in Equation (3.1) if

(1) for any pair of states $x_1, x_2 \in \mathbb{R}^n$, and any time $t \geq 0$,

$$\|\xi(x_1, t) - \xi(x_2, t)\| \leq \beta(\|x_1 - x_2\|, t), \text{and} \tag{4.2}$$

(2) for any t, $\lim_{\|x_1 - x_2\| \to 0^+} \beta(\|x_1 - x_2\|, t) = 0$.

The general properties needed for soundness and completeness of verification based on discrepancy functions were identified in [87], though the key step of computing discrepancy functions automatically for general nonlinear systems remained an open problem. A straightforward discrepancy function can be obtained using the Lipschitz constant of f:

Proposition 4.1 (Proposition 1 in [88])**.** Consider a dynamical system as in Equation (3.1), and suppose $L > 0$ is the Lipschitz constant for $f(x)$. Then $\beta(\|x_1 - x_2\|, t) = e^{Lt}\|x_1 - x_2\|$ is a discrepancy function.

In Definition 4.3 and Proposition 4.1, the norm can be any norm. We will make specific choices for designing algorithms. For Example 3.2, $L = 2$ is a Lipschitz constant with 2-norm, and therefore, $e^{2t}\|x_1 - x_2\|_2$ can be used as a discrepancy function for the jet engine system. However, since Lipschitz constants are always positive, Proposition 4.1 will always produce a discrepancy function with a positive exponent, which could be very conservative for the over-approximation of the reachable sets.

According to the definition of discrepancy function, for system (3.1), at any time t, the ball centered at $\xi(x_0, t)$ with radius $\beta(\delta, t)$ contains the reachable set of (3.1) starting from $B_\delta(x_0)$. Therefore, by bloating the simulation trajectories using the corresponding discrepancy function, we can obtain the over-approximating reachtube. Similar ideas have been considered based on abstraction techniques to synthesize controllers [102]. Definition 4.3 corresponds to the definition of discrepancy function (Definition 2) in [87], except that we allow an arbitrary M-norm as a metric. We also remark that this definition of discrepancy function is similar to the incremental Lyapunov functions [101]; however, here we do not require that trajectories converge to each other.

As noted in [87, 49], several techniques (contraction metric [110], incremental stability [101], matrix measure [49], Lyapunov functions [9], Lyapunov exponents [111], etc.) can be used to find discrepancy functions; however, those techniques either restrict the class of nonlinear systems (e.g., polynomial systems, as in [97]) or require crucial user-supplied inputs (e.g., the closed-form expression of matrix measure function, as in [49]).

The matrix measure has long been used to provide estimates on solutions of systems of ordinary differential equations. Unlike the Lipschitz constants, the matrix measures can be negative. The next proposition is the key that provides a bound on the distance between trajectories in terms of their initial distance and the rate of expansion of the system given by the matrix measure of the Jacobian matrix $J(x)$ with respect to x.

Proposition 4.2. [51]. Consider a dynamical system as in Equation (3.1). Let $S \subseteq \mathbb{R}^n$ and let the Jacobian $J(x) = \frac{\partial f}{\partial x}(x)$ satisfy $\mu(J(x)) \leq c$ for all $x \in S$. If every trajectory of Equation (3.1) with initial conditions in the line segment $\{hx_1 + (1-h)x_2 \mid h \in [0,1]\}$ remains in S until time T, then the solutions $\xi(x_1, t)$ and $\xi(x_2, t)$ satisfy

$$\|\xi(x_1,t) - \xi(x_2,t)\| \leq \|x_1 - x_2\| e^{ct} \tag{4.3}$$

for all $t \in [0, T]$.

Proposition 4.2 provides a bound on the global divergence between trajectories of dynamical system (3.1) using only information about the system's Jacobian at each point. It provides a new way to compute discrepancy function. If there exists $c < 0$ such that for all $(t, x) \in [0, \infty) \times S$ we have $\mu(J(x)) \leq c$, then system (3.1) or the vector field $f(x)$ is said to be contracting with respect to $\|\cdot\|$. But here we do not assume the sign of c. For dynamical systems (3.1), the norm of the Jacobian $\|J(x)\|$ can be used as a Lipschitz constant for $f(x)$. From Lemma 2.2, we know that $\mu(J(x)) \leq \|J(x)\|$. Therefore, theoretically, the discrepancy function we get using matrix measures (Proposition 4.2) gives less conservative bounds on the distance between any pair of trajectories than the one we get using Lipschitz constant (Proposition 4.1). Later, in Chapter 5, we will see that using matrix measures, we can get locally optimal exponential rates for the discrepancy function.

4.5 Verification Algorithm

We are now ready to present the verification algorithm (Algorithm 1). The basic idea appeared earlier in [87, 112] at different levels of generality. Recall that a verification algorithm is said to be *sound* if it answers the safety question correctly and it is said to be *complete* if it is guaranteed to terminate on any input. We know that for general nonlinear and hybrid models, the unbounded time verification problem is undecidable, that is, no algorithm exists that is both sound and complete. Even bounded time versions of this problem are known to be undecidable [113]. Algorithm 1 is sound and is guaranteed to terminate under a mild assumption on the inputs.

If there exists some $\epsilon > 0$ such that $B_\epsilon(\mathsf{Reach}(\Theta, [0, T])) \cap F = \emptyset$, we say the system is *robustly safe*. That is, all states in some envelope around the system behaviors are safe. If there exists some $\epsilon, x \in \Theta$, such that $B_\epsilon(\xi(x, t)) \subseteq F$ over some interval $[t_1, t_2], 0 \leq t_1 < t_2 \leq T$, we say the system is *robustly unsafe*. An algorithm is said to be *relatively complete* if it is guaranteed to terminate when the system is either robustly safe or robustly unsafe. Algorithm 1 is relatively complete. Another way of saying this is that Algorithm 1 is a semi-decision procedure for robust safety verification.

The algorithm consists of the following five main steps:

1. Compute a δ-cover $C = \{\theta_i\}_{i=1}^k$ of the initial set Θ, i.e., $\Theta \subseteq \cup_i B_\delta(\theta_i)$, where $B_\delta(\theta_i)$ is a δ-ball around x_i.

2. For each $x_i \in C$, simulation $\xi(\theta_i, t)$ from θ_i is computed.

3. For every initial state $x \in B_\delta(\theta_i)$, at any time t, we have $||\xi(\theta_i, t) - \xi(x, t)|| \leq \beta(\delta, t)$. Therefore, $B_{\beta(\delta,t)}(\xi(\theta_i, t))$ over-approximates all the states reachable from $B_\delta(\theta_i)$ at time t. Taking an union of such sets over intervals of time up to T we compute a reachtube $\mathcal{R}$, which is an over-approximation of $\mathsf{Reach}(B_\delta(\theta_i), [0, T])$.

4. If $\mathcal{R} \cap F = \emptyset$ then θ_i is removed from the cover C. Else if any interval of the simulation $\xi(\theta_i, t)$ is contained in F then output **Unsafe** and the simulation of $\xi(\theta_i, t)$ serves as a counter-example. Otherwise, θ_i is replaced in C by a finer cover of $B_\delta(\theta_i)$ and steps 2-4 are repeated.

5. If the cover C becomes empty, then output **Safe** and the union of reachtubes $\mathcal{R}_{\texttt{all}}$ that contains the over-approximation of $\mathsf{Reach}(\Theta, [0, T])$.

There are several functions referred to in Algorithm 1.

1. Function $\mathsf{Rad}(S)$ returns the radius of a set S.

2. Function `Simulate`() returns a $(\theta, T, \epsilon, \tau)$-simulation of the system.

3. Function `Generalize`() takes as the inputs the simulation ψ starting from θ, the size of the initial cover δ and the simulation precision ϵ, and

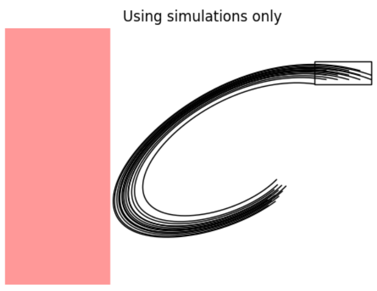
Algorithm 1: Data-driven verification algorithm for dynamical systems.

```
input: $\Theta, T, F, \epsilon_0, \tau_0$
1  $\delta \leftarrow \mathsf{Rad}(\Theta); \epsilon \leftarrow \epsilon_0; \tau \leftarrow \tau_0; \mathcal{R}_{\mathtt{all}} \leftarrow \emptyset$;
2  $\mathcal{C} \leftarrow \mathtt{Cover}(\Theta, \delta, \epsilon)$;
3  while $\mathcal{C} \neq \emptyset$ do
4      for $\langle \theta, \delta, \epsilon \rangle \in \mathcal{C}$ do
5          $\psi = \{(R_i, t_i)_{i=0}^{k}\} \leftarrow \mathtt{Simulate}(\theta, T, \epsilon, \tau)$;
6          $\mathcal{R} \leftarrow \mathtt{Generalize}(\psi, \delta, \epsilon)$;
7          if $\mathcal{R} \cap F = \emptyset$ then
8              $\mathcal{C} \leftarrow \mathcal{C} \backslash \{\langle \theta, \delta, \epsilon \rangle\}$;  $\mathcal{R}_{\mathtt{all}} \leftarrow \mathcal{R}_{\mathtt{all}} \cup \mathcal{R}$ ;
9          else if $\exists j, R_j \subseteq F$ then
10             return (Unsafe, $\psi$)
11         else
12             $\mathcal{C} \leftarrow \mathcal{C} \cup \mathtt{Cover}(B_\delta(\theta), \frac{\delta}{2}, \frac{\epsilon}{2}) \backslash \{\langle \theta, \delta, \epsilon \rangle\}$;
13             $\tau \leftarrow \frac{\tau}{2}$ ;
14         end
15     end
16 end
17 return (Safe, $\mathcal{R}_{\mathtt{all}}$);
```

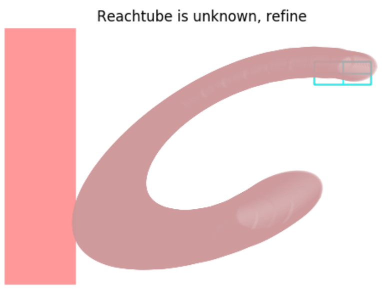
returns a reachtube that contains all the trajectories starting from the initial cover $B_\delta(\theta)$. This can be done by bloating the simulation using a discrepancy function as described in Section 4.4, which is an over-approximation of the distance between any neighboring trajectories starting from $B_\delta(\theta)$.

4. Function `Cover()` returns a set of triples $\{\langle \theta, \delta, \epsilon \rangle\}$, where θ's are sample states, the union of $B_\delta(\theta)$ covers Θ, and ϵ is the precision of simulation.

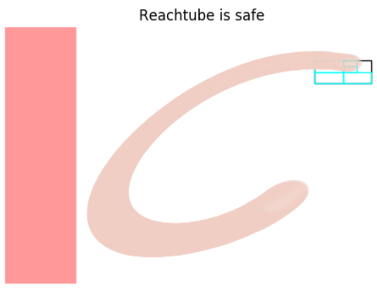
Algorithm 1 proceeds as follows. Initially, $\mathcal{C}$ contains a singleton $\langle \theta_0, \delta_0 = \mathsf{Rad}(\Theta), \epsilon_0 \rangle$, where $\Theta \subseteq B_{\delta_0}(\theta_0)$ and ϵ_0 is a small positive constant. For each triple $\langle \theta, \delta, \epsilon \rangle \in \mathcal{C}$, the **while**-loop from Line 3 checks the safety of the reachtube from $B_\delta(\theta)$, which is computed in Line 5-6. ψ is a $(\theta, T, \epsilon, \tau)$-simulation, which is a sequence of time-stamped rectangles $\{(R_i, t_i)\}$ and is guaranteed to contain the trajectory $\xi(\theta, T)$ by Definition 4.1. Generalizing the simulation result ψ by the discrepancy function to get $\mathcal{R}$, a $(B_\delta(\theta), T)$-reachtube, we have an over-approximation of $\mathsf{Reach}(B_\delta(\theta), [0, T])$. If $\mathcal{R}$ is disjoint from F, then the reachtube from $B_\delta(\theta)$ is safe and the corresponding triple can be safely removed from $\mathcal{C}$. If for some j, R_j (one rectangle of the simulation) is completely contained in the unsafe set, then we can obtain a counterexample in the form of a trajectory that violates the safety property. Otherwise the safety of $\mathsf{Reach}(B_\delta(\theta), [0,T])$ is not determined, and in Line 12 a refinement of $B_\delta(\theta)$ needs to be made with smaller δ and smaller ϵ, τ.



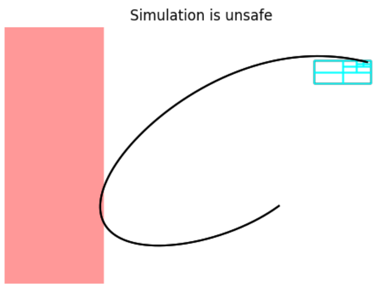
(a)



(b)



(c)



(d)

Figure 4.1: Conceptual demonstration of the data-driven verification algorithm for dynamical systems. The pink region on the left is the unsafe set F; the black hollow box is the initial set Θ with successive refinements and the blue boxes are initial covers that need to be checked for safety. (a): A set of simulations are shown with black lines in the top left figure. (b): dark brown region is a reachtube over-approximations computed using sensitivity analysis with piecewise discrepancy function (will be introduced in Chapter 5) which intersects with F, and therefore, it is inconclusive. (c): The light brown region is the reachtube over-approximation from a part of the initial set that is proven to be safe. (d): A counterexample unsafe simulation is shown as the black line.

Figure 4.1 gives a conceptual demonstration of Algorithm 1 running on the jet engine example (Example 3.2).

Theorem 4.1. Algorithm 1 is sound. That is, if it returns **Safe** then indeed $\mathsf{Reach}(\Theta, [0,T]) \cap F = \emptyset$; if it returns **Unsafe** then it also finds a counterexample, the simulation ψ which enters F. Algorithm 1 is also relatively complete. That is, for any robustly safe or unsafe system, it will terminate and decide either **Safe** or **Unsafe**.

The detailed proof of Theorem 4.1 can be found in pages 50-51 of [88]. A crucial and challenging aspect of Algorithm 1 is choosing an appropriate discrepancy function with which to implement the `Generalize`() function. In the next chapter, we introduce algorithms that implement this function for various types of dynamical systems.

Algorithm 1 can be extended for the safety verification of hybrid systems by replacing $\mathcal{R}$ in Line 6 with the reachtube of the given hybrid system. Chapter 4 of [88] provides a detailed version of the data-driven verification algorithm for hybrid systems with proofs for soundness and relative completeness.

4.6 Summary

In this chapter, we reviewed the data-driven verification approach for dynamical systems. The key to its success is the powerful amalgamation of the speed of numerical simulations with the guarantees coming from sensitivity analysis (discrepancy computation). In next chapter, we will introduce multiple methods to obtain discrepancy functions and look at several case studies.

Chapter 5

Computing Discrepancy

In this chapter, we discuss several approaches for computing discrepancy functions for dynamical systems. We start with the simplest case of stable linear systems where Lyapunov equations can be used for computing discrepancy. Then we move on to discuss nonlinear models and contraction metrics. Next, we introduce one of the major contributions of this dissertation: a locally optimal approach for the discrepancy computation of general nonlinear systems. Finally, we extend the method to compute reachtubes for hybrid systems.

5.1 Introduction

As noted in the last chapter, the performance of the data-driven verification methods relies greatly on having a good discrepancy function that can give sound and precise over-approximations on the distances between any pair of trajectories of the system.

Several techniques like contraction metric, incremental stability, and Lyapunov functions can be used to find discrepancy functions. Indeed, for a linear system, we show that a discrepancy function can be easily constructed using its Lyapunov function (Section 5.2). For a special class of nonlinear systems where a polynomial contraction metric exists, several optimization based methods can also be utilized to discover a discrepancy function for them (Section 5.3). However, those techniques either restrict the class of nonlinear systems or require nontrivial user inputs (e.g., the closed-form expression of a matrix measure function, as in [49]). In this section, we address this problem by providing algorithms that compute discrepancy functions automatically for general nonlinear dynamical systems. The proposed algorithm can provide locally optimal reach set over-approximations.

Our approach for computing discrepancy is based on the well-known result that an upper bound on the matrix measure of the system's Jacobian matrix $J_f(x)$ can be used as an exponential upper bound on the distance between neighboring trajectories [103, 51]. Closed-form expressions for matrix measures are in general difficult to obtain for nonlinear systems. For example, for matrix $J_f(x)$, the matrix measure under Euclidean norm is the largest eigenvalue of the symmetric part of the matrix $\lambda_{\max}((J_f(x)+J_f^T(x))/2)$. However, if we can over-approximate all possible values of the system's Jacobian matrix $J_f(x)$ over some compact set $S \subset \mathbb{R}^n$, we can obtain an upper bound on the matrix measure of the Jacobian matrix over S without knowing its closed form. This two-step computation proceeds as follows: (a) use interval matrices to bound the variation of the Jacobian matrix over S, and (b) compute the upper bound of the matrix measure of the interval matrix. We introduce an algorithm `Generalize` (Algorithm 2) to achieve the second step. The idea is to search all possible linear coordinate transformations of the norms to minimize the matrix measures, which involves solving several optimization problems using semidefinite programming.

We also introduce an algorithm in Section 5.4.2 that uses such discrepancy function based on matrix measures and include discussion of the accuracy and speed of the algorithm (Section 5.4.2). We extend the algorithm to compute the reachable sets for hybrid systems (Section 5.5) and introduce the verification tool C2E2 which implements these algorithms. We show that the resulting algorithm and tool scale to large (up to 28) dimensions and complex nonlinear (with hundreds of nonlinear terms) systems. We apply C2E2 to several challenging hybrid benchmark models including Toyota engine control models (Example 5.3) and a spacecraft rendezvous model (Example 5.4).

5.2 Linear Models

For a linear time invariant (LTI) system $\dot{x} = Ax$, if the system is asymptotically stable we can find a discrepancy function by solving the Lyapunov equation:

Theorem 5.1. For asymptotically stable linear system $\dot{x} = Ax$, given any positive definite matrix $Q \in \mathbb{R}^{n\times n}$, $\beta(\|x_1 - x_2\|_M, t) = e^{-\gamma t}\|x_1 - x_2\|_M$ is a discrepancy function, where $M \succ 0$ can be found by solving the Lyapunov equation $A^T M + MA + Q = 0$ and $\gamma = \frac{\lambda_{\min}(Q)}{2\lambda_{\max}(M)}$.

Proof. Fix any $x_1, x_2 \in \mathbb{R}^n$, and let $y(t) = \xi(x_1, t) - \xi(x_2, t)$, we have

$$\begin{aligned} d\frac{\|y(t)\|_M^2}{dt} &= \dot{y}^T(t)My(t) + y(t)M\dot{y}(t) = y^T(t)(A^T M + MA)y(t) \\ &= -y^T(t)Qy(t) \leq -\lambda_{\min}(Q)y^T(t)y(t) \\ &\leq -\frac{\lambda_{\min}(Q)}{\lambda_{\max}(M)}y^T(t)My(t) = -\frac{\lambda_{\min}(Q)}{\lambda_{\max}(M)}\|y(t)\|_M^2. \end{aligned}$$

By applying Grönwall's inequality, we obtain $\|y(t)\|_M \leq e^{-\frac{\lambda_{\min}(Q)}{2\lambda_{\max}(M)}}\|y(0)\|_M$. □

Example 5.1. A simple linear model of a country's economy by shifting the equilibrium point to the origin is given by

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} 1 & -c_0 \\ c_1(1-c_2) & -c_1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}, \tag{5.1}$$

where x and y represents the national income and the rate of consumer spending respectively. Let $c_0 = 3, c_1 = 2$, and $c_2 = 0$. Consider the Lyapunov function $V(x) = x^\top Mx$ with $M = \begin{bmatrix} 6 & -4 \\ -4 & 7 \end{bmatrix}$. We have $A^T M + MA + Q = 0$ with $Q = \begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix}$.

The eigenvalues of Q are $\lambda_{\max}(Q) = \lambda_{\min}(Q) = 4$, which are all positive. The eigenvalues of M are $\lambda_{\max}(M) = \frac{13+\sqrt{65}}{2}$ and $\lambda_{\min}(M) = \frac{13-\sqrt{65}}{2}$. Therefore, a discrepancy function for the linear system of Equation (5.1) is $\beta(\|x_1 - x_2\|_M, t) = e^{-\frac{8}{13+\sqrt{65}}}\|x_1 - x_2\|_M$ with $M = \begin{bmatrix} 6 & -4 \\ -4 & 7 \end{bmatrix}$.

5.3 Nonlinear Models: Optimization-based Approaches

For nonlinear systems with trajectories that exponentially converge to each other, contraction metrics can be used as a certificate for this convergence [110]). Discrepancy functions can be computed from contraction metrics.

Definition 5.1 (From [110])**.** A uniform metric $\mathcal{M} : \mathbb{R}^n \times \mathbb{R}^{\geq 0} \to \mathbb{R}^{n\times n}$ is called a contraction metric for (3.1) if $\exists \gamma \in \mathbb{R}^{\geq 0}$ such that

$$J_f^T(x)\mathcal{M}(x,t) + \mathcal{M}(x,t)J_f(x) + \dot{\mathcal{M}}(x,t) + \gamma\mathcal{M}(x,t) \preceq 0.$$

Theorem 5.2 (Theorem 2 from [110])**.** For system given by (3.1) that admits a contraction metric $\mathcal{M}$, the trajectories converge exponentially with time, i.e. $\exists k \geq 1, \gamma > 0$ such that, $\forall x_1, x_2 \in \mathbb{R}^n$, $y^T(t)y(t) \leq ky^T(0)y(0)e^{-\gamma t}$, where $y(t) = \xi(x_1,t) - \xi(x_2,t)$.

Proposition 5.1 (Proposition 5 from [88])**.** For system given by (3.1) that admits a contraction metric $\mathcal{M}$, $\beta(\|x_1 - x_2\|_2, t) = \sqrt{k}e^{-\frac{\gamma}{2}t}\|x_1 - x_2\|_2$ is a discrepancy function, where k, γ are from Theorem 5.2.

In [114], a technique for establishing exponential convergence among trajectories using Sum of Squares (SOS) optimization is proposed. It fixes the degree d of the polynomial for all terms in the contraction metric $\mathcal{M}(x)$, then imposes constraints on the coefficients of the polynomials such that $\mathcal{M}(x)$ satisfies conditions given in Definition 5.1. If the contraction metric cannot be found with the d-degree polynomials, the algorithm will increase the value of d and repeat. In [115] the authors presented a simulation-guided approach for discovering contraction metric. It also fixes the degree of the polynomial in the contraction metric $\mathcal{M}(x)$, then samples a finite number of simulations and computes the coefficients of the polynomials such that $\mathcal{M}(x)$ is a valid contraction metric only for that finite number of simulations. Then the algorithm will call an SMT solver to find counter-examples in the entire state space such that $\mathcal{M}(x)$ does not satisfy conditions given in Definition 5.1 for those counter-examples. Then the sampled simulation set is enhanced by those counter-examples and the algorithm repeats using the enhanced sample simulation set.

For a given nonlinear ODE, a contraction metric that has only polynomial terms is not guaranteed to exist, and hence, none of the above procedures is guaranteed to terminate.

5.4 Nonlinear Models: Local Discrepancy

The main obstacle to finding a (global) discrepancy function for general nonlinear systems is the difficulty of globally bounding the convergence (or divergence) rates across all trajectories. By restricting the definition of discrepancy functions over carefully computed parts of the state space, we will gain two benefits. First, such local discrepancy functions will still be adequate to compute `Generalize` needed in Algorithm 1. Second, it will become possible to compute a *local* discrepancy function automatically from simulation traces. The success of the data-driven verification approach (Algorithm 1) hinges on good discrepancy that can lead to coverage of all possible behaviors from only finitely many simulations to ensure soundness of the algorithm. On the other hand, if the discrepancy function gives too-conservative over-approximations, Algorithm 1 will need to refine many times before hitting an answer. Therefore, we want to compute a discrepancy function that is less conservative and in a computationally efficient way. In this section, we introduce methods that use matrix measures to obtain good discrepancy functions for a general class of nonlinear systems. This work was originally presented in [24].

5.4.1 Compute Discrepancy Function with Matrix Measures

We begin by observing that, over a compact set $S \subseteq \mathbb{R}^n$, the Jacobian J_f of the system described by Equation (3.1) can be over-approximated by an interval matrix. Then we establish that the distance between two trajectories in S satisfies a differential equation from a set of differential equations described using the interval matrix. By bounding the matrix measure of the interval matrix, we can get a discrepancy function.

Since we assume the system is continuously differentiable, the Jacobian matrix is continuous, and therefore, over a compact set S, the elements of $J_f(x)$ are bounded. That is, there exists an interval matrix $\mathbb{A}$ such that $\forall x \in S, J_f(x) \in \mathbb{A}$. For interval matrix $\mathbb{A} = \texttt{IntV}(B, C)$, the bounds B and C can be obtained using interval arithmetic or an optimization toolbox by maximizing and minimizing the terms of J_f over S. (The set S can be chosen to be a coarse over-approximation of the reach set, obtained using the Lipschitz constant as in Proposition 4.1).

Once the bounds are obtained, we use the interval matrix that over-approximates the behavior of $J_f(x)$ over S and matrix measures to analyze the rate of convergence or divergence between trajectories:

Lemma 5.1. For system (3.1) with a compact initial set Θ starting from time t_1, suppose $S \subseteq \mathbb{R}^n$ is a compact convex set, and $[t_1, t_2]$ is a time interval such that for any $x \in \Theta$, $t \in [t_1, t_2]$, $\xi(x, t) \in S$. Let $M \in \mathbb{R}^{n \times n}$ be an invertible matrix. If there exists an interval matrix $\mathbb{A}$ such that

(a) $\forall\ x \in S$, $J_f(x) \in \mathbb{A}$, and

(b) $\exists\ \gamma \in \mathbb{R}$, $\forall A \in \mathbb{A}$, $MAM^{-1} + (M^{-1})^\top A^\top M^\top \preceq 2\gamma I$,

then for any $x_1, x_2 \in \Theta$ and $t \in [t_1, t_2]$:

$$\|\xi(x_1, t) - \xi(x_2, t)\|_M \leq e^{\gamma(t-t_1)}\|x_1 - x_2\|_M.$$

Proof. Let the matrix measure of the Jacobian matrix correspond to the M-norm $\mu_M(J(x)) = \mu_2(MJ(x)M^{-1})$ (Lemma 2.2); then from the definition of matrix measures, we have

$$\mu_M(J(x)) = \frac{1}{2}\lambda_{\max}((MJ(x)M^{-1} + (MJ(x)M^{-1})^\top)).$$

From the condition of the proposition, we know that $\forall x \in S$, $MJ(x)M^{-1} + (M^{-1})^\top J^\top(x)M^\top \preceq 2\gamma I$. So we have

$$\mu_M(J(x)) \leq \gamma.$$

Since for any $x \in \Theta$, $t \in [t_1, t_2]$, $\xi(x, t) \in S$, from Proposition 4.2, we know that for any $t \in [t_1, t_2]$,

$$\|\xi(x_1, t) - \xi(x_2, t)\|_M \leq \|x_1 - x_2\|_M e^{\gamma(t-t_1)}.$$

□

Lemma 5.1 provides a discrepancy function: $\beta(\|x_1 - x_2\|_M, t) = \|x_1 - x_2\|_M e^{\gamma(t-t_1)}$. This discrepancy function could result in more or less conservative reachtubes, depending on the selection of M and γ. Ideally, we would like to identify the optimal M such that we can obtain the tightest bound γ. This problem is formulated as follows:

$$\begin{aligned} \min_{\gamma\in\mathbb{R}, M\in\mathbb{R}^{n\times n}} \quad & \gamma \\ \text{s.t} \quad & MAM^{-1} + (M^{-1})^\top A^\top M^\top \preceq 2\gamma I, \quad \forall A \in \mathbb{A}, \\ & M \text{ is invertible.} \end{aligned} \tag{5.2}$$

Pre- and post-multiplying the matrix inequality of (5.2) by $M^\top$ and M, then letting $P = M^\top M$ be a positive semi-definite matrix, we arrive at

$$\begin{aligned} \min_{\gamma\in\mathbb{R}, P\succ 0} \quad & \gamma \\ \text{s.t} \quad & PA + A^\top P \preceq 2\gamma P, \quad \forall A \in \mathbb{A}, \end{aligned} \tag{5.3}$$

which is closer to an semi-definite programing (SDP) problem. However, solving (5.3) to obtain the optimal γ for each time interval involves solving optimization problems with infinite numbers of constraints, imposed by the infinite set of matrices in $\mathbb{A}$. To overcome this problem, we introduce a strategy to transform (5.3) to an equivalent problem with finitely many constraints based on the vertex matrices.

Proposition 2.1 establishes that an interval matrix is equivalent to the convex hull of its vertex matrices. That means each constant matrix A in the interval matrix $\mathbb{A}$ will have a representation based on elements of $\mathtt{VT}(\mathbb{A})$. This allows us to simplify the optimization problem in Equation (5.3) to one with a finite number of constraints, based on the vertex matrices. The next lemma provides a method for computing discrepancy functions from the vertex matrices of an interval matrix.

Lemma 5.2 (Lemma 4.1 from [24])**.** For system (3.1) with a compact initial set Θ starting from time t_1, suppose $S \subseteq \mathbb{R}^n$ is a compact convex set, and $[t_1, t_2]$ is a time interval such that for any $x \in \Theta$, $t \in [t_1, t_2]$, $\xi(x,t) \in S$. Let $M \in \mathbb{R}^{n\times n}$ be an invertible matrix. If there exists an interval matrix $\mathbb{A}$ such that

(a) $\forall\, x \in S$, $J_f(x) \in \mathbb{A}$, and

(b) $\exists\, \gamma \in \mathbb{R}, \forall\, A_i \in \mathtt{VT}(\mathbb{A}),\ \ MA_iM^{-1} + (M^{-1})^\top A_i^\top M^\top \preceq 2\gamma I,$

then for any $x_1, x_2 \in \Theta$ and $t \in [t_1, t_2]$:

$$\|\xi(x_1, t) - \xi(x_2, t)\|_M \leq e^{\gamma(t-t_1)}\|x_1 - x_2\|_M.$$

Proof. From Proposition 2.1, we know that for any interval matrix $\mathbb{A}$,

$$\texttt{hull}(\texttt{VT}(\mathbb{A})) = \mathbb{A}.$$

Therefore, each $A \in \mathbb{A}$ is also in the convex hull of all vertex matrices in $\texttt{VT}(\mathbb{A})$. That is, suppose there are k matrices in the set of vertex matrices $\texttt{VT}(\mathbb{A})$: $\{A_1, \cdots, A_k\}$. For each $A \in \mathbb{A}$, there exists k constant numbers $\alpha_1, \cdots, \alpha_k \in [0, 1]$ such that $A = \sum_{i=1}^k \alpha_i A_i$ and $\sum_{i=1}^k \alpha_i = 1$.

Therefore, if $\forall\ A_i \in \texttt{VT}(\mathbb{A}),\ \ MA_iM^{-1} + (M^{-1})^\top A_i^\top M^\top \preceq 2\gamma I$, then

$$\begin{aligned}
&MAM^{-1} + (M^{-1})^\top A^\top M^\top \\
&= M\textstyle\sum_{i=1}^k \alpha_i A_i M^{-1} + (M^{-1})^\top(\sum_{i=1}^k \alpha_i A_i)^\top M^\top \\
&= \textstyle\sum_{i=1}^k \alpha_i M A_i M^{-1} + \sum_{i=1}^k \alpha_i (M^{-1})^\top A_i^\top M^\top \\
&\preceq 2\textstyle\sum_{i=1}^k \alpha_i \gamma I = 2\gamma I.
\end{aligned}$$

From Lemma 5.1, we can get that for any $t \in [t_1, t_2]$,

$$\|\xi(x_1, t) - \xi(x_2, t)\|_M \leq \|x_1 - x_2\|_M e^{\gamma(t-t_1)}.$$

So the lemma holds. □

Similar to (5.3), Lemma 5.2 suggests the following bilinear optimization problem for finding discrepancy over compact subsets of the state space:

$$\begin{aligned}
\min_{\gamma \in \mathbb{R}, P \succ 0} \quad & \gamma \\
\text{s.t.} \quad & \text{for each } A_i \in \texttt{VT}(\mathbb{A}), PA_i + A_i^\top P \preceq 2\gamma P.
\end{aligned} \tag{5.4}$$

Letting $\gamma_{\max}$ be the maximum of the eigenvalues of $A_i + A_i^\top$ for all i, then $A_i + A_i^\top \preceq \gamma_{\max} I$ (i.e., $M = I$) holds for every A_i, so a feasible solution exists for (5.4). To obtain a minimum feasible solution for γ, we choose a range of $\gamma \in [\gamma_{\min}, \gamma_{\max}]$, where $\gamma_{\min} < \gamma_{\max}$ and perform a line search of γ over $[\gamma_{\min}, \gamma_{\max}]$. Note that if γ is fixed, then (5.4) is a semidefinite program

(SDP), and a feasible solution can be obtained by an SDP solver. As a result, we can solve (5.4) using a line search strategy, where an SDP is solved at each step.

This approach using vertex matrices is computationally intensive due to the potentially $O(2^{n^2})$ matrices in $\mathtt{VT}(\mathbb{A})$ that appear in the SDP (5.4). In [24], a second method is shown to avoid the exponential increase in the number of constraints in (5.4), at the expense of lower accuracy (i.e., increasing the conservativeness). We give the Lemma below without the detailed proof.

Lemma 5.3 (Lemma 4.2 from [24])**.** For system (3.1) with a compact initial set Θ starting from time t_1, suppose $S \subseteq \mathbb{R}^n$ is a compact convex set, and $[t_1, t_2]$ is a time interval such that for any $x \in \Theta$, $t \in [t_1, t_2]$, $\xi(x,t) \in S$. Let $M \in \mathbb{R}^{n \times n}$ be an invertible matrix. If there exists an interval matrix $\mathbb{A} = \mathtt{IntV}([B, C])$ such that

(a) $\forall\, x \in S$, $J_f(x) \in \mathbb{A}$, and

(b) $\exists\, \gamma \in \mathbb{R}$, such that $M(B+C)M^{-1} + (M^{-1})^\top (B+C)^\top M^\top \preceq 4\gamma I$,

then for any $x_1, x_2 \in \Theta$ and $t \in [t_1, t_2]$:

$$\|\xi(x_1,t) - \xi(x_2,t)\|_M \leq e^{\left(\gamma + \frac{\delta}{\lambda_{\min}(M^\top M)}\right)(t-t_1)} \|x_1 - x_2\|_M, \tag{5.5}$$

where $\delta = \sup_{D \in \mathbb{D}} \|D\|_2/2$, and

$$\mathbb{D} = \{D \mid \exists A \in \mathbb{A} \text{ such that } D = (A - \frac{B+C}{2})^\top M^\top M + M^\top M (A - \frac{B+C}{2})\}$$

is also an interval matrix.

In general, Lemma 5.3 provides the discrepancy function $\beta(\|x_1 - x_2\|_M, t) = e^{\left(\gamma + \frac{\delta}{\lambda_{\min}(M)}\right)(t-t_1)} \|x_1 - x_2\|_M$, where an M and γ need to be selected. This suggests solving the optimization problem (5.4) but replacing all the vertex matrices with a single matrix $(B+C)/2$. Then δ is computed as $\sup_{D \in \mathbb{D}} \|D\|_2$, where $\mathbb{D}$ is an interval matrix. In [24], the authors suggested an efficient way to compute $\sup_{D \in \mathbb{D}} \|D\|_2$, which needs only $O(n^2)$ linear computations.

Lemma 5.3 suggests solving the following alternative optimization problem:

$$\begin{aligned} \min_{\gamma\in\mathbb{R},P\succ 0} \quad & \gamma \\ \text{s.t} \quad & P(B+C)+(B+C)^\top P \preceq 4\gamma P. \end{aligned} \tag{5.6}$$

The computations required to produce the discrepancy for Lemma 5.3 are significantly less intensive than for Proposition 5.2, but this comes at the price of decreasing the accuracy (i.e., increasing the conservativeness), due to the positive error term $\frac{\delta}{\lambda_{\min}(M)}$ that is added to γ in (5.5). In practice, we want to make the compact sets S small so that δ (and by extension the exponential term in (5.5)) remains small.

Lemmas 5.2 and 5.3 provide bounds on the M-norm distance between trajectories. Given the simulation result of $\xi(x_1,t)$, for any other initial state x_2 such that $\|x_1-x_2\|_M \le c$, we will have that $\forall t \in [t_1,t_2], \|\xi(x_1,t) - \xi(x_2,t)\|_M \le ce^{\gamma'(t-t_1)}$ ($\gamma' = \gamma$ for Lemma 5.2 and $\gamma' = \gamma + \frac{\delta}{\lambda_{min}(M)}$ for Lemma 5.3). This means that at any time $t\in[t_1,t_2]$, $\xi(x_2,t)$ is contained in the ellipsoid centered at $\xi(x_1,t)$ defined by the set of points x that satisfy $\|(\xi(x_1,t)-x)\|_M \le ce^{\gamma'(t-t_1)}$. That is, $\xi(x_2,t)$ is contained within ellipsoid $E_{ce^{\gamma'(t-t_1)}}(\xi(x_1,t),M)$ (see the definition of ellipsoid in Section 2.2).

Example 5.2. Consider the jet engine example (Example 3.2) over the set $S = \{x = [u,v]^\top \mid u\in[0,0.2], v\in[-0.5,0.5]\}$. The interval matrix that contains every possible value of the Jacobian matrix $J(x)$ is

$$\begin{bmatrix} [-0.66,0] & -1 \\ 3 & -1 \end{bmatrix}. \tag{5.7}$$

Using Lemma 5.2, we obtain the following discrepancy function for this system:

$$\|\xi(x_1,t)-\xi(x_2,t)\|_M \le \|x_1-x_2\|_M e^{-0.4t},$$

with $M = \begin{bmatrix} 2.362 & 0.227 \\ -0.227 & 1.356 \end{bmatrix}$ for as long as the trajectories remain inside S.

Using Lemma 5.3, we obtain the following discrepancy function for this system:

$$\|\xi(x_1,t)-\xi(x_2,t)\|_M \le \|x_1-x_2\|_M e^{-0.165t},$$

with $M = \begin{bmatrix} 2.435 & -0.173 \\ -0.173 & 1.400 \end{bmatrix}$ for as long as the trajectories remain in S.

5.4.2 Algorithm to Compute Local Optimal Reach Set

Given an initial set $B_\delta(x)$ and time bound T, Lemma 5.2 provides discrepancy functions over compact subsets of the state space, and over a bounded time horizon. To compute the reach set of a nonlinear model from a set of initial states over a long time horizon $[0, T]$, we will divide the time interval $[0, T]$ into smaller intervals $[0, t_1], \ldots, [t_{k-1}, t_k = T]$, and compute a piece-wise discrepancy function, where each piece is relevant for a smaller portion of the state space and time.

In this section, we present an algorithm to compute a $(B_\delta(x), T)$-reachtube for system (3.1) using the results from Lemma 5.2. The inputs to Algorithm `Generalize` are as follows:

(1) ψ: a simulation of the trajectory $\xi(x, t)$, where $x = \xi(x, t_0)$ and $t_0 = 0$, represented as a sequence of points $\xi(x, t_0), \ldots, \xi(x, t_k)$ and a sequence of hyper-rectangles $R_i \subseteq \mathbb{R}^n$. such that for any $t \in [t_{i-1}, t_i]$, $\xi(x, t) \in R_i$.

(2) $J_f(\cdot)$: the symbolic Jacobian matrix.

(3) $L \in \mathbb{R}^{\geq 0}$: a Lipschitz constant for the vector field (this can be replaced by a local Lipschitz constant for each time interval).

(4) $M_0 \in \mathbb{R}^{n \times n}, c_0 \in \mathbb{R}^{\geq 0}$: parameters of the initial set such that $B_\delta(x) \subseteq E_{c_0}(x, M_0)$.

The output is a $(B_\delta(x), T)$-reachtube. Note here we assume the simulation trajectory ψ is accurate at discrete time points $t_0, \cdots, t_k$. In Remark 5.2, we will discuss how the algorithm works with validated simulations with guaranteed error bounds.

Algorithm `Generalize` uses Lemma 5.2 to update the matrix M_i to ensure an optimal exponential rate γ_i of the discrepancy function in each time interval $[t_{i-1}, t_i]$. It will solve the optimization problem (5.4) in each time interval to get the local optimal rate. The algorithm proceeds as follows.

1. At Line 1, the radius of the ellipsoid containing the initial set $B_\delta(x)$ is computed as the initial set size.

Algorithm 2: Algorithm `Generalize`

```
input    : $\psi$, $J_f(\cdot)$, $L$, $M_0, c_0$
initially: $\mathcal{R} \leftarrow \emptyset, \gamma_0 \leftarrow -100$
1  $\delta_0 = \mathsf{Rad}\left(E_{c_0}(x, M_0)\right)$ ;
2  for $i$ = $1{:}k$ do
3      $\Delta t \leftarrow t_i - t_{i-1}$ ;
4      $S \leftarrow B_{\delta_{i-1} e^{L\Delta t}}(R_i)$ ;
5      Compute $\mathbb{A}$ such that $\forall x \in S, J_f(x) \in \mathbb{A}$ ;
6      if $\forall V \in \mathtt{VT}(\mathbb{A}) : M_{i-1} V M_{i-1}^{-1} + (M_{i-1}^{-1})^\top V^\top M_{i-1}^\top \preceq 2\gamma I$ then
7          $M_i \leftarrow M_{i-1}$; ;
8          $\gamma_i \leftarrow \arg\min_{\gamma \in \mathbb{R}} \; \forall V \in \mathtt{VT}(\mathbb{A}) : M_i V M_i^{-1} + (M_i V M_i^{-1})^\top \preceq 2\gamma I$ ;
9          $c' \leftarrow c_{i-1}$
10     else
11         compute $M_i, \gamma_i$ from Equation (5.4) ;
12         compute minimum $c'$ such that
             $E_{c_{i-1}}(\xi(x, t_{i-1}), M_{i-1}) \subseteq E_{c'}(\xi(x, t_{i-1}), M_i)$ ;
13     end
14     $c_i \leftarrow c' e^{\gamma_i \Delta t}$ ;
15     $\delta_i \leftarrow \mathsf{Rad}(E_{c_i}(\xi(x, t_i), M_i))$ ;
16     $O_i \leftarrow B_{\delta'}(R_i)$ where $\delta' = \max\{\mathsf{Rad}\left(E_{c'}\left(\xi(x, t_{i-1}), M_i\right)\right), \delta_i\}$ ;
17     $\mathcal{R} \leftarrow \mathcal{R} \cup [O_i, t_i]$ ;
18 end
19 return $\mathcal{R}$ ;
```

2. At Line 4, R_i, which contains the trajectory between $[t_{i-1}, t_i]$, is bloated by the factor $\delta_{i-1} e^{L\Delta t}$ which gives the set S that is guaranteed to contain $\mathsf{Reach}(B_\delta(x), t)$ for every $t \in [t_{i-1}, t_i]$.

3. At Line 5, an interval matrix $\mathbb{A}$ containing $J_f(x)$ for each $x \in S$ is computed.

4. At Line 6, the "if" condition determines whether the M_{i-1}, γ_{i-1} used in the previous iteration satisfy the conditions of Lemma 5.2 (when $i = 1$, γ_0 is an initial guess). This condition will avoid performing updates of the discrepancy function if it is unnecessary. If the condition is satisfied, then M_{i-1} is used again for the current iteration i (Lines 7, 8, and 9) and γ_i will be computed as the smallest possible value such that Lemma 5.2 holds (Line 8) without updating the shape of the ellipsoid (i.e., $M_i = M_{i-1}$). In this case, the γ_i computed using M_{i-1} in the previous iteration $(i - 1)$ may not be ideal (minimum) for the current

iteration (i), but we assume it is acceptable.

5. At Line 11, when the "if" condition at Line 6 does not hold, which means that M_{i-1} and γ_{i-1} do not satisfy the conditions of Lemma 5.2, the previous discrepancy function can no longer ensure an accurate exponential converging or diverging rate between trajectories. Then M_i and γ_i are recomputed by solving (5.4).

6. At Line 12, the algorithm identifies the smallest constant c' for discrepancy function updating such that

$$E_{c_{i-1}}(\xi(x, t_{i-1}), M_{i-1}) \subseteq E_{c'}(\xi(x, t_{i-1}), M_i). \tag{5.8}$$

   This is because the shape matrices M_{i-1} and M_i for the ellipsoids are not equal for the two consecutive time intervals $[t_{i-2}, t_{i-1}]$ (or only t_{i-1} when $i = 1$) and $[t_{i-1}, t_i]$. We need to ensure that at the transition time t_{i-1}, ellipsoid $E_{c'}(\xi(x, t_{i-1}), M_i)$ as the initial set for the time interval $[t_{i-1}, t_i]$ is an over-approximation of the ellipsoid $E_{c_{i-1}}(\xi(x, t_{i-1}), M_{i-1})$, which is the reachable set of at time t_{i-1} for the time interval $[t_{i-2}, t_{i-1}]$.

   It is a standard SDP problem to compute the minimum value for c' that ensures (5.8) (see, for example [116]). We can use the "S procedure" [9] to transfer this optimization problem to the following sum-of-squares problem to make it solvable by SDP solvers:

$$\begin{aligned} \min \quad & c' \\ s.t. \quad & c'^2 - \|x - \xi(x_0, t_{i-1})\|^2_{M_i} - \lambda \left(c^2_{i-1} - \|x - \xi(x_0, t_{i-1})\|^2_{M_{i-1}} \right) \geq 0, \\ & \lambda \geq 0. \end{aligned} \tag{5.9}$$

7. At Line 14, the algorithm computes the updated ellipsoid size c_i such that $E_{c_i}(\xi(x, t_i), M_i)$ contains $\mathsf{Reach}(B_\delta(x), t_i)$.

8. At Line 15, the diameter of $E_{c_i}(\xi(x, t_i), M_i)$ is assigned to δ_i for next iteration.

9. At Line 16 the set O_i is computed such that it contains the reachable set during time interval $[t_{i-1}, t_i]$.

10. Finally, at Line 17, $\mathcal{R}$ is returned as an over-approximation of the entire reachable set.

The next lemma states that the γ produced by Line 11 is a local optimal exponential converging or diverging rate between trajectories.

Lemma 5.4 (Lemma 5.1 from [24])**.** In the i^{th} iteration of Algorithm 2, suppose $\mathbb{A}$ is the approximation of the Jacobian over $[t_{i-1}, t_i]$ computed in Line 5. If E_{i-1} is the reach set at t_{i-1}, then for all M' and γ' such that $\mathsf{Reach}(E_{i-1}, t_i) \subseteq E_{c'}(\xi(x, t_i), M')$ where c' is computed from γ' (Line 14), we have that the γ produced by Line 11 satisfies $\gamma \leq \gamma'$.

Proof. The lemma follows from the fact that any M', γ' that satisfies $MAM^{-1} + (M^{-1})^\top A^\top M^\top \preceq 2\gamma I, \forall A \in \mathbb{A}$ results in an ellipsoidal approximation at t_i that over-approximates the reach set; however, at Line 11 we are computing the minimum exponential change rate γ by searching all possible matrices M for the given interval matrix. Thus, the γ value computed at Line 11 is the optimal exponential change rate over local convex set S for the given interval matrix $\mathbb{A}$. □

In other words, the computed γ is the optimal exponential growth rate for any ellipsoidal reach set approximation, based on a given interval matrix approximation for the Jacobian.

Theorem 5.3 ensures soundness of the verification algorithm.

Theorem 5.3 (Theorem 5.2 from [24])**.** For any (x, T)-simulation $\psi = \xi(x, t_0), \ldots, \xi(x, t_k)$ and any constant $\delta \geq 0$, a call to $\texttt{Generalize}(\psi, \delta)$ returns a $(B_\delta(x), T)$-reachtube.

Proof. By Lemma 5.2, at any time $t \in [t_{i-1}, t_i]$, any other trajectory $\xi(x', t)$ starting from $x' \in E_{c_{i-1}}(\xi(x, t_{i-1}), M_{i-1})$ is guaranteed to satisfy

$$\|\xi(x, t) - \xi(x', t)\|_{M_i} \leq \|\xi(x, t_{i-1}) - x'\|_{M_i} e^{\gamma_i (t - t_{i-1})}. \tag{5.10}$$

Then, at time t_i, the reach set is guaranteed to be contained in the ellipsoid $E_{c_i}(\xi(x, t_i), M_i)$.

At Line 16 we want to compute the set O_i such that it contains the reach set during time interval $[t_{i-1}, t_i]$. According to Equation (5.10), at any time $t \in [t_{i-1}, t_i]$, the reach set is guaranteed to be contained in the ellipsoid

$E_{c(t)}(\xi(x,t), M_i)$, where $c(t) = c' e^{\gamma_i(t-t_{i-1})}$. O_i should contain all the ellipsoids during time $[t_{i-1}, t_i]$. Therefore, it can be obtained by bloating the rectangle R_i using the largest ellipsoid's radius. Since $e^{\gamma_i(t-t_{i-1})}$ is monotonic (increasing when $\gamma_i > 0$ or decreasing when $\gamma_i < 0$) with time, the largest ellipsoid during $[t_{i-1}, t_i]$ is either at t_{i-1} or at t_i. So the largest radius of the ellipsoids is $\max\{\mathsf{Rad}\left(E_{c'}\left(\xi(x, t_{i-1}), M_i\right)\right), \delta_i\}$. Thus, at Line 16, O_i computed at Line 16 is an over-approximation of the reach set during time interval $[t_{i-1}, t_i]$.

When $i = 1$, because the initial ellipsoid $E_{c_0}(x, M_0)$ contains the initial set $B_\delta(x)$, we have that $E_{c_1}(\xi(x, t_1), M_1)$ defined at Line 15 contains $\mathsf{Reach}(B_\delta(x), t_1)$. Also at Line 16, O_1 contains $\mathsf{Reach}(B_\delta(x), [t_0, t_1])$. Repeating this reasoning for subsequent iterations, we have that $E_{c_i}(\xi(x, t_i), M_i)$ contains $\mathsf{Reach}(B_\delta(x), t_i)$, and O_i contains $\mathsf{Reach}(B_\delta(x), [t_{i-1}, t_i])$. Therefore, $\mathcal{R}$ returned at Line 17 is a $(B_\delta(x), T)$-Reachtube. □

Remark 5.1. Algorithm 2 uses Lemma 5.2 to compute discrepancy functions, which can be replaced by Lemma 5.3 to decrease the computational cost by introducing some conservativeness in the over-approximation.

Remark 5.2. To modify Algorithm 2 to accept validated simulations and the error bounds introduced in Section 4.3, at Line 4 and Line 16, we need to bloat `hull`$(\{R_{i-1}, R_i\})$, which is guaranteed to contain the solution $\xi(x,t), \forall t \in [t_{i-1}, t_i]$. Also, at Line 12 and Line 15, the ellipsoid $E_{c_i}(\xi(x, t_i), M_i)$ should be replaced by $E_{c_i}(0, M_i) \oplus R_i$.

5.4.3 Advantages of the `Generalize` Algorithm

At the beginning of this section, we discussed the importance of a good discrepancy function for the overall data-driven verification approach. In the rest of this section, we show that Algorithm 2, using the discrepancy function from matrix measures, produces an accurate reachable set over-approximations with less computational loads.

Proposition 5.2 establishes that the bloating factor δ_i in Line 15 for constructing reachtubes goes to 0 as the size of the initial set $B_\delta(x)$ goes to zero. This implies that the over-approximation error from bloating can be made arbitrarily small by making the uncertainty in the initial cover $\langle x, \delta, \epsilon \rangle$ small.

Proposition 5.2 (Proposition 5.3 from [24]). In Algorithm `Generalize`, for any i, if M_0 and c_0 are optimal, in the sense that no M', c' exists such that $c' < c_0$ and $B_\delta(x) \subseteq E_{M',c'}(x)$, then as the radius of the initial ball $\delta \to 0$ the size of the bloating factor $\delta_i \to 0$ (Line 15).

The contractive system's Jacobian matrix has negative matrix measure under certain coordinate transformation. Next, Corollary 5.1 establishes that for contractive systems the reachtube computed by Algorithm `Generalize` converges to the rectangles that represent the simulation.

Corollary 5.1 (Corollary 5.3 from [24]). Consider a contractive system for which there exists a matrix M such that $\forall x \in \mathbb{R}^n, J_f^\top(x)M + MJ_f(x) \preceq 2\gamma M$, and $\gamma < 0$. Computing the reachtube of the system using Algorithm `Generalize`, we have as $k, T \to \infty$,

$$|\mathsf{Rad}(O_k) - \mathsf{Rad}\,(R_k)\,| \to 0.$$

A linear system is contractive if A is Hurwitz as the real part of its eigenvalues are bounded by some constant $\gamma < 0$. For (even unstable) linear invariant systems, since the Jacobian matrix A does not change over time, the discrepancy function can be computed globally for any time t and $x_1, x_2 \in \mathbb{R}^n$. Therefore, there is no accumulated error introduced using Algorithm `Generalize`. We have also shown the convergence of the algorithm for contractive nonlinear systems in Corollary 5.1. For non-contractive nonlinear systems, the over-approximation error might be accumulated. Such error caused by wrapping effect in the on-the-fly algorithms may not be avoidable. Therefore, for non-contractive or unstable nonlinear systems, it is especially important to reduce the over-approximation error in each time interval, which is what Algorithm `Generalize` aims to achieve.

For an n-dimensional system model, assume that there are N entries of the Jacobian matrix that are not a constant number. At any iteration, at Line 5, the algorithm solves $2N$ optimization problems or uses interval arithmetic to get lower and upper bounds of each component of the Jacobian. For linear time invariant systems, this step is eliminated. At Line 6 using Lemma 5.2 (vertex matrices) the algorithm computes 2^N matrix inequalities; however, using Lemma 5.3 (center matrix with error term) the algorithm computes 1 matrix inequality. At Line 8 or Line 11, using Lemma 5.2 the algorithm solves

1 convex optimization problem with $2^N + 1$ constraints, but using Lemma 5.3 the algorithm solves 1 convex optimization problem with 2 constraints. The discrepancy function updating at Line 12 solves 1 SDP problem. The rest of the algorithm from Line 14 to Line 16 consists of algebraic operations.

From the above analysis, we can see that Lemma 5.3 improves the efficiency of the algorithm as compared to Lemma 5.2, especially when the number of non-constant terms in the Jacobian matrix is large. However, using Lemma 5.3 may result in a more conservative over-approximation. We can consider Lemma 5.2 accurate but with a greater computational burden, and Lemma 5.3 simple but coarse.

As solving the SDP problem is known to be in polynomial time, using Lemma 5.3 will cost time exponentially with respect to the dimensionality, while Lemma 5.3 is still in polynomial time. It is worth noting that other methods like Taylor models suffer from complexity that increases exponentially with both the dimension of the system and the order of the model.

The effective efficiency of the algorithm depends on whether the system is contractive or not. For contractive systems, it is possible that the "if" condition often holds at Line 6, allowing the algorithm to often reuse the previous norm and contraction rate. For non-contractive systems this may not be the case. Also, the algorithm only computes the discrepancy function once for the linear system, since the interval matrix to which the Jacobian matrix belongs is time invariant.

5.4.4 Experimental Evaluation of Algorithm `Generalize`

We implemented `Generalize` in MATLAB and tested it on several benchmark verification problems. Simulations are generated using the validated simulation engine CAPD [109], which returns a sequence of time-stamped rectangles as required. The optimization problems (5.4), (5.6), and the SDP problems are solved using SDP3 [117] and Yalmip [118].

We compare the running time and accuracy of Algorithm `Generalize` against a leading nonlinear verification tool, Flow* [22]. As a measure of precision, we compare the ratio of the reachable set volume to the initial set volume since tools use different set representations. We calculate two volume ratios: (a) average volume of the reachable set divided by the initial volume (sampled at the time steps used in Flow*), (b) the reachable set at the final time point T divided by the initial volume.

We evaluated the algorithm on several nonlinear benchmarks. Van der Pol, Moore-Greitzer, and Brusselator are standard low-dimensional models. The Diode Oscillator from [49] is low dimensional but has complex dynamics described by degree 5 polynomials. Robot Arm is a 4-dimensional model from [119]. Powertrain is the benchmark control system proposed in [120]. This system has highly nonlinear dynamics with polynomials, rational functions, and square roots. Laub-Loomis is a molecular network that produces spontaneous oscillations as a case study for NLTOOLBOX [121]. AS-Polynomial is a 12-dimensional polynomial system [122] that is asymptotically stable around the origin. We also study a 28-dimensional linear model of a helicopter [18]. For the initial condition set of the helicopter model, we used 0.05 as the radius for the first eight dimensions and 0.0005 for the remaining ones, because the reach set estimations of Flow* became unbounded when using 0.05 as the diameter for all dimensions. For systems with fewer than three dimensions, we use Lemma 5.2, and for systems with higher dimensions, we use Lemma 5.3.

Table 5.1: n: Dimension of the system. δ: Radius of the initial set as a ball. T: Time horizon. Run Time: Running Time (includes simulation time for CAPD). A/I VR: the ratio of the average volume of the sampled over-approximation reachable set over the initial set volume. F/I VR: the ratio of the volume of the over-approximation reachable set at the final time T to the initial set volume.

	System	n	δ	T(s)	Algorithm `Generalize`			Flow*		
					Run Time(s)	A/I VR	F/I VR	Run Time(s)	A/I VR	F/I VR
1	Van der Pol	2	0.10	10	0.69	0.28	4.60e-4	1.66	0.26	1.61e-4
2	Moore-Greitzer	2	0.10	10	4.76	0.25	3.79e-4	4.67	0.34	2.16e-3
3	Brusselator	2	0.10	10	5.39	0.69	3.78e-2	6.66	0.33	2.41e-2
4	Diode Oscillator	2	0.005	9	39.98	0.65	9.08e-6	214.40	1.22	0.65
5	Robot Arm	4	0.005	10	9.26	1.83	0.31	354.80	3.35	2.97
6	Powertrain	4	2e-4	5	3.97	10.14	0.77	267.00	3457	1886
7	Saturation	6	0.02	10	1.99	2.19	1.83	5245	1.72	1.16
8	Laub-Loomis	7	0.025	10	5.97	11.11	6.44	355.10	3.72	0.88
9	Biology Model	7	2.5e-3	2	9.25	171.30	344.10	603.60	68.03	343.40
10	AS-Polynomial	12	2.5e-3	2	3.63	45.79	45.74	5525	3.06e10	2.87e10
11	Helicopter (L)	28	0.05	30	2.96	0.75	0.55	288.20	4.24e49	6.02e39

The results are shown in Table 5.1. From Line 1-3 we see that for simple low-dimensional nonlinear systems, the performance of Flow* is comparable to our algorithm. Lines 4-5 and 7-9 show that for more complicated nonlinear systems (with higher order polynomials or higher order dimensionality), our tool performs much better in running time without sacrificing accuracy. Moreover, from Line 6 and Lines 10-11, Algorithm `Generalize` not only finishes reachtube computation much faster, but also provides less conservative results for even more complicated systems (with complicated nonlinear dynamics or even higher dimensions). For linear systems, Algorithm `Generalize` can provide one global discrepancy function that is valid for the entire space to do reach set over-approximation, as compared to Flow*, where even for linear systems, the complexity for each time interval is exponential in both the dimensionality and the order of the Taylor models. Algorithm `Generalize` is more efficient because it is based on the Jacobian, which has n dimensions, so the complexity of Algorithm `Generalize` using the interval matrix norm method increases polynomially with the dimension.

5.5 Reachtube Computation for Hybrid System

In this section, we outline the hybrid extension of Algorithm 2 now presented as Algorithm 3. Algorithm 2 computes the set of reachable states for a given continuous system as described in Equation (3.1) for a given time interval. Therefore, one can essentially apply this algorithm for each of the relevant modes of a hybrid system. For simplicity, let us assume that all the mode invariants and transition guards are convex polyhedra, and that all the reset mappings are linear functions. Without loss of generality, we assume there is only one mode ℓ_{init} in the set of initial locations $\mathsf{L_{init}}$. Algorithm 3 performs the following three steps iteratively until the time horizon for verification.

1. For the given mode ℓ and a given initial set Θ, the algorithm first simulates from the center of Θ, computes the Jacobian of the continuous dynamics in mode ℓ, then computes the reachable set $\mathcal{R}_\ell$ for that mode from Θ for the bounded remaining time specified using Algorithm 2.

2. The reachable set is pruned by removing all the states that violate the mode invariant I_ℓ.

3. The reachable set is checked to satisfy any guards for discrete transitions. If the guards are satisfied, the initial states for the next mode are computed by applying the reset map of the states that satisfy the guard predicate. As the reachable set of states for a hybrid system at a

given time might belong to two different modes, we track the discrete transitions using a queue of tuples $\langle \Theta_{next}, \ell_{next}, t_{left} \rangle$, where ℓ_{next} is the next location that needs to be checked, Θ_{next} is the initial set corresponding to the location ℓ_{next}, and t_{left} is remaining time we need to compute the reachable set in ℓ_{next}.

Algorithm 3: Algorithm `HybridReachtube`

```
   input    : Hybrid System $\mathcal{H} = \langle V = (X \cup L), \Theta, \mathsf{L_{init}}, A, \mathcal{D}, \mathsf{TL} \rangle$, Time
              bound $T$, Lipschitz constants $\{L_\ell\}_{\ell \in \mathsf{L}}$, Parameters for
              validated simulation $\epsilon, \tau$.
   initially: $\mathsf{Q} \leftarrow \langle \Theta, \ell_{init}, T \rangle$, $\mathcal{R}_{\mathtt{hybrid}} \leftarrow \emptyset$
 1 for each $\langle \Theta, \ell, t_{left} \rangle \in \mathsf{Q}$ do
 2     $\psi = \{(R_i, t_i)_{i=0}^{k}\} \leftarrow \mathtt{Simulate}(\mathtt{center}(\Theta), t_{left}, \epsilon, \tau)$;
 3     Compute $M_0, c_0$ such that $\Theta \subseteq E_{c_0}(\mathtt{center}(\Theta), M_0)$;
 4     $J_{f_\ell}(x) \leftarrow$ Jacobian matrix of $f_\ell$ in mode $\ell$;
 5     $\mathcal{R}_\ell \leftarrow \mathtt{Generalize}(\psi, J_{f_\ell}(x), L_\ell, M_0, c_0)$;
 6     $\mathcal{R}_\ell \leftarrow \mathcal{R}_\ell \cap \mathsf{I}_\ell$;
 7     $\{\langle \Theta_{next}, \ell_{next}, t_{left} \rangle\} \leftarrow \mathtt{discreteTransitions}(\mathcal{R}_\ell)$;
 8     $\mathcal{R}_{\mathtt{hybrid}} \leftarrow \mathcal{R}_{\mathtt{hybrid}} \cup \mathcal{R}_\ell$;
 9     $\mathsf{Q}.\mathtt{append}(\{\langle \Theta_{next}, \ell_{next}, t_{left} \rangle\})$;
10 end
11 return $\mathcal{R}_{\mathtt{hybrid}}$ ;
```

Algorithm 3 computes the reachable set for a hybrid system. The main loop that performs the three key steps iteratively happens from Line 2 to Line 9. Line 2 simulates from the center state of Θ. Then at Line 3, we compute an ellipsoid $E_{c_0}(\mathtt{center}(\Theta), M_0)$ to contain the initial set Θ as an ellipsoidal initial set as required by Algorithm 2. Line 4 computes the Jacobian matrix of f_ℓ, continuous dynamics in mode ℓ. With these elements, at Line 5, we can use the `Generalize` function as Algorithm 2 to get the reachable set of states from Θ for the corresponding mode ℓ. Line 6 checks the invariant for the reachable set and line 7 computes the states reached Θ_{next} and the remaining time t_{left} to be checked after discrete transitions.

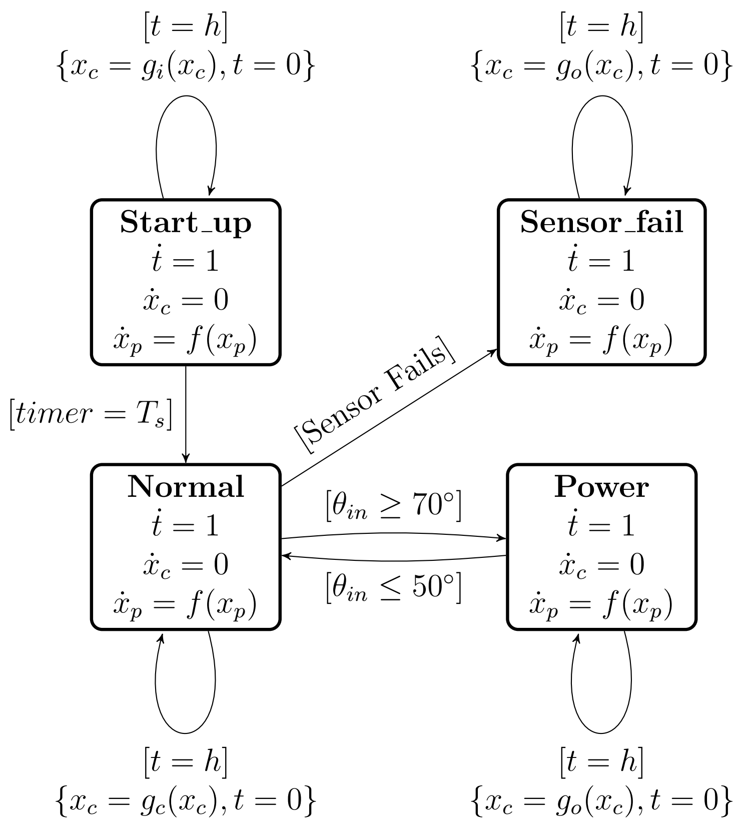
5.5.1 C2E2

Algorithm 2 is the core procedure implemented in the new version of the verification tool C2E2 [28] to automate the reachability analysis for nonlinear hybrid systems. Using the data-driven verification and discrepancy computation algorithms (Algorithms 1, 2, and 3), C2E2 can automatically check bounded time invariant properties of nonlinear hybrid automata and no longer needs the user-specified annotations. Hybrid models and the requirements have to be specified in an xml format. The tool parses the xml model to generate C++ libraries for numerical simulations, reachable set computations, checking safety, and handling discrete transitions. The new version of the tool also supports compositional modeling, a graphical user interface for model editing, and plotting.

In what follows, we introduce two hybrid system examples verified using the latest version of C2E2.

Example 5.3 (Powertrain control)**.** The demands for greater fuel efficiency and lower emissions constantly challenge automotive companies to improve control software in the powertrain systems. Recently a suite of benchmarks were published in [123] to introduce realistic, industrial scale models to the formal verification community. The suite consists of three Simulink models with increasing levels of complexity and sophistication. These models capture the behavior of chemical reactions in internal combustion engines, and hybrid models are deemed suitable for capturing the discrete transitions of control software and the continuous parameters in these models. At a high level, the models take inputs from a driver (throttle angle θ_{in}) and the environment (sensor failures), and define the dynamics of the engine. The key controlled quantity is the air-to-fuel ratio which in turn influences the emissions, the fuel efficiency, and torque generated.

The most complicated model (Model 1) in the suite captures all the interactions taking place in a physical process and faithfully models the control software. It contains several hierarchical components in Simulink with look-up tables, and delay differential equations. Model 1 is simplified to a model with periodic inputs to ordinary differential equations using several heuristics (Model 2), which as per the authors, exhibit behavior similar to that of Model 1. Then Model 2 is further simplified to a hybrid system with only polynomial ODEs (Model 3). At the time of publication of [123], these models were beyond the reach of the then available verification tools, but within a year the simplified models were verified using C2E2 [4], and subsequently, the more complex models were handled by another verification tool DryVR in [25], which will be introduced in Chapter 6.



$[t = h]$ $\{x_c = g_i(x_c), t = 0\}$
$[t = h]$ $\{x_c = g_o(x_c), t = 0\}$
Start_up $\dot{t} = 1$ $\dot{x}_c = 0$ $\dot{x}_p = f(x_p)$
Sensor_fail $\dot{t} = 1$ $\dot{x}_c = 0$ $\dot{x}_p = f(x_p)$
$[timer = T_s]$
[Sensor Fails]
Normal $\dot{t} = 1$ $\dot{x}_c = 0$ $\dot{x}_p = f(x_p)$
$[\theta_{in} \geq 70°]$
$[\theta_{in} \leq 50°]$
Power $\dot{t} = 1$ $\dot{x}_c = 0$ $\dot{x}_p = f(x_p)$
$[t = h]$ $\{x_c = g_c(x_c), t = 0\}$
$[t = h]$ $\{x_c = g_o(x_c), t = 0\}$

Figure 5.1: Hybrid system model of the powertrain control system Model 2.

In more detail, Model 2 and 3 have five variables: regulated throttle plate position θ, intake manifold pressure p, air-fuel ratio λ, intake manifold pressure estimate p_e and integrator state i, and four modes: Start_up, Normal, Power and Sensor_fail. The hybrid model also receives an input signal θ_{in} (throttle angle) as the user input. The required safety specification of powertrain control systems was given in [123] as a number of Signal Temporal Logic properties. Here we only illustrate one primary result for each model, with the simple unsafe set F: in Power mode, $t > 4 \vee \lambda \notin [12.4, 12.6]$, in Normal mode, $t > 4 \vee \lambda \notin [14.6, 14.8]$. [4, 25] provides more comprehensive studies involving other scenarios and requirements.

Figure 5.1 shows the hybrid model of the powertrain control system Model 2. The physical plant dynamics are modeled using continuous variables $x_p = [\theta, p, \lambda]$, which evolve according to a nonlinear ODE $\dot{x}_p = f(x_p)$. The controller variables $x_c = [p_e, i]$ are, instead, updated periodically every h time units by the reset functions $g_i(x_c), g_o(x_c), g_c(x_c)$ in different modes. Since not all the state variables are following ODEs, Algorithm 2 (and therefore C2E2) cannot be used to compute the reachtubes for Model 2. We defer the verification of this model to Chapter 6 where the entire system will be treated as a black-box simulator with the five given variables and four modes.

```
automaton PowertrainModel3
  actions
    Cruise; Throttle_increase ; Throttle_decrease; Fail
  variables
    state variables $[\theta, p, \lambda, p_e, i, t]^\top : \mathbb{R}^6$
    input variables $[T_s, \theta_{in}]^\top : \mathbb{R}^2$
    input variable $fail : \mathbb{B}$
    $\ell$ : enumeration [{Start_up}, {Normal}, {Power}, {Sensor_fail} ]
  initially
    $[\theta, p, \lambda, p_e, i, t]^\top := [8.8^\circ, 0.6215, 14.7, 0.5655, 0, 0]^\top$
    $[T_s, \theta_{in}]^\top := [9.5, 0]^\top$
    $fail :=$ false
    $\ell :=$ Start_up
  transitions
    Cruise
      pre $t = T_s \ \wedge\ \ell =$ Start_up
      eff $\ell :=$ Normal
    Throttle_increase
      pre $\theta_{in} \geq 70^\circ \ \wedge\ \ell =$ Normal
      eff $\ell :=$ Power_up
    Throttle_decrease
      pre $\theta_{in} \leq 50^\circ \ \wedge\ \ell =$ Power_up
      eff $\ell :=$ Normal
    Sensor_fail
      pre $fail =$ true $\wedge\ \ell =$ Normal
      eff $\ell :=$ Sensor_fail
  trajectories
    Start_up
    evolve
      $\dot\theta = 10(\theta_{in} - \theta)$
      $\dot p = c_1(2(c_6 + c_7\theta + c_8\theta^2 + c_9\theta^3)(c_{20}p^2 + c_{21}p$
        $+c_{22}) - c_{12}(c_2 + c_3wp + c_4wp^2 + c_5w^2p))$
      $\dot\lambda = 4(c_{15} + c_{16}c_{25}F_c + c_{17}c_{25}^2F_c^2 +$
        $c_{18}\dot m_c + c_{19}\dot m_c c_{25}F_c - \lambda)$
      $\dot{p_e} = c_1(2c_{23}(c_6 + c_7\theta + c_8\theta^2 + c_9\theta^3)(c_{20}p^2 + c_{21}p$
        $+c_{22}) - (c_2 + c_3wp_e + c_4wp_e^2 + c_5w^2p_e)))$
      $\dot i = 0$
      $\dot t = 1$
      where $F_c = \frac{1}{c_{11}}(c_2 + c_3wp_e + c_4wp_e^2 + c_5p_ew^2)$
        and $\dot m_c = c_{12}(c_2 + c_3wp + c_4wp^2 + c_5w^2p)$
    invariant $\ell =$ Start_up $\wedge\ t \leq 9.5 \wedge fail =$ false

Normal
evolve
  $\dot\theta = 10(\theta_{in} - \theta)$
  $\dot p = c_1(2(c_6 + c_7\theta + c_8\theta^2 + c_9\theta^3)(c_{20}p^2 + c_{21}p$
    $+c_{22}) - c_{12}(c_2 + c_3wp + c_4wp^2 + c_5w^2p))$
  $\dot\lambda = 4(c_{15} + c_{16}c_{25}F_c + c_{17}c_{25}^2F_c^2 +$
    $c_{18}\dot m_c + c_{19}\dot m_c c_{25}F_c - \lambda)$
  $\dot{p_e} = c_1(2c_{23}(c_6 + c_7\theta + c_8\theta^2 + c_9\theta^3)(c_{20}p^2 + c_{21}p$
    $+c_{22}) - (c_2 + c_3wp_e + c_4wp_e^2 + c_5w^2p_e)))$
  $\dot i = c_{14}(c_{24}\lambda - c_{11})$
  $\dot t = 0$
  where $F_c = \frac{1}{c_{11}}(1 + i + c_{13}(c_{24}\lambda - c_{11}))$
    $(c_2 + c_3wp_e + c_4wp_e^2 + c_5p_ew^2)$
  and $\dot m_c = c_{12}(c_2 + c_3wp + c_4wp^2 + c_5w^2p)$
invariant $\ell =$ Normal $\wedge \wedge fail =$ false

Power_up
evolve
  $\dot\theta = 10(\theta_{in} - \theta)$
  $\dot p = c_1(2(c_6 + c_7\theta + c_8\theta^2 + c_9\theta^3)(c_{20}p^2 + c_{21}p$
    $+c_{22}) - c_{12}(c_2 + c_3wp + c_4wp^2 + c_5w^2p))$
  $\dot\lambda = 4(c_{15} + c_{16}c_{25}F_c + c_{17}c_{25}^2F_c^2 +$
    $c_{18}\dot m_c + c_{19}\dot m_c c_{25}F_c - \lambda)$
  $\dot{p_e} = c_1(2c_{23}(c_6 + c_7\theta + c_8\theta^2 + c_9\theta^3)(c_{20}p^2 + c_{21}p$
    $+c_{22}) - (c_2 + c_3wp_e + c_4wp_e^2 + c_5w^2p_e)))$
  $\dot i = 0$
  $\dot t = 0$
  where $F_c = \frac{1}{c_{11}}(c_2 + c_3wp_e + c_4wp_e^2 + c_5p_ew^2)$
    and $\dot m_c = c_{12}(c_2 + c_3wp + c_4wp^2 + c_5w^2p)$
invariant $\ell =$ Power_up $\wedge fail =$ false

Sensor_fail
evolve
  $\dot\theta = 10(\theta_{in} - \theta)$
  $\dot p = c_1(2(c_6 + c_7\theta + c_8\theta^2 + c_9\theta^3)(c_{20}p^2 + c_{21}p$
    $+c_{22}) - c_{12}(c_2 + c_3wp + c_4wp^2 + c_5w^2p))$
  $\dot\lambda = 4(c_{15} + c_{16}c_{25}F_c + c_{17}c_{25}^2F_c^2 +$
    $c_{18}\dot m_c + c_{19}\dot m_c c_{25}F_c - \lambda)$
  $\dot{p_e} = c_1(2c_{23}(c_6 + c_7\theta + c_8\theta^2 + c_9\theta^3)(c_{20}p^2 + c_{21}p$
    $+c_{22}) - (c_2 + c_3wp_e + c_4wp_e^2 + c_5w^2p_e)))$
  $\dot i = 0$
  $\dot t = 0$
  where $F_c = \frac{1}{c_{11}}(c_2 + c_3wp_e + c_4wp_e^2 + c_5p_ew^2)$
    and $\dot m_c = c_{12}(c_2 + c_3wp + c_4wp^2 + c_5w^2p)$
invariant $\ell =$ Sensor_fail $\wedge fail =$ true
```

Figure 5.2: Hybrid automaton model of of the powertrain control system Model 2.

Model 2 got further simplified such that all five variables are continuous and follow a set of polynomial differential equations in Model 3. Figure 5.2 shows the hybrid automaton of Model 3 (see [123] for detailed values of the coefficients c_i). This model can be handled by C2E2. Figure 5.3 also shows the hybrid systems of Model 3, and Figure 5.4 gives a safe reachtube of λ from the initial set $\theta = 8.8^\circ, p \in [0.6115, 0.6315], \lambda \in [14.6, 14.8], p_e \in [0.5555, 0.5755], i \in [0, 0.01]$.

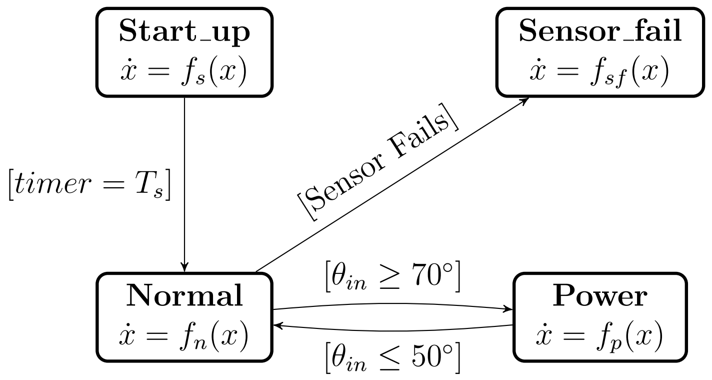


Figure 5.3: Hybrid system model of the powertrain control system Model 3.

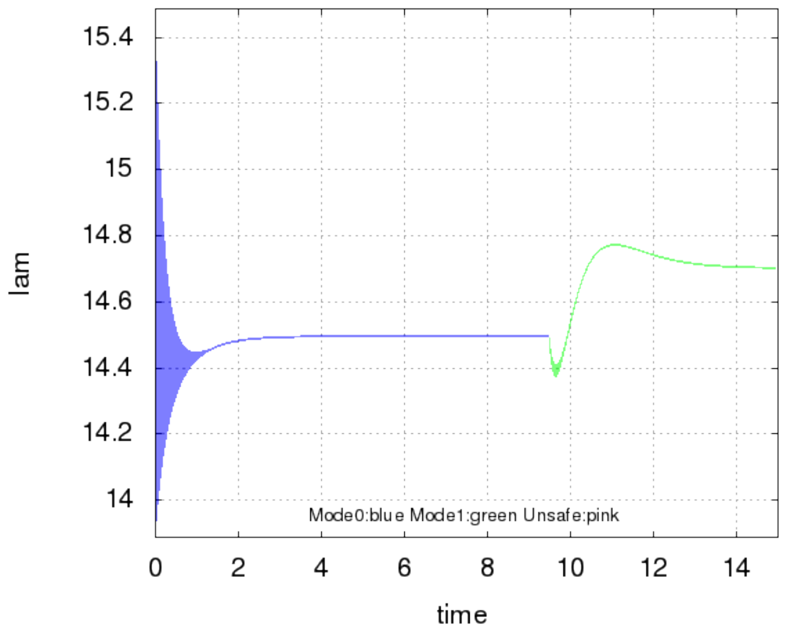


Figure 5.4: Reachtube for λ vs. time of Model 3 produced by C2E2. Blue and green regions correspond to the Start_up and Normal modes respectively.

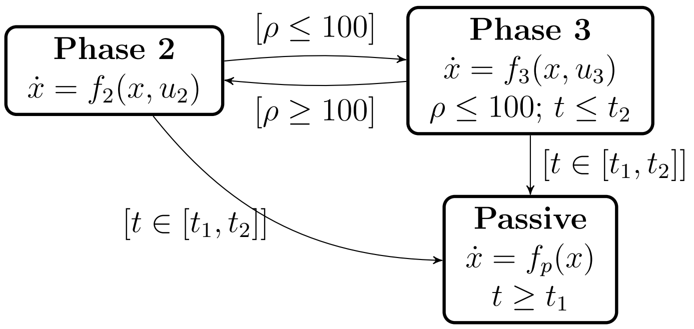


Figure 5.5: Hybrid system model of the autonomous spacecraft rendezvous mission.

Example 5.4 (Spacecraft rendezvous)**.** The extreme cost of failures and the infeasibility of terrestrial testing have made formal methods singularly attractive for space systems. Reachability-based automatic safety verification for satellite control systems were first studied in [124]. At the time of that study, hybrid verification tools were available only for linear hybrid systems, which have restricted applicability because many satellite control problems involve nonlinear orbital dynamics and nonlinear constraints. In this example we present a case study based on the ARPOD problem introduced in [75]. ARPOD stands for autonomous rendezvous proximity operations and docking. It captures an overarching mission needed to assemble a new space station that has been launched in separate modules. Our discussion here is based on the results presented in [30, 31].

A generic ARPOD scenario involves a passive module or a *target* (launched separately into orbit) and a *chaser* spacecraft that must transport the passive module to an on-orbit assembly location. The chaser maintains a relative bearing measurement to the target, but initially it is too far to use its range sensors. Once range measurements become available, the chaser gets more accurate relative positioning data and it can stage itself to dock with the target. Docking must happen with a specific angle of approach and closing velocity, in order to avoid collision and to ensure that the docking mechanisms on each spacecraft will mate.

For simplicity here we discuss the planar (or 2-dimensional) version of the model. The variables of the hybrid model include position (relative to the target) x, y (in meters), time t (in minutes), and horizontal and vertical velocity v_x, v_y. The modes of the hybrid automaton capture four phases of the docking maneuver. Each phase is defined by a separation distance $\rho =$

$\sqrt{x^2+y^2}$ between the chaser and target spacecraft, closing this distance from up to 10 km down to 0, and then performing a maneuver once the satellites are docked. The chaser spacecraft begins in Phase1 while the separation distance ρ is not available, but only has angle of approach $\theta = \texttt{atan}(\frac{y}{x})$ available, and the system is unobservable. While ρ gets small enough, the mission moves into Phase2, where the chaser spacecraft now has a ranging measurement to the chaser spacecraft and must position itself for the Phase 3 docking. After the chaser moves such that $\rho \leq 100$, the docking phase, Phase3 is initiated and additional docking port constraints are active. Once the spacecraft docks (i.e., $\rho = 0$), both spacecraft move into Phase4, where the joint assembly must move to the relocation position.

The chaser must adhere to different sets of constraints in each discrete mode. In [30] a switched linear quadratic regulator (LQR) is designed to meet these constraints, while maintaining liveness in navigating toward the target spacecraft. Figure 5.5 gives the hybrid system model of interest. In addition to the existing mode, the model also has a Passive mode in which the chaser has the thrusters shut down. The system may nondeterministically transition to the Passive mode as a result of a failure or loss of power. The nonlinear dynamic equations describing the motion of the chaser spacecraft relative to the target are:

$$\begin{cases} \dot{x} &= v_x \\ \dot{y} &= v_y \\ \dot{v_x} &= n^2x + 2nv_y + \frac{\mu}{r^2} - \frac{\mu}{r_c^3}(r+x) + \frac{u_x}{m_c} \\ \dot{v_y} &= n^2y - 2nv_x - \frac{\mu}{r_c^3}y + \frac{u_y}{m_c}. \end{cases}$$

The parameters are $\mu = 3.986 \times 10^{14} \times 60^2$ [m^3 / min^2], $r = 42164 \times 10^3$ [m], $m_c = 500$ [kg], $n = \sqrt{\frac{\mu}{r^3}}$ and $r_c = \sqrt{(r+x)^2+y^2}$. The linear feedback controllers for the different modes are defined as $[u_x, u_y]^T = K_1 z$ for mode Phase2, and $[u_x, u_y]^T = K_2 z$ for mode Phase2, where $z = [x, y, v_x, v_y]^T$ is the vector of system states. The feedback matrices K_i were determined with an LQR-approach applied to the linearized system dynamics, where the detailed number can be found at [30]. In mode Passive the system is uncontrolled $[u_x, u_y]^T = [0, 0]^T$. The spacecraft starts from the initial set $x \in [-925, -875]$ [m], $y \in [-425, -375]$ [m], $v_x = 0$ [m/min] and $v_y = 0$ [m/min]. For the considered time horizon of $t \in [0, 200]$ [min], the following

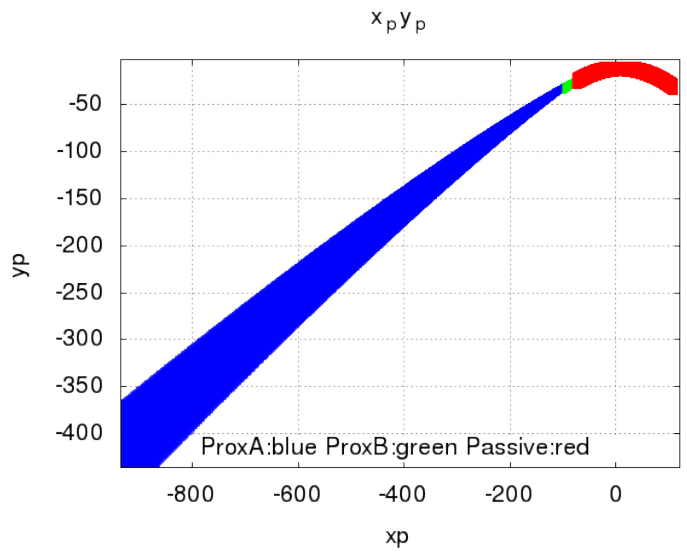


Figure 5.6: Reachtube of x (x-axis) vs y (y-axis) produced by C2E2.

specifications have to be satisfied:

- **Line-of-sight:** In mode Phase3, the spacecraft has to stay inside line-of-sight cone

$$\{[x,y]^T \mid (x \geq -100) \wedge (y \geq x\tan(30^\circ)) \wedge (-y \geq x\tan(30^\circ))\}.$$

- **Collision avoidance:** In mode Passive, the spacecraft has to avoid a collision with the target, which is modeled as a box B with 0.2m edge length and the center placed at the origin.

- **Velocity constraint:** In mode Phase3, the absolute velocity has to stay below 3.3 [m/min]:

$$\sqrt{v_x^2 + v_y^2} \leq 3.3 \text{ [m/min]}.$$

C2E2 was used to prove that the autonomous rendezvous system with the LQR controller satisfies the above requirements. Figure 5.6 shows the reachtube of x (x-axis) vs y (y-axis) produced by C2E2.

Both the powertrain control and the spacecraft rendezvous applications can be verified by C2E2 within a couple of minutes. Other than these two examples, C2E2 has been used to check properties for mixed-signal circuites [7], which have ODEs that contain hundreds of exponential and logarithmic terms. All these challenging models prove the capability of C2E2 as a promising tool for verifying nonlinear hybrid models and provide soundness and relative completeness guarantees.

5.6 Summary

In this chapter, we discussed several techniques to over-approximate reachable states of linear and nonlinear dynamical systems from simulation traces and discrepancy. We proposed a method to compute the upper bounds on the matrix measures, which was used to bloat the simulation traces to obtain the over-approximations. It computes locally optimal coordinate transformations such that the local exponential change rate of the discrepancy is minimized, which leads to over-approximations that are less conservative. To our knowledge, this is the first result with such local optimality guarantees. Our empirical results show that the approaches perform favorably compared to the Flow* tool on examples with higher dimensions or with complex dynamics. We also discussed how to extend the reachability algorithm to handle hybrid systems, which is implemented in the verification tool C2E2. Finally, we studied two applications of C2E2: a powertrain control system from Toyota and a spacecraft rendezvous scenario.

Chapter 6

Verification of Models with Black-box Components

In hybrid system models we have discussed thus far, the evolution of the continuous state variables is explicitly described by differential equations and trajectories. In real world control systems, “models” are typically a heterogeneous mix of simulation code, differential equations, block diagrams, and hand-crafted look-up tables. Extracting clean mathematical models (e.g. ODEs) from these descriptions is usually infeasible. At the same time, rapid developments in advanced driving assist systems (ADAS), autonomous vehicles, robotics, and drones now make the need for effective and sound verification algorithms stronger than ever before. The high-level logic deciding the transitions of when and for how long the system stays in each mode is usually implemented in a relatively clean piece of code and this logical module can be seen as the discrete transition system as Definition 3.1 in Section 3.1. In contrast, the dynamics of physical plant, with hundreds of parameters, is more naturally viewed as a “black-box”. That is, it can be simulated or tested with different initial conditions and inputs, but it is nearly impossible to write down a nice mathematical model. This unavailability of explicit “white-box” models, is a major roadblock to making formal techniques practical for CPS. In this chapter, will view hybrid systems as a combination of a “white-box” discrete transition system that specifies the mode switches and a “black-box” that can simulate the continuous evolution in each mode. The DryVR framework presented in this chapter aims to narrow the gap between sound and practical verification for control systems. This work was also originally presented in [25]. We will also present a case study on analyzing risks of a set of automatic emergency braking systems using DryVR [36] in this chapter.

6.1 Introduction

Consider an ADAS feature like an automatic emergency braking system (AEB). The high-level logic deciding the timing of when and for how long the brakes are engaged after an obstacle is detected by sensors is implemented in a relatively clean piece of code and this logical module can be seen as a *white-box*. In contrast, the dynamics of the vehicle itself, with hundreds of parameters, is more naturally viewed as a *black-box*. That is, it can be simulated or tested with different initial conditions and inputs, but it is nearly impossible to write down a nice mathematical model.

The empirical observation motivating this work is that many control systems, and especially automotive systems, share this combination of white and black boxes (see other examples in Section 6.3.3). In this chapter, we view a hybrid system as a combination of a white-box that specifies the mode switches and a black-box that can simulate the continuous evolution in each mode. We still consider the hybrid systems as defined in Definition 3.2: $\mathcal{H} = \langle V = (X \cup L), \Theta, \mathsf{L}_{\mathsf{init}}, A, \mathcal{D}, \mathsf{TL} \rangle$, where V is the set of variables, Θ and $\mathsf{L}_{\mathsf{init}}$ are initial states for the continuous and discrete variables respectively, A is a set of actions or transition labels, $\mathcal{D}$ is the set of transitions, and TL is a set of deterministic trajectories. Each trajectory $\xi(t)$ in TL is a function of the initial state $\xi(0)$ and time t. Also, TL is not a fixed or finite set of trajectories. We can generate different deterministic trajectories by setting different initial states. Note here the definition of TL does not cover trajectories generated from models with nondeterministic noise. Compared with the hybrid systems we studied in Chapter 4 and Chapter 5, in this chapter, we do not have a closed form description of TL like ODEs, but instead, we have a *simulator*, that can generate sampled data points on individual trajectories for a given initial state and mode.

With black-box modules in our hybrid systems, the main challenge is to compute discrepancy functions (see Section 4.4 for detailed definition) for modes that are now represented by black-boxes. We introduce a probabilistic algorithm that learns the parameters of exponential discrepancy functions from simulation data. The algorithm transforms the problem of learning the parameters of the discrepancy function to the problem of learning a linear separator for a set of points in $\mathbb{R}^2$ that are obtained from transforming the simulation data. A classical result in PAC learning [35] ensures that any such discrepancy function works with high probability for all trajectories.

We performed dozens of experiments with a variety of black-box simulators and observed that 15-20 simulation traces typically give a discrepancy function that works for nearly 100% of all simulations. The reachability algorithm for the hybrid system is still given by Algorithm 3, with the modification that at Line 5, instead of using `Generalize`, we will use the learned discrepancy function to give us the reachtube. The algorithm gives a sound bounded time verification algorithm, provided the learned discrepancy function is correct.

White-box discrete transitions identify the switching sequences under which the black-box modules are exercised. There is a simpler class of hybrid systems where the transitions will only depend on time (for example, the cardiac pacemaker system as in Example 3.3). In this case, the underlying mode switches can be seen as a one-clocked timed automaton [14], or *timed transition graph*. To enable the analysis of such systems, we identify reasoning principles that establish the safety of a system under a complex timed transition graph based on its safety under a simpler timed transition graph. We define a notion of forward simulation between timed transition graphs that provides a sufficient condition of when one timed transition graph "subsumes" another — if G_1 is simulated by G_2 then the reachable states of a hybrid system under G_1 are contained in the reachable states of the system under G_2. Thus the safety of the system under G_2 implies the safety under G_1. Moreover, we give a simple polynomial time algorithm that can check if one timed transition graph is simulated by another.

Our timed transition graphs are acyclic with transitions having bounded switching times. Therefore, the executions of the systems we analyze are over a bounded time, and have a bounded number of mode switches. An important question to investigate is whether establishing the safety for bounded time enables one to establish the safety of the system for an arbitrarily long time and for arbitrarily many mode switches. With this in mind, we define a notion of sequential composition of timed transition graphs G_1 and G_2, such that switching sequences allowed by the composed graph are the concatenation of the sequences allowed by G_1 with those allowed by G_2. Then we prove a sufficient condition on a timed transition graph G such that safety of a system under G implies the safety of the system under arbitrarily many compositions of G with itself.

We have implemented these ideas to create the **D**ata-d**r**iven S**y**stem for

Verification and **R**easoning (DRYVR). The tool is able to automatically verify or find counter-examples in a few minutes, for a suite of automotive systems such as powertrain control [123], automatic transmission control [125], and ADAS features like automatic emergency braking (AEB), lane-change, and auto-passing. Reachability analysis combined with compositional reasoning enabled us to infer safety of systems with respect to arbitrary transitions and duration.

6.2 Hybrid Systems with Black-box Modules

Consider a hybrid system $\mathcal{H} = \langle V = (X \cup L), \Theta, \mathsf{L}_{\mathsf{init}}, A, \mathcal{D}, \mathsf{TL}\rangle$. $\mathsf{L} = \mathsf{val}(L)$ is a finite set of *modes* and $\mathsf{X} = \mathsf{val}(X) \subseteq \mathbb{R}^n$ is the space for continuous states. The black-box generates a set of trajectories TL in X for each mode in L. We denote by $\mathsf{TL}_{init,\ell} = \{\xi.\mathsf{fstate} \mid \langle \xi, \ell\rangle \in \mathsf{TL}\}$ the set of initial states of trajectories in TL that start in mode ℓ. Without loss of generality we assume that $\mathsf{TL}_{init,\ell}$ is a connected, compact subset of X.

Instead of a closed form description of TL as in Definition 3.2, we have a *simulator* that can generate sampled data points on individual trajectories. We will develop techniques that avoid over-reliance on the models generating the trajectories and instead work with sampled data of $\xi(\cdot)$ generated from the simulators. Of course, in order to obtain safety guarantees we will need to make assumptions about the underlying system generating the data.

Definition 6.1. A simulator for a (deterministic and prefix-closed) set TL of trajectories labeled by L is a function (or a program) `sim` that takes as input a mode label $\ell \in \mathsf{L}$, an initial state $x_0 \in \mathsf{TL}_{init,\ell}$, and a finite sequence of time points $t_1, \ldots, t_k$, and returns a sequence of states $\mathtt{sim}(x_0, \ell, t_1), \ldots, \mathtt{sim}(x_0, \ell, t_k)$ such that there exists $\langle \xi, \ell\rangle \in \xi$ with $\xi.\mathsf{fstate} = x_0$ and for each $i \in \{1, \ldots, k\}$, $\mathtt{sim}(x_0, \ell, t_i) = \xi(t_i)$.

For simplicity, we assume that the simulations are perfect (as in the last equality of Definition 6.1). Formal guarantees of soundness are not compromised if we use validated simulations (Definition 4.1) instead. Executions and reachable states are defined in the same way as those in Section 3.4.

6.3 Learning Discrepancy from Simulations

The key subroutine needed for computing the reachable states with Algorithm 1 has to compute a discrepancy function which upper bounds the distance between trajectories. Owing to the absence of ODE models, the `Generalize` function of Algorithm 2 is useless. We will use a probabilistic algorithm for estimating the discrepancy from the data generated by black-box simulators [25].

Recall that a discrepancy function is a continuous function $\beta : \mathbb{R}^n \times \mathbb{R}^{\geq 0} \rightarrow \mathbb{R}^{\geq 0}$, such that for any pair of identically labeled trajectories $\langle \xi_1, \ell \rangle, \langle \xi_2, \ell \rangle \in \mathsf{TL}$, and any $t \in \xi_1.\mathsf{dom} \cap \xi_2.\mathsf{dom}$: (a) β upper-bounds the distance between the trajectories, i.e.,

$$\|\xi_1(t) - \xi_2(t)\| \leq \beta(\|\xi_1.\mathsf{fstate} - \xi_2.\mathsf{fstate}\|, t), \tag{6.1}$$

and (b) β converges to 0 as the initial states converge, i.e., for any trajectory ξ and $t \in \xi.\mathsf{dom}$, if a sequence of trajectories $\xi_1, \ldots, \xi_k, \ldots$ has $\xi_k.\mathsf{fstate} \rightarrow \xi.\mathsf{fstate}$, then $\beta(\|\xi_k.\mathsf{fstate} - \xi.\mathsf{fstate}\|, t) \rightarrow 0$. We present a simple method for discovering discrepancy functions that only uses simulations. Our method is based on a classical result in PAC learning theory [35]. We revisit this result before applying it to finding discrepancy functions.

Definition 6.2 (Learning linear separators)**.** For $\Gamma \subseteq \mathbb{R} \times \mathbb{R}$, a *linear separator* is a pair $(a, b) \in \mathbb{R}^2$ such that

$$\forall (x, y) \in \Gamma.\ x \leq ay + b. \tag{6.2}$$

Let us fix a subset Γ that has a (unknown) linear separator (a_*, b_*). Our goal is to discover some (a, b) that is a linear separator for Γ by sampling points in Γ [1]. The assumption is that elements of Γ can be drawn according to some (unknown) distribution $\mathcal{D}$. With respect to $\mathcal{D}$, the *error* of a pair (a, b) from satisfying Equation 6.2, is defined to be $\mathsf{err}_{\mathcal{D}}(a, b) = \mathcal{D}(\{(x, y) \in \Gamma \mid x > ay + b\})$ where $\mathcal{D}(X)$ is the measure of set X under distribution $\mathcal{D}$. Thus, the error is the measure of points (w.r.t. $\mathcal{D}$) that (a, b) is not a linear separator for. There is a very simple (probabilistic) algorithm that finds a

[1]We prefer to present the learning question in this form as opposed to one where we learn a Boolean concept because it is closer to the task at hand.

pair (a, b) that is a linear separator for a large fraction of points in Γ, as follows.

1. Draw k pairs $(x_1, y_1), \ldots (x_k, y_k)$ from Γ according to $\mathcal{D}$; the value of k will be fixed later.
2. Find $(a, b) \in \mathbb{R}^2$ such that $x_i \leq ay_i + b$ for all $i \in \{1, \ldots k\}$.

Step 2 involves checking feasibility of a linear program, and so can be done quickly. This algorithm, with high probability, finds a linear separator for a large fraction of points.

Proposition 6.1 (Proposition 4 from [25])**.** Let $\epsilon, \delta \in \mathbb{R}^{\geq 0}$. If $k \geq \frac{1}{\epsilon} \ln \frac{1}{\delta}$, then, with probability $\geq 1 - \delta$, the above algorithm finds (a, b) such that $\mathsf{err}_{\mathcal{D}}(a, b) < \epsilon$.

Proof. The result follows from the PAC-learnability of concepts with low VC-dimension [35]. However, since the proof is very simple in this case, we reproduce it here for completeness. Let k be as in the statement of the proposition, and suppose the pair (a, b) identified by the algorithm has error $> \epsilon$. We will bound the probability of this happening.

Let $B = \{(x, y) \mid x > ay + b\}$. We know that $\mathcal{D}(B) > \epsilon$. The algorithm chose (a, b) only because no element from B was sampled in Step 1. The probability that this happens is $\leq (1 - \epsilon)^k$. Observing that $(1 - s) \leq e^{-s}$ for any s, we get $(1 - \epsilon)^k \leq e^{-\epsilon k} \leq e^{-\ln \frac{1}{\delta}} = \delta$. This gives us the desired result. $\square$

6.3.1 Discrepancy Functions as Linear Separators

Using the above result, we will compute discrepancy functions from simulation data, independently for each mode. Let us fix a mode $\ell \in \mathsf{L}$, and a domain $[0, T]$ for each trajectory. The discrepancy functions that we will learn from simulation data will be of two different forms, and we discuss how these are obtained.

Global exponential discrepancy (GED) is a function of the form

$$\beta(\|x_1 - x_2\|, t) = \|x_1 - x_2\| K e^{\gamma t}.$$

Here K and γ are constants. Thus, for any pair of trajectories ξ_1 and ξ_2 (for mode ℓ), we have

$$\forall t \in [0, T].\ \|\xi_1(t) - \xi_2(t)\| \leq \|\xi_1.\mathsf{fstate} - \xi_2.\mathsf{fstate}\| K e^{\gamma t}.$$

Taking logs on both sides and rearranging terms, we have

$$\forall t.\ \ln \frac{\|\xi_1(t) - \xi_2(t)\|}{\|\xi_1.\mathsf{fstate} - \xi_2.\mathsf{fstate}\|} \leq \gamma t + \ln K.$$

It is easy to see that a global exponential discrepancy is nothing but a linear separator for the set Γ consisting of pairs $(\ln \frac{\|\xi_1(t)-\xi_2(t)\|}{\|\xi_1.\mathsf{fstate}-\xi_2.\mathsf{fstate}\|}, t)$ for all pairs of trajectories ξ_1, ξ_2 and time t. Using the sampling based algorithm described under Definition 6.2, we could construct a GED for a mode $\ell \in \mathsf{L}$, where in step 1, drawing samples from Γ reduces to using the simulator to generate traces from different states in $\mathsf{TL}_{init,\ell}$. Proposition 6.1 guarantees the correctness, with high probability, for any separator discovered by the algorithm. However, for our reachability algorithm to not be too conservative, we need K and γ to be small. Thus, when solving the linear program in Step 2 of the algorithm, we search for a solution minimizing $\gamma T + \ln K$.
Piece-wise exponential discrepancy (PED) is the second form of discrepancy functions we consider. It depends on dividing up the time domain $[0, T]$ into smaller intervals, and finding a global exponential discrepancy for each interval. Let $0 = t_0, t_1, \ldots t_N = T$ be an increasing sequence of time points. Let $K, \gamma_1, \gamma_2, \ldots \gamma_N$ be such that for every pair of trajectories ξ_1, ξ_2 (of mode ℓ), for every $i \in \{1, \ldots, N\}$, and $t \in [t_{i-1}, t_i]$, $\|\xi_1(t) - \xi_2(t)\| \leq \|\xi_1(t_{i-1}) - \xi_2(t_{i-1})\| K e^{\gamma_i t}$. Under such circumstances, the discrepancy function itself can be seen to be given as

$$\beta(\|x_1 - x_2\|, t) = \|x_1 - x_2\| K e^{\sum_{j=1}^{i-1} \gamma_j (t_j - t_{j-1}) + \gamma_i (t - t_{i-1})} \qquad \text{for } t \in [t_{i-1}, t_i].$$

If the time points $0 = t_0, t_1, \ldots t_N = T$ are fixed, then the constants $K, \gamma_1, \gamma_2, \ldots$ γ_N can be discovered using the learning approach described for GED; here, to discover γ_i, we take Γ_i to be the pairs obtained by restricting the trajectories to be between times t_{i-1} and t_i. The sequence of time points t_i are also dynamically constructed by our algorithm based on the following approach. Our experience suggests that a value for γ that is ≥ 2 results in very conservative reach tube computation. Therefore, the time points t_i are constructed inductively to be as large as possible, while ensuring that $\gamma_i < 2$.

6.3.2 Learned Discrepancy and Guarantees in Practice

In theory, there is some probability that the learned discrepancy function β is incorrect. That is, some pair of executions $\xi, \xi' \in \mathsf{TL}$ of the system, starting from the same initial state Θ, diverges more than the bound given by the computed β. However, experiments in [25] on dozens of modes with complex, nonlinear trajectories suggest that this almost never happens. In the reported experiments, for each mode a set S_{train} of simulation traces that start from independently drawn random initial states in $\mathsf{TL}_{init,\ell}$ are used to learn a discrepancy function. Each trace has between 100 and $10,000$ data points, depending on the relevant time horizon and sample times. Then another set S_{test} of 1000 simulation traces are drawn for validating the computed discrepancy. For every pair of traces in S_{test} and for every time point, it is checked whether the computed discrepancy satisfies Equation 6.1. It is observed that for $|S_{\mathsf{train}}| > 10$ the computed discrepancy function is correct for 96% of the points in S_{test}, and for $|S_{\mathsf{train}}| > 20$ it is correct for more than 99.9%, across all experiments.

6.3.3 DryVR

Replacing the `Generalize` function in Algorithm 3 with a subroutine for learning discrepancy, we can obtain a complete verification algorithm for black-box hybrid models. This and the data-driven verification algorithm (Algorithm 1) are the core of the approach implemented in the open source DryVR verification tool [25, 34]. The tool supports other forms of discrepancy functions (for example, piece-wise exponential and polynomial) that can also be learned from simulation data with the same type of guarantees. DryVR has been effectively employed to analyze spacecraft control systems and maneuvers involving multiple autonomous and semi-autonomous vehicles.

The mode switches, simulator locations, name of variables, and unsafe set are specified in a JSON file. The black-box simulators have to be provided as separate functions. For example, the following JSON files specify the mode transitions of Example 3.3 (the same as the automaton shown in Figure 3.4), without knowing the underlying dynamics for the variables t, u, v.

```
{
  "vertex":["Stim_on","Stim_off"],
  "edge":[[0,1],[1,0]],
  "variables":["t","u","v"],
  "guards":[
    "(t>=5)",
    "(t>=20)"
  ],
  "resets":[
    "(t==0)",
    "(t==0)"
  ],
  "initialSet":[[0.0,0.0,0.0],[0.0,0.2,0.2]],
  "unsafeSet":"@Allmode:And(u>0.6,v>0.25)",
  "timeHorizon":50.0,
  "directory":"examples/cardiac"
}
```

In order to use the DryVR approach on a system, we have to make some practical choices. First, we could treat the whole system as a black-box with a single mode. At the other extreme, we could first identify the high-level modes and the mode switching logic, for example, by inspecting the model block diagrams or through program analysis, and then treat the individual modes as black-boxes. There are other intermediate policies. Second, for the black-box modules, we need to choose the system variables that are to be considered for reachability analysis. The variable valuations have to be computable from the simulation traces; they should be adequate for inferring the safety properties; and they should be resettable, that is, the simulation engine should be capable of changing the initial values of these variables to generate different simulation traces.

In what follows, we discuss several case studies using DryVR, including ADAS features like automatic emergency braking (AEB), lane-change, and auto-passing. We will also study a more complicated model of the power-train control model (Example 5.3) and spacecraft rendezvous problem (Example 5.4), along with a automatic transmission control system [125].

6.4 Example: ADAS and Autonomous Driving Control

Growth of autonomy and advanced driver assist (ADAS) features in cars has led to significant pressure to assure system-level safety at design time. The broad topic of safety certification for such systems is currently a big technical challenge. While this topic touches multiple technical challenges in several disciplines that are beyond the scope of our discussion (for example, human-autonomy interactions, traffic modeling, and testing for different weather conditions), formal verification, and in particular data-driven verification, can play an effective role for creating safety assurance cases needed for certification with standards like the ISO2626 [126]. Here we discuss several benchmarks we have created representing various common scenarios used for testing ADAS and autonomous driving control systems.

Each scenario for safety verification is constructed by composing several hybrid automaton models—one for each vehicle or road agent. Each vehicle has several continuous variables including the x, y-coordinates of the vehicle on the road, its velocity, heading, and steering angle. The detailed dynamics of each vehicle come from a black-box simulator (for example, written in Python or MatLab). For the experiments in this chapter, we use a standard kinematic steering-based vehicle model from Mathworks as a black-box.[2]

The higher-level decisions about the modes (for example, for "cruising", "speeding", "merging left", etc.) followed by the vehicles are captured by control graphs. In more detail, a vehicle can be controlled by two input signals, namely the throttle (acceleration or brake) and the steering. By choosing appropriate values for these input signals, the modes are defined—cruise: move forward at constant speed, speedup: constant acceleration, brake: constant (slow) deceleration, and em_brake: constant (hard) deceleration. In addition, we have designed lane switching modes change_left and change_right in which the acceleration and steering are controlled in such a manner that the vehicle switches to its left (resp. right) lane in a certain amount of time. The switching rules (guards) between the modes are defined by "driver models". The composed hybrid automaton is then presented to DryVR as the input model.

[2]https://www.mathworks.com/matlabcentral/fileexchange/54852-simple-2d-kinematic-vehicle-steering-model-and-animation?requestedDomain=www.mathworks.com

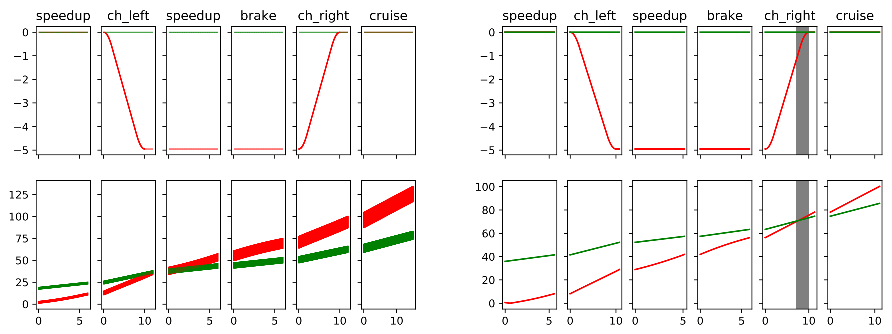


Figure 6.1: Verification of the vehicles overtake scenario. Left: Safe reachtube. Right: Unsafe execution. Vehicle A's (red) modes are shown above each subplot. Vehicle B (green) is in cruise. Top: sx_A, sx_B vs. time. Bottom: sy_A, sy_B vs. time.

Consider a scenario AutoPassing, where Vehicle A is behind B in the same lane starting with the same speed, and A wants to overtake B. A will switch to the left lane after it approaches B, then switch back to the right lane once it is ahead of B. In some cases, A may fail to get ahead of B, in which case it times out and returns in the right lane behind B. The safety requirement is that the vehicles maintain safe separation. Figure 6.1 (*left*) shows a version of this scenario that is verified to be safe by DryVR. The plots show the reachtube over-approximations computed by DryVR. Vehicle B stays in the cruise always but Vehicle A goes through a sequence of modes speedup, change_left, speedup,brake, and change_right, cruise to overtake B. Figure 6.1 left, top shows the projection of reachtubes on lateral positions (sx_A in red and sx_B in green), and the bottom plot shows the positions along the lane (sy_A in red and sy_B in green). Initially, for both $i \in \{A, B\}$, $sx_i = vx_i = 0$ and $vy_i = 1$, i.e., both are cruising at constant speed at the center of the right lane; initial positions along the lane are $sy_A \in [0, 2], sy_B \in [15, 17]$. As time advances, Vehicle A moves to left lane (sx decreases) and then back to the right, while B remains in the right lane, as A overtakes B (bottom plot). With a different initial set, $sy_B \in [30, 40]$, DryVR finds counter-example demonstrating unsafe behavior of the system (Figure 6.1 (right)).

We also construct the following scenarios by defining appropriate sets of initial states and mode transitions of two or more vehicles.

Merge: Initial condition: vehicle A is in left and vehicle B is in the right lane; initial positions and speeds are in some range; A is in cruise mode, and B is in cruise or speedup. Mode transitions: Vehicle A goes through

Table 6.1: Safety verification results. Numbers below benchmark names: # vertices and edges of G, TH: duration of shortest path in G, Ref: # refinements performed, Runtime: overall running time.

Model	TH	Initial set	Ref	Safe	Runtime
AutoPassing (12 vers, 13 edges)	50	$sy_A \in [-1,1]$ $sy_B \in [14,16]$	4	Safe	208.4s
	50	$sy_A \in [-1,1]$ $sy_B \in [4,6.5]$	5	Unsafe	152.5s
Merge (7 vers, 7 edges)	50	$sx_A \in [-5,5]$ $sy_B \in [-2,2]$	0	Safe	55.0s
	50	$sx_A \in [-5,5]$ $sy_B \in [2,10]$	-	Unsafe	38.7s
Merge3 (6 vers, 5 edges)	50	$sy_A \in [-3,3]$ $sy_B \in [14,23]$ $sy_C \in [36,45]$	4	Safe	197.6s
	50	$sy_A \in [-3,3]$ $sy_B \in [14,15]$ $sy_C \in [16,20]$	-	Unsafe	21.3s

the mode sequence cruise, speedup, change_right, cruise over specified intervals to merge to the right lane. Variants of this scenario involve B also switching to speedup or brake. Requirement: A merges to the right lane within a time bound and maintains at least a given safe separation.

AutoPassing**:** Initial condition: vehicle A behind B in the same lane, with A in speedup and B in cruise; initial positions and speeds are in some range. Mode transitions: A goes through the mode sequence change_left, speedup, brake, and change_right, cruise with specified time intervals in each mode to complete the overtake maneuver. If B switches to speedup before A enters speedup then A aborts and changes back to right lane. If B switches to brake before A enters change_left, then A should adjust the time to switch to change_left to avoid collision.

Merge3**:** Initial conditions: same as AutoPassing with a third car C always ahead of B. Mode transitions: vehicle A and vehicle B are the same as AutoPassing, vehicle C is always in the cruise mode. Requirement: vehicle A overtakes B while maintaining minimal safe separation.

Table 6.1 summarizes some of the verification results obtained using DryVR. These experiments were performed on a laptop with Intel Core i7-6600U CPU and 16 GB RAM. The initial range of only the salient continuous variables are shown in the table. The unsafe sets are ($|sx_A - sx_B| < 2$ & $|sy_A - sy_B| < 2$). For all the benchmarks, the algorithm terminated in a few minutes which includes the time to simulate, learn discrepancy, generate reachtubes, check the safety of the reachtube, and make overall refinements. For the results presented in Table 6.1, we used GED. However, the reachtubes and the verification times using both GED and PED were comparable.

6.5 Example: Analyzing Risk in Automatic Emergency Braking Systems

Here we summarize a comprehensive case study from [36] which looks at the most common type of rear-end crashes involving automatic emergency braking (AEB) and forward collision.

6.5.1 Overview

More than 25% of all reported accidents are rear-end crashes [127], of which around 85% happen on straight roads. Emergency braking and forward collision warning systems are becoming standard ADAS features. However, testing the safety of such systems can be nontrivial. The safe braking profiles for a sequence of cars on the highway depend on several factors—initial separation, velocities, vehicle dynamics, reaction times, road surface etc. The DryVRtool works with black-box or unknown vehicle models, using which we can prove, for example, that a given braking profile is safe for a set of scenarios characterized by the ranges of initial separation (d) and reaction times (r). For unsafe scenarios, DryVRcan compute the worst case relative velocity of the collision, which determines the severity of the accident. This type of analysis can be used as a design tool for tuning the braking profiles for different highway speeds, road conditions, etc. In this section, we will analyze hundreds of scenarios to generate a safety surface that can aid such design and analysis.

Consider for example an automatic emergency braking (AEB) system (Figure 6.2): Car1, Car2 and Car3 are cruising with zero relative speeds and certain initial relative separation; Car1 suddenly switches to a braking mode and starts slowing down according to a certain deceleration profile. Irrespective of whether Car2 is human-driven, AEB-equipped, or fully autonomous, a certain amount of time elapses before Car2 switches to a braking mode. We call this the *reaction time* r. Similarly, the mode switch for Car3 happens after a delay. Obviously, Car2's safety is in jeopardy: if it brakes "too hard" it will be rear-ended and if it is "too gentle" then it would have a forward collision. The envelope of safe (no collision) behaviors depends on all the parameters: initial separations, velocities, braking profiles, and reaction times. It is easy to see that if we can solve the safety verification problem described above, then we can also compute this envelope and determine whether a given AEB system is safe over a range of scenarios.

Each car can be viewed as a hybrid system: it has several continuous state variables: deceleration rate $a(t)$, velocity $v(t)$ and position $s(t)$ and two modes: cruise and brake. In the cruise mode, the car maintains a constant speed, and in the brake mode, it decelerates according to a certain braking profile for $a(t)$, which is an input to the system. The initial set K of the system is defined by the uncertainties in the initial vehicle velocities $(v_1(0), v_2(0), v_3(0))$ and the initial separations d_{12}, d_{23} between the vehicles. The unsafe set U corresponds to states where there is a collision, that is, the separation between a pair of cars is less than some threshold. In case of collisions, we would be interested in the maximum possible relative velocity just before the collision, which strongly influences the severity of the accident.

The key parameters we will consider in this chapter are the reaction time or the delay r in switching to the braking mode, and the initial separation d between the cars.

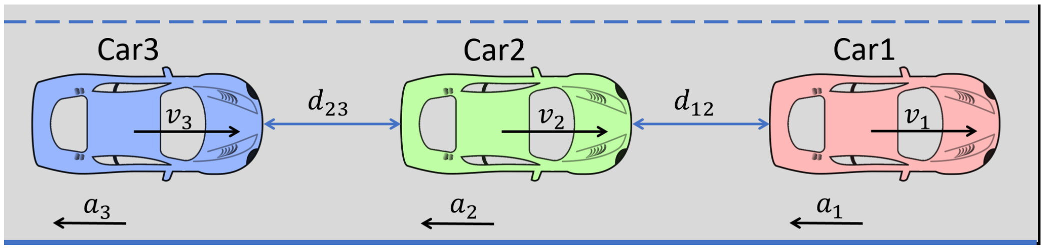
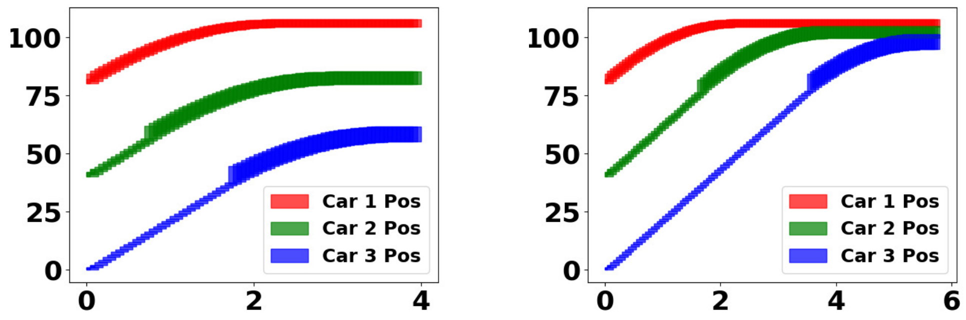


Figure 6.2: Cars cruising and braking in a single lane configuration.

6.5.2 Determining Severity of Collisions using Reachability

Fix the initial velocities and braking profiles for all the cars, fix the ranges for initial separations and reaction times; if DryVR returns safe (and the learned discrepancy is correct), then it means the distance between any two cars is always larger than the threshold value $\theta = 2$ meters at all times,



(a) Safe case: reachtubes of position are separated by a distance ≥ 2 for any time.

(b) Unsafe case: at least a pair of reachtubes of position for some time are contained in the unsafe region (≤ 2 separations).

Figure 6.3: Safety of the AEB system. Horizontal axis is time in seconds, vertical axis is position in meters.

before the cars stop. Figure 6.3a shows the projection of the reachsets plotted against time for the entire range of initial conditions; the positions of Car1 (red), Car2 (green) and Car3 (blue) are separated by at least 2 meters. If DryVR returns unsafe, then it also computes parameter values (initial separations d_{12}, d_{23}, reaction times r_2, r_3) that lead to a state where the cars have less than 2 meters separation. In Figure 6.3b, the reachsets for position overlap indicating a collision and in this case the tool also over-approximates the worst case relative velocity in the collision. For example, in the particular example the worst case collision velocity between Car1 and Car2 is 9.0 (m/s).

6.5.3 Risk Analysis for ASIL

Reachability analysis can be used for determining risk levels of an automotive control system. According to ISO 26262 ASIL classification, risk is broadly defined as

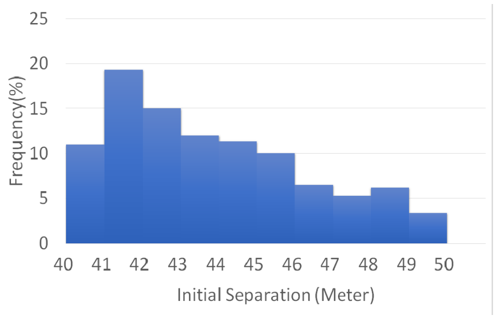
$$severity\ of\ accident \times probability\ of\ occurrence.$$

For the AEB system with 2 cars, the severity is largely determined by the relative velocity of collision, which is approximated from the above reachability analysis.

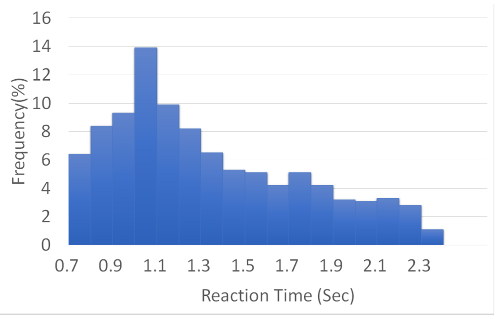
The probability of occurrence depends on the probability distributions on the parameters (d, r, etc.). In general, these distributions can be complicated. As a starting point, the preliminary study presented in [128] uses empirical observations to construct distributions on initial separation (d) which turn out to be a skewed Gaussian with the mean dependent on the car speed. Similarly, the reaction time distribution is also a skewed Gaussian. Examples of such distributions are built using [128, 129] and shown in Figures 6.4a and 6.4b, where the separation d ranges over $[40, 50]$ meters and reaction time r ranges over $[0.7, 2.4]$ seconds.

We analyze the risk by dividing $[40, 50]$ into 10 consecutive small intervals $[d_l^i, d_u^i], i = 1, \ldots, 10$, and $[0.7, 2.4]$ into 17 consecutive intervals $[r_l^i, r_u^i], j = 1, \ldots, 17$. For each region consisting of small intervals of d and r, we use DryVR to verify safety or compute the worst case collision velocity. To compute the probability of the accident occurring, we need to compute the probability that each parameter lies in the given range. For distributions shown in Figures 6.4a and 6.4b, the probability of the region $d \in [d_l^i, d_u^i], r \in [r_l^j, r_u^j]$ is $Pr(d \in [d_l^i, d_u^i]) \times Pr(r \in [r_l^j, r_u^j])$ if we assume the events $d \in [d_l^i, d_u^i])$ and $r \in [r_l^j, r_u^j]$ are independent. For example, $Pr(41 \leq d \leq 42, 1.0 \leq r \leq 1.1) = 0.19 \times 0.139 = 0.026$.

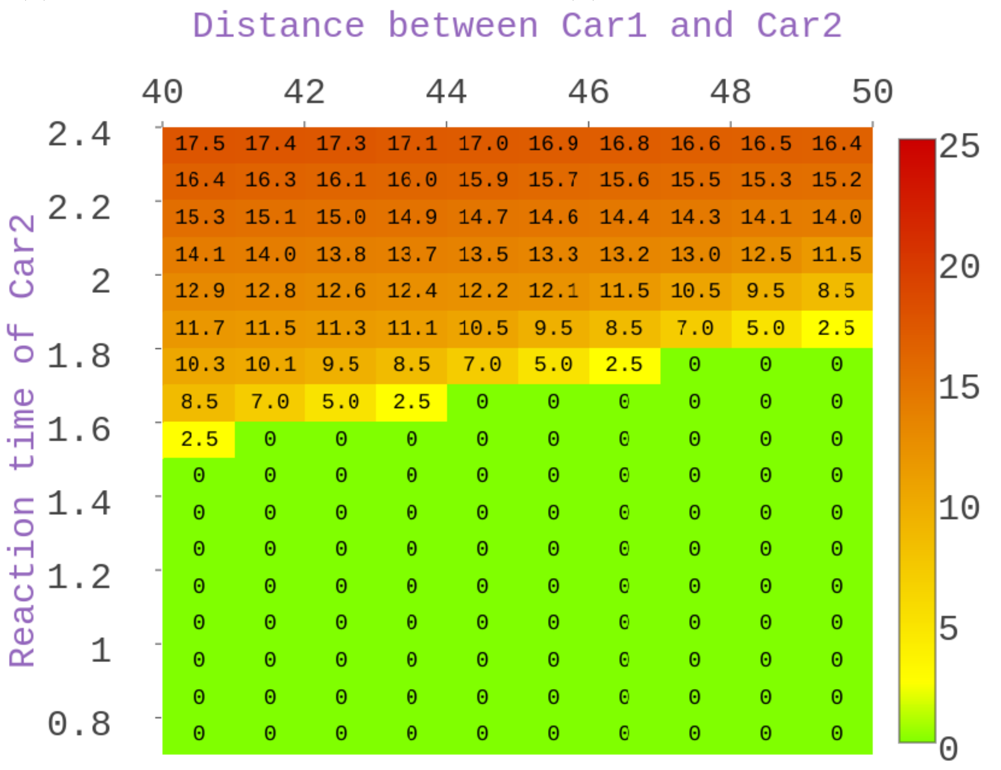
With the given braking profile and initial velocity of both cars, we can compute the worst case relative velocity, for each region of d and r. We report the results in Figure 6.4c. The numbers corresponding to each rectangle in the figure are the worst case relative velocities. For example, for the case $d \in [40, 41]$ and $r \in [2.3, 2.4]$, the worst case relative velocity v_c is 17.5 (m/s). We also plot the heat map of risks as the background of Figure 6.4c, where the green rectangles with number 0 correspond to the safe cases. Combined with the probability of occurrence, we can compute the expected velocity in the collision for Figure 6.4 to be $E[v_c] = \sum_{i=1}^{10} \sum_{j=1}^{17} Pr(d \in [d_l^i, d_u^i]) \times Pr(r \in [r_l^j, r_u^j]) \times v_c(i, j) = 2.86$ (m/s). Therefore, for AEB system with given braking profile and initial velocity for each car, and given distributions for initial separations and reaction times, we can compute the risk defined in ASIL as the expected worst case relative velocity for the collisions.



(a) Initial separation distribution.



(b) Reaction time distribution.



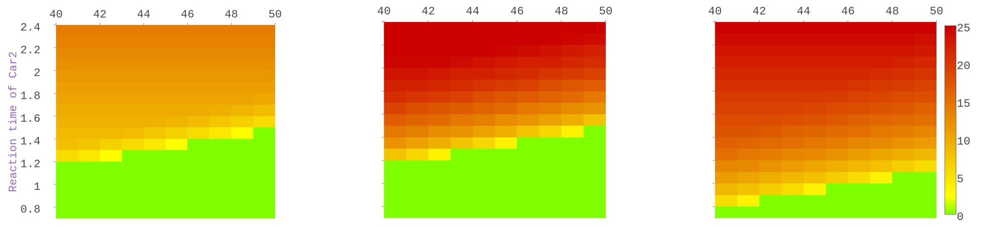
17.5	17.4	17.3	17.1	17.0	16.9	16.8	16.6	16.5	16.4
16.4	16.3	16.1	16.0	15.9	15.7	15.6	15.5	15.3	15.2
15.3	15.1	15.0	14.9	14.7	14.6	14.4	14.3	14.1	14.0
14.1	14.0	13.8	13.7	13.5	13.3	13.2	13.0	12.5	11.5
12.9	12.8	12.6	12.4	12.2	12.1	11.5	10.5	9.5	8.5
11.7	11.5	11.3	11.1	10.5	9.5	8.5	7.0	5.0	2.5
10.3	10.1	9.5	8.5	7.0	5.0	2.5	0	0	0
8.5	7.0	5.0	2.5	0	0	0	0	0	0
2.5	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

(c) Worst case relative velocities (m/s) for the collisions. Braking profiles are fixed (Car1: mild brake, Car2: medium brake) and initial velocities are 30 (m/s).

Figure 6.4: AEB of two cars: probability and severity.

6.5.4 Integrated Safety Analysis

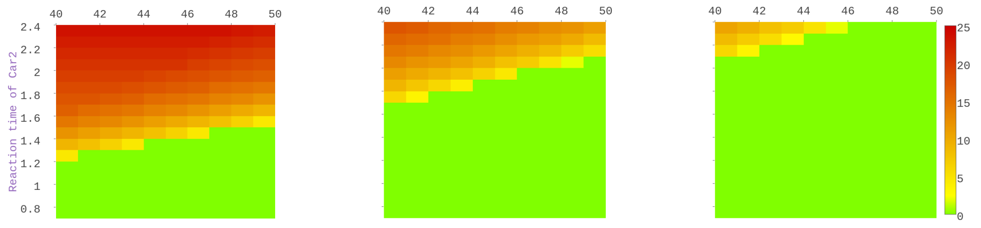
For the emergency braking system with two and three vehicles, we have analyzed hundreds of experiments; the summary of the worst-case relative collision velocities computed from these experiments is shown in Figures 6.5.



(a)
Car1&2: mild brake, initial velocity: 30 (m/s)

(b)
Car1&2: hard brake, initial velocity: 30 (m/s)

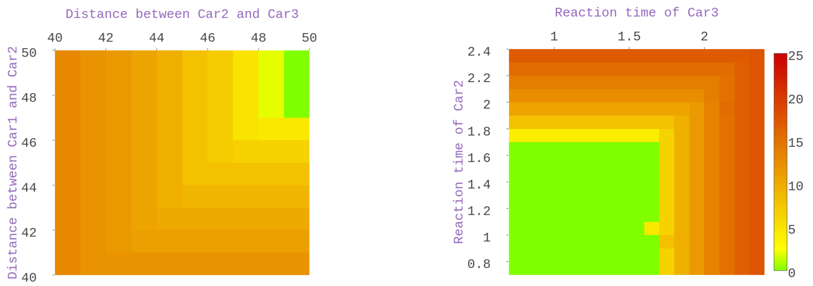
(c)
Car1: medium brake, Car2: mild brake, initial velocity: 30 (m/s)



(d)
Car1&2: medium brake, initial velocity: 30 (m/s)

(e)
Car1&2: medium brake, initial velocity: 22 (m/s)

(f)
Car1&2: medium brake, initial velocity: 18 (m/s)



(g)
Initial separation: d_{23} vs d_{12}. Fix the reaction time of both Car2 and Car3: $r_2, r_3 \in [1.8, 1.9]$ (s)

(h)
Reaction time: r_3 vs r_2. Fix the initial separation of both pairs of cars: $d_{12}, d_{23} \in [44, 45]$ (m)

Figure 6.5: Top row: two cars with different braking profiles and fixed initial velocities. Middle row left to right: two cars with decreasing velocities and fixed braking profiles. Bottom row: three cars, where each car's deceleration is medium-hard, and initial velocity is 22 (m/s). The x-axis of Figure 6.5a to 6.5f is the distance between Car1 and Car2.

We consider 3 different braking profiles for each vehicle: mild, medium and hard. The average deceleration rate increases from mild to hard. The risk analysis can also be applied to any braking profile like the threshold braking and cadence braking used in anti-lock braking systems. Figures 6.5a and 6.5d show the collision heat maps with fixed initial velocities but changing braking profiles for two cars. From Figures 6.5a, 6.5b and 6.5d, we observe that if the lead and the following cars have similar levels of braking, the safe regions are nearly invariant. However, with increasing braking level, the severity (relative velocities) of collisions also increases. Comparing with Figure 6.5d, we can see that if the lead car brakes harder than the follower, then as expected, the safety regions shrink rapidly. Moreover, the collisions are more severe than those in the previous case with both cars braking equally hard. If the lead car brakes more gently (Figure 6.4c), then the severity reduces quickly. Therefore, qualitatively, it is safer for the following car to choose a braking profile harder than or equal to the braking profile of the lead car.

Figures 6.5d-6.5f show a sequence of collision heat maps with fixed braking but changing initial velocities. As expected, both the area of the unsafe regions and severity of collisions decrease with reduction of the initial velocities. The analysis enables us to prove that, for example, the system is safe when the initial velocities of both cars are less than 17 (m/s) for the given braking profiles and reaction times.

For the system with three cars, we consider scenarios with 4 parameters: the initial separations d_{12}, d_{23} and reaction times r_2, r_3. For visualizing the risk, we fix the range of 2 parameters while varying the others. Fixing the reaction times of both Car2 and Car3 to be within the range $[1.8, 1.9]$ (s), we analyze the change of safety envelope with respect to the change of d_{12} and d_{23}. Figure 6.5g shows that the system is collision-free only when both the distances d_{12} and d_{23} are large enough. Compare Figure 6.5g with Figure 6.5e when all the cars have the same initial velocities and braking profiles. We can see that when the reaction time is between $[1.8, 1.9]$ (s), the safe distance changes from $d > 44$ (m) for the system with two cars to $d_{12} > 47$ (m), $d_{23} > 49$ (m) for system with three cars. Therefore, with the increase of number of cars in a chain, the "safe" distance between any pair of cars increases as well.

Next, fixing the distance d_{12}, d_{23} to be within the range $[44, 45]$ (m), we analyze the change of safety envelope with respect to the change of reaction time r_2, r_3. Figure 6.5h shows that the cars are collision free only if both

Car2 and Car3's reaction times are short enough. Compared with Figure 6.5e again, when the distance between the cars is between $[44, 45]$ (m), the safe reaction time changes from $r < 1.9$ (s) for the two cars scenario to $r_2 < 1.7$ (s), $r_3 < 1.6$ (s) for three cars scenario. Both Figure 6.5g and 6.5h show quantitatively that the safety envelope shrinks with the increase of number of cars in the system.

As the running time for each scenario is $3-5$ seconds on a standard laptop, it takes $5 - 30$ minutes to generate a heat map, which suggests that similar analysis could be applied to more complicated scenarios with more modes and parameters, and more sophisticated ADAS systems.

6.6 Other Examples: Powertrain, Spacecraft, and Gear Transmission

We will revisit the powertrain example (Example 5.3) and spacecraft rendezvous example (Example 5.4) introduced in Chapter 5.

6.6.1 Powertrain Control Model

As we have seen in Example 5.3, Model 2's continuous variables can be be described using a set of ODEs (Figure 5.1).The continuous variables for the physical plant $x_p = [\theta, p, \lambda]$ evolve according to ODEs, but the controller variables $x_c = [p_e, i]$ are updated periodically. Using DryVR, we can treat the entire system as a black-box simulator with the five given variables and four modes. With the initial set $p \in [0.6115, 0.6315], \lambda \in [14.6, 14.8], p_e \in [0.5555, 0.5755], i \in [0, 0.01]$, DryVR is able to prove that the system satisfies the safety requirements as stated above. Figure 6.6 shows a safe reachtube of the Air/Fuel variable λ computed using DryVR going through the sequence of modes Start_up, Normal, Power_up (also called Power), Normal.

6.6.2 Spacecraft Rendezvous

As for the spacecraft rendezvous example 5.4, a different control strategy for ARPOD was proposed in [130] which characterizes the family of individual

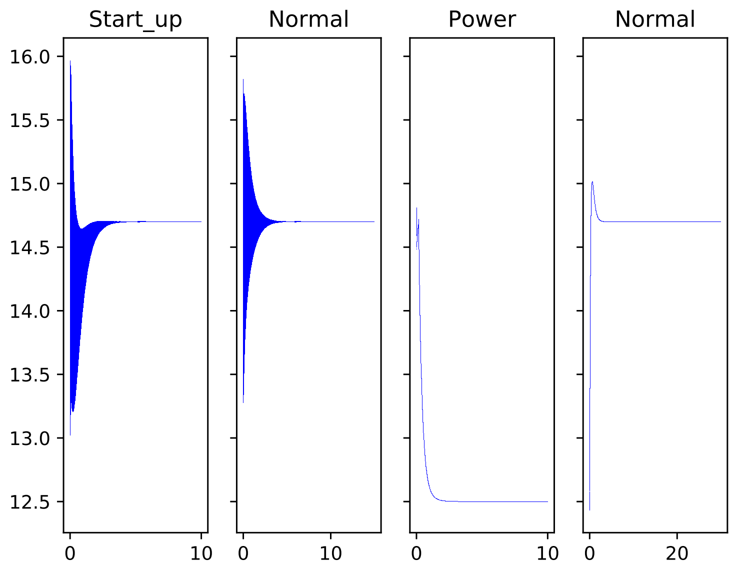


Figure 6.6: Reachtube for λ vs. time of Model 2 produced by DRYVR.

controllers and the required properties they should induce for the closed loop system to solve the problem within each phase, then uses a supervisor that robustly coordinates the individual controllers. Using these controlled subsystems as a black-box, we have been able to check the safety of the overall system using DRYVR. Figure 6.7 (right) shows the reachtube of x and y produced by DRYVR.

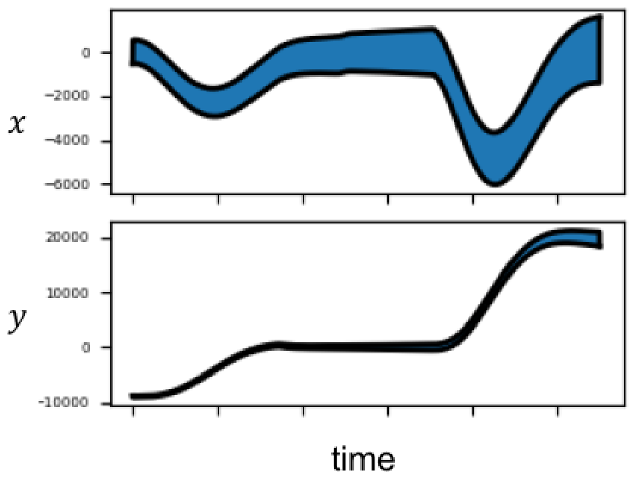


Figure 6.7: Reachtube of x vs. time (above) and y vs. time (below) produced by DRYVR.

6.6.3 Automatic Transmission Control

The automatic transmission control benchmark is a slightly modified version of the Automatic Transmission model provided by Mathworks as a Simulink demo [125]. It is a model of an automatic transmission controller that exhibits both continuous and discrete behavior. The model has been previously used by S-TaLiRo [131] for falsifying certain requirements. We are not aware of any verification results for this system.

For our experiments, we made some minor modifications to the Simulink model to create the hybrid system ATS. This allows us to simulate the vehicle from any one of the four modes, namely, gear1, gear2, gear3 and gear4. Although the system has many variables, we are primarily interested in the car Speed (v), engine RPM (Erpm), impeller torque (T_i), output torque (T_o), and transmission RPM (Trpm), and therefore we use simulations that record these. The timed transition graph of ATS encodes transition sequences and intervals for shifting from gear1 through to gear4. The requirement of interest is that the engine RPM is less than a specified maximum value, which in turn is important for limiting the thermal and mechanical stresses on the cylinders and camshafts. A typical unsafe set $\mathcal{U}_t$ could be Erpm > 4000.

Setting the initial set to be $v = 0, T_i = 394.05, T_o = 52.92$, Trpm $= 0$ and Erpm to be within $[900, 1000]$, DryVR can prove that the engine RPM Erpm is always lower than 4000 with the transition graph from gear1 through gear4 at various time intervals.

6.7 Formal Reasonings on Hybrid Systems with Black-box Modules

In this section, to simplify the analysis, we would like to separate the "white-box" and "black-box". In this way, we will be able to reason over the "white-box" only to infer properties of the overall hybrid systems (see Definition 3.2). We will focus on a special class of hybrid systems where the mode switches will only be decided by time and the transition graph is a directed acyclic graph (DAG). That is, the discrete behaviors or mode transitions are specified by what we call a timed transition graph over L.

Definition 6.3. A *timed transition graph* is a labeled, directed acyclic graph

$G = \langle \mathsf{L}, \mathsf{V}, \mathsf{E}, \mathtt{vlab}, \mathtt{elab} \rangle$, where (a) L is the set of vertex labels also called the set of *modes*, (b) V is the set of vertices, (c) $\mathsf{E} \subseteq \mathsf{V} \times \mathsf{V}$ is the set of edges, (d) $\mathtt{vlab} : \mathsf{V} \to \mathsf{L}$ is a vertex labeling function that labels each vertex with a mode, and (e) $\mathtt{elab} : \mathsf{E} \to \mathbb{R}^{\geq 0} \times \mathbb{R}^{\geq 0}$ is an edge labeling function that labels each edge with a nonempty, closed, bounded interval defined by pair of non-negative reals.

Since G is a DAG, there is a nonempty subset $\mathsf{V}_{\mathsf{init}} \subseteq \mathsf{V}$ of vertices with no incoming edges and a nonempty subset $\mathsf{V}_{\mathsf{term}} \subseteq \mathsf{V}$ of vertices with no outgoing edges. The set of initial locations can also be written as $\mathsf{L}_{\mathsf{init}} = \{\ell \mid \exists\ v \in \mathsf{V}_{\mathsf{init}}, \mathtt{vlab}(v) = \ell\}$. A (maximal) *path* of the graph G is a sequence $\pi = v_1, t_1, v_2, t_2, \ldots, v_k$ such that (a) $v_1 \in \mathsf{V}_{\mathsf{init}}$, (b) $v_k \in \mathsf{V}_{\mathsf{term}}$, and (c) for each (v_i, t_i, v_{i+1}) subsequence, there exist $(v_i, v_{i+1}) \in \mathsf{E}$ and $t_i \in \mathtt{elab}((v_i, v_{i+1}))$. $\mathtt{Path}(G)$ is the set of all possible paths of G. For a given path $\pi = v_1, t_1, v_2, t_2, \ldots,\ v_k$, its *trace*, denoted by $\mathtt{vlab}(\pi)$, is the sequence $\mathtt{vlab}(v_1), t_1, \mathtt{vlab}(v_2), t_2, \ldots,\ \mathtt{vlab}(v_k)$. Since G is a DAG, a trace of G can visit the same mode finitely many times. $\mathsf{Traces}(G)$ is the set of all traces of G.

For a given path $\pi = v_1, t_1, v_2, t_2, \ldots, v_k$, $\sum_{i=1}^{k-1} t_i$ is its time horizon and $T = \max_{\pi = v_1, t_1, \ldots, v_k \in \mathtt{Path}(G)} \sum_{i=1}^{k-1} t_i$ is the time horizon for any path of the timed transition graph G. We can see from Definition 6.3 that a transition graph actually defines the actions set A and transitions $\mathcal{D}$ in a hybrid system $\mathcal{H}$. Therefore, in this section, the hybrid system can also be written as $\mathcal{H} = \langle V = (X \cup L), \Theta, G, \mathsf{TL} \rangle$. The reachable set of hybrid system $\mathsf{Reach}_{\mathcal{H}}(\langle \Theta, \mathsf{L}_{\mathsf{init}} \rangle, [0, T])$ is the set of all states that can be reached from the initial states Θ and initial modes $\mathsf{L}_{\mathsf{init}}$, where T is the time horizon for the corresponding transition graph G. We also write the reachable set as $\mathsf{Reach}_{\mathcal{H}}$ for brevity. For any given vertex $v \in \mathsf{V}$, $\mathsf{Reach}_{\mathcal{H}} \lceil v$ is the set of states that can be reached at vertex v only.

We will develop reasoning techniques based on reachability, abstraction, composition, and substitutivity. To this end, we will need to establish containment relations between the behaviors of systems. Here we define containment of timed transition graph traces. Consider timed transition graphs G_1, G_2, with modes $\mathsf{L}_1, \mathsf{L}_2$, and a mode map $\mathtt{lmap} : \mathsf{L}_1 \to \mathsf{L}_2$. For a trace $\sigma = \ell_1, t_1, \ell_2, t_2, \ldots, \ell_k \in \mathsf{Traces}(G_1)$, simplifying notation, we denote by $\mathtt{lmap}(\sigma)$ the sequence $\mathtt{lmap}(\ell_1), t_1, \mathtt{lmap}(\ell_2), t_2,\ \ldots, \mathtt{lmap}(\ell_k)$. We write

$G_1 \preceq_{\texttt{lmap}} G_2$ iff for every trace $\sigma \in \mathsf{Traces}(G_1)$, there is a trace $\sigma' \in \mathsf{Traces}(G_2)$ such that $\texttt{lmap}(\sigma)$ is a prefix of σ'.

Definition 6.4 (Trace containment)**.** Given graphs G_1, G_2 and a mode map $\texttt{lmap} : \mathsf{L}_1 \to \mathsf{L}_2$, a relation $R \subseteq \mathsf{V}_1 \times \mathsf{V}_2$ is a *forward simulation relation from G_1 to G_2* iff

(a) for each $v \in \mathsf{V}_{1,\mathsf{init}}$, there is $u \in \mathsf{V}_{2,\mathsf{init}}$ such that $(v, u) \in R$,

(b) for every $(v, u) \in R$, $\texttt{lmap}(\texttt{vlab}_1(v)) = \texttt{vlab}_2(u)$, and

(c) for every $(v, v') \in \mathsf{E}_1$ and $(v, u) \in R$, there exists a finite set $u_1, \ldots, u_k$ such that:

  (i) for each u_j, $(v, u_j) \in R$, and

  (ii) $\texttt{elab}_1((v, v')) \subseteq \cup_j \texttt{elab}_2((u, u_j))$.

Proposition 6.2. If there exists a forward simulation relation from G_1 to G_2 with $\texttt{lmap}$, then $G_1 \preceq_{\texttt{lmap}} G_2$.

We will find it convenient to define the *sequential composition* of two timed transition graphs. Intuitively, the traces of the composition of G_1 and G_2 will be those that can be obtained by concatenating a trace of G_1 with a trace of G_2. To keep the definitions and notations simple, we will assume (when taking sequential compositions) $|\mathsf{V}_{\mathsf{init}}| = |\mathsf{V}_{\mathsf{term}}| = 1$. It is easy to generalize to the case when this does not hold. Under this assumption, the unique vertex in $\mathsf{V}_{\mathsf{init}}$ will be denoted as v_{init} and the unique vertex in $\mathsf{V}_{\mathsf{term}}$ will be denoted as v_{term}.

Definition 6.5 (Sequential composition of timed transition graphs)**.** Given graphs $G_1 = \langle \mathsf{L}, \mathsf{V}_1, \mathsf{E}_1, \texttt{vlab}_1, \texttt{elab}_1 \rangle$ and $G_2 = \langle \mathsf{L}, \mathsf{V}_2, \mathsf{E}_2, \texttt{vlab}_2, \texttt{elab}_2 \rangle$ such that $\texttt{vlab}_1(v_{1\mathsf{term}}) = \texttt{vlab}_2(v_{2\mathsf{term}})$, the *sequential composition* of G_1 and G_2 is the graph $G_1 \circ G_2 = \langle \mathsf{L}, \mathsf{V}, \mathsf{E}, \texttt{vlab}, \texttt{elab} \rangle$ where

(a) $\mathsf{V} = (\mathsf{V}_1 \cup \mathsf{V}_2) \setminus \{v_{2,\mathsf{init}})\}$,

(b) $\mathsf{E} = \mathsf{E}_1 \cup \{(v_{1,\mathsf{term}}, u) \mid (v_{2,\mathsf{init}}, u) \in \mathsf{E}_2\} \cup \{(v, u) \in \mathsf{E}_2 \mid v \neq v_{2,\mathsf{init}}\}$,

(c) $\texttt{vlab}(v) = \texttt{vlab}_1(v)$ if $v \in \mathsf{V}_1$ and $\texttt{vlab}(v) = \texttt{vlab}_2(v)$ if $v \in \mathsf{V}_2$,

(d) For edge $(v, u) \in \mathsf{E}$, $\texttt{elab}((v, u))$ equals

  (i) $\texttt{elab}_1((v, u))$, if $u \in \mathsf{V}_1$,

(ii) $\mathtt{elab}_2((v_{2,\mathsf{init}}, u))$, if $v = v_{1,\mathsf{term}}$,

(iii) $\mathtt{elab}_2((v, u))$, *otherwise.*

Given our definition of trace containment between graphs, we can prove a very simple property about sequential composition.

Proposition 6.3. Let G_1 and G_2 be two graphs with modes L that can be sequential composed. Then

$$G_1 \preceq_{\mathsf{id}} G_1 \circ G_2,$$

where id is the identity map on L.

The proposition follows from the fact that every path of G_1 is a prefix of a path of $G_1 \circ G_2$.

After defining trace containment and sequential composition on timed transition graphs, we want to define the containment of trajectories which describes the behaviors of the continuous variables.

Definition 6.6 (Trajectory containment)**.** Consider sets of trajectories TL_1 labeled by L_1 and TL_2 labeled by L_2, and a mode map $\mathtt{lmap} : \mathsf{L}_1 \to \mathcal{L}_2$. If for every labeled trajectory $\langle \xi, \ell \rangle \in \mathsf{TL}_1$, $\langle \xi, \mathtt{lmap}(\ell) \rangle \in \mathsf{TL}_2$, we say TL_1 is contained in TL_2 and denote it by $\mathsf{TL}_1 \preceq_{\mathtt{lmap}} \mathsf{TL}_2$.

6.7.1 Behavior Containment of Hybrid Systems

Containment between graphs and trajectories can be leveraged to establish the containment of the set of reachable states of two hybrid systems.

Proposition 6.4. Considering a pair of hybrid systems $\mathcal{H}_i = \langle V_i = (X_i \cup L_i), \Theta_i, G_i, \mathsf{TL}_i \rangle$, $i \in \{1, 2\}$ and mode map $\mathtt{lmap} : \mathsf{L}_1 \to \mathsf{L}_2$. If $X_1 = X_2, L_1 = L_2$, $\Theta_1 \subseteq \Theta_2$, $G_1 \preceq_{\mathtt{lmap}} G_2$, and $\mathsf{TL}_1 \preceq_{\mathtt{lmap}} \mathsf{TL}_2$, then

$$\mathsf{Reach}_{\mathcal{H}_1} \subseteq \mathsf{Reach}_{\mathcal{H}_2}.$$

For a fixed unsafe set F and two hybrid systems $\mathcal{H}_1$ and $\mathcal{H}_2$, proving $\mathsf{Reach}_{\mathcal{H}_1} \subseteq \mathsf{Reach}_{\mathcal{H}_2}$ and the safety of $\mathcal{H}_2$ allows us to establish the safety of $\mathcal{H}_1$. Proposition 6.4 establishes that proving containment of traces, trajectories, and initial sets of two hybrid systems ensures the containment of their respective reach sets. These two observations together give us a method of concluding the safety of one system, from the safety of another, provided we can check trace containment of two graphs, and trajectory containment of two trajectory sets. In most systems we need to conduct formal reasoning; the set of modes L and the set of trajectories TL are often the same in the hybrid systems we care about. So in this section we present different reasoning principles to check trace containment between two graphs.

Semantically, a timed transition graph G can be viewed as one-clock timed automaton; i.e., one can constructed a timed automaton T with one-clock variable such that the timed traces of T are exactly the traces of G. This observation, coupled with the fact that checking the timed language containment of one-clock timed automata [132] is decidable, allows one to conclude that checking if $G_1 \preceq_{\mathtt{lmap}} G_2$ is decidable. However the algorithm in [132] has non-elementary complexity. Our next observation establishes that forward simulation between graphs can be checked in polynomial time. Combined with Proposition 6.2, this gives a simple sufficient condition for trace containment that can be efficiently checked.

Proposition 6.5. Given graphs G_1 and G_2, and mode map `lmap`, checking if there is a forward simulation from G_1 to G_2 is in polynomial time.

Proof. The result can be seen to follow from the algorithm for checking timed simulations between timed automata [133] and the correspondence between one-clock timed automata; the fact that the automata have only one clock ensures that the region construction is poly-sized as opposed to exponential-sized. However, in the special case of timed transition graphs there is a more direct algorithm which does not involve region construction that we describe here.

Observe that if $\{W_i\}_{i \in I}$ is a family of forward simulations between G_1 and G_2, then $\cup_{i \in I} W_i$ is also a forward simulation. Thus, as in classical simulations, there is a unique largest forward simulation between two graphs that is the greatest fixpoint of a functional on relations over states of the transition graph. Therefore, starting from the relation $\mathsf{V}_1 \times \mathsf{V}_2$, one can progressively remove pairs (v, u) such that v is not simulated by u until a fixpoint is reached. Moreover, in this case, since G_1 is a DAG, one can guarantee that the fixpoint will be reached in $|\mathsf{V}_1|$ iterations. $\square$

Executions of hybrid systems are for bounded time, and bounded number of mode switches. This is because our timed transition graphs are acyclic and the labels on edges are bounded intervals. Sequential composition of graphs allows one to consider switching sequences that are longer and of a longer duration. We now present observations that will allow us to establish the safety of a hybrid system with long switching sequences based on the safety of the system under short switching sequences. To do this we begin by observing simple properties about sequential composition of graphs. In what follows, all hybrid systems we consider will be over a fixed set of modes L and trajectory set TL. Also id will be identity function on L. Our first observation is that trace containment is consistent with sequential composition.

Proposition 6.6. Let G_i, G_i', $i \in \{1,2\}$, be four timed transition graphs over L such that $G_1 \circ G_2$ and $G_1' \circ G_2'$ are defined, and $G_i \preceq_{\mathsf{id}} G_i'$ for $i \in \{1,2\}$. Then $G_1 \circ G_2 \preceq_{\mathsf{id}} G_1' \circ G_2'$.

Next we observe that sequential composition of graphs satisfies the "semi-group property".

Proposition 6.7. Let G_1, G_2 be graphs over L for which $G_1 \circ G_2$ is defined. Let $v_{1,\mathsf{term}}$ be the unique terminal vertex of G_1. Consider the following hybrid systems: $\mathcal{H} = \langle X \cup L, \Theta, G_1 \circ G_2, \mathsf{TL}\rangle$, $\mathcal{H}_1 = \langle X \cup L, \Theta, G_1, \mathsf{TL}\rangle$, and $\mathcal{H}_2 = \langle X \cup L, \mathsf{Reach}_{\mathcal{H}} \lceil v_{1,\mathsf{term}}, G_2, \mathsf{TL}\rangle$. Then

$$\mathsf{Reach}_{\mathcal{H}} = \mathsf{Reach}_{\mathcal{H}_1} \cup \mathsf{Reach}_{\mathcal{H}_2}.$$

Consider a graph G such that $G \circ G$ is defined. Let $\mathcal{H}$ be the hybrid system with timed transition graph G, and $\mathcal{H}'$ be the hybrid system with timed transition graph $G \circ G$; the modes, trajectories, and initial set for $\mathcal{H}$ and $\mathcal{H}'$ are the same. Now by Proposition 6.3 and 6.4, we can establish that $\mathsf{Reach}_{\mathcal{H}} \subseteq \mathsf{Reach}_{\mathcal{H}'}$. Our main result of this section is that under some conditions, the converse also holds. This is useful because it allows us to conclude the safety of $\mathcal{H}'$ from the safety of $\mathcal{H}$. In other words, we can deduce the safety of a hybrid system for long, possibly unbounded, switching sequences (namely $\mathcal{H}'$) from the safety of the system under short switching sequences (namely $\mathcal{H}$).

Theorem 6.1. Suppose G is such that $G \circ G$ is defined. Let v_{term} be the unique terminal vertex of G. For natural number $i \geq 1$, define $\mathcal{H}_i = \langle (X \cup$

$L), \Theta, G^i, \mathsf{TL}\rangle$, where G^i is the i-fold sequential composition of G with itself. In particular, $\mathcal{H}_1 = \langle (X \cup L), \Theta, G, \mathsf{TL}\rangle$. If $\mathsf{Reach}_{\mathcal{H}_1} \lceil v_{\mathsf{term}} \subseteq \Theta$, then

$$\forall i \geq 1, \mathsf{Reach}_{\mathcal{H}_i} \subseteq \mathsf{Reach}_{\mathcal{H}_1}.$$

Proof. Let $\Theta_1 = \mathsf{Reach}_{\mathcal{H}_1} \lceil v_{\mathsf{term}}$. From the condition in the theorem, we know that $\Theta_1 \subseteq \Theta$. Let us define $\mathcal{H}'_i = \langle (X \cup L), \Theta_1, G^i, \mathsf{TL}\rangle$. Observe that from Proposition 6.4, we have $\mathsf{Reach}_{\mathcal{H}'_i} \subseteq \mathsf{Reach}_{\mathcal{H}_i}$.

The theorem is proved by induction on i. The base case (for $i = 1$) trivially holds. For the induction step, assume that $\mathsf{Reach}_{\mathcal{H}_i} \subseteq \mathsf{Reach}_{\mathcal{H}_1}$. Since $\circ$ is associative, using Proposition 6.7 and the induction hypothesis, we have $\mathsf{Reach}_{\mathcal{H}_{i+1}} = \mathsf{Reach}_{\mathcal{H}_1} \cup \mathsf{Reach}_{\mathcal{H}'_i} \subseteq \mathsf{Reach}_{\mathcal{H}_1} \cup \mathsf{Reach}_{\mathcal{H}_i} = \mathsf{Reach}_{\mathcal{H}_1}$. □

Theorem 6.1 allows one to determine the set of reachable states of a set of modes L with respect to graph G^i, provided G satisfies the conditions in the statement. This observation can be generalized. If a graph G_2 satisfies conditions similar to those in Theorem 6.1, then using Proposition 6.7, we can conclude that the reachable set with respect to graph $G_1 \circ G_2^i \circ G_3$ is contained in the reachable set with respect to graph $G_1 \circ G_2 \circ G_3$.

6.8 Experiments on Behavior Containment Reasoning

6.8.1 Trace Containment

Consider an emergency brakes scenario AEB where initially vehicle A is behind B in the same lane with A in cruise, and B is stopped (in cruise mode with velocity 0). A transits from cruise to em_brake over a given interval of time or several disjoint intervals of time. We want that vehicle A stops behind B and maintains at least a given safe separation. In the actual system with timed transition graph G_2 as in Figure 6.8, two different sensor systems trigger the obstacle detection and emergency braking at time intervals $[1, 2]$ and $[2.5, 3.5]$ and take the system from vertex 0 (cruise) to two different vertices labeled with em_brake. To illustrate trace containment reasoning, consider a simpler timed transition graph G_1 as in Figure 6.9 that allows a single transition of A from cruise to em_brake over the interval bigger $[0.5, 4.5]$. Using Proposition 6.4 and checking that graph $G_2 \preceq_{\mathsf{id}} G_1$, it follows that verifying

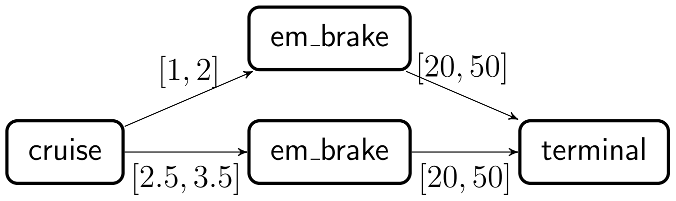


Figure 6.8: Timed transition graph G_2 of the emergency brakes scenario AEB.

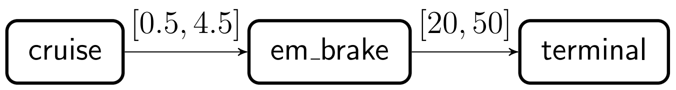


Figure 6.9: Timed transition graph G_1 of the emergency brakes scenario AEB.

the safety of AEB with G_1 is adequate to infer the safety with G_2. Figure 6.10 shows that the safe reachtubes returned by the algorithm for G_1 in red do indeed contain the reachtubes for G_2 (in blue and gray).

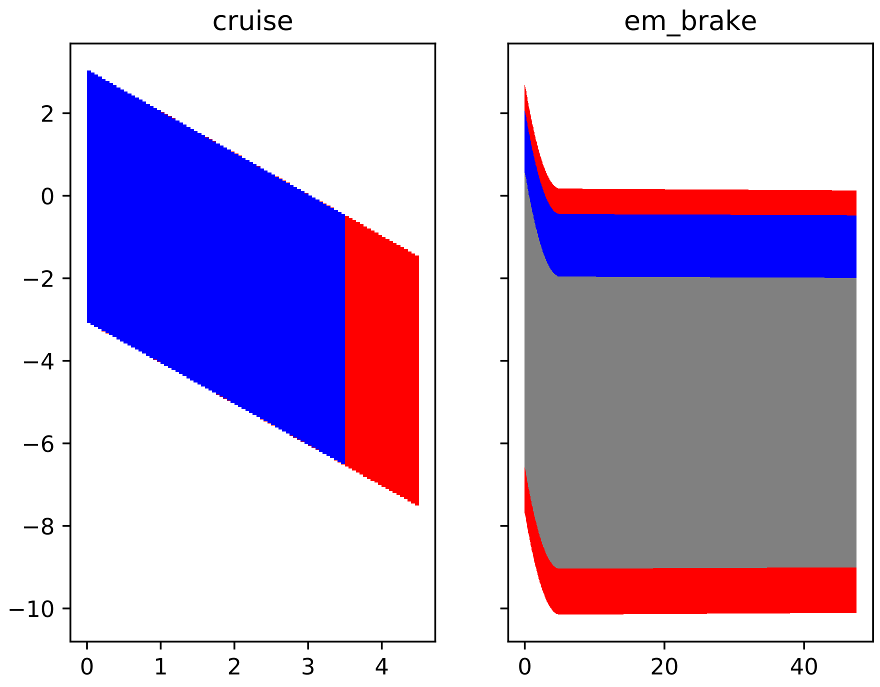


Figure 6.10: AEB reachtubes.

6.8.2 Sequential Composition

We revisit the Powertrain Model 2 of Example 5.3. The initial set Θ is the same as in Example 5.3 Model 3. Let G_A be the graph (v_0,$\mathsf{Start\_up}$) $\xrightarrow{[5,10]}$ (v_1,Normal) $\xrightarrow{[10,15]}$ (v_2,$\mathsf{Power\_up}$), and G_B be the graph (v_0,$\mathsf{Power\_up}$) $\xrightarrow{[5,10]}$ (v_1,Normal) $\xrightarrow{[10,15]}$ (v_2,$\mathsf{Power\_up}$). The graph $G_1 =$ (v_0,$\mathsf{Start\_up}$) $\xrightarrow{[5,10]}$ (v_1,Normal) $\xrightarrow{[10,15]}$ (v_2,$\mathsf{Power\_up}$) $\xrightarrow{[5,10]}$ (v_3,Normal) $\xrightarrow{[10,15]}$ (v_4,$\mathsf{Power\_up}$) can be expressed as the composition $G_1 = G_A \circ G_B$. Consider the two hybrid systems $\mathcal{H}_i = \langle (X \cup L), \Theta_i, G_i, \mathsf{TL} \rangle$, $i \in \{A, B\}$ with $\Theta_A = \Theta$ and $\Theta_B = \mathsf{Reach}_{\mathcal{H}_A} \lceil v_2$. DryVR's estimate of Θ_B had λ in the range from 14.68 to 14.71. The reachset $\mathsf{Reach}_{\mathcal{H}_B} \lceil v_2$ computed by DryVR had λ from 14.69 to 14.70. The remaining variables also were observed to satisfy the containment condition. Therefore, $\mathsf{Reach}_{\mathcal{H}_B} \lceil v_2 \subseteq \Theta_B$. Consider the two hybrid systems $\mathcal{H}_i = \langle (X \cup L), \Theta, G_i, \mathsf{TL} \rangle$, $i \in \{1, 2\}$, where G_1 is (defined above) $G_A \circ G_B$, and $G_2 = G_A \circ G_B \circ G_B \circ G_B$. Using Theorem 6.1 it suffices to analyze $\mathcal{H}_1$ to verify $\mathcal{H}_2$. Moreover, $\mathcal{H}_1$ was been proved to be safe by DryVR.

6.9 Summary

The work presented in this chapter takes an alternative view that complete mathematical models of hybrid systems are unavailable. Instead, the available system description combines a black-box simulator and a white-box transition graph. Starting from this point of view, we have developed the semantic framework, a probabilistic verification algorithm, and results on simulation relations and sequential composition for reasoning about complex hybrid systems over long switching sequences. Through modeling and analysis of a number of automotive control systems using implementations of the proposed approach, we hope to have demonstrated their promise.

Chapter 7

Approximate Partial Order Reduction

In previous chapters, we have seen how to compute discrepancy functions for nonlinear systems and black-box systems, which can be used in the data-driven verification approach. Another challenging problem that arises in distributed autonomous systems is the interleaving of concurrent actions. If we assume every possible ordering of the actions is possible, then we may have to explore exponentially many of such orderings. In this chapter, we study how to overcome this hurdle for an infinite transition system model, which is a special class of hybrid systems where we ignore the continuous evolutions of the states within each mode.

7.1 Introduction

Actions of different computing nodes may interleave arbitrarily in distributed systems. The number of action sequences that have to be examined for state-space exploration grows exponentially with the number of nodes. Partial order reduction methods (POR) [134, 135] tackle this combinatorial explosion by eliminating executions that are *equivalent*, i.e., that do not provide new information about reachable states. This equivalence is based on *independence* of actions: a pair of actions are independent if they commute, i.e., applying them in any order results in the same state. Thus, of all execution branches that start and end at the same state, but perform commuting actions in different order, only one has to be explored.

There are two main classes of POR methods. The *persistent or ample set* is a subset of representative transitions such that the omitted transitions are independent of those in the persistent set. Therefore the persistent set is guaranteed to contain all behaviors of the original system. [134]. The persistent sets are often derived by static analysis of the code. Another line of work is called dynamic POR. It relies on the *sleep set*, which is the set of omitted actions, to avoid the static analysis [136, 137, 138]. These methods examine the history of actions taken by an execution and decide a set of actions that need to be explored in the future.

Current partial order methods are limited when it comes to computation with numerical data and physical quantities (e.g., sensor networks, vehicle platoons, IoT applications, and distributed control and monitoring systems). First, a pair of actions are considered independent only if they commute exactly (see Figure 7.1 left); actions that nearly commute—as are common in these applications—cannot be exploited for pruning the exploration. Second, conventional partial order methods do not eliminate executions that start from nearly similar states and experience equivalent action sequences.

We address these limitations and propose a state space exploration method for nondeterministic, infinite state transition systems based on approximate partial order reduction. Our setup has two mild assumptions: (i) the state space of the transition system has a discrete part L and a continuous part X and the latter is equipped with a metric; (ii) the actions on X are continuous functions. Nondeterminism arises from both the choice of the initial state and the choice of actions. Fixing an initial state v_0 and a sequence of actions τ (also called a *trace*) uniquely defines an execution of the system which we denote by $\xi(v_0, \tau)$. For a given approximation parameter $\varepsilon \geq 0$, we define two actions a and b to be ε-independent if from any state v, the continuous parts of states resulting from applying action sequences ab and ba are ε-close (see Figure 7.1 right). Two *traces* of $\mathcal{A}$ are ε-equivalent if they result from permuting ε-independent actions. To compute the reachable states of $\mathcal{A}$ using a finite (small) number of executions, the key is to generalize or expand an execution $\xi(v_0, \tau)$ by a factor $r \geq 0$, so that this expanded set contains all executions that start δ-close to v_0 and experience action sequences that are ε-equivalent to τ. We call this r a (δ, ε)-*trace equivalent discrepancy factor (ted)* for ξ.

For a fixed trace τ, the only source of nondeterminism is the choice of the initial state. The reachable states from $B_\delta(v_0)$—a δ-ball around v_0—can be over-approximated by expanding $\xi(v_0, \tau)$ by a $(\delta, 0)$-*ted*. This is essentially the sensitivity of $\xi(v_0, \tau)$ to v_0. Fixing v_0, the only source of nondeterminism is the possible sequence of actions in τ. The reachable states from v_0 following all possible valid traces can be over-approximated by expanding

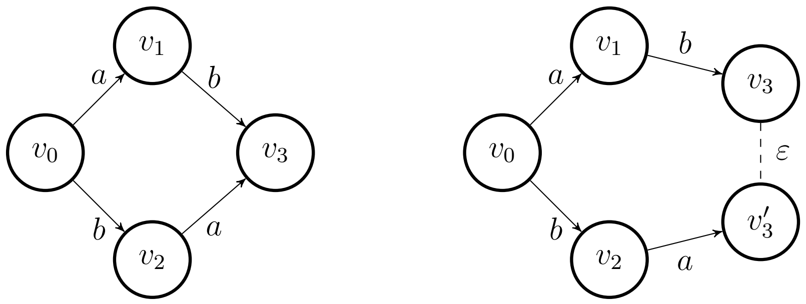

$$\begin{tikzcd} & v_1 \arrow[dr, "b"] & \\ v_0 \arrow[ur, "a"] \arrow[dr, "b"'] & & v_3 \\ & v_2 \arrow[ur, "a"'] & \end{tikzcd} \qquad \begin{tikzcd} & v_1 \arrow[r, "b"] & v_3 \arrow[d, dash, dashed, "\varepsilon"] \\ v_0 \arrow[ur, "a"] \arrow[dr, "b"'] & & v_3' \\ & v_2 \arrow[ur, "a"'] & \end{tikzcd}$$

Figure 7.1: Left: Actions a, b commute exactly. Right: Actions a, b are ε-independent.

$\xi(v_0, \tau)$ by a $(0, \varepsilon)$-*ted*, which includes states reachable by all ε-equivalent action sequences. Computing $(0, \varepsilon)$-*ted* uses the principles of partial order reduction. However, unlike exact equivalence, here, starting from the same state, the states reached at the end of executing two ε-equivalent traces are not necessarily identical. This breaks a key assumption necessary for conventional partial order algorithms: here, an action enabled after ab may not be enabled after ba. Of course, considering disabled actions can still give over-approximation of reachable states, but we show that the precision of approximation can be improved arbitrarily by shrinking δ and ε.

Thus, the reachability analysis in this part brings together two different ideas for handling nondeterminism: it combines sensitivity analysis with respect to initial state and ε-independence of actions in computing (δ, ε)-*ted*, i.e., upper-bounds on the distance between executions starting from initial states that are δ-close to each other and follow ε-equivalent action sequences. As a matter of theoretical interest, we show that the approximation error can be made arbitrarily small by choosing sufficiently small δ and ε. We validate the correctness and effectiveness of the algorithm with three case studies where conventional partial order reduction would not help: an iterative consensus protocol, a simple vehicle platoon control system, and a distributed building heating system. In most cases, our reachability algorithm reduces the number of explored executions by a factor of $O(n!)$, for a time horizon of n, compared with exhaustive enumeration. Using these over-approximations, we could quickly decide safety verification questions. This work was originally presented in [26].

7.2 Background

Recall from Section 3.1 that a deterministic labeled transition system $\mathcal{A}$ is a tuple $\langle V = (X \cup L), \Theta, A, \mathcal{D} \rangle$ (see Definition 3.1). $\mathsf{val}(V)$ is the set of states. A state $v \in \mathsf{val}(V)$ is a valuation of the real-valued and finite-valued variables. We denote by $v.X$ and $v.L$, respectively, the real-valued and discrete (finite-valued) parts of the state v. $\Theta \subseteq \mathsf{val}(V)$ is a set of initial states, A is a finite set of actions, and $\mathcal{D}$ is a the set of transitions. Traces, executions, and reachable sets of the labeled transition system are all defined in Section 3.1.

We have defined discrepancy function for dynamical systems in Section 4.4. A discrepancy function bounds the changes in a system's executions as a continuous function of the changes in its inputs. In this chapter, we extend the notion naturally to labeled transition systems: a discrepancy for an action bounds the changes in the continuous state brought about by its transition function.

Definition 7.1. For an action $a \in A$, a continuous function $\beta_a : \mathbb{R}^{\geq 0} \to \mathbb{R}^{\geq 0}$ is a *discrepancy function* if for any pair of states $v, v' \in V$ with $v.L = v'.L$,

(i) $\|a(v).X - a(v').X\| \leq \beta_a(\|v.X - v'.X\|)$, and

(ii) $\beta_a(\cdot) \to 0$ as $\|v.X - v'.X\| \to 0$.

As the following proposition states, given discrepancy functions for actions, we can reason about distance between executions that share the same trace but have different initial states.

Proposition 7.1. Suppose each action $a \in A$ has a discrepancy function β_a. For any $T \geq 0$ and action sequence $\tau = a_0 a_1 a_2 \ldots a_T$, and for any pair of states $v, v' \in V$ with $v.L = v'.L$, the last states of the pair of potential executions satisfy:

$$\xi_{v,\tau}.\mathsf{lstate}.L = \xi_{v',\tau}.\mathsf{lstate}.L, \tag{7.1}$$

$$\|\xi_{v,\tau}.\mathsf{lstate}.X - \xi_{v',\tau}.\mathsf{lstate}.X\| \leq \beta_{a_T}\beta_{a_{T-1}} \ldots \beta_{a_0}(\|v.X - v'.X\|). \tag{7.2}$$

The proposition can be proved simply by reasoning the distance between the two executions step by step.

Example 7.1. Consider an instance of Consensus of Example 3.1 with $n = 3$ and $N = 3$ with the standard 2-norm on $\mathbb{R}^3$. Let the matrices A_i be

$$A_0 = \begin{bmatrix} 0.2 & -0.2 & -0.3 \\ -0.2 & 0.2 & -0.1 \\ -0.3 & -0.1 & 0.3 \end{bmatrix}, A_1 = \begin{bmatrix} 0.2 & 0.3 & 0.2 \\ 0.3 & -0.2 & 0.3 \\ 0.2 & 0.3 & 0 \end{bmatrix}, A_2 = \begin{bmatrix} -0.1 & 0 & 0.4 \\ 0 & 0.4 & -0.2 \\ 0.4 & -0.2 & -0.1 \end{bmatrix}.$$

It can be checked that for any pair $v, v' \in V$ with $v.L = v'.L$, $\|a_i(v).X - a_i(v').X\|_2 \leq \|A_i\|_2 \|v.X - v'.X\|_2$. Where the induced 2-norms of the matrices are $\|A_0\|_2 = 0.57, \|A_1\|_2 = 0.56, \|A_2\|_2 = 0.53$. Thus, for any $v \in \mathbb{R}^{\geq 0}$, we can use discrepancy functions for a_0, a_1, a_2: $\beta_{a_0}(v) = 0.57v, \beta_{a_1}(v) = 0.56v,$ and $\beta_{a_2}(v) = 0.53v$.

For a finite set of discrepancy functions $\{\beta_a\}_{a \in A'}$ corresponding to a set of actions $A' \subseteq A$, we define $\beta_{max} = \max_{a \in A'}\{\beta_a\}$ as $\beta_{max}(v) = \max_{a \in A'} \beta_a(v)$, for each $v \geq 0$. From Definition 7.1, for each $a \in S$, $\beta_a(\|v.X - v'.X\|) \to 0$ as $\|v.X - v'.X\| \to 0$. Hence, as the maximum of β_a, we have $\beta_{max}(\|v.X - v'.X\|) \to 0$ as $\|v.X - v'.X\| \to 0$. It can be checked that β_{max} is a discrepancy function of each $a \in S$.

For $n \geq 0$ and a function β_{max} defined as above, we define a function $\gamma_n = \sum_{i=0}^{n} \beta_{max}^i$; here $\beta^i = \beta \circ \beta^{i-1}$ for $i \geq 1$ and β^0 is the identity mapping. Using the properties of discrepancy functions as in Definition 7.1, we can show the following properties of $\{\gamma_n\}_{n \in \mathbb{N}}$.

Proposition 7.2. Fix a finite set of discrepancy functions $\{\beta_a\}_{a \in A'}$ with $A' \subseteq A$. Let $\beta_{max} = \max_{a \in A'}\{\beta_a\}$. For any $n \geq 0$, $\gamma_n = \sum_{i=0}^{n} \beta_{max}^i$ satisfies (i) $\forall\ \varepsilon \in \mathbb{R}^{\geq 0}$ and any $n \geq n' \geq 0$, $\gamma_n(\varepsilon) \geq \gamma_{n'}(\varepsilon)$, and (ii) $\lim_{\varepsilon \to 0} \gamma_n(\varepsilon) = 0$.

Proof. (i) For any $n \geq 1$, we have $\gamma_n - \gamma_{n-1} = \beta_{max}^n$. Since $\beta_{max}^n = \max_{a \in S}\{\beta_a\}$ for some finite S, using Definition 7.1, β_{max}^n takes only non-negative values. Hence, the sequence of functions $\{\gamma_n\}_{n \in \mathbb{R}^{\geq 0}}$ is non-decreasing. (ii) Using the property of discrepancy functions, we have $\lim_{\varepsilon \to 0} \beta_{max}(\varepsilon) = 0$. By induction on the nested functions, we have $\lim_{\varepsilon \to 0} \beta_{max}^i(0)$ for any $i \geq 0$. Hence for any $n \in \mathbb{R}^{\geq 0}$, $\lim_{\varepsilon \to 0} \gamma_n(\varepsilon) = \lim_{\varepsilon \to 0} \sum_{i=0}^{n} \beta_{max}^i(\varepsilon) = 0$. □

The function γ_n depends on the set of $\{\beta_a\}_{a \in A'}$, but as the βs will be fixed and clear from context, we write γ_n for brevity.

7.3 Independent Actions and Neighboring Executions

Central to partial order methods is the notion of independent actions. A pair of actions are independent if from any state, the occurrence of the two actions, in either order, results in the same state. We extend this notion and define a pair of actions to be ε-independent (Definition 7.2), for some $\varepsilon > 0$, if the continuous states resulting from swapped action sequences are within ε distance.

7.3.1 Approximately Independent Actions

Definition 7.2. For $\varepsilon \geq 0$, two distinct actions $a, b \in A$ are *ε-independent*, denoted by $a \overset{\varepsilon}{\sim} b$, if for any state $v \in V$

(i) (Commutativity) $ab(v).L = ba(v).L$, and

(ii) (Closeness) $\|ab(v).X - ba(v).X\| \leq \varepsilon$.

The parameter ε captures the degree of the approximation. The smaller the value of ε, the more restrictive the independent relation. If a and b are ε-independent with $\varepsilon = 0$, then $ab(v) = ba(v)$ and the actions are independent in the standard sense (see e.g. Definition 8.3 of [139]). Definition 7.2 extends the standard definition in two ways. First, b need not be enabled at state $a(v)$, and vice versa. That is, if $\xi(v_0, ab)$ is an execution, we can only infer that $\xi(v_0, ba)$ is a potential execution and not necessarily an execution. Secondly, with $\varepsilon > 0$, the continuous states can mismatch by ε when ε-independent actions are swapped. Consequently, an action c may be enabled at $ab(v)$ but not at $ba(v)$. If $\xi(v_0, abc)$ is a valid execution, we can only infer that $\xi(v_0, bac)$ is a potential execution and not necessarily an execution.

We assume that the parameter ε does not depend on the state v. When computing the value of ε for concrete systems, we could first find an invariant for the state's real-valued variable $v.X$ such that $v.X$ is bounded, then find an upper-bound of $\|ab(v).X - ba(v).X\|$ as ε. For example, if a and b are both linear mappings with $a(v).X = A_1 v.X + b_1$ and $b(v).X = A_2 v.X + b_2$ and there is an invariant for $v.X$ is such that $\|v.X\| \leq r$, then it can be checked that $\|ab(v).X - ba(v).X\| = \|(A_2A_1 - A_1A_2)v.X + (A_2b_1 - A_1b_2 + b_2 - b_1)\| \leq \|A_2A_1 - A_1A_2\|r + \|A_2b_1 - A_1b_2 + b_2 - b_1\|$.

For a trace $\tau \in A^*$ and an action $a \in A$, τ is ε-independent to a, written as $\tau \overset{\varepsilon}{\sim} a$, if τ is empty string or for every $i \in \{0, \cdots, len(\tau) - 1\}$, $\tau(i) \overset{\varepsilon}{\sim} a$. It is clear that the approximate independence relation over A is symmetric, but not necessarily transitive.

Example 7.2. Consider approximate independence of actions in $\mathsf{Consensus}$ (Example 3.1). Fix any $i, j \in \{0, \cdots, N\}$ such that $i \neq j$ and any state $v \in V$. It can be checked that $a_i a_j(v).d^{(k)} = a_j a_i(v).d^{(k)} = \mathsf{true}$ *if* $k \in \{i, j\}$, otherwise it is $v.d^{(k)}$. Hence, we have $a_i a_j(v).d = a_j a_i(v).d$ and the commutativity condition of Definition 7.2 holds. For the closeness condition, we have $\|a_i a_j(v).x - a_j a_i(v).x\|_2 = \|(A_i A_j - A_j A_i)v.x\|_2 \leq \|A_i A_j - A_j A_i\|_2 \|v.x\|_2$. If the matrices A_i and A_j commute, then a_i and a_j are ε-approximately independent with $\varepsilon = 0$.

Suppose initially $x \in [-4, 4]^3$ then the 2-norm of the initial state is bounded by the value $4\sqrt{3}$. The specific matrices $A_i, i \in \{0, 1, 2\}$ presented in Example 7.1 are all stable, so $\|a_i(v).x\|_2 \leq \|v.x\|_2$, for each $i \in \{0, 1, 2\}$ and the norm of state is non-increasing in any transitions. Therefore, $\mathsf{I} = \{x \in \mathbb{R}^3 : \|x\|_2 \leq 4\sqrt{3}\}$ is an invariant of the system. Together, we have $\|a_0 a_1(v).x - a_1 a_0(v).x\|_2 \leq 0.1$, $\|a_0 a_2(v).x - a_2 a_0(v).x\|_2 \leq 0.07$, and $\|a_1 a_2(v).x - a_2 a_1(v).x\|_2 \leq 0.17$. Thus, with $\varepsilon = 0.1$, it follows that $a_0 \overset{\varepsilon}{\sim} a_1$ and $a_0 \overset{\varepsilon}{\sim} a_2$ and $\overset{\varepsilon}{\sim}$ is not transitive, but with $\varepsilon = 0.2$, $\overset{\varepsilon}{\sim}$ is transitive.

7.3.2 (δ, ε)-Trace Equivalent Discrepancy for Action Pairs

Definition 7.2 implies that from a single state v, executing two ε-independent actions in either order, we end up in states that are within ε distance. The following proposition uses discrepancy to bound the distance between states reached after performing ε-independent actions starting from *different* initial states v and v'.

Proposition 7.3. If a pair of actions $a, b \in A$ are ε-independent, and the two states $v, v' \in V$ satisfy $v.L = v'.L$, then we have

(i) $ba(v).L = ab(v').L$, and

(ii) $\|ba(v).X - ab(v').X\| \leq \beta_b \circ \beta_a(\|v.X - v'.X\|) + \varepsilon$, where β_a, β_b are discrepancy functions of a, b respectively.

Proof. Fix a pair of states $v, v' \in V$ with $v.L = v'.L$. Since $a \overset{\varepsilon}{\sim} b$, we have $ba(v).L = ab(v).L$. Using the Assumption, we have $ab(v).L = ab(v').L$. Using triangular inequality, we have $\|ba(v).X - ab(v').X\| \leq \|ba(v).X - ba(v').X\| + \|ba(v').X - ab(v').X\|$. The first term is bounded by $\beta_b \circ \beta_a(\|v.X - v'.X\|)$ using Proposition 7.1 and the second is bounded by ε by Definition 7.2, and hence, the result follows. $\square$

7.4 Effect of ε-Independent Traces

In this section, we will develop an analog of Proposition 7.3 for ε-independent traces (action sequences) acting on neighboring states.

7.4.1 ε-Equivalent Traces

First, we define what it means for two finite traces in A^* to be ε-equivalent.

Definition 7.3. For any $\varepsilon \geq 0$, we define a relation $\mathsf{R} \subseteq A^* \times A^*$ such that $\tau \mathsf{R} \tau'$ iff there exists $\sigma, \eta \in A^*$ and $a, b \in A$ such that $a \overset{\varepsilon}{\sim} b$, $\tau = \sigma ab\eta$, and $\tau' = \sigma ba\eta$. We define an equivalence relation $\overset{\varepsilon}{\equiv} \subseteq A^* \times A^*$ called *ε-equivalence*, as the reflexive and transitive closure of R.

That is, two traces $\tau, \tau' \in A^*$ are ε-equivalent if we can construct τ' from τ by performing a sequence of swaps of consecutive ε-independent actions.

The following proposition states that the last states of two potential executions starting from the same initial discrete state (location) and resulting from equivalent traces have identical locations.

Proposition 7.4. Fix potential executions $\xi = \xi(v_0, \tau)$ and $\xi' = \xi(v_0', \tau')$. If $v_0.L = v_0'.L$ and $\tau \overset{\varepsilon}{\equiv} \tau'$, then $\xi.\mathsf{lstate}.L = \xi'.\mathsf{lstate}.L$.

Proof. If $\tau = \tau'$, then the proposition follows from Assumption 3.1. Supposing $\tau \neq \tau'$, from Definition 7.3, there exists a sequence of action sequences $\tau_0, \tau_1, \ldots, \tau_k$ to join τ and τ' by swapping neighboring approximately independent actions. Precisely the sequence $\{\tau_i\}_{i=0}^k$ satisfies:

(i) $\tau_0 = \tau$ and $\tau_k = \tau'$, and

(ii) for each pair τ_i and τ_{i+1}, there exists $\sigma, \eta \in A^*$ and $a, b \in A$ such that $a \overset{\varepsilon}{\sim} b$, $\tau_i = \sigma ab\eta$, and $\tau_{i+1} = \sigma ba\eta$.

From Definition 7.2, swapping approximately independent actions preserves the value of the discrete part of the final state. Hence for any $i \in \{0, \cdots, k-1\}$, $\xi(v_0, \tau_i).\mathsf{lstate}.L = \xi(v_0, \tau_{i+1}).\mathsf{lstate}.L$. Therefore, $\xi.\mathsf{lstate}.L = \xi'.\mathsf{lstate}.L$. □

Next, we relate pairs of potential executions that result from ε-equivalent traces and initial states that are δ-close.

Definition 7.4. Given $\delta, \varepsilon \geq 0$, a pair of initial states v_0, v_0', and a pair traces $\tau, \tau' \in A^*$, the corresponding potential executions $\xi = \xi(v_0, \tau)$ and $\xi' = \xi(v_0', \tau')$ are *(δ, ε)-related*, denoted by $\xi \overset{\delta,\varepsilon}{\approx} \xi'$, if

(i) $v_0.L = v_0'.L$,

(ii) $\|v_0.X - v_0'.X\| \leq \delta$, and

(iii) $\tau \overset{\varepsilon}{\equiv} \tau'$.

Example 7.3. In Example 7.2, we show that $a_0 \overset{\varepsilon}{\sim} a_1$ and $a_0 \overset{\varepsilon}{\sim} a_2$ with $\varepsilon = 0.1$. Consider the executions $\xi = v_0, a_0, v_1, a_1, v_2, a_2, v_3, a_\perp, v_4$ and $\xi' = v_0', a_1, v_1', a_2, v_2', a_0, v_3', a_\perp, v_4'$. with traces $\texttt{trace}(\xi) = a_0 a_1 a_2 a_\perp$ and $\texttt{trace}(\xi') = a_1 a_2 a_0 a_\perp$. For $\varepsilon = 0.1$, we have $a_0 a_1 a_2 a_\perp \overset{\varepsilon}{\equiv} a_1 a_0 a_2 a_\perp$ and $a_1 a_0 a_2 a_\perp \overset{\varepsilon}{\equiv} a_1 a_2 a_0 a_\perp$. Since the equivalence relation $\overset{\varepsilon}{\equiv}$ is transitive, we have $\texttt{trace}(\xi) \overset{\varepsilon}{\equiv} \texttt{trace}(\xi')$. Suppose $v_0 \in B_\delta(v_0')$, then ξ and ξ' are (δ, ε)-related executions with $\varepsilon = 0.1$.

It follows from Proposition 7.4 that the discrete state (locations) reached by any pair of (δ, ε)-related potential executions are the same. In Lemma 7.3 of next section, we will bound the distance between the continuous state reached by (δ, ε)-related potential executions. We define in the following this bound as what we call *trace equivalent discrepancy factor (ted)*, which is a constant number that works for all possible values of the variables starting from the initial set. Looking ahead, by bloating a single potential execution by the corresponding *ted*, we can over-approximate the reachset of all related potential executions. This will be the basis for the reachability analysis in Section 7.5.

Definition 7.5. For any potential execution ξ and constants $\delta, \varepsilon \geq 0$, a (δ, ε)-*trace equivalent discrepancy factor (ted)* is a nonnegative constant $r \geq 0$, such that for any (δ, ε)-related potential finite execution ξ',

$$\|\xi'.\mathsf{lstate}.X - \xi.\mathsf{lstate}.X\| \leq r.$$

That is, if r is a (δ, ε)-*ted*, then the r-neighborhood of ξ's last state $B_r(\xi.\mathsf{lstate})$ contains the last states of all other (δ, ε)-related potential executions.

7.4.2 $(0, \varepsilon)$-Trace Equivalent Discrepancy for Traces (on the Same Initial States)

In this section, we will develop an inductive method for computing (δ, ε)-*ted*. We begin by bounding the distance between potential executions that differ only in the position of a single action.

Lemma 7.1. Consider any $\varepsilon \geq 0$, an initial state $v_0 \in V$, an action $a \in A$ and a trace $\tau \in A^*$ with $len(\tau) \geq 1$. If $\tau \overset{\varepsilon}{\sim} a$, then the potential executions $\xi = \xi(v_0, \tau a)$ and $\xi' = \xi(v_0, a\tau)$ satisfy

(i) $\xi'.\mathsf{lstate}.L = \xi.\mathsf{lstate}.L$ and

(ii) $\|\xi'.\mathsf{lstate}.X - \xi.\mathsf{lstate}.X\| \leq \gamma_{n-1}(\varepsilon)$, where γ_n corresponds to the set of discrepancy functions $\{\beta_c\}_{c \in \tau}$ for the actions in τ.

Proof. Part (i) directly follows from Proposition 7.4. We will prove part (ii) by induction on the length of τ.

Base: For any trace τ of length 1, ξ and ξ' are of the form $\xi = v_0, b_0, v_1, a, v_2$ and $\xi' = v_0, a, v_1', b_0, v_2'$. Since $a \overset{\varepsilon}{\sim} b_0$ and the two executions start from the same state, it follows from Definition 7.2 that $\|v_2'.X - v_2.X\| \leq \varepsilon$.

Recall from the preliminary that $\gamma_0(\varepsilon) = \beta^0(\varepsilon) = \varepsilon$. Hence $\|v_2'.X - v_2.X\| \leq \gamma_0(\varepsilon)$ holds for trace τ with $len(\tau) = 1$.

Induction: Suppose the lemma holds for any τ with length at most $n-1$. Fixing any $\tau = b_0 b_1 \dots b_{n-1}$ of length n, we will show the lemma holds for τ.

Let the potential executions $\xi = \xi(v_0, \tau a)$ and $\xi' = \xi(v_0, a\tau)$ have the form

$$\xi = v_0, b_0, v_1, b_1, ..., b_{n-1}, v_n, a, v_{n+1},$$
$$\xi' = v_0, a, v_1', b_0, v_2', b_1, ..., b_{n-1}, v_{n+1}'.$$

$$\begin{tikzcd}
 & v_1 \arrow[r, "b_1"] \arrow[dr, "a"] & v_2 \arrow[r, "b_2"] & v_3 \arrow[r, "b_3"] & \cdots \arrow[r, "b_{n-1}"] & v_n \arrow[r, "a"] & v_{n+1} & \xi \\
v_0 \arrow[ur, "b_0"] \arrow[dr, "a"'] & & v_2'' \arrow[r, "b_1"] & v_3'' \arrow[r, "b_2"] & \cdots \arrow[r, "b_{n-2}"] & v_n'' \arrow[r, "b_{n-1}"] & v_{n+1}'' & \xi'' \\
 & v_1' \arrow[r, "b_0"] & v_2' \arrow[r, "b_1"] & v_3' \arrow[r, "b_2"] & \cdots \arrow[r, "b_{n-2}"] & v_n' \arrow[r, "b_{n-1}"] & v_{n+1}' & \xi'
\end{tikzcd}$$

Figure 7.2: Potential executions ξ, ξ', and ξ''.

It suffices to prove that $\|\xi.\mathsf{lstate}.X - \xi'.\mathsf{lstate}.X\| = \|v_{n+1}.X - v'_{n+1}.X\| \leq \gamma_{n-1}(\varepsilon)$. We first construct a potential execution $\xi'' = \xi(v_0, b_0 a b_1 \ldots b_{n-1})$ by swapping the first two actions of ξ'. Then, ξ'' is of the form $\xi'' = v_0, b_0, v_1, a, v''_2, b_1, ..., b_{n-1}, v''_{n+1}$. The potential executions ξ, ξ' and ξ'' are shown in Figure 7.2. We first compare the potential executions ξ and ξ''. Notice that ξ and ξ'' share a common prefix v_0, b_0, v_1. Starting from v_1, the action sequence of ξ'' is derived from $\texttt{trace}(\xi)$ by inserting action a in front of the action sequence $\tau' = b_1 b_2 \ldots b_{n-1}$.

Since $\tau' \overset{\varepsilon}{\sim} a$, applying the induction hypothesis on the length $n-1$ action sequence τ', we get

$$\|v_{n+1}.X - v''_{n+1}.X\| \leq \gamma_{n-2}(\varepsilon).$$

Then, we compare the potential executions ξ' and ξ''. Since $b_0 \overset{\varepsilon}{\sim} a$, by applying the property of Definition 7.2 to the first two actions of ξ' and ξ'', we have $\|v'_2.X - v''_2.X\| \leq \varepsilon$. We note that ξ' and ξ'' have the same suffix of action sequence from v'_2 and v''_2. Using Proposition 7.1 from states v'_2 and v''_2, we have

$$\|v'_{n+1}.X - v''_{n+1}.X\| \leq \beta_{b_1}\beta_{b_2}\ldots\beta_{b_{n-1}}(\|v'_2.X - v''_2.X\|) \leq \beta^{n-1}(\varepsilon). \tag{7.3}$$

Combining the bound on $\|v'_2.X - v''_2.X\|$ and (7.3) with triangular inequality, we have

$$\begin{aligned}
\|v_{n+1}.X - v'_{n+1}.X\| &\leq \|v_{n+1}.X - v''_{n+1}.X\| + \|v'_{n+1}.X - v''_{n+1}.X\| \\
&\leq \gamma_{n-2}(\varepsilon) + \beta^{n-1}(\varepsilon) = \gamma_{n-1}(\varepsilon).
\end{aligned}$$

□

7.5 Reachability with Approximate Partial Order Reduction

We will present our main algorithm (Algorithm 6) for reachability analysis with approximate partial order reduction in this section. The core idea is to over-approximate $\mathsf{Reach}(B_\delta(v_0), T)$ by (a) computing the actual execution $\xi(v_0, \tau)$ and (b) expanding this $\xi(v_0, \tau)$ by a (δ, ε)-*ted* to cover all the states reachable from any other (δ_0, ε)-related potential execution. Combining such over-approximations from a cover of Θ, we get over-approximations of $\mathsf{Reach}(\Theta, T)$, and therefore, Algorithm 6 can be used to soundly check for bounded safety or invariance as a subroutine of the data-driven verification approach (Algorithm 1) introduced in Chapter 4. The over-approximations can be made arbitrarily precise by shrinking δ_0 and ε (Theorem 7.2). Of course, at $\varepsilon = 0$ only traces that are exactly equivalent to τ will be covered, and nothing else. Algorithm 6 avoids computing (δ_0, ε)-related executions, and therefore, gains (possibly exponential) speedup.

The key subroutine in Algorithm 6 is `CompTed` which computes the *ted* by adding one more action to the traces. It turns out that the *ted* is independent of v_0, but only depends on the sequence of actions in τ. `CompTed` is used to compute δ_t from δ_{t-1}, such that δ_t is the *ted* for the length t prefix of ξ. Let action a be the t^{th} action and $\xi = \xi(v_0, \tau a)$. The action a may not be ε-independent to the whole sequence τ, but we would still want to compute a set of executions that $\xi(v_0, \tau a)$ can cover. We observe that, with appropriate computation of *ted*, $\xi(v_0, \tau a)$ can cover all executions of the form $\xi(v_0, \phi a \eta)$, where $\phi a \eta$ is ε-equivalent to τa and $a \notin \eta$. In what follows, we introduce this notion of *earliest equivalent position of a in τ* (Definition 7.6), which is the basis for the `CompTed` subroutine and is used in the main reachability Algorithm 6.

7.5.1 Earliest Equivalent Position of an Action in a Trace

For any trace $\tau \in A^*$ and action $a \in \tau$, we define $\texttt{LastPos}(\tau, a)$ as the largest index k such that $\tau(k) = a$. The earliest equivalent position (eep), $\texttt{eep}(\tau, a, \varepsilon)$ is the minimum of $\texttt{LastPos}(\tau', a)$ in any τ' that is ε-equivalent to τa.

Definition 7.6. For any trace $\tau \in A^*$, $a \in A$, and $\varepsilon > 0$, the *earliest equivalent position* of a on τ is

$$\texttt{eep}(\tau, a, \varepsilon) \triangleq \min_{\tau' \overset{\varepsilon}{\equiv} \tau a} \texttt{LastPos}(\tau', a).$$

For any trace τa, its ε-equivalent traces can be derived by swapping consecutive ε-independent action pairs. Hence, the earliest equivalent position of a is the leftmost position it can be swapped to, starting from the end. Any equivalent trace of τa is of the form $\phi a \eta$ where ϕ and η are the prefix and suffix of the last occurrence of action a. Hence, equivalently:

$$\texttt{eep}(\tau, a, \epsilon) = \min_{\phi a \eta \overset{\varepsilon}{\equiv} \tau a,\ a \notin \eta} len(\phi).$$

The following algorithm finds the earliest equivalent point (eep) of action a on an action sequence τ. For any trace τ and action a, $\texttt{eep}(\tau, a, \varepsilon)$ constructs a trace $\phi \in A^*$. Initially ϕ is set to be the empty sequence. Iteratively, from the end of τ, we add action $\tau(t)$ to ϕ if it is not independent to the entire trace ϕa. The length of ϕ gives the $\texttt{eep}$ of action a on trace τ. The time complexity of the algorithm is at most $O((len(\tau))^2)$. If the ε-independence relation is symmetric, then its $\texttt{eep}$ can be computed in $O(len(\tau))$ time.

Algorithm 4: $\texttt{eep}(\tau, a, \varepsilon)$

```
1 $\phi \leftarrow \langle\rangle$ ;
2 $T \leftarrow len(\tau)$;
3 for $t = T - 1 : 0$ do
4     if $\exists b \in \phi a, \tau(t) \overset{\varepsilon}{\not\sim} b$ then
5         $\phi \leftarrow \tau(t)\phi$
6 end
7 return $len(\phi)$;
```

Lemma 7.2 (Lemma 4.7 of [33])**.** For any action $a \in A$ and trace $\tau \in A^*$, the function $\texttt{eep}(\tau, a, \varepsilon)$ computes the $\texttt{eep}$ of a on τ.

Example 7.4. In Example 7.2, we showed that $a_0 \overset{\varepsilon}{\sim} a_1$ and $a_0 \overset{\varepsilon}{\sim} a_2$ with $\varepsilon = 0.1$; $a_\perp$ is not ε-independent to any actions. What is $\texttt{eep}(a_\perp a_0 a_1, a_2, \varepsilon)$? We can swap a_2 ahead following the sequence $\tau a_2 = a_\perp a_0 a_1 a_2 \overset{\varepsilon}{\equiv} a_\perp a_1 a_0 a_2 \overset{\varepsilon}{\equiv} a_\perp a_1 a_2 a_0$. As $a_\perp$ and a_1 are not independent of a_2, it cannot occur earlier. $\texttt{eep}(a_\perp a_0 a_1, a_2, \varepsilon) = 2$.

7.5.2 Reachability using (δ, ε)-Trace Equivalent Discrepancy

CompTed (Algorithm 5) takes inputs of trace τ, a new action to be added a, a parameter $r \geq 0$ such that r is a (δ_0, ε)-*ted* for the potential execution $\xi(v_0, \tau)$ for some initial state v_0, initial set radius δ_0, approximation parameter $\varepsilon \geq 0$, and a set of discrepancy functions $\{\beta_a\}_{a \in A}$. It uses the earliest equivalent position of the action a in τ and returns a (δ_0, ε)-*ted* r' for the potential execution $\xi(v_0, \tau a)$.

Algorithm 5: CompTed($\tau, a, r, \varepsilon, \{\beta_a\}_{a \in A}$)

```
1 $\beta \leftarrow \max_{b \in \tau a}\{\beta_b\}$;
2 $k \leftarrow \mathtt{eep}(\tau, a, \epsilon)$;
3 $t \leftarrow len(\tau)$;
4 if $k = t$ then
5     $r' \leftarrow \beta_a(r)$
6 else
7     $r' \leftarrow \beta_a(r) + \gamma_{t-k-1}(\varepsilon)$ ;
8 end
9 return $r'$;
```

We establish the correctness of Algorithm 5 by showing that r' is indeed an *ted* for the potential execution $\xi(v_0, \tau a)$ and δ_0, ε.

Lemma 7.3. For some initial state v_0 and initial set size δ_0, if r is a (δ_0, ε)-*ted* for $\xi(v_0, \tau)$ then value returned by CompTed() is a (δ_0, ε)-*ted* for $\xi(v_0, \tau a)$.

Proof. Let us fix some initial state v_0 and initial set size δ_0.

Let $\xi_t = \xi(v_0, \tau)$ be the potential execution starting from v_0 by taking the trace τ, and $\xi_{t+1} = \xi(v_0, \tau a)$. Fix any ξ' that is (δ_0, ε)-related to ξ_{t+1}. From Proposition 7.4, $\xi'.\mathsf{lstate}.L = \xi_{t+1}.\mathsf{lstate}.L$. We only need to prove that $\|\xi'.\mathsf{lstate}.X - \xi_{t+1}.\mathsf{lstate}.X\| \leq r'$.

Since $\mathtt{trace}(\xi') \overset{\varepsilon}{\equiv} \tau a$, action a is in the sequence $\mathtt{trace}(\xi')$. Partitioning $\mathtt{trace}(\xi')$ on the last occurrence of a, we get $\mathtt{trace}(\xi') = \phi a \eta$ for some $\phi, \eta \in A^*$ with $a \notin \eta$. Since k is the eep, from Definition 7.6, the position of the last occurrence of a on $\mathtt{trace}(\xi')$ is at least k. Hence we have $len(\phi) \geq k$ and $len(\eta) = t - len(\phi) \leq t - k$. We construct another potential execution $\xi'' = \xi(v_0', \phi\eta a)$ with the same initial state as ξ'. The executions ξ_{t+1}, ξ' and ξ'' are illustrated in Figure 7.3.

The term v_t is the last state of the execution ξ_t. From the condition of the lemma, $B_r(v_t)$ contains the last states of other length t, (δ_0, ϵ)-related

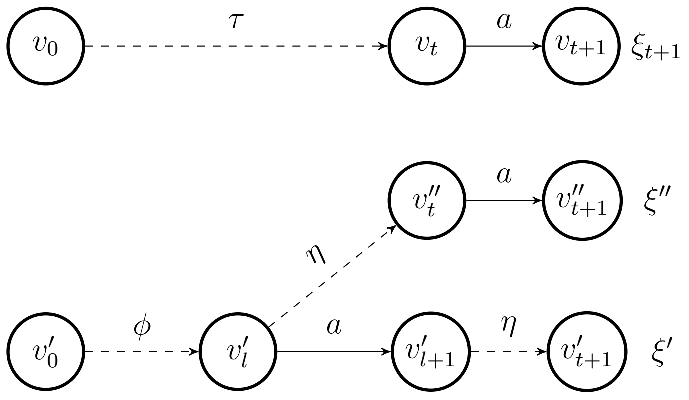


Figure 7.3: Execution ξ, its ε-equivalent execution ξ', and execution ξ'' that is constructed by swapping action a to the back in ξ'.

potential executions. We note that the length t prefix ξ'' is (δ_0, ε)-related to ξ_t. Therefore, we have $\|v_t.X - v''_t.X\| \leq r$. Using the discrepancy function of action a, we have

$$\|v_{t+1}.X - v''_{t+1}.X\| \leq \beta_a(\|v_t.X - v''_t.X\|) \leq \beta_a(r). \tag{7.4}$$

Next, we will quantify the distance between ξ' and ξ''. There are two cases:

(i) If $k = t$ then, $len(\eta) \leq t - k = 0$, that is, η is an empty string. Hence, ξ' and ξ'' are indeed identical and $v'_{t+1} = v''_{t+1}$. Thus from (7.4),

$$\|v_{t+1}.X - v'_{t+1}.X\| = \|v_{t+1}.X - v''_{t+1}.X\| \leq \beta_a(r),$$

and the lemma holds.

(ii) Otherwise $k < t$ and from Lemma 7.1, we can bound the distance between ξ' and ξ'' as

$$\|v'_{t+1}.X - v''_{t+1}.X\| \leq \gamma_{len(\eta)-1}(\varepsilon) \leq \gamma_{t-k-1}(\varepsilon).$$

Combining with (7.4) and using triangular inequality, we get

$$\begin{aligned}\|v_{t+1}.X - v'_{t+1}.X\| &\leq \|v_{t+1}.X - v''_{t+1}.X\| + \|v'_{t+1}.X - v''_{t+1}.X\| \\ &\leq \beta_a(r) + \gamma_{t-k-1}(\varepsilon).\end{aligned}$$

□

We also notice that if a is ε-independent of τ, that is, if a is ε-independent of any action in τ, then the `eep` of a in τ is $\mathtt{eep}(\tau, a, \epsilon) = 0$, so the *ted* r' in Algorithm 5 can be computed from r using the following corollary.

Corollary 7.1. For any potential execution $\xi = \xi(v_0, \tau)$ and constants $\delta, \varepsilon \geq 0$, if r is a (δ, ε)-*ted* for ξ, and the action $a \in A$ satisfies $\tau \overset{\varepsilon}{\sim} a$, then $r' = \beta_a(r) + \gamma_{len(\tau)-1}(\varepsilon)$ is a (δ, ε)-*ted* for $\xi(v_0, \tau a)$.

Next, we present the main reachability algorithm which uses `CompTed`. Algorithm 6 takes inputs of an initial set Θ, time horizon T, two parameters $\delta_0, \varepsilon \geq 0$, and a set of discrepancy functions $\{\beta_a\}_{a \in A}$. It returns the over-approximation of the reach set for each time step.

The algorithm first computes a δ_0-`cover` V_0 of the initial set Θ such that the union of the δ_0 balls around all initial states $v_0 \in V_0$ covers the initial set Θ: $\Theta \subseteq \cup_{v_0 \in V_0} B_\delta(v_0)$ (Line 6). The **for**-loop from Line 1 to Line 16 will compute the over-approximation of the reachset from each initial cover $\mathsf{Reach}(B_{\delta_0}(v_0), t)$. The over-approximation from each cover is represented as a collection $\langle D_0, \ldots, D_T \rangle$, where each D_t is a set of tuples $\langle \tau_t, v_t, \delta_t \rangle$ such that

(i) the traces $D_t \lceil 1$ and their ε-equivalent traces contain the traces of all valid executions of length t,

(ii) the traces in $D_t \lceil 1$ are mutually non-ε-equivalent, and

(iii) for each tuple δ_t is the (δ_0, ε)-*ted* for $\xi(v_0, \tau_t)$.

For each initial cover $B_{\delta_0}(v_0)$, D_0 is initialized as the tuple of empty string, the initial state v_0 and size δ_0 (Line 2). Then the reachset over-approximation is computed recursively for each time step by checking for the maximum set of enabled actions EA for the set of states $B_{\delta_t}(v_t)$ (Line 6), and trying to attach each enabled action $a \in EA$ to τ_t unless $\tau_t a$ is ε-equivalent to some length $t+1$ trace that is already in $D_{t+1} \lceil 1$. This is where the major reduction happens using approximate partial order reduction. If not, the (δ_0, ε)-*ted* for $\xi(v_0, \tau_t a)$ will be computed using `CompTed`, and new tuple $\langle \tau_t a, v_{t+1}, \delta_{t+1} \rangle$ will be added to D_{t+1} (Line 11).

If there are k actions in total and they are mutually ε-independent, then as long as the numbers of each action in τ_t and τ_t' are the same, $\tau_t \overset{\varepsilon}{\equiv} \tau_t'$. Therefore, in this case, D_t contains at most $\binom{t+k-1}{k-1}$ tuples. Furthermore, for any length t trace τ_t, if all actions in τ_t are mutually ε-independent, the algorithm can reduce the number of executions explored by $O(t!)$. Essentially, each $\tau_t \in D_t \lceil 1$ is a representative trace for the length t ε-equivalence class.

Algorithm 6: Reachability algorithm to over-approximate $\mathsf{Reach}(\Theta, T)$

```
input    : $\Theta, T, \varepsilon, \delta_0, \{\beta_a\}$
initially: $V_0 \leftarrow \delta_0$-cover$(\Theta)$, $\mathcal{R} \leftarrow \emptyset$
for $v_0 \in V_0$ do
    $D_0 \leftarrow \{\langle '', v_0, \delta_0 \rangle\}$;
    for $t = 0, \cdots, T-1$ do
        $D_{t+1} \leftarrow \emptyset$;
        for each $\langle \tau_t, v_t, \delta_t \rangle \in D_t$ do
            $EA \leftarrow$ enabledactions$(B_{\delta_t}(v_t))$ ;
            for $a \in EA$ do
                if $\forall \tau_{t+1} \in D_{t+1} \lceil 1, \neg \left(\tau_t a \overset{\varepsilon}{\equiv} \tau_{t+1}\right)$ then
                    $v_{t+1} \leftarrow a(v_t)$ ;
                    $\delta_{t+1} \leftarrow$ CompTed$(\tau_t, a, \delta_t, \varepsilon, \{\beta_a\}_{a \in A})$ ;
                    $D_{t+1} \leftarrow D_{t+1} \cup \langle \tau_t a, v_{t+1}, \delta_{t+1} \rangle$ ;
                end
            end
        end
    end
    $\mathcal{R} \leftarrow \mathcal{R} \cup \langle D_0, \ldots, D_T \rangle$ ;
end
return $\mathcal{R}$ ;
```

Theorem 7.1 shows that Algorithm 7.1 indeed computes an over-approximation for the reachable sets, and Theorem 7.2 states that the over-approximation can be made arbitrarily precise by reducing the size of δ_0, ε.

Theorem 7.1 (Soundness)**.** Set $\mathcal{R}$ returned by Algorithm 6 satisfies $\forall t = 0, \ldots, T$,

$$\mathsf{Reach}(\Theta, t) \subseteq \bigcup_{D_t \in \mathcal{R} \lceil t} \; \bigcup_{\langle \tau, v, \delta \rangle \in D_t} B_\delta(v). \tag{7.5}$$

Proof. Since $\cup_{v_0 \in V_0} B_\delta(v_0) \supseteq \Theta$, it suffices to show that at each time step $t = 0, \ldots, T$, the D_t computed in the **for**-loop from Line 2 to Line 14 satisfies $\mathsf{Reach}(B_{\delta_0}(v_0), t) \subseteq \cup_{\langle \tau, v, \delta \rangle \in D_t} B_\delta(v)$. Fixing any $v_0 \in V_0$, we will prove the theorem by induction.

Base case: Initially before any action happens, the only valid trace is the empty string $''$ and the initial set is indeed $B_{\delta_0}(v_0)$.

Induction: Assume that at time step $t < T$, the union of all the traces $R_t \lceil 1$ and their ε-equivalent traces contains the traces of all length t valid executions, and for each tuple $\langle \tau_t, v_t, \delta_t \rangle \in D_t$, δ_t is a (δ_0, ε)-*ted* for $\xi(v_0, \tau_t)$. That is, $B_{\delta_t}(v_t)$ contains the final states of all (δ_0, ε)-related executions to $\xi(v_0, \tau_t)$. This is sufficient for showing that $\mathsf{Reach}(B_{\delta_0}(v_0), t) \subseteq \cup_{\langle \tau, v, \delta \rangle \in D_t} B_\delta(v)$.

For each tuple contained in D_t, we will consider the maximum possible set of actions enabled at Line 6 and attempt to compute the (δ_0, ε)-*ted* for $\xi(v_0, \tau_t a)$. If $\tau_t a$ is not ε-equivalent to any of the length $t+1$ traces that have already been added to D_{t+1}, then Lemma 7.3 guarantees that the v_{t+1} and δ_{t+1} computed at Line 9 and 10 satisfy that δ_{t+1} is the (δ_0, ε)-*ted* for $\xi(v_0, \tau_t a)$. Otherwise, $\tau_t a$ is ε-equivalent to some trace τ_{t+1} that has already been added to D_{t+1}. In this case, for any initial state v_0' that is δ_0-close to v_0, $\xi(v_0', \tau_t a)$ and $\xi(v_0, \tau_{t+1})$ are (δ_0, ε)-related. It is easy to check that the final state of $\xi(v_0', \tau_t a)$ is already contained in $B_{\delta_{t+1}}(v_{t+1})$. Therefore, the union of all the traces $R_{t+1} \lceil 1$ and their ε-equivalent traces contains the traces of all length $t+1$ valid executions, and for each tuple $\langle \tau_{t+1}, v_{t+1}, \delta_{t+1} \rangle \in D_{t+1}$, δ_{t+1} is a (δ_0, ε)-*ted* for $\xi(v_0, \tau_{t+1})$, which means $\mathsf{Reach}(B_{\delta_0}(v_0), t+1) \subseteq \cup_{\langle \tau, v, \delta \rangle \in D_{t+1}} B_\delta(v)$. So the theorem holds. □

Theorem 7.2 (Precision). For any $r > 0$, there exist $\delta_0, \varepsilon > 0$ such that the reachset over-approximation $\mathcal{R}$ computed by Algorithm 6 satisfies $\forall t = 0, \ldots, T$,

$$\bigcup_{D_t \in \mathcal{R} \lceil t} \bigcup_{\langle \tau, v, \delta \rangle \in D_t} B_\delta(v) \subseteq B_r(\mathsf{Reach}(\Theta, t)). \tag{7.6}$$

Proof. From Proposition 7.2, for any n, $\gamma_n(\varepsilon) \to 0$ as $\varepsilon \to 0$. From Definition 7.1, for any δ_t and discrepancy function β, $\beta(\delta_t) \to 0$ as $\delta_t \to 0$. Therefore, when Line 10 of Algorithm 6 is executed, $\delta_{t+1} \to 0$ as $\delta_t \to 0$ and $\varepsilon \to 0$. Iteratively applying this observation leads to the conclusion that δ_t contained in any set D_t converges to zero as $\delta_0 \to 0$ and $\varepsilon \to 0$.

Let us fix an arbitrary $r > 0$ for the rest of the proof. We know that the set $\mathcal{R}$ is a union of approximations for each $\mathsf{Reach}(B_{\delta_0}(v_0), [0, T])$. Fixing any such v_0, δ_0, it suffices to show that at any time t, $\cup_{\langle \tau, v, \delta \rangle \in D_t} B_\delta(v) \subseteq B_r(\mathsf{Reach}(\Theta, t))$ for small enough δ_0 and ε. Moreover, it suffices to show that fixing any $\langle \tau_t, v_t, \delta_t \rangle \in D_t$, $B_\delta(v_t) \subseteq B_r(\mathsf{Reach}(\Theta, t))$ for small enough δ_0 and ε.

Since each δ_t is a (δ_0, ε)-*ted* of the execution $\xi(v_0, \tau_t)$ and δ_0, ε, there is an execution $\xi' = \xi(v_0', \tau')$ from $v_0' \in B_\delta(v_0)$ following the trace $\tau' \overset{\varepsilon}{\equiv} \tau_t$. By the definition of reachable set, we have $\xi'(t) \in \mathsf{Reach}(\Theta, t)$. On the other hand, ξ' is (δ_0, ε)-related to the potential execution $\xi(v_0, \tau_t)$, so $\xi'(t) \in B_{\delta_t}(v_t)$. That is, $B_{\delta_t}(v_t)$ and the reachset $\mathsf{Reach}(\Theta, t)$ have intersections at the state $\xi'(t)$.

The radius of at each time step δ_t can be made arbitrarily small as δ_0 and ε go to 0. We chose small enough δ_0 and ε, such that the radius of $B_{\delta_t}(v_t)$ is less than $r/2$. Therefore, $B_{\delta_t}(v_t)$ is contained in the radius r ball of the reachset $B_r(\mathsf{Reach}(\Theta, t))$. $\square$

Notice that as δ_0 and ε go to 0, the Algorithm 6 actually converges to a simulation algorithm which simulates every valid execution from a single initial state.

7.6 Experimental Evaluation of Approximate POR

In this section, we discuss the results of evaluating Algorithm 6 in three case studies. Our Python implementation runs on a standard laptop (Intel Core$^{\text{TM}}$ i7-7600 U CPU, 16G RAM).

7.6.1 Iterative Consensus

This is an instance of $\mathsf{Consensus}$ (Example 3.1) with 3 continuous variables and 3 actions a_0, a_1, a_2. We want to check if the continuous states converge to $[-0.4, 0.4]^3$ in 3 rounds starting from a radius 0.5 ball around $[2.5, 0.5, -3]$. Figure 7.4 shows the reachable set over-approximation computed and projected on $x[0]$. The blue and red curves give the bounds. As the figure shows, $x^{(0)}$ converges to $[-0.4, 0.4]$ at round 3; and so do $x^{(1)}$ and $x^{(2])}$ (not shown). We also simulated 100 random valid executions (yellow curves) from the initial set and validate that indeed the over-approximation is sound.

Recall that three actions can occur in any order in each round, i.e., $3! = 6$ traces per round, and $6^3 = 216$ executions from a single initial state up to 3 rounds. We showed in Example 7.2 that $a_0 \overset{\varepsilon}{\sim} a_1$ and $a_0 \overset{\varepsilon}{\sim} a_2$ with $\varepsilon = 0.1$. Therefore, $a_0a_1a_2 \overset{\varepsilon}{\equiv} a_1a_0a_2 \overset{\varepsilon}{\equiv} a_1a_2a_0$ and $a_0a_2a_1 \overset{\varepsilon}{\equiv} a_2a_0a_1 \overset{\varepsilon}{\equiv} a_2a_1a_0$, and Algorithm 6 explored only 2 (length 12) executions from a set of initial states for computing the bounds. The running time for Algorithm 6 is 1 millisecond while exploring all valid executions from even only a single state took 20 milliseconds.

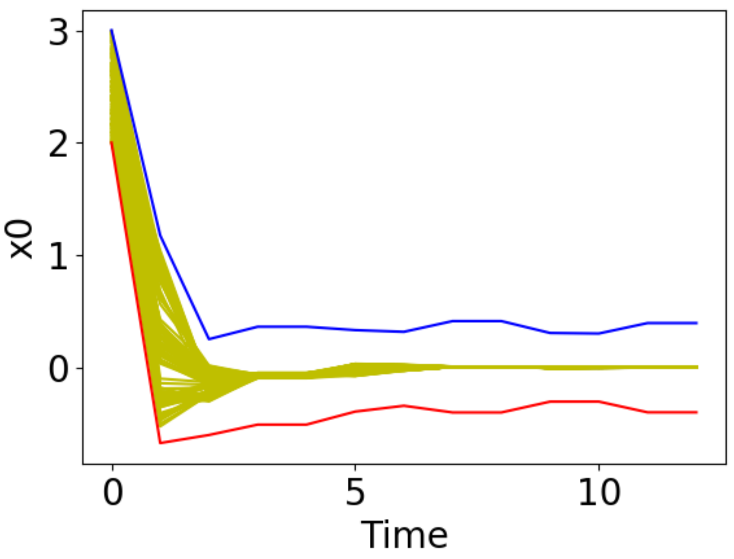


Figure 7.4: Reachable set of the Consensus example. The blue curves are the upper bound of the reachsets and the red curves are the lower bound of the reachsets. Between the blue and red curves, the yellow curves are 100 random simulations of valid executions.

7.6.2 Platoon

Consider an N car platoon on a single-lane road (see Figure 7.5). Each car can choose one of three actions at each time step: a (accelerate), b (brake), or c (cruise). Car 0 can choose any action at each time step; remaining cars try to keep safe distances from preceding cars by choosing to (a) accelerate if the distance is more than 50, (b) brake if the distance is less than 30, and (c) cruise otherwise. For each $i \in \{0, \cdots, N-1\}$, $x^{(2i)}$ is the position, $x^{(2i+1)}$ is the velocity, and $m^{(i)}$ is the chosen action of the i^{th} car. At each step, $m^{(i)}$ is updated using relative positions according to the rule described above, and then x is updated according to the actions. For concreteness, the linear state transition equation for a 2-car platoon is shown below:

```
automaton CarPlatoon($N : \mathbb{N}$)
  variables
    $x : \mathbb{R}^{2N}$ ;
    $m : \{c, a, b\}^N$;

  initially
    for each $i \in \{0, \cdots, N-1\}$
      $m^{(i)}$ := choose $\{c, a, b\}$;

transitions
  move
    pre true
    eff $m^{(0)}$ := choose $\{c, a, b\}$;
      for each $i \in \{1 \cdots, N\}$
        $m^{(i)} := \begin{cases} a & \text{if } x^{(2(i-1))} - x^{(2i)} > 50 \\ b & \text{if } x^{(2(i-1))} - x^{(2i)} < 30 \\ c & \text{else} \end{cases}$;
        $x := Ax + b_m$;
```

Figure 7.5: Labeled transition system model of cars keeping a platoon.

$$x \leftarrow \begin{bmatrix} 1 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 1 \end{bmatrix} x + \begin{bmatrix} \frac{acc_0(\Delta t)^2}{2} \\ acc_0 \Delta t \\ \frac{acc_1(\Delta t)^2}{2} \\ acc_1 \Delta t \end{bmatrix} = Ax + b_m, \tag{7.7}$$

where $acc_i > 0$ if car i accelerates; $acc_i < 0$ if it brakes; and $acc_i = 0$ if it cruises. For any value of m, the discrepancy functions for the corresponding actions are the same: For any v, v' with $v.L = v'.L$, $\beta_a(\|v.x - v'.x\|) = \|A\|\|v.x - v'.x\|$. For any $i, j \in \{0, \cdots, 8\}$ with $i \neq j$, we notice that $\|a_i a_j(v).x - a_j a_i(v).x\| = \|Ab_{m_i} - Ab_{m_j} + b_{m_j} - b_{m_i}\|$ which is a constant number and can be used as ε. If we choose $\Delta t = 0.1$, then the discrepancy function could be $\beta_a(\|v.x - v'.x\|_2) = 1.06\|v.x - v'.x\|_2$. Furthermore, if acc_i can choose from $\{-10, 0\}$, or from $\{10, 0\}$, then the corresponding actions are ε-independent with $\epsilon = 0.141$, and if acc_i can choose from $\{-10, 0, 10\}$, then the corresponding actions are ε-independent with $\epsilon = 0.282$.

Consider a 2-car platoon and a time horizon of $T = 10$. We want to verify that the cars maintain safe separation. Reachset over-approximations projected on the position variables are shown in Figure 7.6, with 100 random simulations of valid executions as a sanity check. Car 0 has lots of choices and its position over-approximation diverges (Figure 7.6). Car 1's position depends on its initial relative distance with Car 0. It is also easy to conclude from Figure 7.6 that two cars maintain safe relative distance for these different initial states.

From a single initial state, in every step, Car 0 has 3 choices, and therefore there are 3^{10} possible executions. Considering a range of initial positions for two cars, there are infinitely many executions, and 9^{10} (around 206 trillion) possible traces. With $\epsilon = 0.282$, Algorithm 6 explored a maximum of $\binom{18}{8} = 43758$ traces; the concrete number varies for different initial sets. The running time for Algorithm 6 is 5.1 milliseconds while exploring all valid executions from even only a single state took 2.9 seconds.

For a 4-car platoon and a time horizon of $T = 10$, there are 81^{10} possible traces considering a range of initial positions. With $\varepsilon = 0.282$, Algorithm 6 explored 7986 traces to conclude that all cars maintain safe separation for the setting where all cars are initially separated by a distance of 40 and have an initial set radius of 4. The running time for Algorithm 6 is 62.3 milliseconds, while exploring all valid executions from even only a single state took 6.2 seconds.

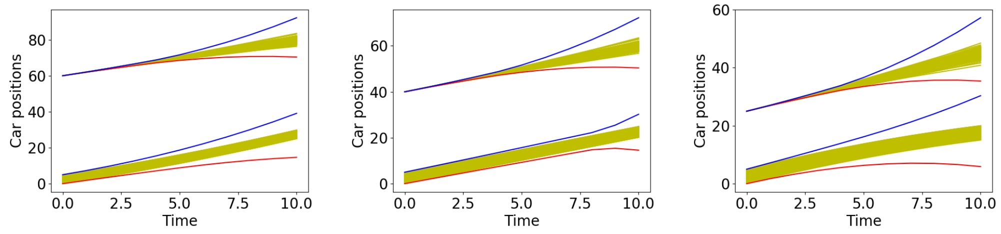


Figure 7.6: Position over-approximations for 2 cars of the CarPlatoon example. The blue curves are the upper bound of the reachsets and the red curves are the lower bound of the reachsets. Between the blue and red curves, the yellow curves are 100 random simulations of valid executions. Car 0's initial position is in the range $[0, 5]$, Car 1's initial position is 60 (Left), 40 (Center) and 25 (Right).

7.6.3 Building Heating System

Consider a building heating system in Figure 7.7. The building has N rooms each with a heater. For $i \in \{0, \cdots, N-1\}$, $x^{(i)} \in \mathbb{R}$ is the temperature of room i and $m^{(i)} \in \{0, 1\}$ captures the off/on state of the heater in the room. The building measures the temperature of rooms periodically every T seconds and saves the measurements to $y^{(i)}$. Based on the measurement $y^{(i)}$, each room takes action a_i to decide whether to turn on or turn off its heater. The Boolean variable $d^{(i)}$ indicates whether room i has made a decision. These decisions are made asynchronously among the rooms with a small delay h. For this system, we want to check whether the temperature of the room remains in an appropriate range.

```
automaton Roomheating($N : \mathbb{N}$)
  variables
    $x : \mathbb{R}^N$ initially $x^{(i)} := 60$;
    $y : \mathbb{R}^N$ initially $y^{(i)} := 60$;
    $d : \mathbb{B}^N$ initially $d := \mathsf{false}^N$;
    $m : \mathbb{B}^N$ initially $m := \mathsf{false}^N$;

  transitions
    on_i, for $i \in \{0, \cdots, N-1\}$
      pre $!d^{(i)} \wedge y^{(i)} <= 72$
      eff $x := W_h x + b_h + C_h m$;
          $d^{(i)} := \mathsf{true} \wedge m^{(i)} := \mathsf{true}$;
    off_i, for $i \in \{0, \cdots, N-1\}$
      pre $!d^{(i)} \wedge y^{(i)} >= 68$
      eff $x := W_h x + b_h + C_h m$;
          $d^{(i)} := \mathsf{true} \wedge m^{(i)} := \mathsf{false}$;

    flow
      pre $\wedge_{i \in \{0, \cdots, N-1\}} d_i$
      eff $x := W_T x + b_T + C_T m$;
          $d^{(i)} := \mathsf{false}, \forall i \in \{0, \cdots, N-1\}$;
          $y := x$;
```

Figure 7.7: Transition system of room heating.

For $i \in \{0, \cdots, N-1\}$, actions $\mathsf{on_i}$,$\mathsf{off_i}$, capture the decision-making process of room i about whether or not to turn on the heater. During the process, time elapses for a (short) period h, which leads to an update of the temperature as an affine function of current temperature x and the heaters state m. The affine function is derived from the thermal equations presented in [140]. In this section, we use an instance of the system with the following matrices:

$$W_h = \begin{bmatrix} 0.96 & 0.01 & 0.01 \\ 0.02 & 0.97 & 0.01 \\ 0 & 0.01 & 0.97 \end{bmatrix}, b_h = \begin{bmatrix} 1.2 \\ 0 \\ 1.2 \end{bmatrix}, C_h = \begin{bmatrix} 0.4 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0.4 \end{bmatrix}. \tag{7.8}$$

After a room controller makes a decision ($\mathsf{on_i}$ or $\mathsf{off_i}$ transition occurs), the variable $d^{(i)}$ changes to true. After all rooms make their decisions, action flow captures the time elapsed for a (longer) period T which also updates the measured values y. We use an instance of this step with the following matrices:

$$W_T = \begin{bmatrix} 0.18 & 0.11 & 0.14 \\ 0.18 & 0.25 & 0.17 \\ 0.09 & 0.13 & 0.28 \end{bmatrix}, b_T = \begin{bmatrix} 34.2 \\ 24 \\ 30 \end{bmatrix}, C_T = \begin{bmatrix} 11.4 & 0 & 0 \\ 0 & 8 & 0 \\ 0 & 0 & 10 \end{bmatrix}. \tag{7.9}$$

For each $i \in \{0, \cdots, N-1\}$ and $a_i \in \{\mathsf{on_i}, \mathsf{off_i}\}$, we will derive the discrepancy function for action a. For any v, v' with $v.L = v'.L$,

$$\begin{aligned} & \|a_i(v).x - a_i(v').x\| \\ = \ & \|W_h v.x + b_h + C_h v.m - W_h v'.x - b_h - C_h v'.m\| \\ \leq \ & \|W_h\| \|v.x - v'.x\|. \end{aligned}$$

We note that $\|W_h\|_2 = 0.99$. Hence, we can define $\beta_a(\|v.x - v'.x\|_2) = 0.99\|v.x - v'.x\|_2$ as the discrepancy functions of each $a \in \{\mathsf{on_i}, \mathsf{off_i}\}_{i \in \{0,1,2\}}$. Similarly, we derived that $\beta_{\mathsf{flow}}(\|v.x - v'.x\|_2) = 0.52\|v.x - v'.x\|_2$.

For any $i, j \in \{0, 1, 2\}$ with $i \neq j$, $a_i \in \{\mathsf{on_i}, \mathit{off}_i\}$ and $a_j \in \{\mathsf{on_j}, \mathsf{off_j}\}$, we can prove $a_i \overset{\varepsilon}{\sim} a_j$ with $\varepsilon = 0.6$. Notice that $a_i(v).x = W_h v.x + b_h + C_h v.m = a_j(v).x$ are identical, but $a_i(v).m$ and $a_j(v).m$ could be different.

$$\begin{aligned} & \|a_i a_j(v).x - a_j a_i(v).x\| \\ = \ & \|W_h a_j(v).x + b_h + C_h a_j(v).m - W_h a_i(v).x - b_h - C_h a_i(v).m\| \\ = \ & \|C_h a_j(v).m - C_h a_i(v).m\| \leq \|C_h\| \|a_j(v).m - a_i(v).m\|. \end{aligned}$$

We note that $\|C_h\|_2 = 0.4$. We will give an upper bound on $\|a_j(v).m - a_i(v).m\|$. Notice that $a_i(v).m$ and $v.m$ can only differ in one bit (m_i). Similarly, $a_j(v).m$ and $v.m$ can only differ in one bit (m_j). Hence $a_i(v).m$ and $a_j(v).m$ can be differ in at most two bits, and $\|a_i(v).m - a_j(v).m\|_2 \leq \|[1, 1, 0]\|_2 = 1.41$. Therefore,

$$\|a_i a_j(v).x - a_j a_i(v).x\|_2 \leq 0.4 * 1.41 \leq 0.6.$$

Thus for any pair of rooms, the on/off decisions are ε-approximately independent with $\varepsilon = 0.6$. The action flow is not independent of any other actions. For a round in which each room makes a decision once in arbitrary order, there are in total $3! = 6$ ε-equivalent action sequences.

We want to verify that the room temperatures remain in the $[60, 79]$ range.

Computed reachable set over-approximation for 8 rounds and projected on the temperature of Room 0 is shown in Figure 7.8. Indeed, the temperature of Room 0 is contained within the range.

For a round in which each room makes a decision once in arbitrary order, there are $3! = 6$ ε-equivalent action sequences. Therefore, from a single initial state, there are 6^8 (1.6 million) valid executions. Algorithm 6 in this case explores only one (length 32) execution with $\varepsilon = 0.6$ to approximate all executions starting from an initial set with radius $\delta = 2$. The running time for Algorithm 6 is 1 millisecond, while exploring all valid executions from only a single state took 434 seconds.

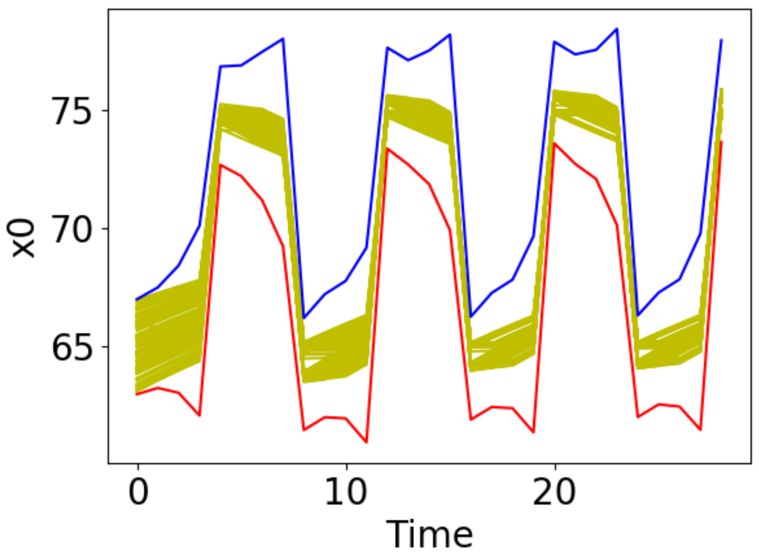


Figure 7.8: Reachable set of the Roomheating example. The blue curves are the upper bound of the reachsets and the red curves are the lower bound of the reachsets. Between the blue and red curves, the yellow curves are 100 random simulations of valid executions.

7.7 Summary

In this chapter, we discussed a partial order reduction technique for reachability analysis of infinite state transition systems that exploits approximate independence and bounded sensitivity of actions to reduce the number of executions explored. This relies on a novel notion of ε-independence that generalizes the traditional notion of independence by allowing approximate commutation of actions. With this ε-independence relation, we have developed an algorithm for soundly over-approximating reachsets of all executions using only ε-equivalent traces. The over-approximation can also be made arbitrarily precise by reducing the size of δ, ε. In experimental evaluation with three case studies we observe that it can reduce the number of executions explored exponentially compared to explicit computation of all executions.

Chapter 8

Controller Synthesis for Linear Systems with Reach-avoid Specification

8.1 Introduction

In this chapter, we study the control synthesis problem for linear, discrete-time, and time-varying plant models with bounded disturbance — a standard and general framework for dynamical systems [38, 39]. We will consider *reach-avoid* specifications which require that starting from any initial state Θ, the controller has to drive the system to a target set G, while avoiding certain unsafe states or obstacles O.

Textbook control design methods address specifications like stability, disturbance rejection, and asymptotic convergence, but they do not provide formal guarantees about reach-avoid specifications. Receding horizon control, also known as model predictive control (MPC), has been broadly used on constrained control problems by solving a constrained optimization problem repeatedly over a moving time horizon. Using MPC for reach-avoid specifications typically solves a sequence of mixed integer linear programming (MILP) [141, 142] or general nonlinear optimization problems [143, 144]. Another approach is based on *discrete abstraction*, where a discrete, finite-state, symbolic abstraction of the original control system is computed, and a discrete controller is synthesized by solving a two-player game on the abstracted game graph. Theoretically, these methods can be applied to systems with nonlinear dynamics and they can synthesize controllers for a general class of linear temporal logic (LTL) specifications. However, in practice, the discretization step leads to a severe state space explosion for higher dimensional models. A detailed comparison between these methods and the proposed approach in this chapter is provided in the related work section (Section 8.2).

In this chapter, the controller we synthesize follows a natural paradigm for designing controllers. The approach is to first design an *open-loop* controller for a single initial state $x_0 \in \Theta$ to meet the reach-avoid specification. This is called the reference trajectory. For the remaining states in the initial set, a *tracking controller* is combined, that drives these other trajectories towards the reference trajectory that starts from x_0. However, designing such a combined controller can be computationally expensive [145] because of the interdependency between the open-loop controller and the tracking controller (Section 8.4.1). Our novel approach to making this approach feasible is to demonstrate that the two controllers can be synthesized in a decoupled way. Our strategy is as follows. We first design a tracking controller using a standard control-theoretical method called LQR (linear quadratic regulator) [146]. The crucial observation that helps decouple the synthesis of the tracking and open-loop controller is that for such a combined controller, once the tracking controller is fixed, the set of states reached from the initial set is contained within a sequence of ellipsoidal sets [83] centered around the reference trajectory. The shape and size of these ellipsoidal sets are solely dependent on the tracking controller and the disturbance, and are independent of the reference trajectory or the open-loop controller. On the flip side, the open-loop controller and the resulting reference trajectory can be chosen independently of the fixed tracking controller. Based on this, the problem of synthesizing the open-loop controller can be completely decoupled from synthesizing the tracking controller.

Our open-loop controller is synthesized by encoding the problem in logic. The straightforward encoding of the synthesis problem is to find an open loop controller that can make sure all states in the reach set ellipsoids satisfy the reach-avoid specification. Such encoding results in a $\exists\forall$ formula in the theory of linear arithmetic. Unfortunately, solving large instances of such formulas using current satisfiability modulo theories (SMT) solvers is challenging. To overcome this, we exploit geometric properties of polytopes and ellipsoids, and reduce the original $\exists\forall$-formula into the quantifier-free fragment of linear arithmetic (QF-LRA). Moreover, assuming that the obstacles and goal set can be represented as polytopes, then the number of linear constraints in the QF-LRA formulas is only linear to time and to the number of hyperplanes as the surfaces of the obstacles and the goal set. In this way, the proposed approach for synthesizing the combined controller can scale to large dimensional systems.

Our overall algorithm (Algorithm 7), after computing an initial tracking controller, iteratively synthesizes open-loop controllers by solving QF-LRA formulas for smaller subsets that cover the initial set. The algorithm will automatically identify the set of initial states for which the combined tracking+open-loop controller is guaranteed to work. Our algorithm is sound (Theorem 8.1), and for a class of robust linear systems, it is also complete (Theorem 8.2).

We have implemented the synthesis algorithm in the tool RealSyn, which first appeared in [27]. We compare the performance of the new algorithm proposed in this chapter with the one in [27] on 10 benchmark problems, whose obstacles are general polytopes instead of only axis-aligned hyper-rectangles. In RealSyn, any SMT solver can be plugged in for solving the open-loop problem. We report the results of using the Yices solver, as Yices outperformed other solvers in [27]. Results show that our new approach can achieve a 2 to 65 times speedup for most benchmark models. The proposed new algorithm also scales well for complex models — including a system with 3 vehicles (12-dimensional) trying to reach a common goal while avoiding collision with the obstacles and each other, and another system with 10 vehicles (20 dimensional) trying to maintain a platoon. For all the benchmark models, RealSyn with the new algorithm finds a controller within 3 minutes using the Yices solver, and for most benchmarks it finds a controller within 10 seconds.

An early version of this approach was presented in [27]. Compared with [27], the major improvements in this chapter are:

1. In this dissertation the approach works for linear time-varying systems, while in [27] only linear time-invariant systems were considered.

2. In [27] the sets of the reachable states were represented as a sequence of ellipsoids with same shape and orientation (but different sizes), which were always over-approximations of the exact reach sets. However, in this dissertation, we allow the reach set ellipsoids to change their shape and orientation at different time steps, which leads to a more accurate over-approximation.

3. In this dissertation, we exploit more efficient methods to encode the constraints of reach sets being separated from the obstacles (or contained in the goal set). As a result, the final formulas for synthesizing

the open-loop controller consist of $O(k)$ linear constraints, where k is the number of hyperplanes as the surfaces of the obstacles and the goal set. In contrast, in [27], such formulas have $O(2^n k)$ linear constraints, where n is the dimensionality of the state space. Experimental results also show the improvements using the new approach, especially on large dimensional ($n > 10$) systems.

8.2 Related Work on Synthesis

Controller synthesis techniques have been the center of extensive investigation with numerous publications every year lately. Here we briefly review related works based on different plant models, specifications, and several major approaches.

Models and Specifications for Synthesis. In increasing order of generality, the types of plant models that have been considered for controller synthesis are double-integrator models [147], linear dynamical models [148, 149, 150, 151, 152, 141, 153], piecewise affine models [41, 154], and nonlinear (possibly switched) models [155, 40, 156, 157, 144]. There is also a line of work on synthesis approaches for stochastic plants (see [158], and the references therein). For each of the classes, both continuous and discrete-time models have been addressed with possibly different approaches.

There are several classes of specifications typically used for synthesis: (1) stabilization for system with special properties, including positive systems [148] and systems with quantized measurements [159, 160], (2) pure safety or invariance specifications [156, 161, 162], (3) reach-avoid [42, 156, 161, 40, 41], and (4) general LTL, GR(1) [163, 149, 164] [150, 154, 165], Metric Temporal Logic [166], and Signal Temporal Logic [167, 142]. For each of these classes both bounded and unbounded-time variants have been considered.

In this work, we focus on linear, discrete-time, time-varying systems with reach-avoid specifications.

Model Predictive Control. MPC [168] utilizes an explicit plant model to predict the plant state and compute the control input to the plant based on this prediction. At each control interval, an MPC algorithm attempts to solve a constrained, discrete-time, optimal control problem in an online setting, with the objective of optimizing future plant behavior based on current state. Without loss of generality, assume the current state of the system is $x[0]$, MPC solves a finite horizon (N steps) optimal control problem defined by:

$$\begin{array}{ll} \min_{u[0],\ldots,u[N-1]} & V_f(x[N]) + \sum_{i=0}^{N-1} \ell(x[i], u[i]) \\ \text{s.t.} & \bigwedge_{i=0}^{N} x[i] \in X, \bigwedge_{j=0}^{N-1} u[j] \in U, \end{array} \tag{8.1}$$

where in the objective function V_f defines cost of the final state of the controlled system $x[N]$, ℓ defines the cost of the rest of the states and control inputs, and the controlled system is required to satisfy the state and control constraints $x[i] \in X, u[i] \in U$, respectively. The *implicit MPC* law asks that at the state $x[0]$, the first control $u[0]$ of the computed optimal control sequence is applied, and the entire calculation is repeated at subsequent control intervals. When optimal control problems admit an explicit offline solution, online operations reduce to a simple function evaluation of a function of the state vectors. Such approach is referred to as *explicit MPC* and has been exploited in many applications including motion planing [152, 141, 153]. The idea of explicit MPC is to solve the optimization problem (8.1) offline for all x within a given set, and to make the dependence of $u(t)$ on $x(t)$ explicit. The resulting MPC control law is a piecewise affine function of the state x defined over a polyhedral partition of the feasible set X_f. For systems with large dimensions of states and controls, explicit MPC is not practically feasible since the number of partitions can be very large. Furthermore, it is hard to make explicit MPC handle cases where the system, cost function, or constraints are time-varying [169].

Using MPC for controller synthesis typically requires model reduction for casting the optimization problem (8.1) as a linear programming (LP) [152], quadratic programming (QP) [170], mixed integer linear programming (MILP) [141, 167, 142] or general nonlinear optimization problems [143, 144].

The major differences between our approach and the MPC-based approaches include:

1. Our approach does not require the help of a cost function.

2. We solve a controller that works for an initial set X_0 with radius r.

Implicit MPC solves for a single initial point, and explicit MPC solves for all states in the feasible set X_f.

3. We use different approaches to incorporate the avoidance condition $x[i] \in X$ in the optimization problem (8.1). In this dissertation, the obstacles at each step are specified by a collection of polytopes. Therefore, the safe region X, as the complement of the obstacles, is usually non-convex.

   To encode such avoidance condition, one has to introduce disjunctions to the constraints. In [144], the authors use Farkas' lemma to change the avoidance condition into its dual form that is compatible for MPC formulation. However, the extra variables introduced by Farkas' lemma will lead to nonlinear constraints. In [141], the authors introduce extra Boolean variables to eliminate the disjunctions, and make the original optimization problem (8.1) an MILP. Both the works use implicit MPC law. The main drawback of implicit MPC is the need to solve a mathematical program online or within the sampling time to compute the control action. Therefore, it is hard to use on systems with large dimensionality [171] and when the sampling period is short. Explicit MPC can help relieve the heavy online computation load, especially when the optimization problem is a LP or QP. However, in this case, the explicit solution for nonlinear optimization and MILP cannot be solved very efficiently in practice [171].

   Compared with these MPC-based approaches, our proposed method benefits from the fact that the tracking controller can fix the shapes and sizes of the reach set ellipsoids from an initial set. We further exploit special properties of the separation between ellipsoids and polytopes to make the constraints quantifier-free over linear real arithmetics, which can be efficiently solved using state-of-the-art SMT solvers. As a result, our approach can scale to large dimensional systems and the computation can be performed offline.

Discrete Abstractions. In recent years, controller synthesis techniques based on so-called *symbolic models* or *discrete abstractions* have received considerable attention within the control systems community, see e.g. [156, 161, 163, 149, 164, 149, 150, 151]. These techniques involve constructing a finite partition of the continuous state space with respect to a set-valued map. Following these methods, it is possible to synthesize controllers for general nonlinear systems to enforce complex specifications formulated in LTL.

There is a growing set of controller synthesis tools and libraries based on the discrete abstraction approach. These include tools like CoSyMA [172], Pessoa [173], LTLMop [174, 175], Tulip [164, 176], and SCOTS [155]. Compared with these methods, our proposed solution takes a different route by "designing" the shape of reach sets first with the tracking controller, then "placing" the reach sets using the open loop controller. The entire process does not involve any partition of the state space, and therefore avoids the potential problem of exponentially growing partitions for large dimensional systems. Our trial with a 4-dimensional example on Tulip [164, 176] did not finish the discretization step in one hour. LTLMop [174, 175] handles GR(1) LTL specifications, which are more general than reach-avoid specifications considered in this dissertation, but it is designed for 2-dimensional robot models working in the Euclidean plane. It generates a hybrid controller as a combination of discrete controllers and continuous controllers to meet the high-level specification under certain assumptions on the environment.

Sampling Based Path Planning. Sampling based methods such as Probabilistic Road Maps (PoMP) [177], Rapidly-exploring Random Trees (RRT) [178], and fast marching tree (FMT) [179] have gained much interest in recent years. They offer the benefits of generating feasible trajectories through known or partially known environments. Compared with the deterministic guarantees provided by our proposed method, the sampling based methods usually come with stochastic guarantees. Also, they are not designed to be robust to model uncertainty or disturbances.

In addition to the above approaches, an alternative synthesis technique generates mode switching sequences for switched system models [180, 181, 182, 183, 184] to meet the specifications. This line of work focuses on a finite input space, instead of the infinite input space we are considering in this paper.

Abate et al. [162] use a controller template similar to the one considered in this chapter for invariant specifications. A counter-example guided inductive synthesis (CEGIS) approach is used to first find a feedback controller for stabilizing the system. Since this feedback controller may not be safe for all initial states of the system, a separate verification step is employed to verify safety, or alternatively to find a counter-example. In the latter case, the process is repeated until a valid controller is found. This is different from our approach, where any controller found needs no further verification.

8.3 Bounded Controller Synthesis Problem on Discrete-time Linear Control Systems

An (n, m)-*dimensional time-varying discrete-time linear system* $\mathcal{A}$ is a 5-tuple $\langle A, B, \Theta, U, D\rangle$, where

(i) A is an infinite sequence of $\mathbb{R}^{n\times n}$ matrices, called *dynamic matrices.*

(ii) B is an infinite sequence of $\mathbb{R}^{n\times m}$ matrices, called *input matrices*, and at each time step t, we ask that the pair $(A[t], B[t])$ is controllable.

(iii) $\Theta \subseteq \mathbb{R}^n$ is a *set of initial states.*

(iv) $U \subseteq \mathbb{R}^m$ is the *space of inputs.*

(v) $D \subseteq \mathbb{R}^n$ is the *space of disturbances.*

A *control sequence* for an (n, m)-dimensional system $\mathcal{A}$ is a (possibly infinite) sequence $u = u[0], u[1], \ldots$, where each $u[t] \in U$. Similarly, a *disturbance sequence* for $\mathcal{A}$ is a (possibly infinite) sequence $d = d[0], d[1], \ldots$, where each $d[t] \in D$. Given control u and disturbance d, and an initial state $x[0] \in \Theta$, the *execution of* $\mathcal{A}$ is uniquely defined as the (possibly infinite) sequence of states $x = x[0], x[1], \ldots$, where for each $t > 0$,

$$x[t+1] = A[t]x[t] + B[t]u[t] + d[t]. \tag{8.2}$$

A *(state feedback) controller* for $\mathcal{A}$ is a function $g : \Theta \times \mathbb{R}^n \to \mathbb{R}^m$ that maps an intial state and a (current) state to an input. That is, given an initial state $x_0 \in \Theta$ and state $x \in \mathbb{R}^n$ at time t, the control input to the plant at time t is:

$$u[t] = g(x_0, x). \tag{8.3}$$

This controller is allowed to use the memory of some initial state x_0 (not necessarily the current execution's initial state) for deciding the current state-dependent feedback. Thus, given an initial state $x[0]$, a disturbance d, and a state feedback controller g, Equations (8.2) and (8.3) define a unique execution x of $\mathcal{A}$. A state x is *reachable at the t^{th}-step* if there exists an execution x of $\mathcal{A}$ such that $x[t] = x$. The set of all reachable states from some set $S \subseteq \Theta$ in exactly T steps using the controller g is denoted by $\mathsf{Reach}_{\mathcal{A},g}(S, T)$. When $\mathcal{A}$ and g are clear from the context, we write $\mathsf{Reach}(S, T)$.

Given an (n, m)-dimensional time-varying discrete-time linear system $\mathcal{A}$, a sequence O of *obstacles* or unsafe sets (with $O[t] \subseteq \mathbb{R}^n$, for each t), a *goal* $G \subseteq \mathbb{R}^n$, and a time bound T, the *bounded time controller synthesis problem* is to find, a state feedback controller g such that for every initial state $\theta \in \Theta$ and disturbance sequence $d \in D^*$ of length T, the unique execution x of $\mathcal{A}$ with g, starting from $x[0] = \theta$, satisfies

(i) for all $t \leq T$, $u[t] \in U$,

(ii) for all $t \leq T$, $x[t] \notin O[t]$, and

(iii) $x[T] \in G$.

For the rest of this section, we will assume that each of the sets in $\{O[t]\}_{t\in\mathbb{N}}$, G and U are closed polytopes. Moreover, we assume that the pairs $(A[t], B[t])$ at each time step are controllable [39].

The controller synthesis problem requires one to find a state feedback controller that ensures that the trajectory starting from any initial state in Θ will meet the reach-avoid specification. Since the set of initial states Θ will typically be an infinite set, this requires the synthesized feedback controller g to have an effective representation. Thus, an "enumerative" representation, where a (separate) *open-loop controller* is constructed for each initial state, is not feasible — by an open-loop controller for initial state $x_0 \in \Theta$, we mean a control sequence u such that the corresponding execution x with $x[0] = x_0$ and 0 disturbance satisfies the reach-avoid constraints. We therefore need a useful template that will serve as the representation for the feedback controller. We address this challenge in our work and articulate the details in the remaining article.

Example 8.1. Consider a mobile robot that needs to reach the green area of an apartment starting from the entrance area, while avoiding the red areas

(Figure 8.1). The robot's dynamics are described by a linear model (for example the navigation model from [140]). The obstacle sequence O here is static, that is, $O[t] = O[0]$ for all $t \geq 0$. Both Θ and G are rectangles (which are also polytopes). Although these sets are depicted in 2D, the dynamics of the robot may involve a higher dimensional state space.

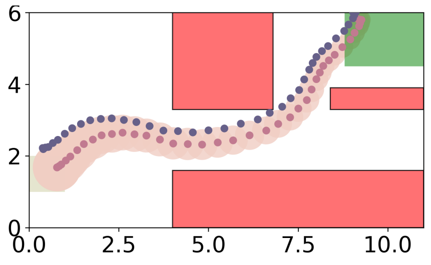


Figure 8.1: The settings for controller synthesis of a mobile robot with reach-avoid specification.

In this example, there is no disturbance, but a similar problem can be formulated for a drone flying outdoors, in which case the disturbance input would model the effect of wind. Time-varying obstacle sets are useful for modeling safety requirements of multi-robot systems.

Suppose the robot is asked to reach the target set in 40 steps. The dotted curves are two executions from Θ and the pink ellipsoids show the projection of the reachset on the robot's position with synthesized controller.

8.4 Synthesis Algorithm

8.4.1 Algorithm Overview

In control theory, one natural controller design paradigm is to first find a *reference execution* x_{ref} which uses an *open-loop controller*, then add a *tracking controller* which tries to force other executions x starting from different initial states $x[0]$ to get close to x_{ref} by minimizing the distance between x_{ref} and x. This form of controller combining open-loop control with tracking control is also proposed in [145] for reach-avoid specifications. For the discrete-time linear control system defined as Equation (8.2), the combined controller is formally defined as follows:

Definition 8.1. Given a discrete-time linear system as Equation (8.2), the combined controller g is a tuple $\langle K, x_{\mathsf{ref}}[0], u_{\mathsf{ref}}\rangle$ such that the control input $u[t]$ to the system is

$$u[t] = u_{\mathsf{ref}}[t] + K[t](x[t] - x_{\mathsf{ref}}[t]), \text{with} \tag{8.4}$$

$$x_{\mathsf{ref}}[t+1] = A[t]x_{\mathsf{ref}}[t] + B[t]u_{\mathsf{ref}}[t], \tag{8.5}$$

where

(1) u_{ref} is called the **open-loop control sequence**, which determines the value of the reference execution $x_{\mathsf{ref}}[t]$ at each time step $t \in \mathbb{N}$ once $x_{\mathsf{ref}}[0]$ is fixed, and

(2) K is called the **tracking controller**, which is a sequence of matrices that determine the additive component of the input based on the difference between the current state and the reference execution.

Given the combined feedback controller g as the tuple $\langle K, x_{\mathsf{ref}}[0], u_{\mathsf{ref}}\rangle$, we could rewrite the linear system in (8.4) as an augmented system

$$\begin{bmatrix} x \\ x_{\mathsf{ref}} \end{bmatrix}[t+1] = \begin{bmatrix} A[t] + B[t]K[t] & -B[t]K[t] \\ 0 & A[t] \end{bmatrix} \begin{bmatrix} x \\ x_{\mathsf{ref}} \end{bmatrix}[t]$$
$$+ \begin{bmatrix} B[t] & 0 \\ 0 & B[t] \end{bmatrix} \begin{bmatrix} u_{\mathsf{ref}} \\ u_{\mathsf{ref}} \end{bmatrix}[t] + \begin{bmatrix} d \\ 0 \end{bmatrix}[t].$$

Observe that the above augmented system has the form:

$$\hat{x}[t+1] = \hat{A}[t]\hat{x}[t] + \hat{B}[t]\hat{u}[t] + \hat{d}[t],$$

and the closed-form solution of the above augmented system is given by

$$\hat{x}[t] = \Big(\prod_{i=0}^{t-1} \hat{A}[i]\Big)\hat{x}[0] + \sum_{i=0}^{t-1}\Big(\prod_{j=i+1}^{t-1} \hat{A}[j]\Big)\big(\hat{B}[i]\hat{u}[i] + \hat{d}[i]\big). \tag{8.6}$$

To synthesize a controller g of this form, therefore, requires finding $K, x_{\mathsf{ref}}[0], u_{\mathsf{ref}}$ such that the closed-form solution meets the reach-avoid specification. This is indeed the approach followed in [145], albeit in the continuous time setting. Observe that in the closed-form solution, $\hat{A}[t]$, $\hat{u}$, and $\hat{x}[0]$ all depend on parameters that we need to synthesize. Therefore, solving such constraints involves polynomials whose degrees grow with the time bound. This is very expensive, and unlikely to scale to large dimensions and time bounds.

In this dissertation, to achieve scalability, we take a slightly different approach than the one where $K, x_{\mathsf{ref}}[0]$, and u_{ref} are simultaneously synthesized. We first synthesize a tracking controller K, *independent* of $x_{\mathsf{ref}}[0]$ and u_{ref}, using the standard LQR method. Once K is synthesized, we show that, no matter what $x_{\mathsf{ref}}[0]$ and u_{ref} are, the state of the system at time t starting from x_0 is guaranteed to be contained within an ellipsoid centered at $x_{\mathsf{ref}}[t]$ with shape and radius that depend only on K, the initial distance between x_0 and $x_{\mathsf{ref}}[0]$, time t, and disturbance set D. Moreover, this radius is only a *linear* function of the initial distance (Lemma 8.1). Thus, if we can synthesize an open-loop controller u_{ref} starting from some state $x_{\mathsf{ref}}[0]$, such that ellipsoids centered around x_{ref} satisfy the reach-avoid specification, we can conclude that the combined controller will work correctly for all initial states in some ball around the initial state $x_{\mathsf{ref}}[0]$. The radius of the ball around $x_{\mathsf{ref}}[0]$ for which the controller is guaranteed to work will depend on the radii of the ellipsoids around x_{ref} that satisfy the reach-avoid specification. This decoupled approach to synthesis is the first key idea in our algorithm.

Synthesizing the tracking controller K still leaves open the problem of synthesizing an open-loop controller for an initial state $x_{\mathsf{ref}}[0]$. A straightforward encoding of the problem could be to find an open-loop controller that works for all initial states in some ball around $x_{\mathsf{ref}}[0]$. That is, finding a satisfying solution for the formula $\exists u_{\mathsf{ref}}, \exists r$, such that $\forall x[0] \in B_r(x_{\mathsf{ref}}[0]), \bigwedge_{t=0}^{T} x[t] \notin O[t] \wedge x[T] \in G$. This results in a $\exists\forall$-formula in the theory of real arithmetic. Unfortunately, solving such formulas does not scale to large dimensional systems using current SMT solvers. The next key idea in our algorithm is to simplify these constraints and make the formula quantifier free. We reduce the problem of deciding whether an ellipsoid (the set of reachable states) is separated from (or contained in) a polytope (the obstacles or the goal) to measuring the distances of the center of the ellipsoid to surfaces of the polytopes in a linearly transformed coordinate. In this way we are able to reduce the original $\exists\forall$-formula into the *quantifier-free* fragment of *linear* real arithmetic (QF-LRA) (Section 8.4.4).

Putting it all together, the overall algorithm (Algorithm 7) works as follows. After computing an initial tracking controller K, it synthesizes open-loop controllers for different initial states by solving QF-LRA formulas. After each open-loop controller is synthesized, the algorithm identifies the set of initial states for which the combined tracking+open-loop controller is guaranteed to work, and removes this set from Θ. In each new iteration, it picks a new initial state not covered by previous combined controllers, and the process terminates when all of Θ is covered. Our algorithm is sound (Theorem 8.1)—whenever a controller is synthesized, it meets the specifications. Further, for robust systems (defined later in the dissertation), our algorithm is guaranteed to terminate when the system has a combined controller for all initial states (Theorem 8.2).

8.4.2 Synthesizing the Tracking Controller K

Given any open-loop controller u_{ref} and the corresponding reference execution x_{ref}, by replacing in Equation (8.2) the controller of Equation (8.4), we get:

$$x[t+1] = (A[t] + B[t]K[t])\, x[t] - B[t]K[t]x_{\mathsf{ref}}[t] + B[t]u_{\mathsf{ref}}[t] + d[t]. \quad (8.7)$$

Subtracting $x_{\mathsf{ref}}[t+1]$ from both sides, we have that for any execution x starting from the initial states $x[0]$ and with disturbance d, the distance between x and x_{ref} changes with time as:

$$x[t+1] - x_{\mathsf{ref}}[t+1] = (A[t] + B[t]K[t])\,(x[t] - x_{\mathsf{ref}}[t]) + d[t]. \quad (8.8)$$

With $A_c[t] \triangleq A[t] + B[t]K[t]$, $y[t] \triangleq x[t] - x_{\mathsf{ref}}[t]$, Equation (8.8) becomes

$$y[t+1] = A_c[t]y[t] + d[t].$$

We want $x[t]$ to be as close to $x_{\mathsf{ref}}[t]$ as possible, which means $K[t]$ should be designed to make $|y[t]|$ converge. Equivalently, $K[t]$ should be designed as a linear feedback controller such that the system $y[t+1] = A_c[t]y[t]$ is stable. Such a matrix $K[t]$ can be computed using classical control theoretic methods. In this work, we compute $K[t]$ as finding a linear state feedback controller by solving the LQR problem [146], stated as follows.

Definition 8.2 (LQR)**.** For a time-varying linear system $\mathcal{A}$ as defined in Section 8.3 with 0 disturbance and a time bound T, the linear quadratic regulator (LQR) problem is the optimal control problem of finding open loop control $u[0], \cdots, u[T-1]$, such that the following objective function is minimized:

$$J(x[0], u, T) \triangleq x[T]^\top Q[T]x[T] + \textstyle\sum_{t=0}^{T-1}(x[t]^\top Q[t]x[t] + u[t]^\top R[t]u[t]),$$

where Q and R are sequences of symmetric positive definite matrices.

The optimal control for LQR is given by $\forall t = 0, \cdots, T-1, u[t] = K[t]x[t]$ where

$$K[t] \triangleq -\left(B[t]^\top P[t+1]B[t] + R[t]\right)^{-1} B[t]^\top P[t+1]A[t], \tag{8.9}$$

and $P[t]$ is computed by solving the discrete time Riccati difference equation:

$$\begin{aligned}&P[t] = A[t]^\top P[t+1]A[t] + Q[t] - A[t]^\top P[t+1]B[t]\\ &(B[t]^\top P[t+1]B[t] + R[t])^{-1}B[t]^\top P[t+1]A[t]\end{aligned}$$

with boundary condition $P[T] = Q[T]$ [185]. The matrices K in Equation (8.9) can be used as a tracking controller as in Definition 8.1.

When $T \to \infty$ and $\forall t \geq 0, A[t] = A, B[t] = B, Q[t] = Q, andR[t] = R$ are all constant matrices, and $K[t]$ computed using Equation (8.9) will also become a constant matrix K. Furthermore, if the pair (A, B) is controllable (or stabilizable), the closed-loop system $x[t+1] = (A + BK)x[t]$ is stable. That is, the eigenvalues of $A_c = A+BK$ with K given by Equation (8.9) have magnitudes less than 1. Therefore, when $T \to \infty$, the tracking controller K computed using LQR can guarantee that the any execution x will converge to x_{ref} asymptotically when there is no disturbance.

For most of the experiments presented in Section 8.5, we fix each $Q[t]$ and $R[t]$ to be identity matrices. Roughly, for a given R, scaling up Q results in a K that makes an execution x converge faster to the reference execution x_{ref}. In this dissertation, the detailed tradeoffs involved in the choices of $Q[t]$ and $R[t]$ will not be pursued further.

With the synthesized K, we are able to compute the set of reachable states for $\mathcal{A}$ with an arbitrary reference trajectory x_{ref}, as shown in the following section.

8.4.3 Reachable Set Over-approximation with the Tracking Controller

In this section, we assume that the tracking controller, which is a sequence of matrices K, computed as in Section 8.4.2, will make $A[t] + B[t]K[t]$ invertible for any time t. We do not need $A[t] + B[t]K[t]$ to be stable for the analysis of the rest of the section. However, later on we will see that if K can make the other trajectories x converge to x_{ref}, the set of reachable states will also converge to its center x_{ref}, which is desirable for the overall synthesis algorithm.

Once we fix K, we show that the reachable states of the system $\mathcal{A}$ with an open-loop controller u_{ref} (to be computed in Section 8.4.4) can be over-approximated using a sequence of ellipsoids centered at the corresponding x_{ref} with shapes and radii depending on A, B, K, the initial set, and the disturbances (Lemma 8.1). Moreover, for systems with 0 disturbances (i.e., $D = \{0\}$), Corollary 8.1 shows that the set of reachable states can be computed precisely (i.e., there is no over-approximation error).

Lemma 8.1. Consider a linear system $\mathcal{A} = \langle A, B, \Theta, U, D\rangle$ with a controller defined as in Equation (8.4). Fix

(1) a tracking controller K such that $A[t] + B[t]K[t]$ is invertible for each time t,

(2) an open-loop controller u_{ref} with the corresponding reference execution x_{ref}, and

(3) an ellipsoidal initial set $S = E_{r[0]}(x_{\mathsf{ref}}[0], M[0]) \subseteq \Theta$, where $r[0]$ and $M[0]$ are the radius and shape of the ellipsoid respectively. Then

$$\mathsf{Reach}(S, t) \subseteq E_{r[t]}(x_{\mathsf{ref}}[t], M[t]), \forall\, t \leq T, \tag{8.10}$$

where $M[t] = M[0]\left(\prod_{i=0}^{t-1}(A[i] + B[i]K[i])^{-1}\right), r[t] = r[0] + \sum_{i=0}^{t-1}\delta[i]$, and $\delta[i]$ is chosen such that $\forall i \geq 0, E_{\delta[i]}(0, M[i+1]) \supseteq D$.

Proof. We prove this lemma by induction on t.

Base case: When $t = 0$, from the condition (3) of the Lemma we know that $\mathsf{Reach}(S, 0) = S = E_{r[0]}(x_{\mathsf{ref}}[0], M[0])$.

Induction step: Assume that at time step t we have $\mathsf{Reach}(S, t) \subseteq E_{r[t]}(x_{\mathsf{ref}}[t], M[t])$.

Let $A_c[t] = A[t] + B[t]K[t]$. At time step $t+1$, from Equation (8.8), we have that

$$x[t+1] = x_{\mathsf{ref}}[t+1] + A_c[t](x[t] - x_{\mathsf{ref}}[t]) + d[t].$$

It is easy to check that $\forall x[t] \in \mathsf{Reach}(S,t)$, $x[t] - x_{\mathsf{ref}}[t] \in E_{r[t]}(0, M[t])$. Moreover, since $d[t] \in D$, we have that

$$x[t+1] \in x_{\mathsf{ref}}[t+1] \oplus A_c[t]E_{r[t]}(0, M[t]) \oplus D. \tag{8.11}$$

Recall that $\oplus$ is the addition of all elements of sets, and $A_c[t]E_{r[t]}(0, M[t])$ means multiplying each vector in $E_{r[t]}(0, M[t])$ with $A_c[t]$.

The right-hand side of Equation (8.11) can be computed as follows:

(1) The second item $A_c[t]E_{r[t]}(0, M[t])$, which contains all possible values of $A_c[t](x[t] - x_{\mathsf{ref}}[t])$, can be computed as:

$$A_c[t]E_{r[t]}(0, M[t]) = \{A_c[t]x \mid \|x\|_{M[t]} \leq r[t]\} = \{A_c[t]x \mid \|M[t]x\|_2 \leq r[t]\}.$$

Letting $y = A_c[t]x$, then we have

$$\begin{aligned} A_c[t]E_{r[t]}(0, M[t]) &= \{y \mid \|M[t]A_c^{-1}[t]y\|_2 \leq r[t]\} \\ &= \{y \mid \|y\|_{M[t]A_c[t]^{-1}} \leq r[t]\} = E_{r[t]}(0, M[t+1]). \end{aligned}$$

(2) Then, since $D \subseteq E_{\delta[t]}(0, M[t+1])$, which means $\forall d \in D$, $\|d\|_{M[t+1]} \leq \delta[t]$. Therefore, we have

$$E_{r[t]}(0, M[t+1]) \oplus D = \{x + d \mid \|x\|_{M[t+1]} \leq r[t], \|d\|_{M[t+1]} \leq \delta[t]\}.$$

Using triangular inequality of the $M[t+1]$ norm, we have

$$E_{r[t]}(0, M[t+1]) \oplus D \subseteq \{y \mid \|y\|_{M[t+1]} \leq r[t] + \delta[t]\} = E_{r[t+1]}(0, M[t+1]).$$

(3) Finally, it is easy to see that

$$x_{\mathsf{ref}}[t+1] \oplus E_{r[t+1]}(0, M[t+1]) = E_{r[t+1]}(x_{\mathsf{ref}}[t+1], M[t+1]).$$

Therefore, we have

$$\mathsf{Reach}(S, t+1) \subseteq E_{r[t+1]}(x_{\mathsf{ref}}[t+1], M[t+1]).$$

□

In the above proof, the only over-approximation happened in Step 2, as we over-approximate the disturbance D using an ellipsoid with shape $M[t+1]$. This is because we want to keep reach sets ellipsoidal all the time. It is straightforward to check that if there is no disturbance, i.e. $D = \{0\}$, we do not need to conduct Step 2, and Lemma 8.1 can give us exact reach sets:

Corollary 8.1. Consider a linear system $\mathcal{A} = \langle A, B, \Theta, U, D = \{0\}\rangle$ with a controller defined as in Equation (8.4). Fix

(1) a tracking controller K,

(2) an open-loop controller u_{ref} with the corresponding reference execution x_{ref}, and

(3) an ellipsoidal initial set $S = E_{r[0]}(x_{\mathsf{ref}}[0], M[0]) \subseteq \Theta$, where $r[0]$ and $M[0]$ are the radius and shape of the ellipsoid respectively. Then,

$$\mathsf{Reach}(S, t) = E_{r[t]}(x_{\mathsf{ref}}[t], M[t]), \forall\ t \leq T, \tag{8.12}$$

where $M[t] = M[0]\left(\prod_{i=0}^{t-1}(A[i] + B[i]K[i])^{-1}\right)$.

In Lemma 8.1, $r[0]$ and $M[0]$ can be chosen arbitrarily as long as the corresponding ellipsoid $E_{r[0]}(x_{\mathsf{ref}}[0], M[0])$ contains (or is equals to) the initial set S. It follows that given any sequence of u_{ref} as the open-loop controller, which leads to a corresponding reference trajectory x_{ref}, the reachable states from S $\mathsf{Reach}(S, t)$ can be over-approximated by an ellipsoid centered at $x_{\mathsf{ref}}[t+1]$ with shape $M[t] = M[0]\left(\prod_{i=0}^{t-1}(A[i] + B[i]K[i])^{-1}\right)$ and radius $r[0]$ (when there is no disturbance) or $r[0]$ plus an additive term $\sum_{i=0}^{t-1}\delta[i]$ which accounts for bounded disturbance. Note that the shapes and radii of the ellipsoids are all independent of the open-loop controller u_{ref} and the reference trajectory x_{ref}. This is the key step to decouple the synthesis of the tracking controller K and rest of the parameters in the feedback controller (u_{ref}, $x_{\mathsf{ref}}[0]$). In the next section, we discuss a novel approach to finding the latter two efficiently.

8.4.4 Synthesis of the Open-loop Controller

In this section, we will discuss the synthesis of the open-loop controller u_{ref} and $x_{\mathsf{ref}}[0]$ in $\langle K, x_{\mathsf{ref}}[0], u_{\mathsf{ref}}\rangle$. From the previous section, we know that given an initial set S, a tracking controller K, and an open-loop controller u_{ref}, the reachable set (under any disturbance) at time t is over-approximated by $E_{r[t]}(x_{\mathsf{ref}}[t], M[t])$. Thus, once we fix K, the problem of synthesizing a controller reduces to the problem of synthesizing an appropriate u_{ref} and $x_{\mathsf{ref}}[0]$ such that the reachset over-approximations meet the reach-avoid specification. Indeed, for the rest of the this section, we will assume fixed K.

For synthesizing u_{ref} and $x_{\mathsf{ref}}[0]$, we would like to formalize the problem in terms of constraints that will allow us to use SMT solvers. In the following, we describe the details of how this problem can be formalized as a quantifier-free first-order formula over the theory of reals. We will then lay out specific assumptions and/or simplifications required to reduce the problem to QF-LRA theory, which is implemented efficiently in existing state-of-the-art SMT solvers. Most SMT solvers also provide the functionality of explicit model generation, and the concrete controller values can be read-off from the models generated when the constraints are satisfiable.

Constraints for Synthesizing u_{ref} The u_{ref} synthesis problem can be stated as finding satisfying solutions for the formula ϕ_{synth}, where the initial set of states is $S = B_{r[0]}(x_{\mathsf{ref}}[0])$.

$$\begin{aligned} \phi_{\mathsf{synth}} \triangleq\ & \exists u_{\mathsf{ref}}[0], u_{\mathsf{ref}}[1], \ldots u_{\mathsf{ref}}[T-1], r[0] \\ & \exists x_{\mathsf{ref}}[0], x_{\mathsf{ref}}[1], \ldots x_{\mathsf{ref}}[T], \\ & \phi_{\mathsf{control}}(u_{\mathsf{ref}}) \wedge \phi_{\mathsf{execution}}(u_{\mathsf{ref}}, x_{\mathsf{ref}}) \\ & \wedge \phi_{\mathsf{avoid}}(r[0], u_{\mathsf{ref}}, x_{\mathsf{ref}}) \wedge \phi_{\mathsf{reach}}(r[0], u_{\mathsf{ref}}, x_{\mathsf{ref}}) \end{aligned} \tag{8.13}$$

where ϕ_{control} constrains the space of inputs, $\phi_{\mathsf{execution}}$ states that the sequence x_{ref} is a reference execution following Equation (8.4), ϕ_{avoid} specifies the safety constraint, and ϕ_{reach} specifies that the system reaches G:

$$\begin{aligned}
&\phi_{\mathsf{control}}(u_{\mathsf{ref}}) \triangleq \bigwedge_{t=0}^{T-1} u_{\mathsf{ref}}[t] \oplus \big(K[t] \otimes E_{r[t]}(0, M[t])\big) \subseteq U \\
&\phi_{\mathsf{execution}}(u_{\mathsf{ref}}, x_{\mathsf{ref}}) \triangleq \bigwedge_{t=0}^{T-1} (x_{\mathsf{ref}}[t+1] = A[t]x_{\mathsf{ref}}[t] + B[t]u_{\mathsf{ref}}[t]) \\
&\phi_{\mathsf{avoid}}(r[0], u_{\mathsf{ref}}, x_{\mathsf{ref}}) \triangleq \bigwedge_{t=0}^{T} E_{r[t]}(x_{\mathsf{ref}}[t], M[t]) \cap O[t] = \emptyset \\
&\phi_{\mathsf{reach}}(r[0], u_{\mathsf{ref}}, x_{\mathsf{ref}}) \triangleq E_{r[T]}(x_{\mathsf{ref}}[T], M[T]) \subseteq G.
\end{aligned} \tag{8.14}$$

We make a few remarks about this formulation. First, each of the formulas $\phi_{\mathsf{control}}, \phi_{\mathsf{avoid}}$ and ϕ_{reach} represent sufficient conditions to check for the existence of u_{ref}. Second, the constraints stated above belong to the (decidable) theory of reals. However, $\phi_{\mathsf{control}}, \phi_{\mathsf{avoid}}$ and ϕ_{reach}, and thus ϕ_{synth}, are not quantifier free as they use subset and disjointness checks. This is because for sets S, T expressed as predicates $\varphi_S(\cdot)$ and $\varphi_T(\cdot)$, $S \cap T = \emptyset$ corresponds to the formula $\forall x \cdot \neg(\varphi_S(x) \wedge \varphi_T(x))$ and $S \subseteq T$ (or equivalently $S \cap T^c = \emptyset$) corresponds to the formula $\forall x \cdot \varphi_S(x) \implies \varphi_T(x)$.

Reduction to QF-LRA. The central idea behind eliminating the universal quantification in the disjointness predicates in ϕ_{avoid}, or in the inferred disjointness predicates in ϕ_{reach} and ϕ_{control}, is to check whether an ellipsoid is disjointed or contained in a polytope. Lemmas 8.2 and 8.3 state that the disjointness and containment checks can be done through linear constraints.

Lemma 8.2. For an ellipsoid $E_{r[t]}(x_{\mathsf{ref}}[t], M[t])$ and a polytope $\{x \in \mathbb{R}^k \mid Ax \leq b\}$, if

$$\begin{aligned}
&\bigvee_{i=1}^{k} \big(A^{(i)}x_{\mathsf{ref}}[t] > b(i)\big) \wedge \\
&\Big(\tfrac{A^{(i)}x_{\mathsf{ref}}[t]-b(i)}{\|\tilde{A}^{(i)}\|_2} > r[t] \vee \tfrac{A^{(i)}x_{\mathsf{ref}}[t]-b(i)}{\|\tilde{A}^{(i)}\|_2} < -r[t]\Big)
\end{aligned} \tag{8.15}$$

where $\tilde{A} = AM^{-1}[t]$, then

$$E_{r[t]}(x_{\mathsf{ref}}[t], M[t]) \cap \{x \mid Ax \leq b\} = \emptyset.$$

Proof. Take an affine coordinate transformation $y = M[t]x$ and let $\tilde{x}_{\mathsf{ref}}[t] = M[t]x_{\mathsf{ref}}[t]$. Under the transformed coordinate, the ellipsoid $E_{r[t]}(x_{\mathsf{ref}}[t], M[t])$

becomes a ball:

$$E_{r[t]}(M[t]x_{\mathsf{ref}}[t], I) = B_{r[t]}(\tilde{x}_{\mathsf{ref}}[t]),$$

and the polytope also becomes $\tilde{A}y \leq b$. Affine transformation preserves the disjointness between objects. As long as the ball $B_{r[t]}(\tilde{x}_{\mathsf{ref}}[t])$ is disjointed from the polytope $\tilde{A}y \leq b$, the original ellipsoid and polytope are disjointed.

Consider the ball $B_{r[t]}(\tilde{x}_{\mathsf{ref}}[t])$ in the transformed coordinate, if the center $\tilde{x}_{\mathsf{ref}}[t]$ is outside the polytope $\tilde{A}y \leq b$ and its distance to any surface of the polytope is greater than $r[t]$, then the ball $B_{r[t]}(\tilde{x}_{\mathsf{ref}}[t])$ is not intersecting with any surfaces of the polytope, and therefore is disjointed from the polytope. Equivalently, this means that there exists an $i \leq k$, such that $\tilde{A}^{(i)}\tilde{x}_{\mathsf{ref}}[t] > b(i)$, and the distance from $\tilde{x}_{\mathsf{ref}}[t]$ to any surface, which is a hyperplane $\tilde{A}^{(i)}x = b(i)$, is greater than $r[t]$. Recall that $A^{(i)}$ and $b(i)$ are the i^{th} row of A and b respectively.

The distance from $\tilde{x}_{\mathsf{ref}}[t]$ to a hyperplane $\tilde{A}^{(i)}x = b(i)$ is $\frac{|\tilde{A}^{(i)}\tilde{x}_{\mathsf{ref}}[t]-b(i)|}{\|\tilde{A}^{(i)}\|_2}$. Therefore, the ball $B_{r[t]}(\tilde{x}_{\mathsf{ref}}[t])$ is disjointed from the polytope $\tilde{A}y \leq b$ if the following is true:

$$\bigvee_{i=1}^{k} \left(\tilde{A}^{(i)}\tilde{x}_{\mathsf{ref}}[t] > b(i)\right) \wedge \left(\frac{\tilde{A}^{(i)}\tilde{x}_{\mathsf{ref}}[t]-b(i)}{\|\tilde{A}^{(i)}\|_2} > r[t] \vee \frac{\tilde{A}^{(i)}\tilde{x}_{\mathsf{ref}}[t]-b(i)}{\|\tilde{A}^{(i)}\|_2} < -r[t]\right),$$

which is equivalent to Equation (8.15). □

In Lemma 8.2, to check whether an ellipsoid is disjointed from a polytope (obstacle) with k surfaces using Equation (8.15), the formula contains $3k$ linear inequalities with conjunctions and disjunctions. In [27] the reach set over-approximations are represented using hyper-rectangles. The hyber-rectangle is disjointed from the polytope if there is a surface of the polytope such that the vertices of the hyber-rectangle lie on the other side of the surface. Such a formula has $2^n k$ linear inequalities, where n is the dimensionality of the state space. Compared with the methods used in [27], Lemma 8.2 reduces the number of constraints in ϕ_{avoid} from $2^n k$ to $3k$, which is the key fact that makes the proposed approach scale to systems with large n. We will also see the same improvement in ϕ_{reach} and ϕ_{control}.

Similar to Lemma 8.2, as long as the center of the ball $B_{r[t]}(\tilde{x}_{\mathsf{ref}}[t])$ is inside the polytope $\tilde{A}y \leq b$, and the distances from $\tilde{x}_{\mathsf{ref}}[t]$ to all surfaces of the polytope $\tilde{A}^{(i)}x = b(i)$ are greater than the radius $r[t]$, the ball is entirely contained in the polytope:

Lemma 8.3. For any ellipsoid $E_{r[t]}(x_{\mathsf{ref}}[t], M[t])$ and a polytope $\{x \in \mathbb{R}^k \mid Ax \leq b\}$, if

$$\begin{aligned} &\textstyle\bigwedge_{i=1}^{k} \left(A^{(i)}x_{\mathsf{ref}}[t] \leq b(i)\right) \wedge \\ &\left(\tfrac{A^{(i)}x_{\mathsf{ref}}[t]-b(i)}{\|\tilde{A}^{(i)}\|_2} \geq r[t] \vee \tfrac{A^{(i)}x_{\mathsf{ref}}[t]-b(i)}{\|\tilde{A}^{(i)}\|_2} \leq -r[t]\right) \end{aligned} \tag{8.16}$$

where $\tilde{A} = AM^{-1}[t]$, then

$$E_{r[t]}(x_{\mathsf{ref}}[t], M[t]) \subseteq \{x \mid Ax \leq b\}.$$

With Lemma 8.2 and 8.3, we can rewrite ϕ_{avoid} and ϕ_{reach} in Equation 8.14 as:

$$\begin{aligned} \phi_{\mathsf{avoid}}(r[0], u_{\mathsf{ref}}, x_{\mathsf{ref}}) &\triangleq \bigwedge_{t=0}^{T} \bigwedge_{\{x|Ax\leq b\}\in O[t]} \textstyle\bigvee_{i=1}^{k} \left(A^{(i)}x_{\mathsf{ref}}[t] > b(i)\right) \\ &\wedge \left(\tfrac{A^{(i)}x_{\mathsf{ref}}[t]-b(i)}{\|\tilde{A}^{(i)}\|_2} > r[t] \vee \tfrac{A^{(i)}x_{\mathsf{ref}}[t]-b(i)}{\|\tilde{A}^{(i)}\|_2} < -r[t]\right), \\ \phi_{\mathsf{reach}}(r[0], u_{\mathsf{ref}}, x_{\mathsf{ref}}) &\triangleq \textstyle\bigwedge_{i=1}^{k} \left(A_{G}^{(i)}x_{\mathsf{ref}}[T] \leq b_{G}(i)\right) \\ &\wedge \left(\tfrac{{A_G}^{(i)}x_{\mathsf{ref}}[T]-b_G(i)}{\|\tilde{A_G}^{(i)}\|_2} \geq r[T] \vee \tfrac{{A_G}^{(i)}x_{\mathsf{ref}}[T]-b_G(i)}{\|\tilde{A_G}^{(i)}\|_2} \leq -r[T]\right), \end{aligned} \tag{8.17}$$

where in ϕ_{reach}, the goal set G is represented as an ellipsoid $\{x|A_G x \leq b_G\}$. Once the tracking controller K is fixed, the matrices $\tilde{A}$ (or $\tilde{A}_G$) are constants. Moreover, $r[t] = r[0] + \sum_{i=0}^{t-1} \delta[i]$ and δ are also constants. Therefore, ϕ_{avoid} and ϕ_{reach} are linear expressions of $r[0], u_{\mathsf{ref}}, x_{\mathsf{ref}}$ with disjunctions. In the expression ϕ_{control} of Equation 8.14, $u_{\mathsf{ref}}[t] \oplus \left(K[t] \otimes E_{r[t]}(0, M[t])\right)$ is essentially also an ellipsoid $E_{r[t]}(u_{\mathsf{ref}}[t], M[t]K^{-1}[t])$. Therefore, ϕ_{control} can also be represented as a linear expression of u_{ref} and $r[0]$.

As discussed above, the constraints as in $\phi_{\mathsf{control}}, \phi_{\mathsf{execution}}, \phi_{\mathsf{avoid}}$, and ϕ_{reach} only give rise to linear constraints, do not have the $\forall$ quantification over states, and are sound transformations of ϕ_{synth} into QF-LRA. Moreover, the number of linear inequality constraints in ϕ_{synth} is only linear to the time steps T, the number of obstacles in $\mathcal{O}$ and the number of surfaces in each obstacle and goal set. In Section 8.4.5 we will see that as the reach set is exact when the disturbance is 0 (Corollary 8.1), these checks will also turn out to be sufficient to ensure that if there exists a controller, ϕ_{synth} is satisfiable.

Lemma 8.4. If the formula ϕ_{synth} is satisfiable, then there is a control sequence u_{ref} such that for every $x \in B_{r[0]}(x_{\mathsf{ref}}[0])$ and for every $d \in D^T$, the unique execution x defined by the controller $\langle K, x_{\mathsf{ref}}[0], u_{\mathsf{ref}} \rangle$ and d, starting at x, satisfies $x[T] \in G \wedge \forall t \leq T \cdot x[t] \notin O[t]$.

We remark that a possible alternative for eliminating the $\forall$ quantifier is the use of Farkas' lemma, but this gives rise to nonlinear constraints.[1] Indeed, in our experimental evaluation, we observed the downside of resorting to Farkas' lemma in this problem.

8.4.5 Synthesis Algorithm Putting It All Together

Section 8.4.4 describes how to formalize constraints to generate a control sequence that works for S, which could be a subset of the initial set Θ. The overall synthesis procedure (Algorithm 7), first computes a tracking controller K, then generates open-loop control sequences and reference executions in order to cover the entire set Θ.

The procedure `bloatParams` computes the tracking controller K, based on which it further computes a sequence of shape matrices M and disturbance bounds δ using Lemma 8.1, for the system $\mathcal{A}$ and time bound T with Q, R for the LQR method. Given any reference execution x_{ref} and initial set $B_{r[0]}(x_{\mathsf{ref}}[0])$, the parameters computed by `bloatParams` can be used to over-approximate $\mathsf{Reach}(B_{r[0]}(x_{\mathsf{ref}}[0]), t)$ with the ellipsoid $E_{r[t]}(x_{\mathsf{ref}}[t], M[t])$, where $r[t] = r[0] + \sum_{i=0}^{t-1} \delta[i]$.

The procedure `getConstraints` constructs the logical formula ψ_{synth} (Equation (8.18)). Whenever ψ_{synth} holds, we can find an initial radius $r[0]$ that is above some threshold r^*, a center $x_{\mathsf{ref}}[0]$ in the set $\Theta \setminus$ `cover`, and a control sequence u_{ref}, such that any controlled execution starting from $B_r[0](x_{\mathsf{ref}}[0])$

[1]Farkas' lemma introduces auxiliary variables that get multiplied with existing variables $x_{\mathsf{ref}}[0], \ldots, x_{\mathsf{ref}}[T]$, leading to nonlinear constraints.

Algorithm 7: Algorithm for Synthesizing Combined Controller

```
   input    : $\mathcal{A}, T, O[0], \ldots, O[T], G, Q, R$
   initially: $r^* \leftarrow \mathsf{Rad}(\Theta)$ ;
   $K, M, \delta \leftarrow$ bloatParams($\mathcal{A}, T, Q, R$) ;
   cover $\leftarrow \emptyset$;
   controllers $\leftarrow \emptyset$
1  while $\Theta \not\subseteq$ cover do
2      $\psi_{\mathsf{synth}} \leftarrow$ getConstraints($\mathcal{A}, T, F[0], \ldots, F[T], G, M, \delta, r^*$, cover)
        ;
3      if checkSat($\psi_{synth}$) = SAT then
4          $r, u_{\mathsf{ref}}, x_{\mathsf{ref}} \leftarrow$ model($\psi_{\mathsf{synth}}$) ;
5          cover $\leftarrow$ cover $\cup B_r(x_{\mathsf{ref}}[0])$;
6          controllers $\leftarrow$
             controllers $\cup \{ \, ( \langle K, x_{\mathsf{ref}}[0], u_{\mathsf{ref}} \rangle \, , \; B_r(x_{\mathsf{ref}}[0]) \, ) \, \}$ ;
7      else
8          $r^* \leftarrow r^*/2$ ;
9      end
10 end
11 return controllers ;
```

satisfies the reach-avoid requirements.

$$\psi_{\mathsf{synth}} \triangleq \phi_{\mathsf{synth}} \wedge x_{\mathsf{ref}}[0] \in \Theta \wedge x_{\mathsf{ref}}[0] \notin \texttt{cover} \wedge r[0] > r^* \tag{8.18}$$

Line 3 checks for the satisfiability of ψ_{synth}. If satisfiable, we extract the model generated to get the radius of the initial ball, the control sequence u_{ref} and the reference execution x_{ref} in Line 4. The generated controller $\langle K, x_{\mathsf{ref}}[0], u_{\mathsf{ref}} \rangle$ is guaranteed to work for the ball $B_{r[0]}(x_{\mathsf{ref}}[0])$, which can be marked *covered* by adding it to the set `cover`. In order to keep all the constraints linear, one can further under-approximate $B_{r[0]}(x_{\mathsf{ref}}[0])$ with a hypercube $\{x \in \mathbb{R}^n \mid \wedge_{i=1}^{n} x_{\mathsf{ref}}[0](i) - r[0](i)/\sqrt{n} \leq x \leq x_{\mathsf{ref}}[0](i) + r[0](i)/\sqrt{n}\}$. If ψ_{synth} is unsatisfiable, then we reduce the minimum radius r^* (Line 8) and continue to look for controllers, until we find that $\Theta \subseteq$ `cover`.

The set `controllers` is the set of pairs $(\langle K, x_{\mathsf{ref}}[0], u_{\mathsf{ref}} \rangle, S)$, such that the controller $\langle K, x_{\mathsf{ref}}[0], u_{\mathsf{ref}} \rangle$ drives the set S to meet the desired specification. Each time a new controller is found, it is added to the set `controllers` together with the initial set for which it works (Line 6).

The following theorem asserts the soundness of Algorithm 7, and it follows from Lemmas 8.1 and 8.4.

Theorem 8.1. If Algorithm 7 terminates, then the synthesized controller is correct. That is, (a) for each $x \in \Theta$, there is a $(\langle K, x_{\mathsf{ref}}[0], u_{\mathsf{ref}}\rangle, S) \in$ `controllers`, such that $x \in S$, and (b) for each $(\langle K, x_{\mathsf{ref}}[0], u_{\mathsf{ref}}\rangle, S) \in$ `controllers`, the unique controller $\langle K, x_{\mathsf{ref}}[0], u_{\mathsf{ref}}\rangle$ is such that for every $x \in S$ and for every $d \in D^T$, the unique execution defined by $\langle K, x_{\mathsf{ref}}[0], u_{\mathsf{ref}}\rangle$ and d, starting at x, satisfies the reach-avoid specification.

Algorithm 7 ensures that, upon termination, every $x \in \Theta$ is covered, i.e., one can construct a combined controller that drives x to G while avoiding O. However it may find multiple controllers for a point $x \in \Theta$. This non-determinism can be easily resolved by picking any controller assigned for x. Below, we show that, under certain robustness assumptions on the system $\mathcal{A}$, G and the sets O, and in the absence of disturbance, Algorithm 7 terminates.

Definition 8.3 (Robustly controllable systems)**.** A system $\mathcal{A} = \langle A, B, \Theta, U, D\rangle$ is said to be ε-robustly controllable ($\varepsilon > 0$) with respect to the reach-avoid specification (O, G) and matrices K, if (a) $D = \{0\}$, and (b) for every initial state $\theta \in \Theta$ there is an open loop-controller $u_{\mathsf{ref}} \in U^T$ such that the unique execution starting from θ using the open-loop controller u_{ref} satisfies the reach-avoid specification. Moreover, with the controller $\langle K, \theta, u_{\mathsf{ref}}\rangle$ defined as in Equation (8.4), $\forall x \in B_\varepsilon(\theta)$, the unique trajectory x defined by the controller $\langle K, \theta, u_{\mathsf{ref}}\rangle$ starting from x also satisfies the reach avoid specification.

Theorem 8.2. If $\mathcal{A}$ is an ε-robust controllable system with respect to the reach-avoid specification (O, G), the tracking controller K, and an arbitrarily small $\varepsilon > 0$, then Algorithm 7 terminates.

Proof. As seen in Corollary 8.1, when the system is robust, then (in the absence of any disturbance i.e., $D = \{0\}$), the computed ellipsoids are exact reach sets starting from $B_{r[0]}(x_{\mathsf{ref}}[0])$. Moreover, as r^* approaches 0, $r[0]$ can also approach 0. From Corollary 8.1 we know that $\forall t \geq 0, r[t] = r[0]$, so the radii of the reach sets' ellipsoids all converge to 0. With $r[t] \to 0$, Equation (8.15) and Equation (8.16) in Lemmas 8.2 and 8.3 (therefore Equation (8.17)) also become satisfiable whenever there is a controller. The correctness of Theorem 8.2 then follows from the above observations. $\square$

We remark that an alternative approach to solve the bounded controller synthesis problem is to synthesize an open-loop control sequence u_{ref} for a single initial condition $x_{\mathsf{ref}}[0]$ first, and then find the maximum cover such that there exists a tracking controller K to make every execution starting from the cover also satisfy the reach-avoid specification. However, when implemented this approach, we observed that the synthesized reference trajectory x_{ref} always got very close to the obstacles. Therefore, the maximum initial cover for which this reference trajectory works would be minuscule, and result in a very large number of partitions in the initial set. In contrast, Algorithm 7 asks the SMT solver to search for a reference that works for an initial cover with the size of at least r^* with any disturbance (and r^* is adjusted iteratively), resulting in a much smaller solution space.

8.5 RealSyn Implementation and Evaluation

We have implemented our new synthesis algorithm in a tool called RealSyn, which first appeared in [27]. We implement the proposed new algorithm in this dissertation and compare it with the algorithms used in [27]. RealSyn is written in Python. For solving Equation (8.18) it can interface with any SMT solver through Python APIs. We present experimental results with Yices (version 2.5.4) [186], as Yices outperformed the other solvers in [27].

We use 10 benchmark examples to evaluate the performance of the proposed algorithm in RealSyn on a standard laptop with Intel Core i7 processor, 16GB RAM, running Ubuntu 16.04. The results are reported in Table 8.1. The results are encouraging and demonstrate the effectiveness of our approach and the feasibility of scalable controller synthesis for high dimensional systems and complex reach-avoid specifications.

Comparison with Other Tools. We considered other controller synthesis tools for possible comparison with RealSyn. In summary, CoSyMa [172], Pessoa [173], and SCOTS [155] do not explicitly support discrete-time sytems. LTLMop [174, 175] is designed to analyze robotic systems in the (2-dimensional) Euclidean plane and thus is not suitable for most of our examples. TuLiP [164, 176] comes closest to addressing the same class of problems. TuLip relies on discretization of the state space and a receding horizon approach for synthe-

Table 8.1: Controller synthesis using RealSyn with original and improved synthesis algorithms.

	Model	n	m	CAV Algorithm [27]		Algorithm 7	
				#iter	time(s)	#iter	time(s)
1	1-robot	2	1	7	0.06	7	0.03
2	2-robot	4	2	183	2.26	1	0.04
3	Example 8.1	4	2	1	319.97	1	104
4	1-car dynamic avoid	4	2	12	8.49	12	8.12
5	1-car navigation	4	2	17	6.73	15	1.14
6	2-car navigation	8	4	1	4.07	1	1.86
7	3-car navigation	12	6	1	741.73	1	159.38
8	4-car platoon	8	4	1	0.15	1	0.03
9	8-car platoon	16	8	1	0.62	1	0.10
10	10-car platoon	20	10	1	7.74	1	0.12

sizing controllers for more general GR(1) specifications. However, we found that TuLip succumbs to the state space explosion problem when discretizing the state space, and it did not work on most of our examples. For instance, TuLiP was unable to synthesize a controller for the 2-dimensional system '1-robot' (Table 8.1), and returned `unrealizable`. On the benchmark '2-robot' ($n = 4$), TuLip did not return any answer within 1 hour.

Benchmarks. Our benchmarks are mainly vehicle motion planning examples from [27] with reach-avoid specifications. Benchmarks 1-2 model robots moving on the Euclidean plane, where each robot is a 2-dimensional system and admits a 1-dimensional input. Starting from some initial region on the plane, the robots are required to reach the common goal area within the given time steps, while avoiding certain obstacles. For '2-robot', the robots are also required to maintain a minimum separation. Benchmarks 3-7 are discrete vehicular models adopted from [140]. Each vehicle is a 4-dimensional system with 2-dimensional input. Benchmark 3 is the running example in [27], which describes a mobile robot that needs to accomplish a reach-avoid goal in an apartment. Benchmark 4 describes one *ego* vehicle running on a two-lane road, trying to overtake a vehicle in front of it. The second vehicle serves as the obstacle. Benchmarks 5-7 are similar to Benchmark 2 where the vehicles are required to reach a common goal area while avoiding collision with the obstacles and with each other (inspired by a merge). The velocities and accelerations of the vehicles are also constrained in each of these benchmarks.

Benchmarks 8-10 model multiple vehicles trying to form a platoon by maintaining the safe relative distance between consecutive vehicles. The models are adopted (and discretized) from [145]. Each vehicle is a 2-dimensional system with 1-dimensional input. For the 4-car platoon model, the running times reported in Table 8.1 are much smaller than the time (5 minutes) reported in [145]. This observation aligns with our analysis in Section 8.4.1.

Synthesis Performance. In Table 8.1, columns 'n' and 'm' stand for the dimensions of the state space and input space. For each background solver, '#iter' is the number of iterations Algorithm 7 required to synthesize a controller, and 'time' is the respective running times. All benchmarks are synthesized for a specification with $10 - 20$ steps.

In general, the proposed algorithm improves the performance of RealSyn with the running time 2 to 65 times faster than the original algorithm as in [27]. The only exception is Benchmark 4 where the running time stays almost the same. This is because in Benchmark 4, all obstacles, goal set, and reach set over-approximations in [27] were represented as axis-aligned hyper-rectangles. To check the disjointness and containment of axis-aligned hyper-rectangles, [27] used a much simpler method with $O(n)$ linear inequalities, instead of enumerating all the vertices of the hyper-rectangles, which introduces $O(2^n)$ linear inequalities. Therefore, the improvement of the proposed algorithm in this dissertation on Benchmark 4 is minor.

However, for the rest of the benchmarks where the obstacles are not axis-aligned hyper-rectangles, the proposed new algorithm can reduce the number of linear constraints in the final SAT problem (Equation (8.18)) from $O(2^n)$ to $O(1)$ with respect to the dimensionality of the system. The results in Table 8.1 verify our analysis in Section 8.4.

8.6 Summary

In this chapter, we proposed a novel technique for synthesizing controllers for systems with time-varying discrete-time linear dynamics, operating under bounded disturbances, and for reach-avoid specifications. Our approach relies on generating controllers that combine an open-loop controller with a tracking controller, thereby allowing a decoupled approach for synthesizing each component independently. Experimental evaluation using our tool RealSyn demonstrates the value of the approach when analyzing systems with complex dynamics and specifications.

Chapter 9

Conclusions

Verification and synthesis for safety-critical autonomous systems are important but hard problems. The undecidability results for very simple systems (e.g. rectangular hybrid automaton) have led researchers to focus on methods to compute conservative approximations of the behaviors for various types of systems. However, we are still facing profound challenges like scalability, high computation cost, and the lack of precise mathematical models.

In this dissertation, we took a novel data-driven approach that combined the advantages of simulation data and sensitivity. The former is computationally inexpensive and easily accessible, while the latter can provide coverage guarantees through rigorous analysis. We explored several techniques that advance the data-driven verification and synthesis of different models of autonomous systems. All of the techniques rely on numerical or symbolic executions of the underlying model and on sensitivity analysis of the model. Sensitivity analysis gives probabilistic or worst-case bounds on how much the states or outputs of a system will change, with small changes in the input. For different formalisms of autonomous systems, including discrete transition systems, dynamical systems, and hybrid systems, we showed how to compute the bounds on sensitivity and used them to generalize a single execution of the system to over-approximate all neighboring executions. Such generalized over-approximations can be used in two ways. For data-driven verification, they can be used in a semi-decision procedure to prove safety or invariance of the system. For controller synthesis, the sensitivity can be computed symbolically to simplify the formula of finding a correct-by-construction controller.

We used different mathematical tools to address different challenges that arise in different models. In Chapter 5, we discussed how to use matrix measures to find locally tightest over-approximations of reachable states for nonlinear systems. As a result, we could use fewer simulations to establish safety or find counter-examples for nonlinear hybrid systems. Our approach also opened the gate to reason about sample complexity of the verification problem. In Chapter 6, we overcame the hurdle that complete mathematical models of hybrid systems could be unavailable by developing a new semantic framework that combines the known and unknown parts of hybrid systems. The resulting tool, DryVR, implemented a probabilistic sensitivity analysis algorithm and narrowed the gap between sound and practical verification for control systems. In Chapter 7, we introduced an approximate partial order reduction method to analyze the sensitivity of nondeterministic, infinite state discrete transition systems. It reduced the number of explored executions by a factor of $O(n!)$, for a time horizon of n, compared with exhaustive enumeration. Finally in Chapter 8, we showed that for linear systems to meet reach-avoid specification, we could use a controller that combined an open-loop controller with a tracking controller. Once again, sensitivity analysis of the tracking controller allowed us to show that neighboring trajectories stay close to the reference. Using that, we were able to reduce the synthesis of this type of controller to computationally efficient problems: finding LQR controllers and solving satisfiability problems over quantifier-free linear arithmetic.

Using the above techniques, we were able to implement tools and tackle real-world challenging systems like Toyota powertrain control systems, spacecraft rendezvous control systems, and risk analysis of automatic braking systems. These applications were initial attempts of the the data-driven approach but showed the promise of the technique for bridging rigorous analysis and practical systems. Nevertheless, there are still many relevant problems be to solved.

The techniques proposed in this dissertation could be further explored to solve problems related to large-scale, multi-agent autonomous systems. A summary of possible future work follows.

Synthesize controllers for nonlinear systems Following the concepts in Chapter 8, we could use a combined open-loop and tracking controller to make nonlinear systems meet reach-avoid specifications. The open-loop controller generates a reference trajectory while the tracking controller can force the other trajectories with disturbed inputs to converge to the reference trajectory. For nonlinear systems, it is difficult to decouple the synthesis of these two controllers since one may influence the behavior of the other. One idea is to use Lyapunov functions to generate the tracking controller, then use the Lyapunov level sets to bound the divergence of other trajectories. Then we can explore path-planing techniques to generate the reference trajectory.

Parallelize data-driven verification for online reachability Formal methods are usually performed off-line and ask for specific models and requirements. However, in reality, there are always unforeseen circumstances, so the model and requirements for autonomous systems are always evolving. One future extension is to parallelize the data-driven verification approach to expedite the computation of reachable sets. The data-driven verification technique is naturally parallelizable to offer online reachability analysis: we can generate the representative simulations and perform corresponding sensitivity analysis independently. However, with the representative simulations being generated simultaneously, we need to optimize the trade-off between the quality and efficiency. That is, the algorithm will need to figure out the optimal sampling strategy so the simulations can best represent the reachable states to hit the answer for the verification problem.

Use partial order reduction on nonlinear hybrid systems In Chapter 7, we ignored the continuous evolution of the system inside each mode but only considered the ordering issues brought by discrete actions. One natural extension of the approximate POR work is to extend the method to handle nonlinear hybrid systems. When we take into account continuous dynamical behaviors, we need to analyze trajectories inside and across different modes of the hybrid system, where trajectories are (nonlinear, continuous) functions of the initial states, inputs, and time. Therefore, the notion of approximately commutable action pairs need to be extended to approximately commutable continuous functions. Moreover, because hybrid systems have guards and reset functions, the transition time might be shifted and the enabled transitions might be changed by swapping the order of transitions. In this case, we will have to analyze possible consequences of such time shifting and transition enabling/disabling.

References

[1] P. Koopman and M. Wagner, "Autonomous vehicle safety: An interdisciplinary challenge," *IEEE Intelligent Transportation Systems Magazine*, vol. 9, no. 1, pp. 90–96, 2017.

[2] P. Koopman and F. Fratrik, "How many operational design domains, objects, and events?" in *Workshop on Artificial Intelligence Safety 2019 co-located with the Thirty-Third AAAI Conference on Artificial Intelligence 2019 (AAAI-19), Honolulu, Hawaii, January 27, 2019.*, 2019.

[3] B. Cook, "Formal reasoning about the security of Amazon web services," in *International Conference on Computer Aided Verification.* Springer, 2018, pp. 38–47.

[4] P. S. Duggirala, C. Fan, S. Mitra, and M. Viswanathan, "Meeting a powertrain verification challenge," in *International Conference on Computer Aided Verification.* Springer, 2015, pp. 536–543.

[5] M. Althoff and J. M. Dolan, "Online verification of automated road vehicles using reachability analysis," *IEEE Transactions on Robotics*, vol. 30, no. 4, pp. 903–918, 2014.

[6] H. Kress-Gazit, M. Lahijanian, and V. Raman, "Synthesis for robots: Guarantees and feedback for robot behavior," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, no. 1, 2018.

[7] C. Fan, Y. Meng, U. Maier, E. Bartocci, S. Mitra, and U. Schmid, "Verifying nonlinear analog and mixed-signal circuits with inputs," in *IFAC Conference on Analysis and Design of Hybrid Systems*, 2018.

[8] H. K. Khalil, *Nonlinear Systems.* Prentice Hall, 2002.

[9] D. Liberzon, *Switching in Systems and Control.* Springer, 2012.

[10] P. J. Antsaklis, J. A. Stiver, and M. Lemmon, "Hybrid system modeling and autonomous control systems," in *Hybrid Systems.* Springer, 1992, pp. 366–392.

[11] "Autonomous system," https://www.encyclopediaofmath.org/index.php/Autonomous_system, accessed: 2019-09-30.

[12] P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, *Hybrid Systems II.* Springer, 1995, vol. 999.

[13] T. A. Henzinger, "The theory of hybrid automata," in *Verification of Digital and Hybrid Systems.* Springer, 2000, pp. 265–292.

[14] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, no. 2, pp. 183–235, 1994.

[15] D. M. Gabbay, I. Hodkinson, and M. Reynolds, *Temporal Logic: Mathematical Foundations and Computational Aspects.* Clarendon Press, 1994.

[16] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya, "What's decidable about hybrid automata?" in *ACM Symposium on Theory of Computing.* ACM, 1995, pp. 373–382.

[17] V. Vladimerou, P. Prabhakar, M. Viswanathan, and G. E. Dullerud, "STORMED hybrid systems," in *International Colloquium on Automata, Languages, and Programming*, vol. 5126. Springer, 2008, pp. 136–147.

[18] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler, "SpaceEx: scalable verification of hybrid systems," in *International Conference on Computer Aided Verification*, S. Q. Ganesh Gopalakrishnan, Ed. Springer, 2011.

[19] G. Frehse, "PHAVer: Algorithmic verification of hybrid systems past HyTech." in *International Conference on Hybrid Systems: Computation and Control*, 2005, pp. 258–273.

[20] E. Asarin, O. Bournez, T. Dang, and O. Maler, "Approximate reachability analysis of piecewise-linear dynamical systems," in *Hybrid Systems: Computation and Control*, B. Krogh and N. Lynch, Eds., vol. 1790, 2000, pp. 20–31.

[21] S. Kong, S. Gao, W. Chen, and E. Clarke, "dReach: δ-reachability analysis for hybrid systems," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems.* Springer, 2015, pp. 200–205.

[22] X. Chen, E. Ábrahám, and S. Sankaranarayanan, "Flow*: An analyzer for non-linear hybrid systems," in *International Conference on Computer Aided Verification.* Springer, 2013, pp. 258–263.

[23] M. Althoff and D. Grebenyuk, "Implementation of interval arithmetic in CORA 2016," in *International Workshop on Applied Verification of Continuous and Hybrid Systems*, 2016, pp. 91–105.

[24] C. Fan, J. Kapinski, and X. Jin, "Locally optimal reach set over-approximation for nonlinear systems," in *International Conference on Embedded Software*. ACM Press, 2016, pp. 1–10.

[25] C. Fan, B. Qi, S. Mitra, and M. Viswanathan, "DryVR: Data-driven verification and compositional reasoning for automotive systems," in *International Conference on Computer Aided Verification*, vol. 10426 LNCS. Springer, 2017, pp. 441–461.

[26] C. Fan, Z. Huang, and S. Mitra, "Approximate partial order reduction," in *International Symposium on Formal Methods*. Springer, 2018, pp. 588–607.

[27] C. Fan, U. Mathur, S. Mitra, and M. Viswanathan, "Controller synthesis made real: reach-avoid specifications and linear dynamics," in *International Conference on Computer Aided Verification*. Springer, 2018, pp. 347–366.

[28] C. Fan, B. Qi, S. Mitra, M. Viswanathan, and P. S. Duggirala, "Automatic reachability analysis for nonlinear hybrid models with C2E2," in *International Conference on Computer Aided Verification*. Springer, 2016, pp. 531–538.

[29] P. S. Duggirala, L. Wang, S. Mitra, M. Viswanathan, and C. Muñoz, "Temporal precedence checking for switched models and its application to a parallel landing protocol," in *Formal Methods*. Springer, 2014, pp. 215–229.

[30] N. Chan and S. Mitra, "Verified hybrid LQ control for autonomous spacecraft rendezvous," in *Decision and Control (CDC), 2017 IEEE 56th Annual Conference on*. IEEE, 2017, pp. 1427–1432.

[31] N. Chan and S. Mitra, "Verifying safety of an autonomous spacecraft rendezvous mission," in *International Workshop on Applied Verification of Continuous and Hybrid Systems*, 2017, pp. 20–32.

[32] Z. Huang, C. Fan, A. Mereacre, S. Mitra, and M. Kwiatkowska, "Simulation-based verification of cardiac pacemakers with guaranteed coverage," *IEEE Design and Test*, vol. 32, no. 5, pp. 27–34, 2015.

[33] Z. Huang, "Compositional analysis of networked cyber-physical systems: safety and privacy," Ph.D. dissertation, University of Illinois at Urbana-Champaign, 2016.

[34] B. Qi, C. Fan, M. Jiang, and S. Mitra, "DryVR 2.0: A tool for verification and controller synthesis of black-box cyber-physical systems," in *International Conference on Hybrid Systems: Computation and Control.* ACM Press, 2018, pp. 269–270.

[35] M. J. Kearns and U. V. Vazirani, *An Introduction to Computational Learning Theory.* MIT Press, 1994.

[36] C. Fan, B. Qi, and S. Mitra, "Data-driven formal reasoning and their applications in safety analysis of vehicle autonomy features," in *IEEE Design and Test*, vol. 35, 2018, pp. 31–38.

[37] N. Chan and S. Mitra, "Verified hybrid LQ control for autonomous spacecraft rendezvous," in *Decision and Control (CDC), 2017 IEEE 56th Annual Conference on.* IEEE, 2017, pp. 1427–1432.

[38] J. P. Hespanha, *Linear Systems Theory.* Princeton University Press, 2009.

[39] P. J. Antsaklis and A. N. Michel, *A Linear Systems Primer.* Birkhäuser Boston, 2007.

[40] J. Ding and C. J. Tomlin, "Robust reach-avoid controller synthesis for switched nonlinear systems," in *Decision and Control (CDC), 2010 IEEE 49th Annual Conference on*, 2010, pp. 6481–6486.

[41] Z. Huang, Y. Wang, S. Mitra, G. E. Dullerud, and S. Chaudhuri, "Controller synthesis with inductive proofs for piecewise linear systems: An smt-based algorithm," in *Decision and Control (CDC), 2015 IEEE 54th Annual Conference on*, 2015, pp. 7434–7439.

[42] J. F. Fisac, M. Chen, C. J. Tomlin, and S. S. Sastry, "Reach-avoid problems with time-varying dynamics, targets and constraints," in *International Conference on Hybrid Systems: Computation and Control*, 2015, pp. 11–20.

[43] P. M. Esfahani, D. Chatterjee, and J. Lygeros, "The stochastic reach-avoid problem and set characterization for diffusions," *Automatica*, vol. 70, pp. 43–56, 2016.

[44] Z. Huang, C. Fan, A. Mereacre, S. Mitra, and M. Kwiatkowska, "Invariant verification of nonlinear hybrid automata networks of cardiac cells," in *International Conference on Computer Aided Verification*, vol. 8559 LNCS. Springer, 2014, pp. 373–390.

[45] Z. Huang, C. Fan, A. Mereacre, S. Mitra, and M. Z. Kwiatkowska, "Invariant verification of nonlinear hybrid automata networks of cardiac cells," in *International Conference on Computer Aided Verification.* Springer, 2014, pp. 373–390.

[46] Z. Jiang, M. Pajic, S. Moarref, R. Alur, and R. Mangharam, "Modeling and verification of a dual chamber implantable pacemaker," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems.* Springer, 2012, pp. 188–203.

[47] A. El-Guindy, D. Han, and M. Althoff, "Formal analysis of drum-boiler units to maximize the load-following capabilities of power plants," *IEEE Transactions on Power Systems*, vol. PP, no. 99, pp. 1–12, 2016.

[48] G. H. Golub and C. F. Van Loan, *Matrix Computations (3rd Ed.).* Baltimore, MD, USA: Johns Hopkins University Press, 1996.

[49] J. Maidens and M. Arcak, "Reachability analysis of nonlinear systems using matrix measures," *IEEE Transactions on Automatic Control*, vol. 60, no. 1, pp. 265–270, 2015.

[50] C. A. Desoer and M. Vidyasagar, *Feedback Systems: Input-Output Properties.* SIAM, 1975.

[51] E. D. Sontag, "Contractive systems with inputs," in *Perspectives in Mathematical System Theory, Control, and Signal Processing.* Springer, 2010, pp. 217–228.

[52] R. Alur, R. Grosu, Y. Hur, V. Kumar, and I. Lee, "Modular specification of hybrid systems in CHARON," in *International Workshop on Hybrid Systems: Computation and Control.* Springer, 2000, pp. 6–19.

[53] R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho, "Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems," in *Hybrid Systems.* Springer, 1992, pp. 209–229.

[54] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine, "The algorithmic analysis of hybrid systems," *Theoretical Computer Science*, vol. 138, no. 1, pp. 3–34, 1995.

[55] N. Lynch, R. Segala, F. Vaandrager, and H. B. Weinberg, "Hybrid I/O automata," in *International Hybrid Systems Workshop.* Springer, 1995, pp. 496–510.

[56] M. S. Branicky, V. S. Borkar, and S. K. Mitter, "A unified framework for hybrid control: Model and optimal control theory," *IEEE Transactions on Automatic Control*, vol. 43, no. 1, pp. 31–45, 1998.

[57] S. Mitra, "A verification framework for hybrid systems," Ph.D. dissertation, Massachusetts Institute of Technology, 2007.

[58] L. Fang and P. J. Antsaklis, "Information consensus of asynchronous discrete-time multi-agent systems," in *American Control Conference*. IEEE, 2005, pp. 1883–1888.

[59] Z. Huang, S. Mitra, and G. Dullerud, "Differentially private iterative synchronous consensus," in *ACM Workshop on Privacy in the Electronic Society*. ACM, 2012, pp. 81–90.

[60] D. K. Kaynar, N. Lynch, R. Segala, and F. Vaandrager, "The theory of timed I/O automata," *Synthesis Lectures on Distributed Computing Theory*, vol. 1, no. 1, pp. 1–137, 2010.

[61] C. Fan and S. Mitra, "Bounded verification with on-the-fly discrepancy computation," in *International Symposium on Automated Technology for Verification and Analysis*, vol. 9364. Springer, 2015, pp. 446–463.

[62] M. Krstic, I. Kanellakopoulos, P. V. Kokotovic et al., *Nonlinear and Adaptive Control Design*. Wiley New York, 1995, vol. 222.

[63] S. Mitra, *Verifying Cyberphysical Systems*. MIT Press (to be published).

[64] E. Hainry, "Reachability in linear dynamical systems," in *Logic and Theory of Algorithms*. Springer, 2008, pp. 241–250.

[65] G. Lafferriere, G. J. Pappas, and S. Sastry, "O-minimal hybrid systems," *Mathematics of Control, Signals and Systems*, vol. 13, no. 1, pp. 1–21, 2000.

[66] K. Makino and M. Berz, "Taylor models and other validated functional inclusion methods," *International Journal of Pure and Applied Mathematics*, vol. 4, no. 4, pp. 379–456, 2003.

[67] R. E. Moore, R. B. Kearfott, and M. J. Cloud, *Introduction to Interval Analysis*. SIAM, 2009.

[68] A. Platzer and J.-D. Quesel, "KeYmaera: A hybrid theorem prover for hybrid systems (system description)," in *International Joint Conference on Automated Reasoning*. Springer, 2008, pp. 171–178.

[69] A. Platzer, "Quantified differential dynamic logic for distributed hybrid systems," in *International Workshop on Computer Science Logic*. Springer, 2010, pp. 469–483.

[70] Z. Chaochen, W. Ji, and A. P. Ravn, "A formal description of hybrid systems," in *International Hybrid Systems Workshop*. Springer, 1995, pp. 511–530.

[71] P. W. Kopke, “The theory of rectangular hybrid automata,” Ph.D. dissertation, Cornell University, 1996.

[72] E. Bartocci, F. Corradini, M. R. Di Berardini, E. Entcheva, S. A. Smolka, and R. Grosu, “Modeling and simulation of cardiac tissue using hybrid i/o automata,” *Theoretical Computer Science*, vol. 410, no. 33-34, pp. 3149–3165, 2009.

[73] A. Fehnker, F. Vaandrager, and M. Zhang, “Modeling and verifying a lego car using hybrid I/O automata,” in *International Conference on Quality Software*. IEEE, 2003, pp. 280–289.

[74] S. Mitra, Y. Wang, N. Lynch, and E. Feron, “Safety verification of model helicopter controller using hybrid input/output automata,” in *International Workshop on Hybrid Systems: Computation and Control*. Springer, 2003, pp. 343–358.

[75] C. Jewison and R. S. Erwin, “A spacecraft benchmark problem for hybrid control and estimation,” in *Decision and Control (CDC), 2016 IEEE 55th Conference on*. IEEE, 2016, pp. 3300–3305.

[76] K. L. Man and M. P. Schellekens, “Analysis of a mixed-signal circuit in hybrid process algebra $\mathrm{ACP}^{\mathrm{srt}}_{\mathrm{hs}}$,” *Engineering Letters*, vol. 15, no. 2, 2007.

[77] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi, “HyTech: A model checker for hybrid systems,” in *International Conference on Computer Aided Verification*. Springer, 1997, pp. 460–463.

[78] C. Yan and M. R. Greenstreet, “Verifying an arbiter circuit,” in *Formal Methods in Computer-Aided Design*. IEEE, 2008, pp. 7:1–7:9.

[79] S. Gupta, B. H. Krogh, and R. A. Rutenbar, “Towards formal verification of analog designs,” in *International Conference On Computer Aided Design*, 2004, pp. 210–217.

[80] S. Bak and P. S. Duggirala, “Hylaa: A tool for computing simulation-equivalent reachability for linear systems,” in *International Conference on Hybrid Systems: Computation and Control*. ACM, 2017, pp. 173–178.

[81] S. Schupp, E. Abraham, I. Ben Makhlouf, and S. Kowalewski, “HyPro: A C++ library for state set representations for hybrid systems reachability analysis,” in *NASA Formal Methods Symposium*, vol. 10227. Springer, 2017, pp. 288–294.

[82] T. Dang, A. Donzé, and O. Maler, “Verification of analog and mixed-signal circuits using hybrid system techniques,” in *Formal Methods in Computer-Aided Design*, 2004, pp. 21–36.

[83] A. A. Kurzhanskiy and P. Varaiya, "Ellipsoidal techniques for reachability analysis of discrete-time linear systems," *IEEE Transactions on Automatic Control*, vol. 52, no. 1, pp. 26–38, 2007.

[84] T. Dang, C. Le Guernic, and O. Maler, "Computing reachable states for nonlinear biological models," in *International Conference on Computational Methods in Systems Biology*, vol. 5688. Springer, 2009, pp. 126–141.

[85] M. Fränzle, C. Herde, T. Teige, S. Ratschan, and T. Schubert, "Efficient solving of large non-linear arithmetic constraint systems with complex Boolean structure," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 1, no. 3-4, pp. 209–236, 2007.

[86] F. Immler, "Verified reachability analysis of continuous systems," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems.* Springer, 2015, pp. 37–51.

[87] P. S. Duggirala, S. Mitra, and M. Viswanathan, "Verification of annotated models from executions," in *International Conference on Embedded Software.* IEEE Press, 2013, pp. 26:1–26:10.

[88] P. S. Duggirala, "Dynamic analysis of cyber-physical systems," Ph.D. dissertation, University of Illinois at Urbana-Champaign, 2015.

[89] X. Chen, "Reachability analysis of non-linear hybrid systems using taylor models," Ph.D. dissertation, RWTH Aachen University, 2015.

[90] S. Bak and P. S. Duggirala, "Rigorous simulation-based analysis of linear hybrid systems," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems.* Springer, 2017, pp. 555–572.

[91] S. Bak and P. S. Duggirala, "Simulation-equivalent reachability of large linear systems with inputs," in *International Conference on Computer Aided Verification.* Springer, 2017, pp. 401–420.

[92] S. Bogomolov, M. Forets, G. Frehse, F. Viry, A. Podelski, and C. Schilling, "Reach set approximation through decomposition with low-dimensional sets and high-dimensional matrices," in *International Conference on Hybrid Systems: Computation and Control.* ACM, 2018, pp. 41–50.

[93] G. Frehse, "Reachability of hybrid systems in space-time," in *International Conference on Embedded Software.* IEEE Press, 2015, pp. 41–50.

[94] R. Ray, A. Gurung, B. Das, E. Bartocci, S. Bogomolov, and R. Grosu, "XSpeed: Accelerating reachability analysis on multi-core processors," in *Haifa Verification Conference*, vol. 9434, 2015, pp. 3–18.

[95] A. Donzé and O. Maler, "Systematic simulation using sensitivity analysis," in *International Conference on Hybrid Systems: Computation and Control*. Springer, 2007, pp. 174–189.

[96] A. Girard, G. Pola, and P. Tabuada, "Approximately bisimilar symbolic models for incrementally stable switched systems," *IEEE Transactions on Automatic Control*, vol. 55, no. 1, pp. 116–126, 2010.

[97] A. A. Julius and G. J. Pappas, "Trajectory based verification using local finite-time invariance," in *International Conference on Hybrid Systems: Computation and Control*. Springer, 2009, pp. 223–236.

[98] S. Chaudhuri, S. Gulwani, and R. Lublinerman, "Continuity and robustness of programs," *Communications of the ACM*, vol. 55, no. 8, pp. 107–115, 2012.

[99] R. Samanta, J. V. Deshmukh, and S. Chaudhuri, "Robustness analysis of string transducers," in *Automated Technology for Verification and Analysis*. Springer, 2013, pp. 427–441.

[100] R. Majumdar and I. Saha, "Symbolic robustness analysis," in *IEEE Real-Time Systems Symposium*. IEEE, 2009, pp. 355–363.

[101] D. Angeli, "A Lyapunov approach to incremental stability properties," *IEEE Transactions on Automatic Control*, vol. 47, no. 3, pp. 410–421, 2002.

[102] M. Zamani, G. Pola, M. Mazo, and P. Tabuada, "Symbolic models for nonlinear control systems without stability assumptions," *IEEE Transactions on Automatic Control*, vol. 57, no. 7, pp. 1804–1809, 2012.

[103] V. Boichenko and G. Leonov, "Lyapunov's direct method in estimates of topological entropy," *Journal of Mathematical Sciences*, vol. 91, no. 6, pp. 3370–3379, 1998.

[104] H. Abbas and G. Fainekos, "Linear hybrid system falsification through local search," in *International Symposium on Automated Technology for Verification and Analysis*. Springer, 2011, pp. 503–510.

[105] G. E. Fainekos, S. Sankaranarayanan, K. Ueda, and H. Yazarel, "Verification of automotive control applications using S-TaLiRo," in *American Control Conference*. Citeseer, 2012, pp. 3567–3572.

[106] S. Sankaranarayanan, S. A. Kumar, F. Cameron, B. W. Bequette, G. Fainekos, and D. M. Maahs, "Model-based falsification of an artificial pancreas control system," *ACM SIGBED Review*, vol. 14, no. 2, pp. 24–33, 2017.

[107] G. Fainekos, "Automotive control design bug-finding with the S-TaLiRo tool," in *American Control Conference*. IEEE, 2015, pp. 4096–4096.

[108] N. Nedialkov, "VNODE-LP: Validated solutions for initial value problem for ODEs," McMaster University, Tech. Rep., 2006.

[109] CAPD, "Computer assisted proofs in dynamics," 2002. [Online]. Available: http://www.capd.ii.uj.edu.pl/

[110] W. Lohmiller and J.-J. E. Slotine, "On contraction analysis for nonlinear systems," *Automatica*, vol. 34, no. 6, pp. 683–696, 1998.

[111] Y. B. Pesin, "Characteristic Lyapunov exponents and smooth ergodic theory," *Uspekhi Matematicheskikh Nauk*, vol. 32, no. 4, pp. 55–112, 1977.

[112] A. Donzé, "Breach, a toolbox for verification and parameter synthesis of hybrid systems," in *International Conference on Computer Aided Verification*. Springer, 2010, pp. 167–170.

[113] S. Gao, J. Avigad, and E. M. Clarke, "Delta-decidability over the reals," in *IEEE/ACM Symposium on Logic in Computer Science*. IEEE Computer Society, 2012, pp. 305–314.

[114] E. M. Aylward, P. A. Parrilo, and J.-J. E. Slotine, "Stability and robustness analysis of nonlinear systems via contraction metrics and sos programming," *Automatica*, vol. 44, no. 8, pp. 2163–2170, 2008.

[115] A. Balkan, J. V. Deshmukh, J. Kapinski, and P. Tabuada, "Simulation-guided contraction analysis," in *Indian Control Conference*, 2015, pp. 71–75.

[116] S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan, *Linear Matrix Inequalities in System and Control Theory*, ser. Studies in Applied Mathematics. Philadelphia, PA: SIAM, 1994, vol. 15.

[117] R. H. Tütüncü, K. C. Toh, and M. J. Todd, "Solving semidefinite-quadratic-linear programs using SDPT3," *Mathematical Programming*, vol. 95, no. 2, pp. 189–217, 2003.

[118] J. Löfberg, "YALMIP: A toolbox for modeling and optimization in MATLAB," in *International Symposium on Computer-Aided Control System Design*, 2004.

[119] D. Angeli, E. D. Sontag, and Y. Wang, "A characterization of integral input-to-state stability," *IEEE Transactions on Automatic Control*, vol. 45, no. 6, pp. 1082–1097, 2000.

[120] J. Kapinski, J. V. Deshmukh, S. Sankaranarayanan, and N. Aréchiga, "Simulation-guided Lyapunov analysis for hybrid dynamical systems," in *International Conference on Hybrid Systems: Computation and Control*. ACM, 2014, pp. 133–142.

[121] R. Testylier and T. Dang, "NLTOOLBOX: A library for reachability computation of nonlinear dynamical systems," in *International Symposium on Automated Technology for Verification and Analysis*. Springer, 2013, pp. 469–473.

[122] J. Anderson and A. Papachristodoulou, "Dynamical system decomposition for efficient, sparse analysis," in *Decision and Control (CDC), 2010 IEEE 49th Annual Conference on*. IEEE, 2010, pp. 6565–6570.

[123] X. Jin, J. V. Deshmukh, J. Kapinski, K. Ueda, and K. Butts, "Powertrain control verification benchmark," in *International Conference on Hybrid Systems: Computation and Control*. New York, NY, USA: ACM, 2014, pp. 253–262.

[124] T. T. Johnson, J. Green, S. Mitra, R. Dudley, and R. S. Erwin, "Satellite rendezvous and conjunction avoidance: Case studies in verification of nonlinear hybrid systems," in *International Symposium on Formal Methods*. Springer, 2012, pp. 252–266.

[125] Mathworks, "Modeling an Automatic Transmission and Controller," http://www.mathworks.com/videos/modeling-an-automatic-transmission-and-controller-68823.html.

[126] "Road vehicles — Functional safety," International Organization for Standardization (ISO), Geneva, Switzerland, Standard, Nov. 2011.

[127] K. Kodaka, M. Otabe, Y. Urai, and H. Koike, "Rear-end collision velocity reduction system," SAE Technical Paper, Tech. Rep., 2003.

[128] S. Fabris, "Method for hazard severity assessment for the case of undemanded deceleration," *TRW Automotive, Berlin*, 2012.

[129] J. Piao and M. McDonald, "Low speed car following behaviour from floating vehicle data," in *Intelligent Vehicles Symposium*, June 2003, pp. 462–467.

[130] B. P. Malladi, R. G. Sanfelice, E. Butcher, and J. Wang, "Robust hybrid supervisory control for rendezvous and docking of a spacecraft," in *Decision and Control (CDC), 2016 IEEE 55th Conference on*. IEEE, 2016, pp. 3325–3330.

[131] Y. Annpureddy, C. Liu, G. Fainekos, and S. Sankaranarayanan, "S-TaLiRo: A tool for temporal logic falsification for hybrid systems," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems.* Springer, 2011.

[132] J. Ouaknine and J. Worrell, "On the language inclusion problem for timed automata: Closing a decidability gap," in *IEEE Symposium on Logic in Computer Science.* IEEE, 2004, pp. 54–63.

[133] K. Čerāns, "Decidability of bisimulation equivalences for parallel timer processes," in *International Conference on Computer Aided Verification.* Springer, 1992, pp. 302–315.

[134] E. M. Clarke, O. Grumberg, and D. Peled, *Model Checking.* MIT Press, 1999.

[135] R. Alur, R. K. Brayton, T. A. Henzinger, S. Qadeer, and S. K. Rajamani, "Partial-order reduction in symbolic state space exploration," in *International Conference on Computer Aided Verification.* Springer, 1997, pp. 340–351.

[136] Y. Yang, X. Chen, G. Gopalakrishnan, and R. M. Kirby, "Efficient stateful dynamic partial order reduction," in *International SPIN Workshop on Model Checking of Software.* Springer, 2008, pp. 288–305.

[137] P. Abdulla, S. Aronis, B. Jonsson, and K. Sagonas, "Optimal dynamic partial order reduction," in *ACM SIGPLAN Notices*, vol. 49, no. 1. ACM, 2014, pp. 373–384.

[138] C. Flanagan and P. Godefroid, "Dynamic partial-order reduction for model checking software," in *ACM Sigplan Notices*, vol. 40, no. 1. ACM, 2005, pp. 110–121.

[139] C. Baier, J.-P. Katoen, and K. G. Larsen, *Principles of Model Checking.* MIT Press, 2008.

[140] A. Fehnker and F. Ivančić, "Benchmarks for hybrid systems verification," in *International Workshop on Hybrid Systems: Computation and Control.* Springer, 2004, pp. 326–341.

[141] M. Vitus, V. Pradeep, G. Hoffmann, S. Waslander, and C. Tomlin, "Tunnel-milp: Path planning with sequential convex polytopes," in *AIAA Guidance, Navigation and Control Conference and Exhibit*, 2008, p. 7132.

[142] V. Raman, A. Donzé, D. Sadigh, R. M. Murray, and S. A. Seshia, "Reactive synthesis from signal temporal logic specifications," in *International Conference on Hybrid Systems: Computation and Control.* ACM, 2015, pp. 239–248.

[143] J. M. Filho, E. Lucet, and D. Filliat, "Real-time distributed receding horizon motion planning and control for mobile multi-robot dynamic systems," in *International Conference on Robotics and Automation*. IEEE, 2017, pp. 657–663.

[144] C. Liu, S. Lee, S. Varnhagen, and H. E. Tseng, "Path planning for autonomous vehicles using model predictive control," in *IEEE Intelligent Vehicles Symposium*. IEEE, 2017, pp. 174–179.

[145] B. Schürmann and M. Althoff, "Optimal control of sets of solutions to formally guarantee constraints of disturbed linear systems," in *American Control Conference, (ACC)*, 2017, pp. 2522–2529.

[146] D. Liberzon, *Calculus of Variations and Optimal Control Theory: A Concise Introduction*. Princeton University Press, 2011.

[147] G. E. Fainekos, A. Girard, H. Kress-Gazit, and G. J. Pappas, "Temporal logic motion planning for dynamic robots," *Automatica*, vol. 45, no. 2, pp. 343–352, 2009.

[148] M. A. Rami and F. Tadeo, "Controller synthesis for positive linear systems with bounded controls," *IEEE Transactions on Circuits and Systems*, vol. 54-II, no. 2, pp. 151–155, 2007.

[149] M. Kloetzer and C. Belta, "A fully automated framework for control of linear systems from temporal logic specifications," *IEEE Transactions on Automatic Control*, vol. 53, no. 1, pp. 287–297, 2008.

[150] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon temporal logic planning," *IEEE Transactions on Automatic Control*, vol. 57, no. 11, pp. 2817–2830, 2012.

[151] P. Tabuada and G. J. Pappas, "Linear time logic control of discrete-time linear systems," *IEEE Transactions on Automatic Control*, vol. 51, no. 12, pp. 1862–1877, 2006.

[152] A. Bemporad, F. Borrelli, and M. Morari, "Model predictive control based on linear programming - The explicit solution," *IEEE Transactions on Automatic Control*, vol. 47, no. 12, pp. 1974–1985, 2002.

[153] M. N. Zeilinger, C. N. Jones, and M. Morari, "Real-time suboptimal model predictive control using a combination of explicit MPC and online optimization," *IEEE Transactions on Automatic Control*, vol. 56, no. 7, pp. 1524–1534, 2011.

[154] B. Yordanov, J. Tumova, I. Cerna, J. Barnat, and C. Belta, "Temporal logic control of discrete-time piecewise affine systems," *IEEE Transactions on Automatic Control*, vol. 57, no. 6, pp. 1491–1504, 2012.

[155] M. Rungger and M. Zamani, "Scots: A tool for the synthesis of symbolic controllers," in *International Conference on Hybrid Systems: Computation and Control*. ACM, 2016, pp. 99–104.

[156] P. Tabuada, *Verification and Control of Hybrid Systems - A Symbolic Approach*. Springer, 2009.

[157] J. Liu, N. Ozay, U. Topcu, and R. M. Murray, "Synthesis of reactive switching protocols from temporal logic specifications," *IEEE Transactions on Automatic Control*, vol. 58, no. 7, pp. 1771–1785, 2013.

[158] A. Abate, S. Amin, M. Prandini, J. Lygeros, and S. Sastry, "Computational approaches to reachability analysis of stochastic hybrid systems," in *International Workshop on Hybrid Systems: Computation and Control,*, 2007, pp. 4–17.

[159] R. W. Brockett and D. Liberzon, "Quantized feedback stabilization of linear systems," *IEEE transactions on Automatic Control*, vol. 45, no. 7, pp. 1279–1289, 2000.

[160] D. Liberzon, "Observer-based quantized output feedback control of nonlinear systems," *IFAC Proceedings Volumes*, vol. 41, no. 2, pp. 8039–8043, 2008.

[161] A. Girard, "Controller synthesis for safety and reachability via approximate bisimulation," *Automatica*, vol. 48, no. 5, pp. 947–953, 2012.

[162] A. Abate, I. Bessa, D. Cattaruzza, L. C. Cordeiro, C. David, P. Kesseli, D. Kroening, and E. Polgreen, "Automated formal synthesis of digital controllers for state-space physical plants," in *International Conference on Computer Aided Verification*, 2017, pp. 462–482.

[163] K. Mallik, A.-K. Schmuck, S. Soudjani, and R. Majumdar, "Compositional synthesis of finite-state abstractions," *IEEE Transactions on Automatic Control*, vol. 64, no. 6, pp. 2629–2636, 2018.

[164] T. Wongpiromsarn, U. Topcu, N. Ozay, H. Xu, and R. M. Murray, "Tulip: A software toolbox for receding horizon temporal logic planning," in *International Conference on Hybrid Systems: Computation and Control*. ACM, 2011, pp. 313–314.

[165] E. A. Gol, M. Lazar, and C. Belta, "Language-guided controller synthesis for linear systems," *IEEE Transactions on Automatic Control*, vol. 59, no. 5, pp. 1163–1176, 2014.

[166] P. Bouyer, L. Bozzelli, and F. Chevalier, "Controller synthesis for mtl specifications," in *International Conference on Concurrency Theory*. Springer, 2006, pp. 450–464.

[167] V. Raman, A. Donzé, M. Maasoumy, R. M. Murray, A. Sangiovanni-Vincentelli, and S. A. Seshia, "Model predictive control with signal temporal logic specifications," in *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on.* IEEE, 2014, pp. 81–87.

[168] D. Q. Mayne, "Model predictive control: Recent developments and future promise," in *Automatica*, vol. 50, no. 12. Pergamon, 2014, pp. 2967–2986.

[169] Y. Wang and S. Boyd, "Fast model predictive control using online optimization," *IEEE Transactions on Control Systems Technology*, vol. 18, no. 2, pp. 267–278, 2009.

[170] M. M. G. Ardakani, B. Olofsson, A. Robertsson, and R. Johansson, "Real-time trajectory generation using model predictive control," in *IEEE International Conference on Automation Science and Engineering.* IEEE, 2015, pp. 942–948.

[171] A. Alessio and A. Bemporad, "A survey on explicit model predictive control," in *Nonlinear Model Predictive Control: Towards New Challenging Applications*, vol. 384. Springer, 2009, pp. 345–369.

[172] S. Mouelhi, A. Girard, and G. Gössler, "CoSyMA: A tool for controller synthesis using multi-scale abstractions," in *International Conference on Hybrid Systems: Computation and Control.* ACM, 2013, pp. 83–88.

[173] P. Roy, P. Tabuada, and R. Majumdar, "Pessoa 2.0: A controller synthesis tool for cyber-physical systems," in *International Conference on Hybrid Systems: Computation and Control.* ACM, 2011, pp. 315–316.

[174] K. W. Wong, C. Finucane, and H. Kress-Gazit, "Provably-correct robot control with LTLMoP, OMPL and ROS," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, p. 2073.

[175] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Temporal logic based reactive mission and motion planning," *IEEE Transactions on Robotics*, vol. 25, no. 6, p. 1370–1381, 2009.

[176] I. Filippidis, S. Dathathri, S. C. Livingston, N. Ozay, and R. M. Murray, "Control design for hybrid systems with tulip: The temporal logic planning toolbox," in *IEEE Conference on Control Applications*, 2016, pp. 1030–1041.

[177] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[178] J. J. Kuffner and S. M. LaValle, “RRT-Connect: An efficient approach to single-query path planning,” in *IEEE International Conference on Robotics and Automation*, vol. 2. IEEE, 2000, pp. 995–1001.

[179] L. Janson, E. Schmerling, A. Clark, and M. Pavone, “Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions,” *International Journal of Robotics Research*, vol. 34, no. 7, pp. 883–921, 2015.

[180] H. Ravanbakhsh and S. Sankaranarayanan, “Robust controller synthesis of switched systems using counterexample guided framework,” in *International Conference on Embedded Software*. ACM, 2016, pp. 8:1–8:10.

[181] T. Koo, G. J. Pappas, and S. Sastry, “Mode switching synthesis for reachability specifications,” in *International Workshop on Hybrid Systems: Computation and Control*, 2001, pp. 333–346.

[182] A. Taly, S. Gulwani, and A. Tiwari, “Synthesizing switching logic using constraint solving,” *International Journal on Software Tools for Technology Transfer*, vol. 13, no. 6, pp. 519–535, 2011.

[183] S. Jha, S. A. Seshia, and A. Tiwari, “Synthesis of optimal switching logic for hybrid systems,” in *International Conference on Embedded Software*, 2011, pp. 107–116.

[184] H. Zhao, N. Zhan, and D. Kapur, “Synthesizing switching controllers for hybrid systems by generating invariants,” in *Theories of Programming and Formal Methods - Essays Dedicated to Jifeng He on the Occasion of His 70th Birthday*, 2013, pp. 354–373.

[185] S. Bittanti, A. J. Laub, and J. C. Willems, *The Riccati Equation*. Springer Science & Business Media, 2012.

[186] B. Dutertre, “Yices 2.2,” in *International Conference on Computer Aided Verification*, vol. 8559. Springer, 2014, pp. 737–744.