RWTH Aachen

Department of Computer Science
Technical Report

Reachability Analysis of Non-Linear Hybrid Systems Using Taylor Models

Xin Chen

ISSN 0935–3232 · Aachener Informatik-Berichte · AIB-2015-09

RWTH Aachen · Department of Computer Science · April 2015

Reachability Analysis of Non-Linear Hybrid Systems Using Taylor Models

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der RWTH Aachen University zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von

M. Sc.
Xin Chen
aus
Shanghai, China

Berichter: Prof. Dr. Erika Ábrahám
Prof. Dr. Sriram Sankaranarayanan

Tag der mündlichen Prüfung: 24. März 2015

Xin Chen
Theory of Hybrid Systems
xin.chen@informatik.rwth-aachen.de

Aachener Informatik Bericht AIB-2015-09

Herausgeber: Fachgruppe Informatik
RWTH Aachen University
Ahornstr. 55
52074 Aachen
GERMANY

ISSN 0935-3232

Abstract

With the ubiquitous use of computers in controlling physical systems, it requires to have a new formalism that could model both continuous flows and discrete jumps. *Hybrid systems* are introduced to this purpose. A hybrid system, which is modeled by a *hybrid automaton* in the thesis, is equipped with finitely many discrete modes and continuous real-valued variables. A state of it is then represented by a mode along with a valuation of the variables. Given that the system is in a mode ℓ, the variable values are changed continuously according to the Ordinary Differential Equation (ODE) associated to ℓ, or discretely by a jump starting from ℓ. The thesis focuses on the techniques to compute all reachable states over a bounded time horizon and finitely many jumps for a hybrid system with non-linear dynamics. The results of that can then be used in safety verification of the system.

Although a great amount of work has been devoted to the reachability analysis of hybrid systems with linear dynamics, there are few effective approaches proposed for the non-linear case which is very often in applications. The difficulty is twofold. Firstly, it is not easy to find an over-approximation with acceptable accuracy for a set of the solutions of a non-linear ODE. Secondly, to detect and compute the reachable states under a jump requires solving non-linear real arithmetic problems which is also difficult in general. In the thesis, we present our approaches to deal with the above difficulties. For the first one, we present the use of Taylor models as the over-approximate representations for non-linear ODE solutions. Our work can be viewed as a variant of the Taylor model method proposed by Berz et al., such that we are able to efficiently deal with some examples with more than 10 variables. Besides, we also extend the work of Lin and Stadtherr to handle the ODEs with bounded time-varying parameters. For the second difficulty, we present two techniques: (a) domain contraction and (b) range over-approximation to compute an enclosure for the reachable set from which a jump is enabled. They can be seen as Satisfiability Modulo Theories (SMT) solving algorithms which are specialized for the reachability analysis of hybrid systems. In order to reduce the computational cost, we also propose different heuristics for aggregating Taylor models. Besides the above contributions, we describe a method to fast generate Taylor model over-approximations for linear ODE solutions. Its performance is demonstrated via a comparison with the tool SpaceEx.

To make our methods accessible by other people, we implement them in a tool named FLOW*. To examine the effectiveness, we thoroughly compare it with some related tools which are popularly used, according to their functionalities, over a set of non-trivial benchmarks that are collected by us from the areas of mechanics, biology, electronic engineering and medicine. From the experimental results, the advantage of FLOW* over the other tools becomes more apparent when the scale of the system grows. On the other

hand, it also shows that FLOW* can be applied to analyzing realistic systems.

Zusammenfassung

Mit der allgegenwärtigen Verwendung von Computern in der Regelung von physikalischen Systemen entstand der Bedarf für neue Formalismen, die in der Lage sind, sowohl kontinuierliches als auch diskretes Verhalten zu behandeln. *Hybride Systeme* wurden für diesen Zweck eingeführt. Ein hybrides System, welches in dieser Arbeit durch *hybride Automaten* modelliert wird, ist ausgestattet mit endlich vielen diskreten Modi und kontinuierlichen reell-wertigen Variablen. Der Systemzustand ist repräsentiert durch den aktuellen Modus zusammen mit der Evaluierung der Variablen. In einem gegebenen Modus ℓ, ändern sich die Variablenwerte entweder kontinuierlich entsprechend den Differentialgleichungen die ℓ zugeordnet sind, oder aber diskret einer von ℓ ausgehenden Transition folgend. Diese Arbeit konzentriert sich auf Techniken, die es ermöglichen, alle in einem vorgegebenen Zeithorizont mit einer eingeschränkten Anzahl von diskreten Transitionen erreichbaren Zustände eines hybriden Systems mit nicht-linearer Dynamik zu bestimmen.

Obwohl die Erreichbarkeitsanalyse für hybride Systeme mit linearer Dynamik bereits intensiv erforscht wurde, stehen nur wenige Techniken für nicht-lineare Dynamiken, die in Anwendungen sehr häufig vorkommen, zur Verfügung. Solche Techniken zu entwickeln ist aus zwei Gründen schwierig. Erstens ist es nicht einfach eine hinreichend präzise Überapproximation für die Lösungen der nicht-linearen Differentialgleichungen zu finden. Zweitens müssen, um die Erreichbarkeit entlang diskreter Transitionen zu berechnen, schwere nicht-lineare reell-arithmetische Probleme gelöst werden. Wir stellen in dieser Arbeit Ansätze zur Lösung dieser Probleme vor. Um das erste Problem zu lösen, verwenden wir Taylor Modelle, um die Lösungen von nicht-linearen Differentialgleichungen überapproximierend zu repräsentieren. Unser Ansatz kann als eine Variante der Taylor Model Methode von Berz et al. betrachtet werden. Wir sind in der Lage, hinreichend genaue überapproximierende Repräsentierungen für die Lösungsmengen von Beispielen mit mehr als 10 Variablen effizient zu bestimmen. Zusätzlich erweitern wir die Arbeit von Lin und Stadtherr um Differentialgleichungen mit beschränkten zeit-varianten Parametern behandeln zu können. Um die zweite Schwierigkeit zu meistern, stellen wir zwei Techniken vor: (a) Domänen-Verengung (domain contraction) und (b) Wertebereich-Überapproximation (range over-approximation), um erreichbare Zustandsmengen, aus denen eine Transition genommen werden kann, überapproximierend zu beschreiben. Diese Methoden verwenden SAT Modulo Theories (SMT) Algorithmen, die auf die Erreichbarkeitsanalyse von hybriden Systemen angepasst wurden. Um den Berechnungsaufwand zu reduzieren schlagen wir unterschiedliche Methoden für die Zusammenfassung von mehreren Taylor Modellen vor. Zusätzlich stellen wir eine Methode zur schnellen Berechnung von Taylor-Modell-Überapproximationen für Lösungen von linearen Differentialgleichungen vor.

Damit unsere Methoden anderen zugänglich wird, haben wir sie in dem FLOW* Programm implementiert. Um die Wirksamkeit zu untersuchen, vergleichen wir FLOW*

ausführlich zu anderen häufig verwendeten Programmen anhand mehrerer, nicht-trivialer Anwendungen, die wir aus den Bereichen der Mechanik, Biologie, Elektrotechnik und Medizin entnahmen. Aus den experimentellen Ergebnissen werden mit immer größer werdenden Beispielen die Vorteile von Flow* im Vergleich zu anderen Programmen klar ersichtlich. Die Ergebnisse zeigen, dass das Programm auf Beispiele realistischer Größe angewandt werden kann.

(I thank Prof. Erika Ábrahám and Stefan Schupp for the German translation.)

Acknowledgements

The contributions in the thesis are made under the help of many people. First of all, I thank my advisor Erika Ábrahám for leading me to the topic of hybrid systems. From her, I learned the basic knowledge of doing research such as finding and solving problems, reading and writing papers. I am not able to complete my doctoral study without the help from her.

I am very grateful to Sriram Sankaranarayanan who gave me a great help in my research work. He leads me to being interested in many research topics. His encouragement and active involvement gave me a tremendous support in my research life.

I want to give my great thanks to my parents Guohua Chen and Weiwei Jin for their selfless love and support all the time.

I thank Goran Frehse and Oded Maler for the discussion on linear hybrid systems during my visit to Grenoble. Special thanks to Scott Stoller and Stanley Bak for giving me valuable advices to improve Flow*.

I thank Yan Zhang, Aleksandar Chakarov, Aditya Zutshi, Bai Xue, Colas Le Guernic, Matthias Althoff and Ibtissem Ben Makhlouf for many interesting discussions on my research work. I thank Ming Li for inviting me several times to enjoy typical Chinese food, and Hao Wu for helping me many times with German translations.

Last but not least, I also thank the members of the group Theory of Hybrid Systems for sharing fun and happiness during my stay in Aachen.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Hybrid systems are a class of dynamical systems which exhibit both continuous and discrete behaviors. They are natural modeling formalism for the systems composed of a discrete controller interacting with a physical environment. Nowadays, such systems are also named *Cyber-Physical Systems (CPSs)* which are ubiquitous in various areas such as automotive, biology, manufacturing, transportation and so on. Hybrid systems often appear in safety-critical situations, thus it is significant to verify their safety properties. Unfortunately, the safety verification of hybrid systems is notoriously difficult. The existing theories on pure discrete or continuous systems can not be easily extended and applied to dealing with the mixed behaviors, and hence new techniques are required.

The purpose of a safety verification task is to ensure that no system behavior violates the given safety property. To do that, most approaches derive a set which captures all system behaviors and then verify that the safety property is satisfied by the set. A great amount of work has been devoted to developing effective algorithmic approaches to produce an over-approximation set of the behavior of a hybrid system. Most of them combine the existing analysis techniques from computer science and control theory. For example, we may first compute a finite discrete abstraction of a hybrid system and then apply a *model checking* [CGP99] routine to prove the safety. To make the abstraction as small as possible, we may need to use the techniques developed in control theory for studying the system behavior under continuous dynamics.

In this thesis, we focus on proving safety properties of *non-linear* hybrid systems. We want to compute an *over-approximation* for the reachable set of a system. If the over-approximation set does not contain any unsafe state, then we can conclude that the system is safe. Otherwise we try to improve the approximation quality and check the inclusion of an unsafe state again.

1.1 Topic

1.1.1 State of the art

One extensively used methodology for detecting unsafe behaviors is *simulation* [GP06, DM07, Don07]. A simulation method examines finitely many executions of a system and tries to find the ones that violate the safety property. Such a method is not able to handle the case that the system has infinitely many executions unless all executions are guaranteed to be in a computable region around the exemplary ones. Besides, it is also not easy to obtain exact executions for the formal models of hybrid systems, such as hybrid automata. Thus a safety property may not be easily proved or disproved by simulations.

Another way to verify a safety property on a system is *formal verification* and it is our main topic. In a formal verification task, the system is usually defined by a *mathematical model*, and we try to prove that no behavior of the model violates the given property. We usually compute all reachable states of the model, if no unsafe state is included then the system is safe. Such an approach is also called *reachability analysis*. For hybrid systems, widely used mathematical models are *hybrid automata*. Unfortunately, the problem of determining the reachability of a state for a hybrid automaton is not *decidable* [ACH+95], and hence there is no complete algorithm to determine whether a given hybrid automaton is safe or not. However, we may resort to computing an over-approximation for the reachable state set. If the over-approximation does not contain any unsafe state then neither does the exact reachable set and the system is safe. Otherwise, the safety is unknown and we may try to refine the over-approximation.

In the past decades, over-approximate reachability analysis for the hybrid automata with all dynamics defined by linear expressions is intensively studied. Most techniques use a scheme called *flowpipe construction* [Zha92]. That is, given a bounded time horizon $[0, \Delta]$, an over-approximation of the reachable set in $[0, \Delta]$ is iteratively computed as a group of sets $\mathcal{F}_1, \ldots, \mathcal{F}_N$ such that $\mathcal{F}_i$ for each $1 \leq i \leq N$ is an over-approximation of the reachable set over a time segment in $[0, \Delta]$, it is also called a *flowpipe over-approximation*. Convex geometric objects have been recognized as suitable representations of flowpipe over-approximations, since the computations on them can be done based on existing algorithms. It has been shown that convex polyhedra [CK98, SDI08], ellipsoids [KV00], zonotopes [Gir05] and hyper-rectangles [CÁ11, CÁF11] can be successfully used as flowpipe over-approximations in the reachability analysis of linear hybrid automata. Besides, support functions [LG09] which are symbolic representations for general convex sets even provide a good applicability to handle hundreds of state variables.

For the hybrid automata with non-linear dynamics, convex representations however are not suitable for flowpipe over-approximations. Since the exact reachable set at a time point is usually non-convex under a non-linear continuous dynamics, it is better to use non-convex over-approximations. Some proposed representations are orthogonal polyhedron [BMP99, Dan00], interval sets [RN11, ERNF11, Gao12], and Taylor models [BM98, CÁS12]. Besides, the reachability analysis on a non-linear hybrid automaton may also be done by first deriving a linear or even discrete abstraction of the system and then performing the reachability analysis on the abstracted model. The abstraction can be obtained by applying predicate abstraction [ADI03], hybridization [ADG07, ASB08, DLM09, DMT10] or bisimulation abstraction [HTP05]. A common difficulty of those techniques is to control the size of the abstraction.

Other than the above methods, *invariant computation* may also be used to prove safety of hybrid automata. The main idea is widely used in program analysis [CC77, CSS03]. Unlike the computation of flowpipe over-approximations, it tries to derive a system of constraints such that all system states satisfy them. Then if the constraints are inconsistent with the specification of the unsafe set, then the system is safe. Such methods may give answers to the verification problems on which flowpipe construction can not work. Some related work can be found in [SSM04, GT08, Pla10].

All of the methods have advantages and disadvantages in different situations. For example, flowpipe construction does not work well on unbounded time horizons to which invariant computation is often applied, but the latter one generally requires a much higher time cost than the former one in a time-bounded analysis task. For the abstraction methods, although an abstraction model has simpler dynamics than the original system, the size of it is often exponential in the size of the original one. Henceforth, a combinatorial use of those techniques is an interesting topic.

1.1.2 Related tools

The development of the reachability analysis techniques results in the presence of many tools. We summarize some of them in Table 1.1, the techniques used by those tools are also briefly described. Since different tools may accept different hybrid system models, we also list the model name for each tool in the table. We should mention that some of the tools may have more features, but we only concern the ones which are related to reachability analysis. More details could be found in the listed references.

Name	Hybrid system model	Techniques	Ref.
HyTech	hybrid automata	conservative abstraction, polyhedral computation	[HHWT95, HHWT97]
CHARON	composed agents and modes	simulation	[AGH+00, ADE+01]
CheckMate	hybrid automata	convex polyhedron-based flowpipe construction	[CK98, SRKC00]
d/dt	hybrid automata	orthogonal polyhedron-based flowpipe construction	[ADM02, Dan00]
PHAVer	hybrid automata	conservative abstraction, polyhedral computation	[Fre05a, Fre05b]
MATISSE	constrained linear systems	bisimulation abstraction, zonotope computation	[GP05]
HSolver	hybrid automata	conservative abstraction, constraint solving	[RS05]
HYSDEL	discrete hybrid automata	simulation	[Bem04, Kva08]
Level set toolbox	hybrid automata	level set method	[MT00, MT05]
Ellipsoidal Toolbox	hybrid automaton with linear time-varying continuous dynamics	Ellipsoidal calculus	[KV06]
HySAT/iSAT	hybrid automata	interval verified integration, constraint solving	[FHT+07, FH07]
Ariadne	hybrid automata	interval verified integration, constraint solving	[BBC+08]
KeYmaera	hybrid programs	automated theorem proving	[PQ08, Pla10]
SpaceEx	hybrid automata with linear dynamics	support function-based flowpipe construction	[Le 09, FLD+11]
NLTOOLBOX	continuous systems, discrete polynomial systems	hybridization, Bernstein polynomial technique	[ADG07, TD13]
dReach	hybrid automata	interval verified integration, constraint solving	[Gao12, GKC13]

Table 1.1: Tools for reachability analysis of dynamical systems

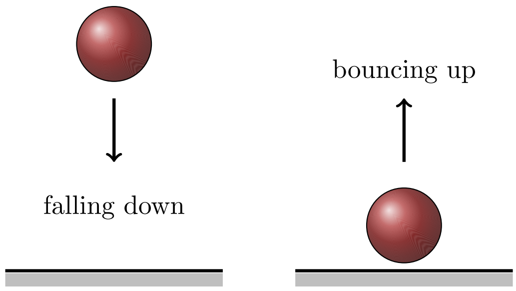


Figure 1.1: Bouncing ball

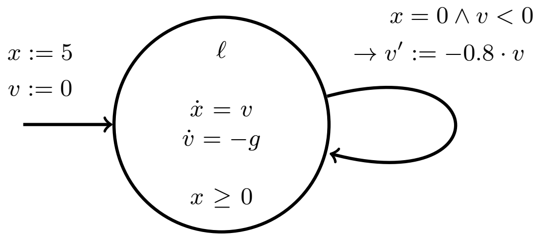


Figure 1.2: Hybird automaton model of a bouncing ball

1.1.3 Models of hybrid systems

The models of hybrid systems under our concern are *hybrid automata*. They are widely used mathematical models for hybrid systems. The formal definition of a hybrid automaton will be given in Chapter 4. Here, we present a simple example.

Example 1.1.1. *A bouncing ball is illustrated in Figure 1.1. We study its vertical motion such that the ball is initially in a position which is* 5*-meter high from the ground and its velocity is zero. Under the influence of gravity, the ball starts to fall. When it hits the ground, the velocity is reversed immediately with some loss in the speed. The ball goes upward until the velocity becomes zero and it then starts to fall again.*

The bouncing ball example can be viewed as a hybrid system equipped with two state variables x and v which represent vertical distance from the ball to the ground and the velocity of the ball, respectively. The velocity v evolves both continuously when the ball is moving in the air and discretely when the ball is bouncing up. We give the hybrid automaton $\mathcal{A}$ which models the bouncing ball system in Figure 1.2. The model has two real-valued variables x, v, a discrete state (or mode, location) ℓ and a discrete jump. The constant g is the gravitational acceleration. A state of $\mathcal{A}$ is a tuple $\langle \ell, \nu \rangle$ such that ℓ denotes the mode name while $\nu : \{x, v\} \to \mathbb{R}$ is a valuation of the variables. The condition $x \geq 0$ is the mode invariant which means that if $\langle \ell, \nu \rangle$ is a valid state then $\nu(x) \geq 0$ must hold. The notation $x = 0 \wedge v < 0 \to v := -0.8 \cdot v$ denotes that if the value of x is zero and v has a negative value then the jump may be executed and after that the value of v is updated by multiplying it with -0.8. Note that the execution of a jump is not mandatory in a hybrid automaton. The leftmost assignment $x := 5$, $v := 0$ denotes the initial valuation of the variables.

1.2 Contributions

The main contributions in the thesis are summarized as follows.

A framework of generating Taylor model flowpipes for continuous systems. It has already been shown in [Ber99, Mak98, BM98, MB03, NJN06, MB09] that Taylor models are powerful flowpipe representations for continuous systems. Even for some chaotic systems, such as Lorentz system and Rössler attractor, their behaviors in a considerably long time can be tightly wrapped by Taylor models. However, Taylor models have a terrible scalability on the number of system variables since the largest size (number of terms) of a fixed-degree polynomial rises heavily with regard to the number of its variables. In Chapter 3, a general framework for computing Taylor model flowpipes is presented. We propose some efficient techniques as well as heuristics which can be easily embedded into the main framework. In order to improve the overall performance, we present the techniques to adaptively change the time step-sizes or Taylor model orders during a flowpipe construction according to a user-specified remainder bound. To deal with the continuous systems with bounded time-varying uncertainties, we describe an approach using Taylor model arithmetic. Furthermore, to optimize the performance on linear continuous systems, we present a flowpipe construction method which combines Taylor model and support function calculus. Based on the experiments, it can be seen that our method is very competitive to the support function methods implemented in SpaceEx.

Taylor model flowpipe construction for hybrid systems. The extension of using Taylor models to generate flowpipe over-approximations for hybrid automata is non-trivial. Based on the techniques for continuous dynamics, we are able to over-approximate the reachable set in a mode. However, new techniques are still required to handle jumps and mode invariants. That is, we need to compute the intersection of a flowpipe over-approximation and a jump guard (or mode invariant). Such a task is not easy in general since the intersection is often not a Taylor model. Hence, we seek to compute an over-approximation of it. To do so, we present two techniques which are named *domain contraction* and *range over-approximation*. In the former one, we try to contract the domain of the Taylor model as much as possible such that the exact intersection is still in it. In the latter one, we over-approximate the Taylor model by a polytope, and then compute the polytope/guard (or polytope/invariant) intersection. An advantage of the former technique is that we do not need additional over-approximation on the result, since it is already a Taylor model. For the latter one, we need to further over-approximate the intersection by a Taylor model, but we can make it entirely lie in the guard (or invariant) set. Moreover, we also show that the two techniques can be used in combination. To reduce the computational cost in a reachability analysis task, we present several methods to aggregate a set of Taylor models. The details will be described in Chapter 4.

Implementation of the tool Flow*. We implement most of our techniques in the tool Flow*. The main functionalities of it is as follows.

(1) Computing Taylor model flowpipes for non-linear hybrid automata. The results form an over-approximation of the states which can be reached within a bounded time

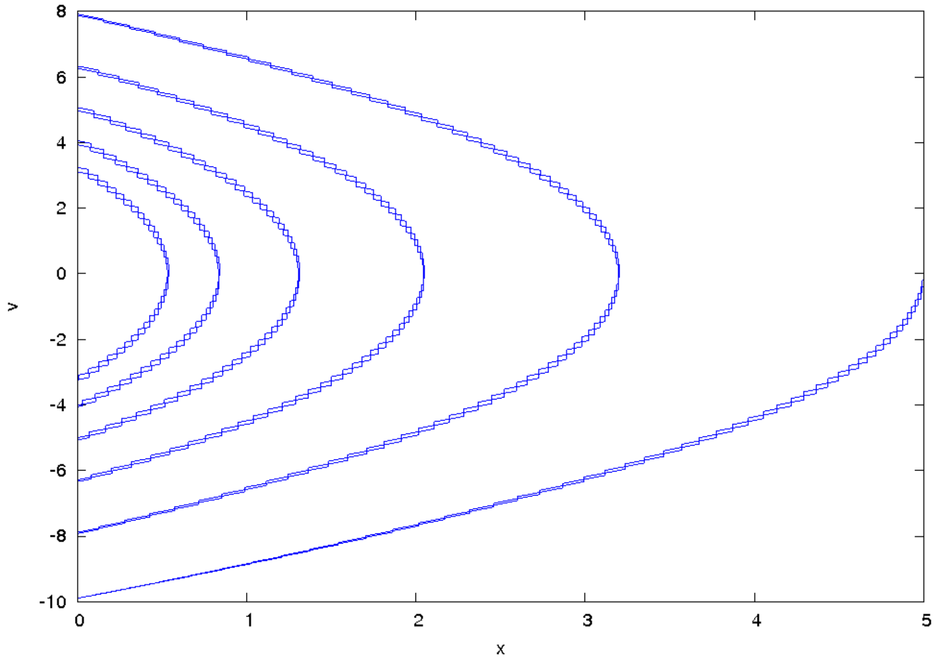


Figure 1.3: Flowpipe over-approximations of the bouncing ball model

horizon and number of jumps.

(2) Verifying a safety property on a given set of Taylor models. An unsafe set is specified by a system of polynomial inequalities in Flow*. To ensure that all Taylor models are safe, we verify the inconsistency of the unsafe set and the Taylor models.

(3) If an unsafe flowpipe is detected, the tool outputs an execution of the automaton which may possibly lead the system to an unsafe set.

The tool consists of the following main components:

(a) a basic arithmetic library for intervals and Taylor models,

(b) a Taylor model integrator for computing the flowpipes under a continuous dynamics, and

(c) a Taylor model library for handling mode invariants, jump guards and proving safety properties.

The basic arithmetic library is implemented based on the GNU Multiple Precision Arithmetic Library (GMP) and the Multiple Precision Floating-Point Reliable Library (MPFR). To ensure the conservativeness during a reachability analysis task, we treat every real number as an interval. Besides, the tool also produces a 2D visualization of Taylor model flowpipes. For the bouncing ball model given in Example 1.1.1, the octagon enclosures of the Taylor model flowpipes in 5 jumps are illustrated in Figure 1.3. Heuristics for improving the overall performance in different situations are also implemented in Flow*. A detailed description is given in Chapter 5.

The content in the thesis covers some work packages in the DFG project HyPro, in which we also plan to collect a set of linear and non-linear hybrid system benchmarks for evaluating existing tools.

1.3 Outline

The thesis is organized as follows. In Chapter 2, we list our notation convention and introduce the basic definitions and theorems of interval and Taylor model arithmetic. A general framework of Taylor model flowpipe construction for continuous systems is presented in Chapter 3. We extend the related work by introducing more efficient techniques and the functionality to handle time-varying parameters. In Chapter 4, the approach to generate Taylor model flowpipes for hybrid automata is discussed in detail. We also show various techniques and heuristics to improve the overall performance. The implementation of FLOW* is described in Chapter 5.

1.4 Personal publications

1.4.1 Publications included in the thesis

The thesis is partially composed of the work in the following publications.

[1] Xin Chen and Erika Ábrahám. *Choice of Directions for the Approximation of Reachable Sets for Hybrid Systems.* In Proceedings of the 13th International Conference on Computer Aided Systems Theory (EUROCAST'11), volume 6927 of Lecture Notes in Computer Science, pages 535-542. Springer, 2011.

Abstract. In this paper we propose an approach to over-approximate the reachable set (with bounded time and number of transitions) of a hybrid system by a finite set of polytopes. The constraints of the polytope are determined by a direction choice method. For the hybrid systems whose (1) continuous dynamics are linear, (2) invariants and guards are defined by linear inequalities, and (3) variable resets are expressed by invertible affine maps, we show that the over-approximations can be computed in polynomial time, and the overestimation can be arbitrarily reduced by decreasing the discretization time step if the continuous dynamics are all deterministic. Some experimental results are also presented to show the effectiveness of our approach.
My contribution: main approach and the experiments.

[2] Xin Chen, Erika Ábrahám, and Goran Frehse. *Efficient bounded reachability computation for rectangular automata.* In Proceedings of the 5th International Workshop on Reachability Problems (RP'11), volume 6945 of Lecture Notes in Computer Science, pages 139-152. Springer, 2011.

Abstract. We present a new approach to compute the reachable set with a bounded number of jumps for a rectangular automaton. The reachable set under a flow transition is computed as a polyhedron which is represented by a conjunction of finitely

many linear constraints. If the bound is viewed as a constant, the computation time is polynomial in the number of variables.
My contribution: main approach and the experiments.

[3] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. *Taylor model flowpipe construction for non-linear hybrid systems.* In Proceedings of the 33rd IEEE Real-Time Systems Symposium (RTSS'12), pages 183-192. IEEE Computer Society, 2012.

Abstract. We propose an approach for verifying non-linear hybrid systems using higher-order Taylor models that are a combination of bounded degree polynomials over the initial conditions and time, bloated by an interval. Taylor models are an effective mean for computing rigorous bounds on the complex time trajectories of non-linear differential equations. As a result, Taylor models have been successfully used to verify properties of non-linear continuous systems. However, the handling of discrete (controller) transitions remains a challenging problem. In this paper, we provide techniques for handling the effect of discrete transitions on Taylor model flowpipe construction. We explore various solutions based on two ideas: domain contraction and range over-approximation. Instead of explicitly computing the intersection of a Taylor model with a guard set, domain contraction makes the domain of a Taylor model smaller by cutting away parts for which the intersection is empty. It is complemented by range over-approximation that translates Taylor models into commonly used representations such as template polyhedra or zonotopes, on which intersections with guard sets have been previously studied. We provide an implementation of the techniques described in the paper and evaluate the various design choices over a set of challenging benchmarks.
My contribution: main approach and the experiments.

[4] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. *Flow*: An analyzer for non-linear hybrid systems.* In Proceedings of the 25th International Conference on Computer Aided Verification (CAV'13), volume 8044 of Lecture Notes in Computer Science, pages 258-263. Springer, 2013.

Abstract. The tool Flow* performs Taylor model-based flowpipe construction for non-linear (polynomial) hybrid systems. Flow* combines well-known Taylor model arithmetic techniques for guaranteed approximations of the continuous dynamics in each mode with a combination of approaches for handling mode invariants and discrete transitions. Flow* supports a wide variety of optimizations including adaptive step sizes, adaptive selection of approximation orders and the heuristic selection of template directions for aggregating flowpipes. This paper describes Flow* and demonstrates its performance on a series of non-linear continuous and hybrid system benchmarks. Our comparisons show that Flow* is competitive with other tools.
My contribution: improvements of the previous techniques, implementation of the tool, and experiments.

[5] Xin Chen, Stefan Schupp, Ibtissem Ben Makhlouf, Erika Ábrahám, Goran Frehse, and Stefan Kowalewski. *A Benchmark Suite for Hybrid Systems Reachability Analysis.* To appear in the 7th NASA Formal Methods Symposium (NFM'15), 2015.

Abstract. Since about two decades, formal methods for continuous and hybrid systems enjoy increasing interest in the research community. A wide range of analysis techniques were developed and implemented in powerful tools. However, the lack of appropriate benchmarks make the testing, evaluation and comparison of those tools difficult. To support these processes and to ease exchange and repeatability, we present a manifold benchmark suite for the reachability analysis of hybrid systems. Detailed model descriptions, classification schemes, and experimental evaluations help to find the right models for a given purpose.
My contribution: collection of the non-linear benchmarks and the experiments on FLOW*.

The following contributions which have not been published are also included in the thesis.

(i) *An algorithm for fast remainder refinement.* The method is described in Section 3.3.3. By a comparison with the original method proposed by Berz et al., our algorithm may dramatically reduce the time cost of a flowpipe construction task.

(ii) *Generating Taylor model flowpipes for the ODEs with time-varying uncertainties.* We present the method in Section 3.5. Although it is similar to the approach proposed by Lin and Stadtherr for treating time-invariant uncertainties, our method can also handle the time-varying case.

(iii) *Fast Taylor model flowpipe generation for linear ODEs.* The standard Taylor model integration method can not efficiently compute flowpipe over-approximations for linear ODEs. To improve it, we present a method that combines Taylor model as well as support function calculus, such that the performance is very competitive to the algorithms implemented in SpaceEx. The details are described in Section 3.6.

(iv) *An efficient vector selection algorithm for determining a template for Taylor model aggregation.* It is often necessary to over-approximate several Taylor models by one to relieve the burden in the subsequent computation. However, it is not easy to keep a good accuracy. In Section 4.4.2, we present an algorithm for determining a template of the over-approximation. For a system with n variables, the method selects n linearly independent vectors from the vector set which is derived from the mode invariants, jump guards and users to form an aggregation template.

1.4.2 Other publications

[6] Sriram Sankaranarayanan, Xin Chen and Erika Ábrahám. *Lyapunov Function Synthesis using Handelman Representations.* In IFAC Symposium on Nonlinear Control Systems (NOLCOS'13). 2013.

Abstract. We investigate linear programming relaxations to synthesize Lyapunov functions that establish the stability of a given system over a bounded region. In particular, we attempt to discover functions that are more readily useful inside symbolic verification tools for proving the correctness of control systems. Our approach

searches for a Lyapunov function, given a parametric form with unknown coefficients, by constructing a system of linear inequality constraints over the unknown parameters. We examine two complementary ideas for the linear programming relaxation, including interval evaluation of the polynomial form and "Handelman representations" for positive polynomials over polyhedral sets. Our approach is implemented as part of a branch-and-relax scheme for discovering Lyapunov functions. We evaluate our approach using a prototype implementation, comparing it with techniques based on Sum-of-Squares (SOS) programming. A comparison with SOSTOOLS is carried out over a set of benchmarks gathered from the related work. The evaluation suggests that our approach using Simplex is generally fast, and discovers Lyapunov functions that are simpler and easy to check. They are suitable for use inside symbolic formal verification tools for reasoning about continuous systems.
My contribution: discussion on the main approach.

[7] Yan Zhang, Sriram Sankaranarayanan, Fabio Somenzi, Xin Chen and Erika Ábrahám. *From Statistical Model Checking to Statistical Model Inference: Characterizing the Effect of Process Variations in Analog Circuits.* In Proceedings of the 32nd IEEE/ACM International Conference on Computer-Aided Design (ICCAD'13), pages 662-669, IEEE/ACM, 2013.

Abstract. This paper studies the effect of parameter variation on the behavior of analog circuits at the transistor (netlist) level. It is well known that variation in key circuit parameters can often adversely impact the correctness and performance of analog circuits during fabrication. An important problem lies in characterizing a safe subset of the parameter space for which the circuit can be guaranteed to satisfy the design specification. Due to the sheer size and complexity of analog circuits, a formal approach to the problem remains out of reach, especially at the transistor level. Therefore, we present a statistical model inference approach that exploits recent advances in statistical verification techniques. Our approach uses extensive circuit simulations to infer polynomials that approximate the behavior of a circuit. A procedure inspired by statistical model checking is then introduced to produce "statistically sound" models that extend the polynomial approximation. The resulting model can be viewed as a statistically guaranteed over-approximation of the circuit behavior. The proposed technique is demonstrated with two case studies in which it identifies subsets of parameters that satisfy the design specifications.
My contribution: discussion on the main approach.

[8] Johanna Nellen, Erika Ábrahám, Xin Chen and Pieter Collins. *Counterexample Generation for Hybrid Automata.* In Proceedings of the 2nd International Workshop on Formal Techniques for Safety-Critical Systems (FTSCS'13), volume 419 of Communications in Computer and Information Science, pages 88-106. Springer, 2013.

Abstract. The last decade brought us a whole range of over-approximative algorithms for the reachability analysis of hybrid automata, a widely used modeling language for systems with combined discrete-continuous behavior. Besides theoretical results, there are also some tools available for proving safety in the continuous time domain. However, if a given set of critical states is found to be reachable, these tools do not provide counterexamples for models beyond timed automata.

This paper investigates the question whether and how available tools can be used to generate counterexamples, even if this functionality is not directly supported. Using the tools SpaceEx and Flow*, we discuss possibilities to solve our task with and without modifying the tools' source code, report on the effort and the efficiency of implementation, and propose a simulation-based approach for the validation of the resulting (possibly spurious) counterexamples.
My contribution: discussion on the main approach and design of the experiments.

[9] Xin Chen, Sriram Sankaranarayanan and Erika Ábrahám. *Under-approximate Flowpipes for Non-linear Continuous Systems.* In Proceedings of the 14th Conference on Formal Methods in Computer-Aided Design (FMCAD'14), pages 59-66. IEEE, 2014.

Abstract. We propose an approach for computing under- as well as over-approximations for the reachable sets of continuous systems which are defined by non-linear Ordinary Differential Equations (ODEs). Given a compact and connected initial set of states, described by a system of polynomial inequalities, we compute under-approximations of the set of states reachable over time. Our approach is based on a simple yet elegant technique to obtain an accurate Taylor model over-approximation for a backward flowmap based on well-known techniques to over-approximate the forward map. Next, we show that this over-approximation can be used to yield both over- and under-approximations for the forward reachable sets. Based on the result, we are able to conclude "may" as well as "must" reachability to prove properties or conclude the existence of counterexamples. A prototype of the approach is implemented and its performance is evaluated over a reasonable number of benchmarks.
My contribution: main approach and the experiments.

[10] Mohamed Amin Ben Sassi, Sriram Sankaranarayanan, Xin Chen, and Erika Ábrahám. *Linear Relaxations of Polynomial Positivity for Polynomial Lyapunov Function Synthesis.* IMA Journal of Mathematical Control and Information. Accepted January 11, 2015.

Abstract. We examine linear programming (LP) based relaxations for synthesizing polynomial Lyapunov functions to prove the stability of polynomial ordinary differential equations (ODEs). Our approach starts from a desired parametric polynomial form of the polynomial Lyapunov function. Subsequently, we encode the positive definiteness of the function, and the negation of its derivative, over the domain of interest. We first compare two classes of relaxations for encoding polynomial positivity: relaxations by sum-of-squares (SOS) programmes, against relaxations based on Handelman representations and Bernstein polynomials, that produce linear programmes. Next, we present a series of increasingly powerful LP relaxations based on expressing the given polynomial in its Bernstein form, as a linear combination of Bernstein polynomials. Subsequently, we show how these LP relaxations can be used to search for Lyapunov functions for polynomial ODEs by formulating LP instances. We compare our techniques with approaches based on SOS on a suite of automatically synthesized benchmarks.
My contribution: discussion on the methods.

Chapter 2

Preliminaries

2.1 Notations

Our notation convention is given as follows.

Sets. We use capital letters such as A, B, etc. to denote *sets* of elements. A set is called a *singleton* if it contains exactly one element. Given a set A, we denote by $\overline{A}$ its *complement set* and by 2^A its *power set* which consists of all subsets of A. For two sets A and B, we denote by $A \cap B$ their *intersection* and by $A \cup B$ their *union*. The *relative complement* of B w.r.t. A is denoted by $A \backslash B$. Besides, we use the following notations to represent sets of numbers. The sets of *reals* and *rationals* are denoted by $\mathbb{R}$ and $\mathbb{Q}$ respectively. We use $\mathbb{Z}$ to denote the set of *integers*, and use $\mathbb{N}$ to denote the set of *natural numbers* which are non-negative integers. In the Euclidean space $\mathbb{R}^n$ for some $n \in \mathbb{N}$ and $n > 0$, we call an element of $\mathbb{R}^n$ a *vector* as well as a *point*.

Vectors and matrices. We denote a *(column) vector* by a tuple $(x_1, \ldots, x_n)$ or a letter with a vector over it, such as $\vec{x}$. *Row vectors* are represented by *transposes* of column vectors, for example $\vec{x}^T$. Given a vector $\vec{x}$, we denote by x_i or $\vec{x}[i]$ the i-th component of $\vec{x}$, and we use $\mathrm{Dim}(\vec{x})$ to denote the *dimension* of $\vec{x}$, that is, the number of components in $\vec{x}$. The symbol $\|\cdot\|$ stands for *Euclidean norm* of vectors. Vectors are also used to collectively represent a set of ordered variables or constants in the thesis. For example, the variables $x_1, \ldots, x_n$ can be denoted by $\vec{x}$. *Matrices* are denoted by capital letters. We occasionally use a matrix to collectively represent a set of column vectors which are columns of the matrix.

Functions. Given a function f, we denote by $\mathtt{Dom}(f)$ the domain and by $\mathtt{Rng}(f)$ the range of f respectively. The dimension of f is the dimension of its values. Given a set D, we use $\mathcal{C}^k(D)$ to denote the set of functions f such that for any $\vec{x} \in D$, all partial derivatives of f at $\vec{x}$ up to order k exists and are continuous. Note that D is not necessarily same as the domain of f but should be a subset of it. Thereby, the symbols $\mathcal{C}^0(D)$ and $\mathcal{C}^\omega(D)$ stand for the sets of *continuous functions* and *analytic functions* respectively. A *vector-valued function* can be defined by a vector of real-valued functions. In the thesis, we do not distinguish real-valued and vector-valued functions unless it is intentionally pointed out.

2.2 Interval arithmetic

2.2.1 Basic definitions and theorems

A *(real) interval* is the set of real numbers which are between two real endpoints $a, b \in \mathbb{R}$ such that $a \leq b$. In the thesis, the four types of intervals are denoted in the following way.

$$\begin{array}{rcl}(a,b) & = & \{c \in \mathbb{R} \,|\, a < c < b\} \\ [a,b) & = & \{c \in \mathbb{R} \,|\, a \leq c < b\} \\ (a,b] & = & \{c \in \mathbb{R} \,|\, a < c \leq b\} \\ [a,b] & = & \{c \in \mathbb{R} \,|\, a \leq c \leq b\}\end{array}$$

The endpoints a, b are also allowed to take the infinities $-\infty$ and $+\infty$ when the bound is strict. We call the intervals with non-strict bounds *closed* intervals, and we denote the set of them by $\mathbb{IR}$. For an interval $I = [a,b] \in \mathbb{IR}$, we call a the *lower bound* and b the *upper bound* of I. Both of them are also called *endpoints*.

The *width*, *midpoint* and *magnitude* of an interval $[a,b] \in \mathbb{IR}$ are defined by

$$\begin{array}{lrcl}\text{Width:} & \mathrm{W}([a,b]) & = & b - a \\ \text{Midpoint:} & \mathrm{Mid}([a,b]) & = & (a+b)/2 \\ \text{Magnitude:} & \mathrm{Mag}([a,b]) & = & \max\{|a|, |b|\}.\end{array}$$

We call an interval I *degenerate* if $\mathrm{W}(I) = 0$, i.e., I is singleton, and *symmetric* if $\mathrm{Mid}(I) = 0$.

Similar to reals, intervals can also be organized as vectors or matrices. Given two interval vectors $\vec{X}$, $\vec{Y}$ with $\mathrm{Dim}(\vec{X}) = \mathrm{Dim}(\vec{Y})$, we use $\vec{X} \subseteq \vec{Y}$ to denote that $\vec{X}[i] \subseteq \vec{Y}[i]$ for all $1 \leq i \leq \mathrm{Dim}(\vec{X})$. The above three operators on an interval vector $\vec{I}$ are defined as follows. The midpoint of an interval vector $\vec{I}$ is also a vector such that

$$\text{Midpoint:} \quad \mathrm{Mid}(\vec{I})[i] \quad = \quad \mathrm{Mid}(\vec{I}[i])$$

for $1 \leq i \leq \mathrm{Dim}(\vec{I})$. The width or magnitude of an interval vector is a real value which is the maximum width or magnitude respectively of the components:

$$\begin{array}{lrcl}\text{Width:} & \mathrm{W}(\vec{I}) & = & \max_{1 \leq i \leq \mathrm{Dim}(\vec{I})}\{\mathrm{W}(\vec{I}[i])\} \\ \text{Magnitude:} & \mathrm{Mag}(\vec{I}) & = & \max_{1 \leq i \leq \mathrm{Dim}(\vec{I})}\{\mathrm{Mag}(\vec{I}[i])\}\end{array}$$

In the rest of the thesis, we also call interval vectors intervals and simply discard the arrow symbol in their notations. For simplicity, we sometimes also denote an interval vector by a Cartesian product, such as $[a,b]^n$. Besides, intervals are also called *boxes* or *grids*.

Given two intervals $X, Y \in \mathbb{IR}$ and a binary operator $\diamond$ over reals, the result $X \diamond Y$ is defined by the set

$$X \diamond Y = \{x \diamond y \,|\, x \in X, y \in Y\} \tag{2.1}$$

which consists of the elements $x \diamond y$ for all $x \in X$ and $y \in Y$. If $\diamond$ defines a continuous mapping from $\mathbb{R}^2$ to $\mathbb{R}$, the set (2.1) is also an interval. Hence, the basic four operators $+, -, \cdot, /$ over reals can be extended to deal with intervals, and the endpoints of the results can be computed from the endpoints of the operands.

$$
\begin{array}{llcll}
\text{Addition:} & [a,b] & + & [c,d] & = [a+c, b+d] \\
\text{Subtraction:} & [a,b] & - & [c,d] & = [a-d, b-c] \\
\text{Multiplication:} & [a,b] & \cdot & [c,d] & = [\min\{a\cdot c, a\cdot d, b\cdot c, b\cdot d\}, \\
 & & & & \quad \max\{a\cdot c, a\cdot d, b\cdot c, b\cdot d\}] \\
\text{Division:} & [a,b] & / & [c,d] & = [a,b]\cdot[1/d, 1/c]
\end{array}
$$

wherein $0 \notin [c,d]$ for division. Similar to real arithmetic, the addition and multiplication operators on intervals are also *commutative* and *associative*. However, multiplication is only *sub-distributive* over addition on intervals. That is, for any intervals $[a_1, b_1]$, $[a_2, b_2]$ and $[a_3, b_3]$, we have the following inclusion

$$[a_1,b_1]\cdot([a_2,b_2]+[a_3,b_3]) \;\subseteq\; ([a_1,b_1]\cdot[a_2,b_2])+([a_1,b_1]\cdot[a_3,b_3])$$

in which the equivalence does not hold generally.

Standard functions such as exponential, logarithm and square root can also be extended to their interval versions:

$$
\begin{array}{llcl}
\text{Exponential:} & \exp([a,b]) & = & [\exp(a), \exp(b)] \\
\text{Logarithm:} & \log_c([a,b]) & = & [\log_c(a), \log_c(b)] \\
\text{Square root:} & \sqrt{[a,b]} & = & [\sqrt{a}, \sqrt{b}]
\end{array}
$$

wherein $c > 1$, $0 < a \le b$ must hold for logarithm, and $0 \le a \le b$ must hold for square root. The n-th power of an interval is also an interval which can be computed by Algorithm 1. Note that in general, the algorithm yields more accurate results than those computed by merely applying interval multiplications.

Given $[a,b] \in \mathbb{IR}$, the interval of $\sin([a,b])$ is computed in the following way. We check the inclusions of the points $(2n-\frac{1}{2})\pi$ and $(2n+\frac{1}{2})\pi$ for $n \in \mathbb{Z}$ in $[a,b]$. If $(2n-\frac{1}{2})\pi \in [a,b]$ holds for some $n \in \mathbb{Z}$, the lower bound of $\sin([a,b])$ is -1. Otherwise the lower bound is $\min\{\sin(a), \sin(b)\}$. On the other hand, if $(2n+\frac{1}{2})\pi \in [a,b]$ holds for some $n \in \mathbb{Z}$, the upper bound of $\sin([a,b])$ is 1. Otherwise the upper bound is $\max\{\sin(a), \sin(b)\}$. We give the computation by Algorithm 2. The interval computations for the other trigonometric functions can be done analogously. Based on the above basic operators, an elementary function may also be extended to handle interval arguments.

Computing the set of values of a function according to a set of inputs is often required in many analysis tasks such as verification. Such a computation can not be done by computing the result for each input, since they are usually infinitely many. Then it is necessary to introduce set-based arithmetic to function evaluations. Given a function f, a united extension F of f, as stated by Definition 2.2.1, always returns the exact value set of f according to the input of F. However, if we restrict the sets to intervals, the interval united extension of a function does not exist in general. Therefore, we resort to a weaker version which is called *interval extension*. It deals with intervals and produces the same results as that of f on the degenerate interval (real) inputs.

Definition 2.2.1 (United extension)**.** *Given a function $f(\vec{x})$ over the domain $D \subseteq \mathbb{R}^n$ for some $n \in \mathbb{N}$. We say that a set-valued function $F(X)$ over $X \in 2^D$ is the* united extension *of f, if*

$$F(C) = \{f(\vec{c}) \mid \vec{c} \in C\} \tag{2.2}$$

for all $C \in 2^D$.

Algorithm 1 The n-th power of an interval

Input: an interval $[a, b]$ and $n \in \mathbb{N}$
Output: the interval of $([a, b])^n$

```
if n is odd then
    c ← a^n;
    d ← b^n;
else
    if a ≥ 0 then
        c ← a^n;
        d ← b^n;
    else if b < 0 then
        c ← b^n;
        d ← a^n;
    else
        c ← 0;
        d ← max{a^n, b^n};
    end if
end if
return [c, d];
```

Algorithm 2 The interval sine function

Input: an interval $[a, b]$
Output: the interval of $\sin([a, b])$

```
if ∃n ∈ ℤ.((2n − 1/2)π ∈ [a, b]) then
    c ← −1;
else
    c ← min(sin(a), sin(b));
end if
if ∃n ∈ ℤ.((2n + 1/2)π ∈ [a, b]) then
    d ← 1;
else
    d ← max{sin(a), sin(b)};
end if
return [c, d];
```

Definition 2.2.2 (Interval extension). *Given a function $f(\vec{x})$ over the domain $D \subseteq \mathbb{R}^n$ for some $n \in \mathbb{N}$. We say that the interval-valued function $F(\vec{X})$ is an* interval extension *of f if for all $\vec{c} \in D$ there is*

$$F(\vec{C}) = [f(\vec{c}), f(\vec{c})] \tag{2.3}$$

wherein $C_i = [c_i, c_i]$ for $1 \leq i \leq n$.

Definition 2.2.3 (Inclusion isotonicity). *We call an interval extension F over the domain $D \subseteq \mathbb{IR}^n$* inclusion isotonic, *if for all $\vec{X}, \vec{Y} \in D$ and $\vec{X} \subseteq \vec{Y}$, there is $F(\vec{X}) \subseteq F(\vec{Y})$.*

By the fundamental theorem of interval analysis, if an interval extension F of a real-valued function f is *inclusion isotonic*, then for any interval input $\vec{C} \subseteq \mathtt{Dom}(f)$, $F(\vec{C})$ is always an interval enclosure (or over-approximation) of the exact function range $\{f(\vec{c}) \,|\, \vec{c} \in \vec{C}\}$.

Theorem 2.2.4 (Fundamental theorem of interval analysis [MKC09]). *If F is an inclusion isotonic interval extension of f, then we have that $\{f(\vec{c}) \,|\, \vec{c} \in \vec{C}\} \subseteq F(\vec{C})$ for all interval inputs $\vec{C} \subseteq \mathtt{Dom}(f)$.*

For a real-valued function f over an interval domain D, in general, it is not easy to find an interval extension F of it such that the set $F(\vec{C})$ for an input interval $\vec{C} \subseteq D$ is always the smallest interval enclosure of $\{f(\vec{c}) \,|\, \vec{c} \in \vec{C}\}$. As stated by Theorem 6.1 in [MKC09], if the extension F is *inclusion isotonic* as well as *Lipschitz* in D, then we may arbitrarily approach the smallest interval enclosure by performing a finer uniform subdivision on the input interval. A detailed description is given as follows.

Definition 2.2.5. *An interval extension $F(\vec{X})$ is* Lipschitz *in D if there exists $L \in \mathbb{R}$ such that $W(F(\vec{C})) \leq L \cdot W(\vec{C})$ for all $\vec{C} \subseteq D$. F is also called an L-*Lipschitz *interval extension of f in D.*

Given an n-dimensional interval $\vec{C} = ([a_1, b_1], \ldots, [a_n, b_n])$ and a number $N \in \mathbb{N}$, the *N-uniform subdivision* of $\vec{C}$ is the set of intervals

$$\vec{C_{\vec{i}}} = ([a_1 + (i_1 - 1) \cdot (\frac{b_1 - a_1}{N}), a_1 + i_1 \cdot (\frac{b_1 - a_1}{N})], \ldots, [a_n + (i_n - 1) \cdot (\frac{b_n - a_n}{N}), a_n + i_n \cdot (\frac{b_n - a_n}{N})])$$

for $1 \leq i_1, \ldots, i_n \leq N$, i.e., we uniformly divide $\vec{C}$ into N^n grids. Then, for an inclusion isotonic and Lipschitz interval extension F of f, a smaller interval enclosure than $F(\vec{C})$ can often be obtained by computing the smallest interval enclosure of $\bigcup\{F(\vec{C_{\vec{i}}}) \,|\, 1 \leq i_1, \ldots, i_n \leq N\}$ for some $N > 1$. Moreover, when $N \to +\infty$, the result converges to the smallest interval enclosure of $\{f(\vec{c}) \,|\, \vec{c} \in \vec{C}\}$.

The above fact also holds when f is a vector-valued function, although the range of it is often not an interval. In the following content, we consider the over-approximation quality of the set $\bigcup\{F(\vec{C_{\vec{i}}}) \,|\, 1 \leq i_1, \ldots, i_n \leq N\}$ instead of its smallest interval enclosure.

Definition 2.2.6 (Hausdorff distance). *Given two* non-empty *sets X, Y, the* Hausdorff distance *between X, Y is given by*

$$d_H(X, Y) = \sup\{\sup_{\vec{x} \in X} \inf_{\vec{y} \in Y} \|\vec{x} - \vec{y}\|, \sup_{\vec{y} \in Y} \inf_{\vec{x} \in X} \|\vec{x} - \vec{y}\|\}.$$

For a non-empty set S and its over-approximation S', the overestimation in S' can be measured by the *Hausdorff distance* between S and S'. If the Hausdorff distance is zero, then the two sets are identical and there is no overestimation. Hausdorff distance has the following properties (see [Edg08]).

- Given non-empty sets X, Y, Z, we have that
$$d_H(X,Y) + d_H(Y,Z) \geq d_H(X,Z).$$
- Given non-empty sets X_1, X_2, Y_1, Y_2, we have that
$$d_H(X_1 \cup X_2, Y_1 \cup Y_2) \leq \sup\{d_H(X_1,Y_1), d_H(X_2,Y_2)\}.$$

When $S' \in \mathbb{IR}^n$ is an interval over-approximation of $S \subseteq \mathbb{R}^n$, the Hausdorff distance $d_H(S', S)$ is at most $\sqrt{n} \cdot \mathrm{W}(S')$, since the Euclidean distance between two points in S' is bounded by $\sqrt{n} \cdot \mathrm{W}(S')$. If the set S is over-approximated by the union of a set of intervals $S_1, \ldots, S_m$ such that $S_i \cap S \neq \emptyset$ for $1 \leq i \leq m$, then the overestimation is given by
$$\sup_{1 \leq i \leq m} \{d_H(S_i, S_i \cap S)\}.$$

Theorem 2.2.7 tells that if an interval extension F is inclusion isotonic and Lipschitz, then we may always have a more accurate over-approximation, which is a set of intervals obtained by first uniformly subdividing the domain into finer grids and then computing the the interval values of F over them.

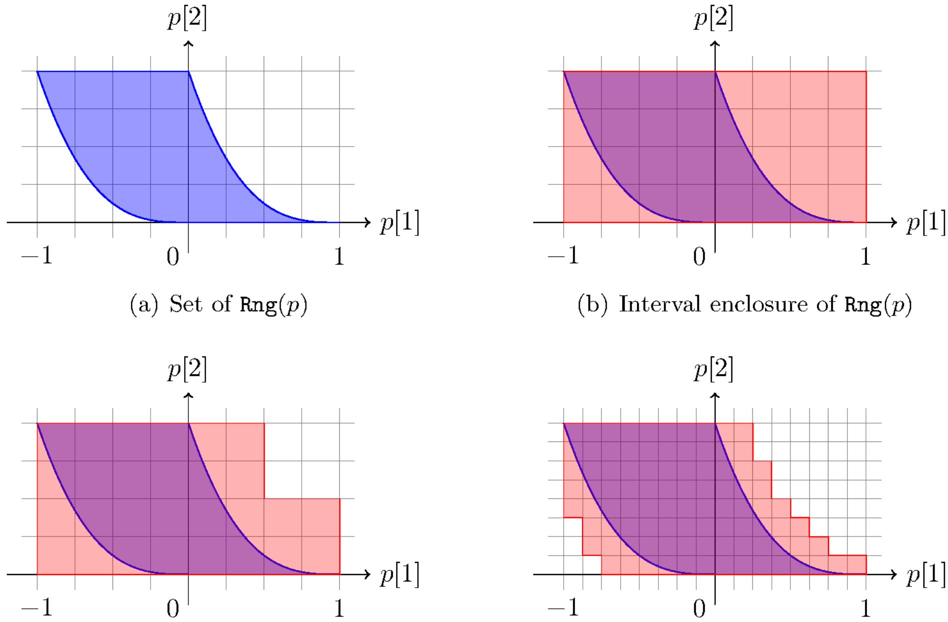
Theorem 2.2.7. *Given that $F(\vec{X})$ is an inclusion isotonic and L-Lipschitz interval extension of the function $f(\vec{x})$ in the domain $D \in \mathbb{IR}^n$, then*
$$d_H(\{f(\vec{x}) \,|\, \vec{x} \in D\}, \bigcup_{\vec{C} \in D_{(N)}} F(\vec{C})) \;\leq\; \frac{\sqrt{m}}{N} \cdot L \cdot W(D)$$
wherein $D_{(N)}$ denotes the N-uniform subdivision of D, and m is the dimension of F.

Proof. We consider the bound on $d_H(\{f(\vec{x}) \,|\, \vec{x} \in \vec{C}\}, F(\vec{C}))$ for a grid $\vec{C} \in D_{(N)}$. Since F is inclusion isotonic and L-Lipschitz, we have that $F(\vec{C})$ is an interval enclosure of $\{f(\vec{x}) \,|\, \vec{x} \in \vec{C}\}$ and $\mathrm{W}(F(\vec{C})) \leq L \cdot \mathrm{W}(\vec{C})$. Then we may also infer that $d_H(\{f(\vec{x}) \,|\, \vec{x} \in \vec{C}\}, F(\vec{C})) \leq \sqrt{m} \cdot \mathrm{W}(F(\vec{C}))$, and thereby, the distance is also bounded by $\sqrt{m} \cdot L \cdot \mathrm{W}(\vec{C})$. By the definition of uniform subdivision, we have that $\mathrm{W}(D) = N \cdot \mathrm{W}(\vec{C})$, hence
$$d_H(\{f(\vec{x}) \,|\, \vec{x} \in \vec{C}\}, F(\vec{C})) \leq \frac{\sqrt{m}}{N} \cdot L \cdot \mathrm{W}(D).$$

Since the sets $\{f(\vec{x}) \,|\, \vec{x} \in \vec{C}\}$ and $F(\vec{C})$ are non-empty for all $\vec{C} \in D_{(N)}$, we may rewrite the Hausdorff distance $d_H(\{f(\vec{x}) \,|\, \vec{x} \in D\}, \bigcup_{\vec{C} \in D_{(N)}} F(\vec{C}))$ as
$$\sup_{\vec{C} \in D_{(N)}} \{d_H(\{f(\vec{x}) \,|\, \vec{x} \in \vec{C}\}, F(\vec{C}))\}.$$

Hence we have that
$$d_H(\{f(\vec{x}) \,|\, \vec{x} \in D\}, \bigcup_{\vec{C} \in D_{(N)}} F(\vec{C})) \leq \frac{\sqrt{m}}{N} \cdot L \cdot \mathrm{W}(D).$$

□



(a) Set of $\mathtt{Rng}(p)$ (b) Interval enclosure of $\mathtt{Rng}(p)$

(c) 4-uniform subdivision (d) 8-uniform subdivision

Figure 2.1: Interval over-approximations of a polynomial function

Example 2.2.8. *We consider the polynomial function* $p(x_1, x_2) = \begin{pmatrix} x_1 - x_2 \\ x_2^3 \end{pmatrix}$ *over the domain* $x_1 \in [0, 1]$ *and* $x_2 \in [0, 1]$*. The function* $P(X_1, X_2) = \begin{pmatrix} X_1 - X_2 \\ X_2^3 \end{pmatrix}$ *over* $X_1 \subseteq [0, 1]$ *and* $X_2 \subseteq [0, 1]$ *is an interval extension of* p*. It is also not difficult to verify that* P *is inclusion isotonic and Lipschitz. The exact range of* p*, shown in Figure 2.1(a), is not an interval. An interval over-approximation of* $\mathtt{Rng}(p)$ *can be computed directly by evaluating* $P([0, 1], [0, 1])$*. As shown in Figure 2.1(b), the overestimation might be too large. However, if we perform a* 4*-uniform subdivision on the domain, a better over-approximation which is presented in Figure 2.1(c) can be obtained. We may even further improve the accuracy by applying an* 8*-uniform subdivision, as shown in Figure 2.1(d).*

For an n-dimensional domain, the N-uniform subdivision yields N^n grids which may easily lead the subsequent computation to be intractable when n is large. Therefore, improving accuracy by subdivision often requires a lot of computational effort.

2.2.2 Interval evaluation for polynomial functions

We focus on the interval evaluation on the functions which are defined by polynomials. Given a polynomial p, an interval extension P of p can be easily derived as an expression which is syntactically same as an equivalent expression of p except that the variables take interval values and the operators are replaced by their interval counterparts. It is also not difficult to verify that such an extension is inclusion isotonic and Lipschitz. For example, an interval extension of $p = 2 - x_1 + x_2^2 x_1$ can be $P = [2, 2] - X_1 + X_2^2 \cdot X_1$ or

$P = [2, 2] - X_1 \cdot ([1, 1] - X_2^2)$. We denote the set of interval extensions derived in such a way by $\mathrm{Ext}(p)$.

Lemma 2.2.9 ([MKC09]). *Given a polynomial function p over an interval domain D, any interval extension in the set $Ext(p)$ is inclusion isotonic and Lipschitz in D.*

If a polynomial is not linear then the interval evaluation on two extensions $P, P' \in \mathrm{Ext}(p)$ may yield different results. To see that, we give Example 2.2.10. The phenomenon is caused by the *dependency problem*. In the example, we break the dependency between the terms x and x^2 in the interval computations.

Example 2.2.10. *We consider the polynomial $p(x) = x - x^2$ over the domain $x \in [-1, 1]$. It can be verified that the exact range of p is $[-2, \frac{1}{4}]$. We consider the interval evaluations on the following two interval extensions $P_1, P_2 \in Ext(p)$:*

$$P_1 = X - X^2, \qquad\qquad P_2 = X \cdot ([1, 1] - X)$$

The interval of $P_1([-1, 1])$ can be computed by

$$P_1([-1, 1]) = [-1, 1] - ([-1, 1])^2 = [-1, 1] - [0, 1] = [-2, 1],$$

and that of $P_2([-1, 1])$ can be computed by

$$P_2([-1, 1]) = [-1, 1] \cdot ([1, 1] - [-1, 1]) = [-1, 1] \cdot [0, 2] = [-2, 2].$$

Both of the results strictly contain the exact range.

In our work, we always do the evaluation job on the interval extension obtained from a *Horner form* of the polynomial. The evaluation on a Horner form-based interval extension often not only helps to relieve the dependency problem but also requires a smaller number of operations than that on a monomial form-based one (see [Pen00, CG02]), although it is not always the case, as it is given by the above example. For a multivariate polynomial, its Horner form is usually not unique and to choose the best one for accurate interval evaluation is not easy. We follow a simple but efficient method which will be given in the next chapter. More heuristics can be found in [CG02, CK04].

2.2.3 Applications

One of the main applications of interval arithmetic is to produce reliable results in numerical computation. Since we can only use finite precision arithmetic on computers, it is necessary to round the numbers during a computation. For example, we want to compute the product of the two decimal numbers 0.872915 and 7.921103, and only six decimal places are allowed to keep due to the machine precision. The exact result should be 6.914449625, however we have to round it to a near number such as 6.914449 or 6.914450. The difference of a rounded number from its exact value is called *round-off error*. Unfortunately, the round-off errors in a computation could be crucial to the final result and hence we should take them into account in some situations. Interval arithmetic provides a feasible and efficient way to carry round-off errors. In the previous example, the number 6.914449625 can be represented by an interval $[6.914449, 6.914450]$ which contains the exact value. Reliable results for complex numerical computation tasks may also be obtained by using interval arithmetic [KC91]. Besides, interval arithmetic is also extensively used in global optimization [IF79], constraint solving [JKDW01, BG06], validated integration of ODEs [NJC99] and so on.

In the next section, we introduce another class of over-approximate representations which are called Taylor models. Taylor models can be viewed as a higher-order extension of intervals, such that the overestimation in a result can be reduced by computing a polynomial part.

2.3 Taylor models

2.3.1 Taylor approximations

We briefly revisit the approximation method of using Taylor polynomials [Apo67, Apo69]. Given a univariate function f which is κ times differentiable over the domain $(a,b) \subseteq \mathbb{R}$. The *order k Taylor approximation (or expansion, polynomial)* wherein $k \leq \kappa$ of f at $x = c$ for some $c \in (a,b)$ is

$$p_k(x) = f(c) + f^{(1)}(c)(x-c) + \frac{1}{2!}f^{(2)}(c)(x-c)^2 + \cdots + \frac{1}{k!}f^{(k)}(c)(x-c)^k \tag{2.4}$$

such that $f^{(i)}(c)$ denotes the i-th order derivative of f at $x = c$. If f is also $(k+1)$ times differentiable, the approximation error of $p_k(x)$ for any $x \in (a,b)$, can be explicitly expressed by the *Lagrange remainder term*

$$r_k(x) = f(x) - p_k(x) = \frac{1}{(k+1)!}f^{(k+1)}(\xi(x))(x-c)^{k+1} \tag{2.5}$$

for some constant $\xi(x)$ between x and c. If $f \in \mathcal{C}^\omega((a,b))$ then there is some $\varepsilon > 0$ such that for any $x \in (c-\varepsilon, c+\varepsilon) \subseteq (a,b)$, $p_k(x)$ converges to $f(x)$ when $k \to \infty$.

Taylor polynomials can also be applied to approximating multivariate functions. Given a multivariate function $f \in \mathcal{C}^\kappa(D)$ wherein $D \subseteq \mathbb{R}^n$, its *order k Taylor approximation* for $k \leq \kappa$ at $\vec{x} = \vec{c}$ for some $\vec{c} \in D$ is the polynomial

$$\begin{aligned} p_k(\vec{x}) \ &= f(\vec{c}) + \sum_{i=1}^{n}\left(\frac{\partial f}{\partial x_i}(\vec{c}) \cdot (x_i - c_i)\right) + \cdots \\ &+ \frac{1}{k!}\sum_{j_1+\cdots+j_n=k}\left(\frac{\partial^k f}{\partial x_1^{j_1}\cdots\partial x_n^{j_n}}(\vec{c}) \cdot \prod_{i=1}^{n}(x_i - c_i)^{j_i}\right) \end{aligned} \tag{2.6}$$

Similarly, if f is also $(k+1)$ times partially differentiable in D, the Lagrange remainder term for any $\vec{x} \in D$ is given by

$$r_k(\vec{x}) = \frac{1}{(k+1)!}\sum_{j_1+\cdots+j_n=k+1}\left(\frac{\partial^{k+1} f}{\partial x_1^{j_1}\cdots\partial x_n^{j_n}}(\vec{\xi}(\vec{x})) \cdot \prod_{i=1}^{n}(x_i - c_i)^{j_i}\right) \tag{2.7}$$

for some constant $\vec{\xi}(\vec{x})$ on the line segment connecting $\vec{x}$ and $\vec{c}$. If $f \in \mathcal{C}^\omega(D)$, there exists a non-empty open set C containing $\vec{c}$ such that $p_k(\vec{x})$ converges to $f(\vec{x})$ when $k \to \infty$ for any $\vec{x} \in C$.

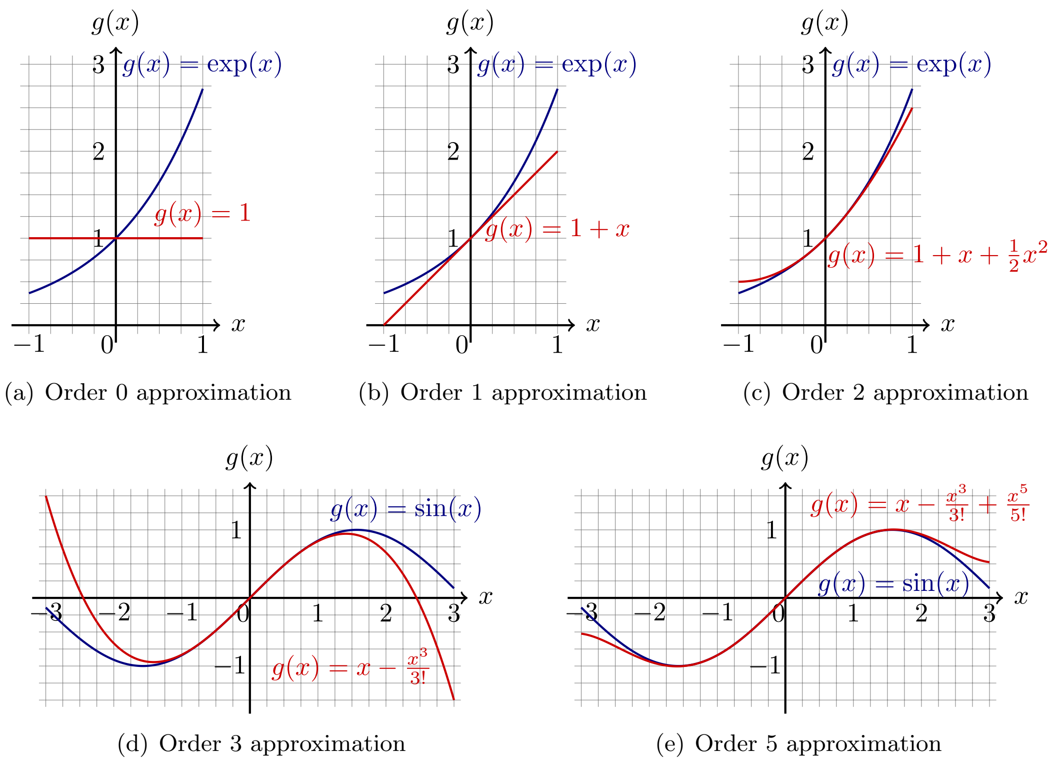


Figure 2.2: Taylor approximations for the functions $\exp(x)$ and $\sin(x)$

Example 2.3.1. *The Taylor approximations at the point* 0 *of some elementary functions are given in Figure 2.2. The Taylor approximations from order* 0 *to* 2 *of the function* $\exp(x)$ *are given by Figure 2.2(a)- 2.2(c). Only the interval* $[-1, 1]$ *of* x *is considered. For the function* $\sin(x)$*, the interval of* x *under consideration is enlarged to* $[-3, 3]$ *while higher-order approximations are presented. Since both of* $\exp(x)$ *and* $\sin(x)$ *are analytic, the approximation qualities can always be improved in those regions by raising the approximation orders.*

Taylor approximations can be computed in a compositional way. Given two univariate functions f, g such that $\texttt{Rng}(f) \subseteq \texttt{Dom}(g)$. Assume that we have the order k Taylor polynomial $p_f(x)$ of $f(x)$ at $x = c$ and the order k Taylor polynomial $p_g(x)$ of $g(x)$ at the point $x = f(c)$. Then the order k Taylor polynomial of the composite function $g \circ f$ at $x = c$ can be obtained by substituting $p_f(x)$ in the place of x in $p_g(x)$ and retaining only the terms of degree $\leq k$. For example, consider the functions $f(x) = \sin(x)$ and $g(x) = \exp(x)$, we choose $c = 0$ and $k = 3$. The polynomials $p_f(x)$ and $p_g(x)$ are given by

$$p_f(x) = x - \frac{1}{3!}x^3 \qquad \text{and} \qquad p_g(x) = 1 + x + \frac{1}{2!}x^2 + \frac{1}{3!}x^3$$

Then the order 3 Taylor approximation of $g \circ f = \exp(\sin(x))$ can be computed by evaluating the polynomial $p_g(p_f(x))$

$$\begin{aligned} p_g(p_f(x)) &= 1 + \left(x - \frac{1}{6}x^3\right) + \frac{1}{2}\left(x - \frac{1}{6}x^3\right)^2 + \frac{1}{6}\left(x - \frac{1}{6}x^3\right)^3 \\ &= 1 + x + \frac{1}{2}x^2 - \frac{1}{6}x^4 - \frac{1}{12}x^5 + \frac{1}{72}x^6 + \frac{1}{72}x^7 - \frac{1}{1296}x^9 \end{aligned}$$

and then removing the items of degrees > 3. The result is $1 + x + \frac{1}{2}x^2$. Taylor approximations for multivariate functions can also be obtained in a similar way.

Polynomial interpolations. Polynomial approximations may also be obtained by using *polynomial interpolations* [Phi03, Tre13]. The task of polynomial interpolation is to compute a polynomial p which interpolates the given finite pairs of points (c_1, d_1), ..., (c_m, d_m), i.e., $p(c_i) = d_i$ for $1 \le i \le m$. If the value d_i for $1 \le i \le m$ is selected as $f(c_i)$ for some function f which is not necessarily continuous, then p can be viewed as a polynomial approximation of f such that they coincide at the points $x = c_1, \ldots, c_m$. For multivariate and vector-valued functions, the points $c_1, \ldots, c_m$ and $d_1, \ldots, d_n$ are real-valued vectors. A Taylor approximation is only guaranteed to touch the original function at the expansion point, however a polynomial interpolation is able to pass through several ones. Unfortunately, this fact does not lead to a conclusion that a polynomial interpolation is always a better approximation than a Taylor expansion of the same order, since the interpolation quality highly depends on the points selected. In [MH02], it is proved that interpolating the *Chebyshev nodes* can produce a *near-best* polynomial approximation for a function. We illustrate a comparison between a Taylor approximation and a Chebyshev interpolation on the function $\cos(3x)$ by Figure 2.3. The order 4 Taylor approximation is the polynomial $1 - \frac{9}{2}x^2 + \frac{27}{8}x^4$, and the order 4 Chebyshev interpolation is approximately equivalent to $0.9751857 - 4.04896519x^2 + 2.104651873x^4$. Although interpolation methods may provide better approximation qualities in general, to replace the Taylor approximations by some polynomial interpolations in a large computation framework is often not easy. We will give further discussions in the subsequent sections.

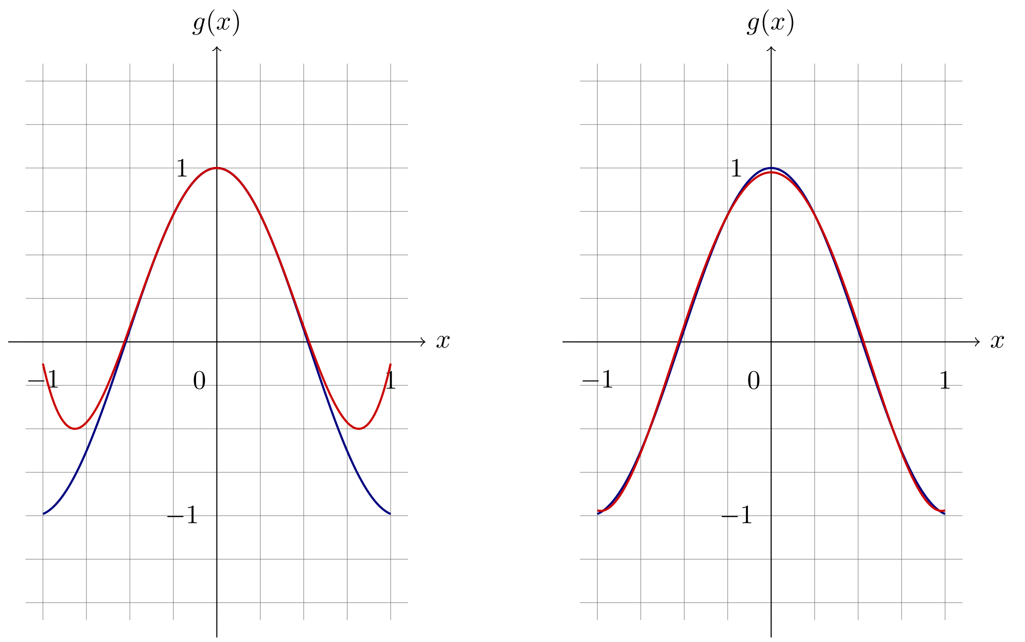
Since the Taylor approximation $p(x)$ of a function $f(x)$ at $x = c$ has an error bound which generally grows with the size of $|x - c|$, we often choose the midpoint of the domain as the expansion point.

2.3.2 Basic theorems of Taylor models

Taylor models are representations which combine Taylor polynomials and intervals. They are originally developed by Berz and Makino [MB96, Ber99, MB03] to provide over-approximate representations for continuous functions.

Definition 2.3.2 (Taylor model)**.** *A* Taylor Model *(TM) is denoted by a pair* (p, I) *such that* p *is a polynomial over a set of variables* $\vec{x}$ *ranging in an interval domain* D*, and* I *is the interval remainder. TMs may also be vector-valued. A vector-valued TM can be viewed as a vector of real-valued TMs, or in a way that both* p *and* I *are vector-valued and are of the same dimension.*

Given a TM (p, I) and a function f which are over the same domain D, we say that



(a) Order 4 Taylor approximation (in red) (b) Order 4 Chebyshev interpolation (in red)

Figure 2.3: Comparison between Taylor approximation and Chebyshev interpolation on $\cos(3x)$

f is *over-approximated* by (p, I), denoted by $f \in (p, I)$ or $f \in p + I$, if

$$f(\vec{x}) \in p(\vec{x}) + I \quad \textit{for all} \quad \vec{x} \in D \ .$$

A TM forms a *convex* and *compact* set of continuous functions over-approximated by it. A proof can be found in [BM98]. In the rest of the thesis, if $f \in (p, I)$, then (p, I) is called a TM of f.

Definition 2.3.3 (Convex set). *A set S is* convex *if and only if for all $x, y \in S$, we have that*

$$\lambda x + (1 - \lambda)y \in S \quad \textit{for all} \quad \lambda \in [0, 1] \ .$$

Definition 2.3.4 (Normed vector space). *A* normed vector space $(M, |\cdot|)$ *is a vector space M equipped with a norm $|\cdot|$.*

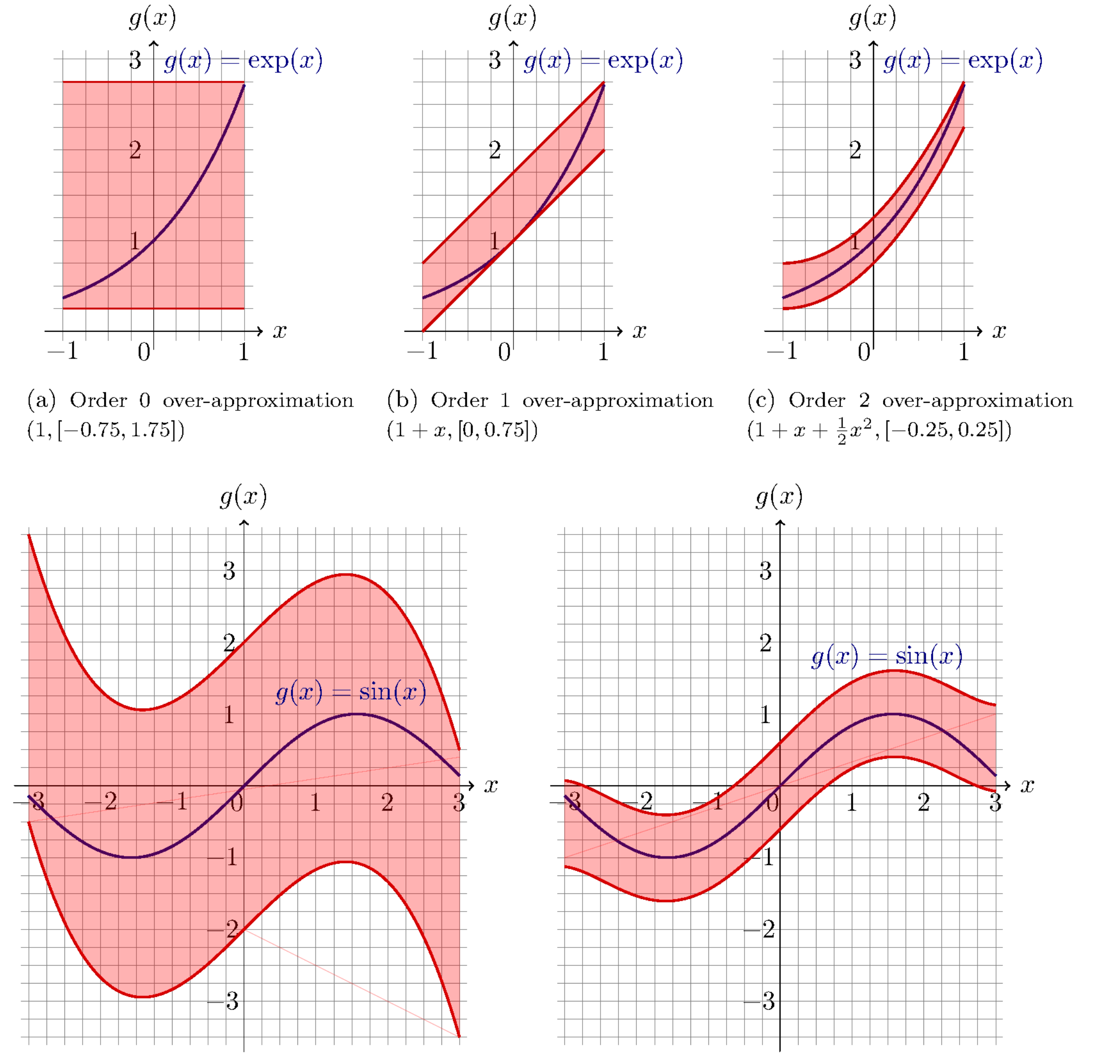
Definition 2.3.5 (Bounded set). *A set S in a normed vector space $(M, |\cdot|)$ is* bounded, *if there exists $x \in S$ and $r > 0$ such that $|x - y| < r$ for all $y \in S$.*

Definition 2.3.6 (Open set). *A set S in a normed vector space $(M, |\cdot|)$ is* open, *if for any $x \in S$ there is some $\varepsilon > 0$ such that for any $y \in M$ if $|x - y| < \varepsilon$ then $y \in S$.*

Definition 2.3.7 (Closed set). *A set S is* closed *if its complement is open.*

Definition 2.3.8 (Compact set). *A set S is* compact *in a normed vector space of finite dimension if it is bounded and closed.*

Theorem 2.3.9. *A TM over an interval domain D defines a* convex *and* compact *set of continuous functions which are over-approximated by it over D.*



(a) Order 0 over-approximation $(1, [-0.75, 1.75])$

(b) Order 1 over-approximation $(1 + x, [0, 0.75])$

(c) Order 2 over-approximation $(1 + x + \frac{1}{2}x^2, [-0.25, 0.25])$

(d) Order 3 over-approximation $(x - \frac{1}{3!}x^3, [-2, 2])$

(e) Order 5 over-approximation $(x - \frac{1}{3!}x^3 + \frac{1}{5!}x^5, [-0.6, 0.6])$

Figure 2.4: Order k over-approximations for the functions $\exp(x)$ and $\sin(x)$

Given a function $f \in \mathcal{C}^k(D)$ for some interval D, an order k TM (p, I) of f can be obtained by first computing the order k Taylor polynomial p of f at $\vec{x} = \mathrm{Mid}(D)$ and then evaluating an interval I which contains all remainder terms for $\vec{x} \in D$. For example, an order 4 TM of the function $\exp(x)$ over $[-1, 1]$ is given by

$$(1 + x + \frac{1}{2!}x^2 + \frac{1}{3!}x^3 + \frac{1}{4!}x^4 \ , \ [-0.02266, 0.02266])$$

such that the remainder interval is obtained by interval evaluating $\frac{1}{5!} \cdot \exp([-1, 1]) \cdot ([-1, 1])^5$. It contains all remainder terms for $x \in [-1, 1]$.

Example 2.3.10. *In Figure 2.4, we extended the order k approximations presented in Example 2.3.1 to their order k TM over-approximations.*

We consider the overestimation in a TM over-approximation. It may also be measured by means of Hausdorff distance. If (p, I) is a TM of the function f over the domain D, then the Hausdorff distance between (p, I) and f can be computed by

$$d_H((p, I), f) = \max_{\vec{x} \in D}\{\|f(\vec{x}) - p(\vec{x})\|\}$$

which is bounded by $\sqrt{m} \cdot \mathrm{W}(I)$ wherein m is the dimension of I. Hence, the overestimation of a TM can be assessed only based on the width of the remainder interval.

Given a function $f \in \mathcal{C}^\omega(D)$ for some D. If we have the order k Taylor polynomial p_k of f at $\vec{x} = \vec{c}$ for some $\vec{c} \in D$, and the remainder term $f(\vec{x}) - p_k(\vec{x})$ converges to zero when $k \to \infty$ for all $\vec{x} \in D$, then we say that a computation (list of code, algorithm) R is *contractible* if the remainder interval $R(k)$ for p_k converges to zero when $k \to \infty$. For example, the interval evaluation $R(k) = \frac{1}{(k+1)!}\exp([2,4])([-1,1])^{k+1}$ is a contractible computation for the remainder interval over the domain $[2, 4]$ for the Taylor expansion of $\exp(x)$ at $x = 3$.

Based on a contractible remainder computation, we may arbitrarily reduce the overestimation of a TM by raising its order. On the other hand, given a fixed-degree Taylor polynomial, if the remainder is computed by an inclusion isotonic and Lipschitz interval extension of the exact remainder expression, then the resulting interval can be made arbitrarily close to the smallest interval enclosure of the remainders by performing finer subdivision on the domain interval.

2.3.3 Taylor model arithmetic

Similar to intervals, basic operators such as addition, multiplication can also be extended to deal with TMs. Based on them, we are able to define a *TM extension* of a given function. Besides, the inclusion isotonicity as well as the Lipschitz property of TM extensions may be defined analogously.

Given a function f and its TM (p, I) over the domain D. A TM for $h(f)$ wherein h is a *unary* operator can be computed only based on (p, I). For example, the additive inverse and integral on a TM (p, I) over D are defined by

$$\begin{array}{lrcl} \text{Additive inverse:} & -(p, I) & = & (-p, -I) \\ \text{Integral:} & \partial_i^{-1}(p, I) & = & (\int_{a_i}^{b_i}(p(\vec{x}) - p_e(\vec{x}))dx_i \ , \ (\mathtt{Int}(p_e) + I) \cdot [a_i, b_i]) \end{array}$$

wherein p_e consists of the terms of degrees $> k$ in p for some *truncation order* $k \geq 0$, and $[a_i, b_i]$ is the range of x_i in D. The operation $\mathtt{Int}(p_e)$ denotes an interval enclosure of p_e over D. Here, the order k is not necessarily same as the degree of p. The reason of specifying it is to limit the representation size of the resulting TM.

Given two functions f, g over the some domain D, assume that (p_1, I_1), (p_2, I_2) are the TMs of them respectively. For a binary operator $\circ$, a TM for $f \circ g$ can be computed only based on (p_1, I_1) and (p_2, I_2). For example, the sum of the two TMs is computed by

$$(p_1, I_1) + (p_2, I_2) \quad = \quad (p_1 + p_2, I_1 + I_2)$$

and for some truncation order $k \geq 0$, their order k product is computed by

$$(p_1, I_1) \cdot (p_2, I_2) \quad = \quad (p_1 \cdot p_2 - p_e \ , \ \mathtt{Int}(p_1) \cdot I_2 + I_1 \cdot \mathtt{Int}(p_2) + I_1 \cdot I_2 + \mathtt{Int}(p_e))$$

wherein p_e consists of the terms of degrees $> k$ in $p_1 \cdot p_2$.

Algorithm 3 Compute an order k TM for $f((p, I))$

Input: a TM (p, I), a continuous function $f \in \mathcal{C}^{\kappa}(D_f)$ such that $(p, I) \subseteq D_f$
Output: an order k TM for $f((p, I))$ such that $k < \kappa$

1: Compute the order k Taylor polynomial $p_f(x)$ of $f(x)$ at the midpoint of $\mathtt{Int}((p, I))$;
2: Evaluate a safe remainder interval I_f for $p_f(x)$;
3: Compute the order k TM (p_r, I_r) for $p_f((p, I))$;
4: $I_r \leftarrow I_r + I_f$;
5: **return** (p_r, I_r);

An order κ TM (p, I) can be simplified by performing a k-*truncation* on it for some $0 \leq k < \kappa$. That is, we remove the terms of degrees $> k$ in p and add their interval enclosure onto the remainder.

$$\text{Truncation:} \quad \mathrm{Trunc}_k((p, I)) \quad = \quad (p - p_e \ , \ I + \mathtt{Int}(p_e))$$

The resulting TM is a simplification as well as an over-approximation of the original one.

The TM division is more complicated. Given two TMs (p_1, I_1), (p_2, I_2) such that $0 \notin \mathtt{Int}((p_2, I_2))$. An order k TM of $(p_1, I_1)/(p_2, I_2)$ is computed in the following way. Firstly, we compute the order k Taylor polynomial p_k for the function $\frac{1}{x}$ at $x = c$ wherein c is the midpoint of $\mathtt{Int}((p_2, I_2))$,

$$p_k(x) = \frac{1}{c} \cdot \left(1 - \frac{x-c}{c} + \left(\frac{x-c}{c}\right)^2 - \left(\frac{x-c}{c}\right)^3 + \cdots + (-1)^k \left(\frac{x-c}{c}\right)^k\right)$$

and a remainder interval for p_k is evaluated by

$$I_r = \mathtt{Int}\left((-1)^{k+1} \frac{1}{x^{k+2}} \cdot (x-c)^{k+1}\right)$$

over $x \in \mathtt{Int}((p_2, I_2))$. Secondly, we compute an order k TM (p_3, I_3) by substituting $(p_2 - c, I_2)$ in the place of x in (p_k, I_r). Then an order k TM of $(p_1, I_1)/(p_2, I_2)$ can be computed by $(p_1, I_1) \cdot (p_3, I_3)$. Such an idea could also be applied to computing TMs for continuous functions. We present a general procedure by Algorithm 3. For more TM operations, one may refer to [MB03].

By TM arithmetic, we may compute a TM for a function based on the TMs of its components. In that case, the resulting remainder interval is computed by a procedure consists of several subroutines. If the procedure is a contractible remainder evaluation, then we may still able to improve the accuracy by using a higher TM order during the computation.

Class of the TM domains. Although the TM domains under our consideration are always intervals, we can still define a large class of non-convex sets by those TMs. For a function f with an arbitrary domain set D, we may compute a set of grids that over-approximate D, and then use the TMs over them to over-approximate the values of f.

2.3.4 Applications

TMs provide higher-order over-approximations for continuous functions. Unlike intervals, the overestimation in a TM can be measured only based on its remainder interval size which is often very small. They considerably relieve the dependency problem in many computation jobs. Therefore, TMs are often used as replacements of intervals in numerical computation tasks when accuracy is critical. However, TM arithmetic is more computational complex than interval arithmetic in general. It can be seen from their representations. An interval could be represented by their endpoints which are only two floating-point numbers. On the other hand, for a TM, we need not only to represent its interval remainder but also to keep a multivariate polynomial. In the worst case, a polynomial of n variables and k degree can have $\binom{n+k}{k}$ terms. Although the Taylor expansion of a function is not always a dense polynomial, the resulting polynomial from a complex computation routine is often of a large size. Lots of heuristics can be used to conservatively simplify the representation of a TM (p, I). For example, we may first remove the terms whose interval enclosures are smaller than a specified box from the polynomial p, then compute an interval enclosure of the removed parts and add it onto the remainder I.

The use of TMs appears in a wide range of numerical analysis tasks. The most well known application is the validated integration techniques for non-linear ODEs [BM98, MB09, NJN06]. Besides, TMs can also be applied to global optimization and satisfiability (SAT) checking [BM09], reachability analysis of non-linear hybrid systems [CÁS12], or even proof systems [BJMD+12].

Polynomial interpolations as TM polynomial parts. As we discussed previously, polynomial interpolations such as Chebyshev polynomials may provide better approximations than Taylor expansions. Thereby if we replace the Taylor approximations in the TM arithmetic framework by Chebyshev polynomials, the overestimation could be greatly reduced. To do that, we have to carry out all computations in Chebyshev basis since a transformation of a polynomial from Chebyshev to monomial basis on a finite-precision machine may lead to a loss of precision. Such work is proved effective for univariate functions [BJ10]. However, the multiplication of two multivariate polynomials in Chebyshev basis seems not as easy as that in monomial basis. Hence, the use of Chebyshev polynomials in the framework of TM arithmetic needs further investigation.

2.4 Representations for reachable sets

We revisit some popular reachable set representations for continuous and hybrid systems in this section. Most of them are convex geometric objects in the Euclidean space $\mathbb{R}^n$ for some $n \in \mathbb{Z}$ and $n > 0$.

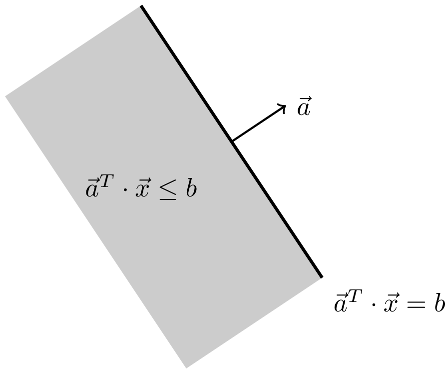


Figure 2.5: Example of a halfspace and its supporting hyperplane

2.4.1 Convex polyhedra and polytopes

An n-dimensional *(closed) halfspace* S is the set of points in $\mathbb{R}^n$ which satisfy a linear polynomial inequality $\vec{a}^T \cdot \vec{x} \leq b$.[1] It is the set defined by $S = \{\vec{x} \in \mathbb{R}^n \mid \vec{a}^T \cdot \vec{x} \leq b\}$ and is denoted by $S : \vec{a}^T \cdot \vec{x} \leq b$.

A *hyperplane* H in the Euclidean space $\mathbb{R}^n$ is the set of points which satisfy a linear polynomial equation $\vec{a}^T \cdot \vec{x} = b$. It is the set $H = \{\vec{x} \in \mathbb{R}^n \mid \vec{a}^T \cdot \vec{x} = b\}$ which is $(n-1)$-dimensional. We denote it by $H : \vec{a}^T \cdot \vec{x} = b$ wherein $\vec{a}$ is a vector which is orthogonal to H, and we also call it a normal vector (or normal) of H. By Definition 2.4.1, a halfspace $S : \vec{a}^T \cdot \vec{x} \leq b$ only has one supporting hyperplane which is $H : \vec{a}^T \cdot \vec{x} = b$. We illustrate an example in Figure 2.5.

Definition 2.4.1 (Supporting hyperplane)**.** *Given a set $S \subseteq \mathbb{R}^n$, a hyperplane $H : \vec{a}^T \cdot \vec{x} = b$ is a* supporting hyperplane *of S if*

- *S is contained in the halfspace defined by $\vec{a}^T \cdot \vec{x} \leq b$, and*
- *$S \cap H \neq \emptyset$.*

An n-dimensional *(convex) polyhedron* is defined by an intersection of finitely many n-dimensional halfspaces. Given m halfspaces $S_1 : \vec{a}_1^T \cdot \vec{x} \leq b_1, \ldots, S_m : \vec{a}_m^T \cdot \vec{x} \leq b_m$, their intersection is the set

$$P = \{\vec{x} \in \mathbb{R}^n \mid \bigwedge_{i=1}^{m} (\vec{a}_i^T \cdot \vec{x} \leq b_i)\}$$

which is denoted by $P : A \cdot \vec{x} \leq \vec{b}$ wherein the i-th row of A is $\vec{a}_i^T$ for $1 \leq i \leq m$. The notation is also called a *$\mathcal{H}$-representation (halfspace representation)*. We give an example in Figure 2.6 in which the polyhedron is defined by the intersection of three halfspaces. When a polyhedron is bounded, we also call it a *polytope*.

Non-redundant $\mathcal{H}$-representations. Although the $\mathcal{H}$-representation of a polyhedron P is not unique, there exists a minimum number m such that P can be defined by m

[1] In the thesis, we also use $\cdot$ to denote the vector or matrix multiplication operator. Therefore, $\vec{a}^T \cdot \vec{x}$ denotes the inner product of $\vec{a}$ and $\vec{x}$.

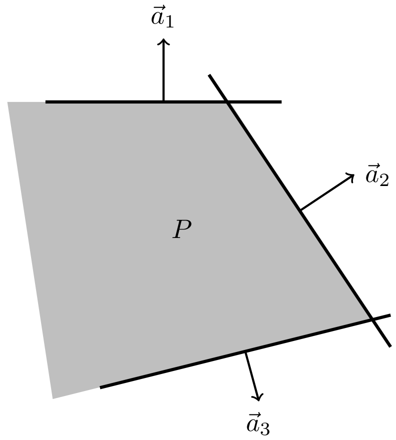


Figure 2.6: A polyhedron P defined by the intersection of three halfspaces

halfspaces. We also call such a representation *non-redundant*. The redundant halfspaces in a $\mathcal{H}$-representation can be removed by *Linear Programming (LP)* [BV04] in polynomial-time complexity. Given a $\mathcal{H}$-representation

$$P : (\vec{a}_1^T, \ldots, \vec{a}_m^T) \cdot \vec{x} \ \leq \ (b_1, \ldots, b_m) \ .$$

To check whether the i-th halfspace for $1 \leq i \leq m$ is redundant or not, we solve the following linear program

$$\sup\{\vec{a}_i^T \cdot \vec{x}\} \quad \text{subject to} \quad \bigwedge_{1 \leq j \leq m, j \neq i} (\vec{a}_j^T \cdot \vec{x} \leq b_j) \text{ and } \vec{x} \in \mathbb{R}^n \ .$$

If the result is no larger than b_i then the halfspace $S_i : \vec{a}_i^T \cdot \vec{x} \leq b_i$ is redundant and can be removed from the representation. By checking every halfspace, we may finally obtain a non-redundant $\mathcal{H}$-representation for P. If P is an n-dimensional polytope then a non-redundant $\mathcal{H}$-representation of it contains at least $n+1$ halfspaces (see [Zie95]).

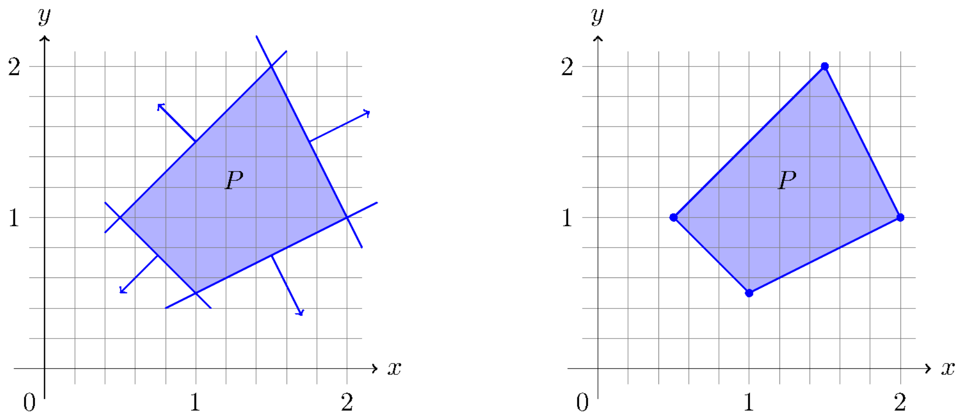
The *faces* of a polyhedron are its subsets on the boundary. We give the definition as follows.

Definition 2.4.2 (d-face)**.** *Given an n-dimensional polyhedron P. For $d \leq n$, a d-dimensional subset F of P is called a* d-face *if there exists a supporting hyperplane H of P such that $H \cap P = F$. We also call the $(n-1)$- and 0-faces* facets *and* vertices *respectively.*

If a polyhedron P is also a polytope, then it can also be defined by the *convex hull* of its vertices. Therefore, we may uniquely represent P by its vertices. Such a representation is called *$\mathcal{V}$-representation*, i.e., vertex representation. The minimum number of vertices in an n-dimensional polytope is $n+1$ [Zie95]. Notice that the $\mathcal{V}$-representation of a polytope is unique. We give Example 2.4.4 to show the $\mathcal{H}$- and $\mathcal{V}$-representations of a polytope.

Definition 2.4.3 (Convex hull)**.** *The convex hull of a set $V \subseteq \mathbb{R}^n$ is the smallest convex set in $\mathbb{R}^n$ which contains V. If $V = \{\vec{v}_1, \ldots, \vec{v}_m\}$ is finite, then the convex hull of V is defined by*

$$Conv(V) = \left\{ \sum_{i=1}^{m} \lambda_i \cdot v_i \;\middle|\; \lambda_1, \ldots, \lambda_m \in [0,1], \sum_{j=1}^{m} \lambda_j = 1 \right\} \ . \tag{2.8}$$



(a) $\mathcal{H}$-representation (b) $\mathcal{V}$-representation

Figure 2.7: Two representations of polytope P

Example 2.4.4. *We consider the polytope P defined by the intersection of the following halfspaces:*

$$S_1 = \{(x, y) \in \mathbb{R}^2 \mid 2x + y - 5 \le 0\}$$
$$S_2 = \{(x, y) \in \mathbb{R}^2 \mid x - 2y \le 0\}$$
$$S_3 = \{(x, y) \in \mathbb{R}^2 \mid -x - y + 1.5 \le 0\}$$
$$S_4 = \{(x, y) \in \mathbb{R}^2 \mid -x + y - 0.5 \le 0\}$$

which are illustrated in Figure 2.7(a). Then a $\mathcal{H}$-representation of P is given by

$$P : \begin{pmatrix} 2 & 1 \\ 1 & -2 \\ -1 & -1 \\ -1 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} \le \begin{pmatrix} 5 \\ 0 \\ -1.5 \\ 0.5 \end{pmatrix}.$$

Since the polytope P has the vertices $(2, 1)$, $(1, 0.5)$, $(0.5, 1)$, $(1.5, 2)$, see Figure 2.7(b), the $\mathcal{V}$-representation of it is $P : \{(2, 1), (1, 0.5), (0.5, 1), (1.5, 2)\}$.

Redundant points in defining a convex hull. Given a set of points $V = \{\vec{v}_1, \ldots, \vec{v}_m\}$, we say that a point $\vec{v}_i \in V$ is *redundant* in defining the convex hull $\mathrm{Conv}(V)$ if $\mathrm{Conv}(V) = \mathrm{Conv}(V \backslash \{\vec{v}_i\})$. Such redundancy can be checked by solving the following linear problem

$$\text{find } \lambda_1, \ldots, \lambda_{i-1}, \lambda_{i+1}, \ldots, \lambda_m \in [0, 1] \text{ subject to } \vec{v}_i = \sum_{1 \le j \le m, j \ne i} \lambda_j \cdot v_j \wedge \sum_{1 \le j \le m, j \ne i} \lambda_j = 1$$

that is to check whether the point $\vec{v}_i$ is in the convex hull of the other points. If so, then $\vec{v}_i$ is redundant.

Translations between $\mathcal{H}$- and $\mathcal{V}$-representations. The equivalent translation from a $\mathcal{H}$-representation to a $\mathcal{V}$-representation is an interesting problem in computational complexity, it is known as the *vertex enumeration problem*. Such a problem is intractable when the dimension of the polytope is high (see [AF92, Tiw08]). On the other hand, the reverse translation is known as the *facet enumeration problem* which is as hard as the vertex enumeration problem. Hence, it is often to only use one of the representations during a computation. When a translation is not avoidable and the efficiency is also required, we can make it approximately.

Computational aspects of $\mathcal{H}$- and $\mathcal{V}$-representations. Polytopes are closed under many operations which are needed in the reachability analysis for hybrid automata. However, the computational complexity of an operation highly depends on the representations in use. We give a brief summary as below, more details can be found in [Zie95, HRGZ97, Tiw08].

- *Emptiness* - To check whether a polytope $P \subseteq \mathbb{R}^n$ is empty. If P is given by a $\mathcal{H}$-representation $P : A \cdot \vec{x} \leq \vec{b}$, then we only need to solve the following linear program
$$\text{find } \vec{x} \in \mathbb{R}^n \quad \text{subject to} \quad A \cdot \vec{x} \leq \vec{b} \ .$$
If no solution can be found, then the polytope is empty. If it is given by a $\mathcal{V}$-representation $P : \{\vec{v}_1, \ldots, \vec{v}_m\}$, then the emptiness of P may be checked by solving
$$\text{find } \vec{x} \in \mathbb{R}^n \quad \text{subject to} \quad \vec{x} = \sum_{i=1}^{m} \lambda_i \cdot v_i \wedge \bigwedge_{i=1}^{m} (\lambda_i \in [0,1]) \wedge \sum_{i=1}^{m} \lambda_i = 1 \ .$$
Hence the emptiness checking is easy on both $\mathcal{H}$- and $\mathcal{V}$-representations in practice.

- *Membership* - To check whether $\vec{c} \in P$ holds for a given point $\vec{c} \in \mathbb{R}^n$ and a polytope $P \subseteq \mathbb{R}^n$. Such a problem can be efficiently solved on a $\mathcal{H}$-representation $P : A \cdot \vec{x} \leq \vec{b}$, since we only need to verify whether $A \cdot \vec{c} \leq \vec{b}$ holds or not. For a $\mathcal{V}$-representation $P : \{\vec{v}_1, \ldots, \vec{v}_m\}$, the problem may be solved by linear programming:
$$\text{find } \lambda_1, \ldots, \lambda_m \in [0,1] \quad \text{subject to} \quad \vec{c} = \sum_{i=1}^{m} \lambda_i \cdot v_i \wedge \sum_{i=1}^{m} \lambda_i = 1 \ .$$
Therefore, the membership checking is easy on both of the representations.

- *Affine mapping* - To compute the image of a polytope $P \subseteq \mathbb{R}^n$ under an affine mapping $\pi : \vec{x} \mapsto M \cdot \vec{x} + \vec{c}$ for $M \in \mathbb{R}^{n' \times n}$, $\vec{c} \in \mathbb{R}^n$. It is the polytope defined by $P' = \{M \cdot \vec{x} + \vec{c} \,|\, \vec{x} \in P\}$. If P is given by a $\mathcal{V}$-representation $P : \{\vec{v}_1, \ldots, \vec{v}_m\}$, then the $\mathcal{V}$-representation of P' can be computed by removing the redundant points from $\{M \cdot \vec{v}_1 + \vec{c}, \ldots, M \cdot \vec{v}_m + \vec{c}\}$ in defining $\mathrm{Conv}(\{M \cdot \vec{v}_1 + \vec{c}, \ldots, M \cdot \vec{v}_m + \vec{c}\})$. However, the image is easy to compute for a $\mathcal{H}$-representation $P : A \cdot \vec{x} \leq \vec{b}$ only in the case that M is an invertible matrix. The result is of the $\mathcal{H}$-representation
$$P' : \ (A \cdot M^{-1}) \cdot \vec{x} \leq (\vec{b} + (A \cdot M^{-1}) \cdot \vec{c}) \ .$$
Otherwise the computation is as hard as the vertex numeration problem on P.

- *Intersection* - To compute $P \cap Q$ for the given polytopes P, Q. If both of P, Q are given by $\mathcal{H}$-representations, a $\mathcal{H}$-representation of $P \cap Q$ can be computed by first collecting the halfspaces defining P, Q and removing the redundant ones. On the other hand, the computation of either a $\mathcal{H}$-representation or a $\mathcal{V}$-representation for the intersection is hard when one of P, Q is in $\mathcal{V}$-representation.

P	Q	$P \cap Q$	$\mathrm{Conv}(P \cup Q)$	$P \oplus Q$
$\mathcal{H}$	$\mathcal{H}$	$+$ $(\mathcal{H})$	$-$	$-$
$\mathcal{H}$	$\mathcal{V}$	$-$	$-$	$-$
$\mathcal{V}$	$\mathcal{H}$	$-$	$-$	$-$
$\mathcal{V}$	$\mathcal{V}$	$-$	$+$ $(\mathcal{V})$	$+$ $(\mathcal{V})$

Table 2.1: Complexities of the binary operators on polytopes. Legends: $\mathcal{H}$: $\mathcal{H}$-representation, $\mathcal{V}$: $\mathcal{V}$-representation, $+$: easy, $-$: hard.

- *Convex hull* - To compute $\mathrm{Conv}(P \cup Q)$ for the given polytopes P, Q. If the polytopes P, Q are in $\mathcal{V}$-representations, the $\mathcal{V}$-representation of $\mathrm{Conv}(P \cup Q)$ can be computed by collecting the vertices of P, Q and removing the redundant ones in defining $\mathrm{Conv}(P \cup Q)$. If one of P, Q is in $\mathcal{H}$-representation, then to compute either of the representations for the convex hull is hard.

- *Minkowski sum* - To compute $P \oplus Q$ for the given polytopes P, Q. The Minkowski sum of P, Q is defined by

$$P \oplus Q = \{\vec{x} + \vec{y} \,|\, \vec{x} \in P, \vec{y} \in Q\}$$

  which is also a polytope. When P, Q are in $\mathcal{V}$-representations, then the $\mathcal{V}$-representation of their Minkowski sum can be obtained by first computing the Minkowski sum of the vertex sets V_P, V_Q of P, Q, and then removing the redundant points in defining $\mathrm{Conv}(V_P \oplus V_Q)$. However, if one of the polytopes is in $\mathcal{H}$-representation, then either of the representations for $P \oplus Q$ is hard to compute.

In Table 2.1, we summarize the hardness of the binary operations on the two polytope representations. Polytopes could also be used as approximations for a bounded subset of $\mathbb{R}^n$.

Polytopes as over-approximations. A bounded set $S \subset \mathbb{R}^n$ for some $n \geq 0$ can be over-approximated by a polytope $\overline{P}$ based on a template $\vec{l}_1, \ldots, \vec{l}_m \in \mathbb{R}^n$. A $\mathcal{H}$-representation of it is of the form

$$\overline{P} : \; (\vec{l}_1^{\,T}, \ldots, \vec{l}_m^{\,T}) \cdot \vec{x} \leq (b_1, \ldots, b_m)$$

wherein the value b_i for $1 \leq i \leq m$ is obtained by solving the optimization problem

$$\sup\{\vec{l}_i^{\,T} \cdot \vec{x}\} \quad \text{subject to} \quad \vec{x} \in S \tag{2.9}$$

We illustrate an example in Figure 2.8.

Definition 2.4.5 (Template of a polytopic approximation)**.** *A* template *of a polytopic over- or under-approximation is a set of vectors $\vec{l}_1, \ldots, \vec{l}_m$ which are nonzero. For an n-dimensional over-approximation, we also require that there are at least $n+1$ vectors* linearly independent*.*

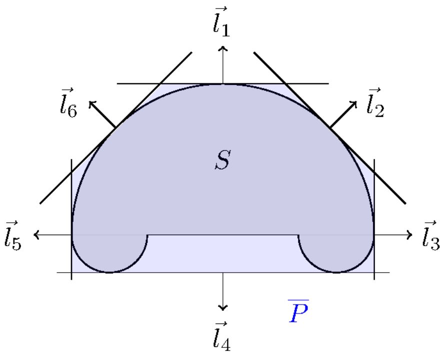


Figure 2.8: Polytopic over-approximation of a bounded set

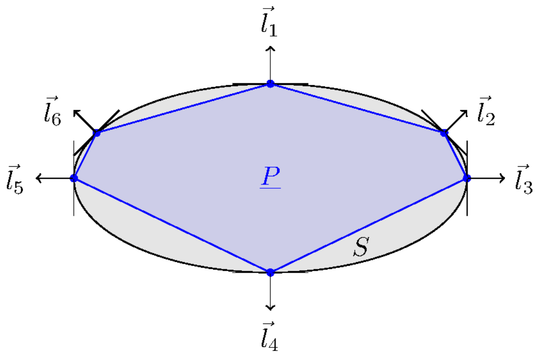


Figure 2.9: Polytopic under-approximation of a closed and bounded convex set

Polytopes as under-approximations. If a bounded set $S \subset \mathbb{R}^n$ for some $n \geq 0$ is closed and convex, we may compute an under-approximate polytope $\underline{P}$ for it based on a template $\vec{l}_1, \ldots, \vec{l}_m \in \mathbb{R}^n$. The polytope $\underline{P}$ can be computed as the convex hull of the points $\vec{v}_1, \ldots, \vec{v}_m \in \mathbb{R}^n$ such that for each $1 \leq i \leq m$, the point $\vec{v}_i$ is obtained from a solution of the optimization problem (2.9). Therefore, the $\mathcal{V}$-representation of $\underline{P}$ can be obtained as a non-redundant subset of those points in defining $\mathrm{Conv}(\{\vec{v}_1, \ldots, \vec{v}_m\})$.

We give two examples of over- and under-approximation in Figure 2.8 and 2.9. It can be seen that the quality of a polytopic approximation highly depends on the given template. In Chapter 4, several heuristics are proposed to achieve a good accuracy in doing such jobs.

2.4.2 Zonotopes

A zonotope is a special polytope which can be defined as the image of the unit box $[-1, 1]^m$ for some $m \geq 0$ under an affine mapping.

Definition 2.4.6 (Zonotope). *A* zonotope *is defined by the set*

$$\mathcal{Z} = \{G \cdot \vec{x} + \vec{c} \,|\, \vec{x} \in [-1, 1]^m\} \tag{2.10}$$

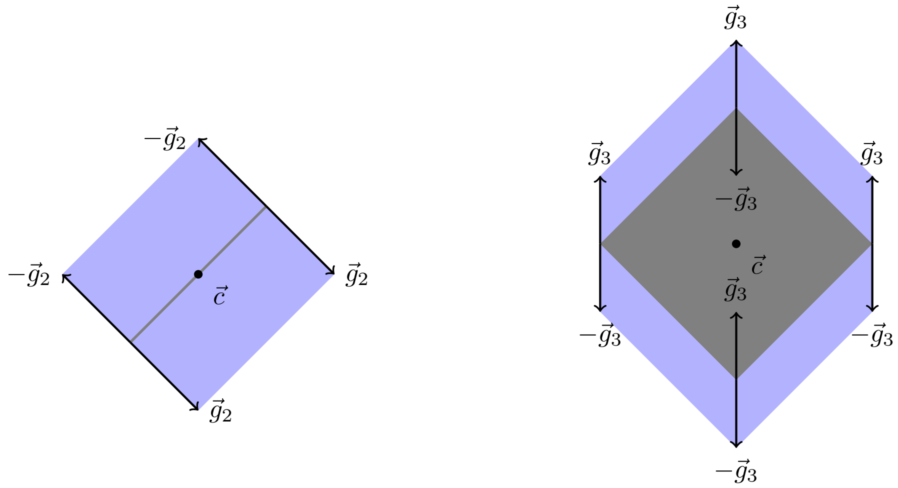
for some $G \in \mathbb{R}^{n \times m}$*,* $\vec{c} \in \mathbb{R}^n$*, and* $m, n \geq 0$*.*

Other than the $\mathcal{H}$- and $\mathcal{V}$-representations for general polytopes, a zonotope can be represented by the matrix G along with the vector $\vec{c}$. More precisely, it can be represented by the tuple $\mathcal{Z} = (\vec{c}, < \vec{g}_1, \ldots, \vec{g}_m >)$ wherein $\vec{g}_1, \ldots, \vec{g}_m$ are the columns of G. Such a representation is called *$\mathcal{G}$-representation (generator representation)* in which $\vec{c}$ is the *center* and $\vec{g}_1, \ldots, \vec{g}_m$ are the *generators*.

On the other hand, the zonotope $\mathcal{Z}$ can also be viewed as the Minkowski sum of the center $\vec{c}$ and the line segments defined by $L_i = \{x \cdot g_i \,|\, x \in [-1, 1]\}$ for $1 \leq i \leq m$. The translation from a $\mathcal{G}$-representation to either a $\mathcal{H}$- or $\mathcal{V}$-representation is difficult, some techniques are discussed elsewhere [Zie95, ASB10, Alt10].

Example 2.4.7. *We give an example of how to construct a zonotope based on its center and generators. Assume that the center is* $\vec{c} = (2, 3)$ *and the generators are given by* $\vec{g}_1 =$

(a) Adding the center (b) Adding the generator $\vec{g}_1$



(c) Adding the generator $\vec{g}_2$ (d) Adding the generator $\vec{g}_3$

Figure 2.10: Construct a zonotope based on the center and generators

$(1,1)$, $\vec{g}_2 = (1,-1)$ *and* $\vec{g}_3 = (0,1)$. *The zonotope* $\mathcal{Z} = ((2,3), < (1,1), (1,-1), (0,1) >)$ *is constructed by the steps shown in Figure 2.10.*

Zonotopes are closed under the operations of linear mapping and Minkowski sum, for both of which the computation can be done efficients on $\mathcal{G}$-representations. Given a zonotope $\mathcal{Z} = (\vec{c}, < \vec{g}_1, \ldots, \vec{g}_m >)$. The image of $\mathcal{Z}$ under the linear mapping $\pi : \vec{x} \mapsto A \cdot \vec{x}$ is still a zonotope whose $\mathcal{G}$-representation is given by

$$\pi(\mathcal{Z}) = (A \cdot \vec{c}, < A \cdot \vec{g}_1, \ldots, A \cdot \vec{g}_m >) \ .$$

Given two zonotopes $\mathcal{Z}_1 = (\vec{c}_1, < \vec{g}_1, \ldots, \vec{g}_m >)$ and $\mathcal{Z}_2 = (\vec{c}_2, < \vec{h}_1, \ldots, \vec{h}_k >)$. The Minkowski sum of $\mathcal{Z}_1$ and $\mathcal{Z}_2$ is also a zonotope whose $\mathcal{G}$-representation is

$$\mathcal{Z}_1 \oplus \mathcal{Z}_2 = (\vec{c}_1 + \vec{c}_2, < \vec{g}_1, \ldots, \vec{g}_m, \vec{h}_1, \ldots, \vec{h}_k >) \ .$$

Zonotopes are however not closed under the intersection with any set given by a polytope, zonotope or even hyperplane. Some techniques for deriving a zonotopic over-approximation for such an intersection are described in [GL08, Alt10].

Lemma 2.4.8. *An order* 1 *TM is a zonotope and vice versa.*

Proof. Given an order 1 TM (p, I), the range of p is the image of the interval domain under the linear mapping defined by p. Then it is a zonotope. Since the range of (p, I) is the Minkowski sum of the range of p and the interval I, it is also a zonotope.

In the other direction, since a zonotope is the image of a unit box under an affine mapping, it can be expressed in the form of an order 1 TM whose remainder interval is zero. □

We show that the translations between order 1 TMs and $\mathcal{G}$-representations are easy. Given a TM (p, I) such that p is a linear polynomial over the variables $\vec{x}$ which range in an interval $D \in \mathbb{IR}^m$ for some $m \geq 0$. Assume that the dimension of the TM is n. The $\mathcal{G}$-representation of the TM range can be obtained as follows.

We first construct the zonotope for the range of p. We reformulate p as a polynomial $q(\vec{y})$ with $\vec{y} \in [-1, 1]^m$ such that the ranges of p, q are same. Then the center of the zonotope is given by the constant part of q. For $1 \leq i \leq m$ and $1 \leq j \leq n$, the j-th component of the generator $\vec{g}_i$ is the coefficient of y_i in the j-th component of q. For example, the polynomial $p = (1 + x_1 - x_2, x_3 - x_1)$ of dimension 2 over $x_1 \in [1, 3]$, $x_2 \in [-1, 1]$ and $x_3 \in [-1, 0]$ can be reformulated as $q = (3 + y_1 - y_2, -2.5 + 0.5y_3 - y_1)$ with $y_1, y_2, y_3 \in [-1, 1]$. Then the $\mathcal{G}$-representation for the range of q is $((3, -2.5), < (1, -1), (-1, 0), (0, 0.5) >)$.

Then the $\mathcal{G}$-representation of the range of (p, I) can be obtained by computing the Minkowski sum of the above zonotope and I.

The translation in the other direction is easier. Given a zonotope $\mathcal{Z} = (\vec{c}_1, < \vec{g}_1, \ldots, \vec{g}_m >)$ which is of dimension n, an equivalent order 1 TM is of the form $(\vec{c} + p(\vec{x}), 0)$ with $\vec{x} \in [-1, 1]^m$. It is derived by setting the coefficient of x_i in the j-th component of p by $\vec{g}_i[j]$ for $1 \leq i \leq m$ and $1 \leq j \leq n$.

2.4.3 Ellipsoids

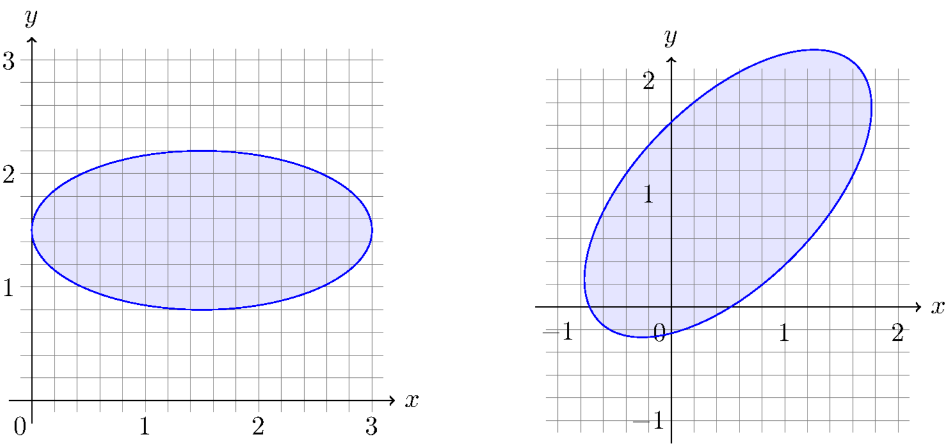
As another class of representations, an n-dimensional ellipsoid consists of the points in $\mathbb{R}^n$ which satisfy a given polynomial inequality of degree 2. The definition is presented as below.

Definition 2.4.9 (Ellipsoid)**.** *An* ellipsoid $\mathcal{E}$ *in* $\mathbb{R}^n$ *is defined by the set*

$$\mathcal{E} = \{\vec{x} \in \mathbb{R}^n \mid (\vec{x} - \vec{c})^T \cdot Q \cdot (\vec{x} - \vec{c}) \leq 1\} \tag{2.11}$$

wherein $\vec{c} \in \mathbb{R}^n$ *is the* center *of* $\mathcal{E}$ *and* $Q \in \mathbb{R}^{n \times n}$ *is the* shape matrix *which is* positive definite, *i.e.,* $\vec{x}^T \cdot Q \cdot \vec{x} > 0$ *for all nonzero* $\vec{x} \in \mathbb{R}^n$. *We denote it by* $\mathcal{E} : (\vec{c}, Q)$.

Intuitively, the eigenvectors of Q define the principal axes of $\mathcal{E}$ and the eigenvalues of Q are the reciprocals of the squares of the semi-axes.



(a) Ellipsoid $\mathcal{E}_1$ (b) Ellipsoid $\mathcal{E}_2$

Figure 2.11: Examples of ellipsoids

Example 2.4.10. *We present two examples of ellipsoids. In Figure 2.11(a), the ellipsoid $\mathcal{E}_1$ is defined by*

$$\mathcal{E}_1 = \{\vec{x} \in \mathbb{R}^2 \mid (\vec{x} - (1.5, 1.5))^T \cdot \begin{pmatrix} \frac{1}{2.25} & 0 \\ 0 & \frac{1}{0.49} \end{pmatrix} \cdot (\vec{x} - (1.5, 1.5))\} \ .$$

For the ellipsoid shown in Figure 2.11(b), the definition is given by

$$\mathcal{E}_2 = \{\vec{x} \in \mathbb{R}^2 \mid (\vec{x} - (0.5, 1))^T \cdot \begin{pmatrix} 0.9765625 & 0.5859375 \\ 0.5859375 & 0.9765625 \end{pmatrix} \cdot (\vec{x} - (0.5, 1))\} \ .$$

The emptiness of an ellipsoid $\mathcal{E} : (\vec{c}, Q)$ can be verified by solving the convex feasibility problem

$$\text{find } \vec{x} \in \mathbb{R}^n \quad \text{subject to} \quad (\vec{x} - \vec{c})^T \cdot Q \cdot (\vec{x} - \vec{c}) \leq 1 \ .$$

The ellipsoid is empty if and only if no solution is found.

For the operators we introduced on polytopes, ellipsoids are only closed under affine mapping and the intersection with a hyperplane. Given an ellipsoid $\mathcal{E} : (\vec{c}, Q)$, its image under an affine mapping $\pi : \vec{x} \mapsto M \cdot \vec{x} + \vec{b}$ is the ellipsoid defined by

$$\pi(\mathcal{E}) : \ (M \cdot \vec{c} + \vec{b}, (M \cdot Q^{-1} \cdot M^T)^{-1}) \ .$$

Some exact and approximate methods to compute the other operations on ellipsoids are described elsewhere [KV00, KV06, RST02].

However, ellipsoids are not closed under intersection, Minkowski sum and convex hull. Some over- and under-approximation techniques for those operations are given in [KV00, KV06, RST02].

2.4.4 Support functions

Support functions can be used as symbolic representations for convex sets. If two convex and compact sets $S_1, S_2 \subseteq \mathbb{R}^n$ have the same support function, then $S_1 = S_2$.

Definition 2.4.11 (Support function). *Given a set S in the n-dimensional Euclidean space $\mathbb{R}^n$, the* support function $\rho_S : \mathbb{R}^n \to \mathbb{R} \cup \{-\infty, \infty\}$ *is defined by*

$$\rho_S(\vec{l}) = \sup\{\vec{l}^{\,T} \cdot \vec{x} \mid \vec{x} \in S\} \ . \tag{2.12}$$

Support function of a polytope. The support function value of a polytope in either $\mathcal{H}$- or $\mathcal{V}$-representation w.r.t. a vector can be computed by LP. Given $P : A \cdot \vec{x} \leq \vec{b}$, the support function value $\rho_P(\vec{l})$ for any constant vector $\vec{l}$ can be obtained by solving

$$\sup\{\vec{l}^{\,T} \cdot \vec{x}\} \quad \text{subject to} \quad A \cdot \vec{x} \leq \vec{b} \ .$$

For a $\mathcal{V}$-representation $P : \{\vec{v}_1, \ldots, \vec{v}_m\}$, the support function value $\rho_P(\vec{l})$ can be obtained by solving

$$\sup\{\vec{l}^{\,T} \cdot \vec{x}\} \quad \text{subject to} \quad \vec{x} = \sum_{i=1}^{m} \lambda_i \cdot \vec{v}_i \wedge \bigwedge_{i=1}^{m} (\lambda_i \in [0,1]) \wedge \sum_{i=1}^{m} \lambda_i = 1 \ .$$

Support function of a zonotope. Although the support function value of a zonotope in $\mathcal{G}$-representation w.r.t. a vector $\vec{l}$ may be derived by LP, it is not necessary. Instead, an easier way to do that is computing a vector $\vec{x}^*$ which is the sum of the zonotope center and those vectors $\vec{v}$ such that either $\vec{v}$ or $-\vec{v}$ is a generator and $\vec{l}^{\,T} \cdot \vec{v} > 0$. Then the support function value can be computed by $\vec{l}^{\,T} \cdot \vec{x}^*$.

Support function of an ellispoid. The support function value of an ellipsoid $\mathcal{E}$: $(\vec{c}, Q)$ w.r.t. a vector $\vec{l}$ can be computed by

$$\rho_{\mathcal{E}}(\vec{l}) = \vec{l}^{\,T} \cdot \vec{c} + \sqrt{\vec{l}^{\,T} \cdot Q^{-1} \cdot \vec{l}} \ .$$

Support function representations are closed under affine mapping, Minkowski sum and convex hull [Le 09]. Given the support function ρ_S of a set S, the support function of its image under an affine mapping $\pi : \vec{x} \mapsto M \cdot \vec{x} + \vec{c}$ is given by

$$\rho_{\pi(S)}(\vec{l}) = \rho_S(M^T \cdot \vec{l}) + \vec{l}^{\,T} \cdot \vec{c} \ .$$

Given two sets S_1, S_2 which are represented by their support functions ρ_{S_1}, ρ_{S_2} respectively. The support function of $S_1 \oplus S_2$ can be computed by

$$\rho_{S_1 \oplus S_2}(\vec{l}) = \rho_{S_1}(\vec{l}) + \rho_{S_2}(\vec{l})$$

and the support function of $\mathrm{Conv}(S_1 \cup S_2)$ is given by

$$\rho_{\mathrm{Conv}(S_1 \cup S_2)}(\vec{l}) = \sup\{\rho_{S_1}(\vec{l}), \rho_{S_2}(\vec{l})\} \ .$$

Nevertheless, the support function of the intersection of two support functions is usually difficult to compute.

Chapter 3

Taylor Model Flowpipes for Continuous Systems

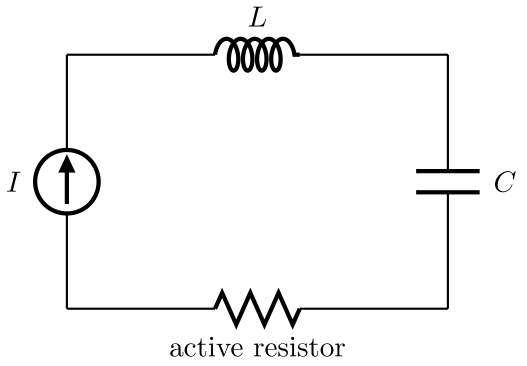
Continuous systems are mathematical formalisms for physical systems which exhibit only continuous behavior. The evolution of a continuous system from an initial state is characterized by a solution of an Ordinary Differential Equation (ODE). Then, to compute the reachable set of a continuous system, it is often needed to solve an initial value problem for the modeling ODE, and unfortunately, such a job can hardly be done explicitly since most ODEs do not have closed-form solutions. In this chapter, we focus on the methods to generate an over-approximation for the reachable set in a bounded time horizon for a continuous system. The over-approximation set is computed as finitely many Taylor Model flowpipes.

We briefly revisit the background of validated integration techniques, and then present the method of Taylor model integration which is originally developed by Berz and Makino [BM98, MB09]. The method shares the basic framework that is used in the interval-based integration method [NJC99] but shows better accuracy in most of the applications. We present our approach to compute Taylor model flowpipes. It follows the main outline proposed by Berz and Makino but contains new techniques and heuristics to improve the overall performance. To handle the ODEs with time-varying uncertainties, we present a Taylor model extension of the method for dealing with time-invariant uncertainties proposed by Lin and Stadtherr [LS07]. We prove that all behaviors in the time-varying case can be captured by our method. To improve the performance on linear ODEs, we propose an efficient method that combines the use of Taylor models and support functions. Its effectiveness is shown via a comparison with SpaceEx.

3.1 Continuous systems

Given a function g, we denote its time derivative by $\dot{g}$ or $\frac{dg}{dt}$. Then an ODE is given by the form $\dot{\vec{x}} = f(\vec{x}, t)$.

Definition 3.1.1 (Continuous system)**.** *A* continuous system $\mathcal{S}$ *is defined by an ODE* $\dot{\vec{x}} = f(\vec{x}, t)$ *wherein* $\vec{x}$ *are the* state variables *and* t *is the* time variable. *The function* f *is called the* vector field *of* $\mathcal{S}$ *which associates each state* $\vec{c}$ *a derivative vector* $f(\vec{c}, t)$ *at time* t.



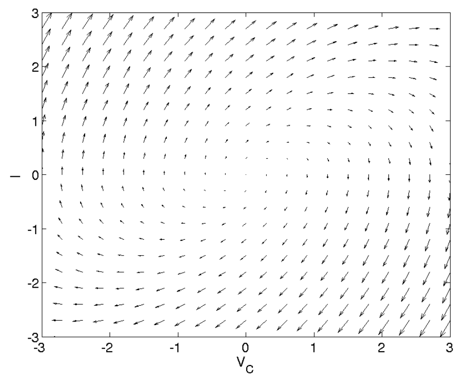


Figure 3.1: Van der Pol circuit

Figure 3.2: Vector field of the Van der Pol circuit

Example 3.1.2 (Van der Pol circuit)**.** *The Van der Pol circuit given in Figure 3.1 is a typical continuous system. It is an RLC circuit except that the resistor is active. It pumps energy into the system when the current is low and lets the energy dissipate when the current is high. The voltage of the active resistor is governed by*

$$V_R = -\mu I + I^3 \ .$$

Since the voltage drop across an inductor is proportional to the change rate of the current through it, then we have that $V_L = L \cdot \frac{dI}{dt}$*. For a capacitor, the current is proportional to the change rate of the voltage drop, hence we have that* $I = C \cdot \frac{dV_C}{dt}$*. Kirchoff's Voltage Law tells that the sum of the voltage drops around a loop is zero, therefore the following equation holds:*

$$-\mu I + I^3 + V_C + L \cdot \frac{dI}{dt} = 0 \ .$$

Hence, we may derive a 2-dimensional ODE

$$\begin{cases} \dot{V}_C &= \frac{1}{C} \cdot I \\ \dot{I} &= \frac{1}{L} \cdot \left(-V_C + \mu I - I^3\right) \end{cases}$$

which is named Van der Pol equation*. In Figure 3.2, we present the vector field of the Van der Pol circuit in the* V_C*-*I *plane such that the parameters are selected as*

$$C = 1, \quad L = 1, \quad \mu = 1 \ .$$

The (higher-order) Lie derivatives of a continuous function w.r.t. a vector field is given by Definition 3.1.3. As an example, assume that we have the ODE

$$\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = f(x,y) = \begin{pmatrix} -x + y \\ 1 - y^2 \end{pmatrix}$$

and a function $g(x, y, t) = t + x - y$. The Lie derivatives of g up to order 2 are given as

$$\begin{aligned} \mathcal{L}_f(g) &= (-x + y) - (1 - y^2) + 1 = -x + y + y^2 \\ \mathcal{L}_f^2(g) &= \mathcal{L}_f(-x + y + y^2) = -(-x + y) + (1 - y^2) + 2y(1 - y^2) \\ &= 1 + x + y - y^2 - 2y^3 \ . \end{aligned}$$

Definition 3.1.3 (Lie derivative). *Given an n-dimensional ODE $\dot{\vec{x}} = f(\vec{x}, t)$, the* (first-order) Lie derivative *of a differentiable function $g(\vec{x}, t)$ w.r.t. the vector field f is defined by*

$$\mathcal{L}_f(g) = \sum_{i=1}^{n} \left(\frac{\partial g}{\partial x_i} \cdot f_i \right) + \frac{\partial g}{\partial t}$$

wherein f_i denotes the i-th component of f. If g is k times differentiable, the higher-order Lie derivatives of it are defined recursively as

$$\mathcal{L}_f^{m+1}(g) = \mathcal{L}_f(\mathcal{L}_f^m(g)) \quad \textit{for} \quad m = 1, 2, \ldots, k-1 \ .$$

A function $g(t)$ is said to be a *solution* of the ODE $\dot{\vec{x}} = f(\vec{x}, t)$ in a time interval T containing 0 w.r.t. the initial condition $\vec{x} = \vec{x}_0$ when $t = 0$, if

(i) $g(0) = \vec{x}_0$, and

(ii) $\frac{dg}{dt}(t) = f(g(t), t)$ over $t \in T$.

By *Picard-Lindelöf theorem*, if the function f is Lipschitz continuous w.r.t. $\vec{x}$ in a neighborhood of $\vec{x}_0$ and continuous w.r.t. t, then the solution $\vec{x}(t)$ of the ODE w.r.t. the initial condition $\vec{x}(0) = \vec{x}_0$ is unique in an open time interval $(-\Delta(\vec{x}_0), \Delta(\vec{x}_0))$ wherein $\Delta(\vec{x}_0)$ is a constant depending on $\vec{x}_0$ (see [Mei07]). The theorem could also be applied to the case that f is *locally Lipschitz continuous* w.r.t. $\vec{x}$ and continuous w.r.t. t.

Definition 3.1.4 (Neighborhood). *Given a point $\vec{c} \in \mathbb{R}^n$ for some $n \in \mathbb{N}$. A set $U \subseteq \mathbb{R}^n$ is called a* neighborhood *of $\vec{c}$ if there exists an open set $V \subseteq \mathbb{R}^n$ such that $\vec{c} \in V \subseteq U$.*

Definition 3.1.5 (Lipschitz continuity). *We say that a function $f(\vec{x}, t)$ with $t \in T$ is* Lipschitz continuous *w.r.t. $\vec{x}$ in C, if there exists a real constant $L \geq 0$ such that for any $\vec{c}_1, \vec{c}_2 \in C$ and $s \in T$ the following inequality holds.*

$$\|f(\vec{c}_1, s) - f(\vec{c}_2, s)\| \leq L \cdot \|\vec{c}_1 - \vec{c}_2\| \tag{3.1}$$

We also call f locally Lipschitz continuous *w.r.t. $\vec{x}$ if for any $\vec{c} \in C$ there exists a* neighborhood *U of $\vec{c}$ such that f is Lipschitz continuous w.r.t. $\vec{x}$ in U.*

Theorem 3.1.6 (Picard-Lindelöf theorem). *Given an n-dimensional ODE $\dot{\vec{x}} = f(\vec{x}, t)$. If $f(\vec{x}, t)$ is Lipschitz continuous w.r.t. the variables $\vec{x}$ in some open set $C \subseteq \mathbb{R}^n$ and continuous w.r.t. t in an interval T containing 0, then for any $\vec{x}_0 \in C$, the solution $\vec{x}(t)$ w.r.t. $\vec{x}(0) = \vec{x}_0$ is* unique *over $t \in (-\Delta(\vec{x}_0), \Delta(\vec{x}_0)) \subseteq T$ for some real value $\Delta(\vec{x}_0) > 0$ depending on $\vec{x}_0$.*

Example 3.1.7. *The ODE $\dot{x} = 0.5x$ has the unique solution $x(t) = \exp(0.5t) \cdot c$ over $t \in \mathbb{R}$ for all initial state $x(0) = c \in \mathbb{R}$. However, the ODE $\dot{x} = \sqrt{2|x|}$ has the following two solutions w.r.t. the initial condition $x(0) = 0$:*

(1) $\quad x(t) = 0$ $\qquad\qquad$ *(2)* $\quad x(t) = \begin{cases} 0.5t^2, & t \geq 0 \\ -0.5t^2, & t < 0 \end{cases}$

since the function $\sqrt{2|x|}$ is not Lipschitz continuous over any set containing 0.

For the continuous systems under our consideration, we always assume that the vector fields are locally Lipschitz continuous w.r.t. the state variables and continuous w.r.t. the time variable. Therefore, given a time interval $[-\Delta, \Delta] \in \mathbb{IR}$, if a solution is ensured to exist there, then it is also unique (see [Mei07]). Besides the notation $\vec{x}(t)$, we sometimes denote $\varphi_f(\vec{x}_0, t)$ as the unique solution of the ODE $\dot{\vec{x}} = f(\vec{x}, t)$ w.r.t. $\vec{x}(0) = \vec{x}_0$ at time t. It is also known as the *flow* of the continuous system from $\vec{x}_0$. For convenience, we denote the set of flows $\{\varphi_f(\vec{x}_0, t) \mid \vec{x}_0 \in X_0, t \in \Delta\}$ collectively by $\varphi_f(X_0, \Delta)$ when $\varphi_f(\vec{x}_0, t)$ all exists for $\vec{x}_0 \in X_0$ and $t \in \Delta$. The set is also called a *flowpipe*.

An *initial value problem (IVP)* is to find a solution $\vec{x}(t)$ over some time interval T containing 0 for an ODE $\dot{\vec{x}} = f(\vec{x}, t)$ and its initial condition $\vec{x}(0) = \vec{x}_0$. Then for a continuous system defined by $\dot{\vec{x}} = f(\vec{x}, t)$, the problem is similar to computing the *reachable set* $\varphi_f(X_0, t)$ for $t \in T$ according to a given initial set X_0. Since a continuous system is always defined by an ODE in the thesis, we sometimes also call continuous systems ODEs, or ODE solutions reachable sets.

Definition 3.1.8 (Continuous reachability problem). *Given an n-dimensional continuous system $\mathcal{S} : \dot{\vec{x}} = f(\vec{x}, t)$ and an initial set $X_0 \subseteq \mathbb{R}^n$. The* reachability problem *of $\mathcal{S}$ is to verify whether a given state $\vec{c} \in \mathbb{R}^n$ is* reachable *in a given time interval T. We also call the problem* bounded *when T is bounded.*

Since most IVPs do not have a closed-form solution, the reachability problem of a continuous system can hardly be solved explicitly. Hence, we seek to compute an approximation of the result. Two state-of-the-art approaches are widely used to generate approximations for ODE solutions. They are called *numerical integration* and *validated integration* (also named verified integration or guaranteed integration). Both of them can be applied to yielding approximations for the reachable sets of continuous systems.

Numerical integration. Numerical integration is a task to generate numerical solutions for IVPs. More precisely, it computes numerical approximations for ODE solutions. Lots of numerical integration techniques are proposed in the past, such as Euler's method, Taylor's method and Runge-Kutta method (see [AHS09]). Those methods compute an approximation for the solution at a time point by consecutively approaching it via finitely many *time steps* which are also called *integration steps*. Given an ODE $\dot{\vec{x}} = f(\vec{x}, t)$ as well as an initial condition $\vec{x}(0) = \vec{x}_0$, an approximation value $\vec{s}_t$ for the solution $\vec{x}(t)$ at some time $t > 0$ can be computed by the following iterations,

for all $i = 1, \ldots, N$ **do**
  Compute $\vec{s}_i$ as an approximation of $\varphi_f(\vec{s}_{i-1}, \delta_i)$;
end for

wherein $\vec{s}_0 = \vec{x}_0$ and $\delta_1, \ldots, \delta_N$ are called *time step-sizes* such that $\sum_{i=1}^{N} \delta_i = t$. The scheme is based on the fact that a unique solution $\varphi_f(\vec{x}_0, t)$ over a time interval T containing 0 satisfies

$$\varphi_f(\vec{x}_0, t_1 + t_2) = \varphi_f(\varphi_f(\vec{x}_0, t_1), t_2)$$

for $t_1, t_2, t_1 + t_2 \in T$. The reason to compute the approximation by iterations is that the step-size in each iteration can be made small enough to control the local approximation error, although the global error may still accumulate during the iterations. Numerical integration techniques may be used to generate time-bounded simulation paths for

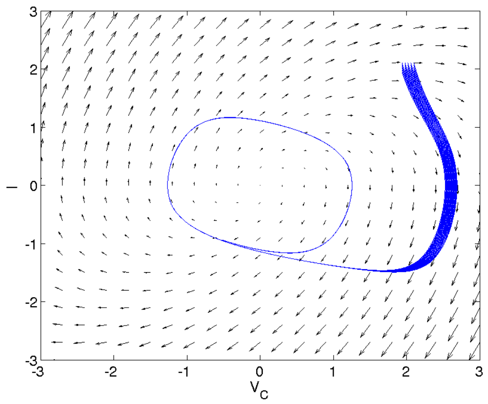


Figure 3.3: Numerical simulations

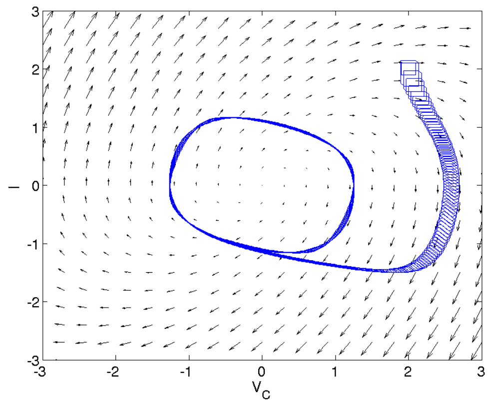


Figure 3.4: Validated integration

continuous systems. However, those approximation paths do not help to solve a reachability problem, since the approximation error is not guaranteed, and we still do not know whether a state is reachable or not.

Validated integration. Validated integration shares the same framework with numerical integration except that a *flowpipe over-approximation* is computed in every integration step to enclose the reachable set in a time segment. For the same example as above, the iterations become

for all i=1,...,N **do**
  Compute $\mathcal{F}_i$ as an over-approximation of $\varphi_f(\mathcal{F}_{i-1}, [0, \delta_i])$;
end for

wherein $\mathcal{F}_0 = \{\vec{x}_0\}$. Here, the initial set may also contain infinitely many states. Validated integration methods need to guarantee the inclusion of the exact solution in every integration step and are usually much more time-consuming than performing a numerical integration. The representation of a flowpipe over-approximation plays an important role in the overall computational performance. The sets on which general operations such as linear mapping, Minkowski sum are easy to compute are considered suitable. For continuous systems, a validated integration method may compute a reachable set over-approximation, and then any state that is not included is absolutely not reachable.

Example 3.1.9. *We consider the Van der Pol circuit given in Example 3.1.2. The initial set of interest is the box* $V_C(0) \in [1.9, 2.1]$ *and* $I(0) \in [1.9, 2.1]$. *In Figure 3.3, we illustrate the* 25 *simulation trajectories generated by* MATLAB *ode45, their starting points are uniformly distributed in the initial box. Figure 3.4 presents the over-approximation sets computed by* FLOW* *for the system behavior, each state which is not included is not reachable.*

3.2 High-level flowpipe construction schemes

We briefly revisit the high-level techniques for validated integration which is also called *flowpipe construction* in this thesis. We consider the schemes for linear and non-linear continuous systems separately, since the linear ones can be handled in more efficient ways.

3.2.1 Schemes for linear continuous systems

We concentrate on the autonomous linear continuous systems defined by homogeneous ODEs of the form $\dot{\vec{x}} = A\cdot\vec{x}$, since a non-homogeneous ODE $\dot{\vec{x}} = A\cdot\vec{x}+\vec{b}$ can be equivalently transformed to a homogeneous one by adding new state variable(s).

Given an n-dimensional linear continuous system $\mathcal{S} : \dot{\vec{x}} = f(\vec{x}) = A\cdot\vec{x}$ and an initial set $X_0 \subseteq \mathbb{R}^n$, the reachable set in the bounded time horizon $[0,\Delta]$ can be explicitly expressed by

$$\varphi_f(X_0, [0,\Delta]) \;=\; \{\exp(A\cdot t)\cdot\vec{x}_0 \,|\, \vec{x}_0 \in X_0, t \in [0,\Delta]\}$$

since $\vec{x}(t) = \exp(A\cdot t)\cdot\vec{x}_0$ is the solution of the ODE w.r.t. the initial condition $\vec{x}(0) = \vec{x}_0$. Unfortunately, the above set can hardly be computed exactly or even approximated accurately by one shot when the time interval $[0,\Delta]$ is large. We may however use flowpipe over-approximations to wrap the reachable set by segments. In each of them we only need to over-approximate the matrix $\exp(A\cdot t)$ over a small interval of t, and it can be done by using the methods of approximating matrix exponentials [MV03] and interval arithmetic.

The flowpipe construction for linear continuous systems can be carried out in various ways. Here, we present two popular schemes. The first one is given by Algorithm 4. For $1 \le i \le N$, the i-th flowpipe over-approximation $\mathcal{F}_i$ is an over-approximation of the reachable set in the time step $[\sum_{j=1}^{i-1}\delta_j, \sum_{j=1}^{i}\delta_j]$, it may be obtained by (conservatively) solving some optimization problems over the ODE solution set [CK98].

Algorithm 4 Flowpipe construction for linear continuous systems - Scheme I

Input: an ODE $\dot{\vec{x}} = A\cdot\vec{x}$, an initial set X_0, N step-sizes $\delta_1,\dots,\delta_N$ such that $\sum_{i=1}^{N}\delta_i = \Delta$
Output: an over-approximation $\mathcal{R}$ for the reachable set from X_0 in time $[0,\Delta]$

1: $\mathcal{R} \leftarrow \emptyset$;
2: **for all** $i = 1,\dots,N$ **do**
3: $\quad\mathcal{F}_i \leftarrow$ `overapprox`$(\{\exp(A\cdot t)\cdot\vec{x}_0 \,|\, \vec{x}_0 \in X_0, t \in [\sum_{j=1}^{i-1}\delta_j, \sum_{j=1}^{i}\delta_j]\})$;
4: $\quad\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{F}_i$;
5: **end for**
6: **return** $\mathcal{R}$;

In the second scheme, as presented by Algorithm 5, a flowpipe over-approximation, except the first one, is computed as the image of the previous one under the linear mapping $\pi : \vec{x} \mapsto \exp(A\cdot\delta)\cdot\vec{x}$. It is because of the fact that

$$\exp(A\cdot a)\cdot\exp(A\cdot b) = \exp(A\cdot(a+b))$$

for all constant $a,b \in \mathbb{R}$, and by the solution form of linear ODEs, we have that

$$\varphi_f(X_0, [i\cdot\delta, (i+1)\cdot\delta]) \;=\; \exp(A\cdot\delta)\cdot\varphi_f(X_0, [(i-1)\cdot\delta, i\cdot\delta])$$

for $1 \leq i \leq N-1$. By using the flowpipe representations which are closed under linear mappings, such as zonotopes [Gir05] and support functions [LG09], the second scheme often costs less time than the first one does, since there is no need to solve optimization problems in a time step.

Algorithm 5 Flowpipe construction for linear continuous systems - Scheme II

Input: an ODE $\dot{\vec{x}} = A \cdot \vec{x}$, an initial set X_0, a time horizon $[0, \Delta]$, a step-size δ
Output: an over-approximation $\mathcal{R}$ for the reachable set from X_0 in time $[0, \Delta]$

$N \leftarrow \lceil \frac{\Delta}{\delta} \rceil$;
$M_\delta \leftarrow$ `overapprox`$(\exp(A \cdot \delta))$; # M_δ is an interval matrix
$\mathcal{F}_1 \leftarrow$ `overapprox`$(\{\exp(A \cdot t) \cdot \vec{x}_0 \,|\, \vec{x}_0 \in X_0, t \in [0, \delta]\})$;
$\mathcal{R} \leftarrow \mathcal{F}_1$;
for all $i = 2, \ldots, N$ **do**
    $\mathcal{F}_i \leftarrow M_\delta \cdot \mathcal{F}_{i-1}$;
    $\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{F}_i$;
end for
return $\mathcal{R}$;

The two schemes are different tradeoffs between efficiency and accuracy. For Scheme I, the overall overestimation does not accumulate during the iterations since the flowpipe over-approximations are computed independently, although in each step we may need to solve some optimization problems which could be time-consuming. In Scheme II, except the first flowpipe over-approximation, we only need to recursively compute images of linear mappings. However the overall overestimation may accumulate since the matrix M_δ used in every iteration is an over-approximation of $\exp(A \cdot \delta)$. In both of the schemes, it is possible to arbitrarily reduce the overall overestimation by shrinking the step-sizes when those operations are implemented properly. In Section 3.6, we describe a novel method which combines the use of Taylor models and support functions in a variant of Scheme II.

3.2.2 General scheme for non-linear continuous systems

When a continuous system is defined by a non-linear ODE, the reachable set over-approximations often can not be computed based on the closed-form solutions. In this case, we may resort to Taylor approximations.

Given a non-linear continuous system $\mathcal{S} : \dot{\vec{x}} = f(\vec{x}, t)$ and an initial set X_0, the i-th flowpipe over-approximation $\mathcal{F}_i$ can be computed by the following procedure.

$\Omega_i \leftarrow$ `overapprox`$(\{p_k(\vec{x}_0, t) \,|\, \vec{x}_0 \in X_l, t \in [0, \delta_i]\})$;
if there is an interval I_l s.t. $\varphi_f(X_l, [0, \delta_i]) \subseteq \Omega_i \oplus I_l$ **then**
    $\mathcal{F}_i \leftarrow$ `overapprox`$(\Omega_i \oplus I_l)$;
else
    Terminate and return FAIL;
end if

wherein X_l denotes the *local initial set* which is X_0 when $i = 1$, and an over-approximation of $\varphi_f(X_0, \sum_{j=1}^{i-1} \delta_j)$ for $i > 1$. It can be derived form the previous flowpipe over-approximation. The value of δ_i is the i-th step-size. The polynomial p_k is a Taylor approximation of the solution in the current time step, and it can be obtained as

$$p_k(\vec{x}_0, t) = \vec{x}_0 + \mathcal{L}_f(\varphi_f(\vec{x}_0, t))|_{t=0} \cdot t + \cdots + \frac{1}{k!}\mathcal{L}_f^k(\varphi_f(\vec{x}_0, t))|_{t=0} \cdot t^k \ .$$

In practice, p_k is not necessarily to be the exact Taylor polynomial but should be made as close as possible. For example, an *Interval Taylor series (ITS)* is computed for p_k in [NJC99]. After over-approximating p_k, we should verify the existence of the solution over the time step $[0, \delta_i]$ and the initial set X_l. To do so, we try to find an interval I_l such that the result of bloating Ω_i by adding I_l contains a solution. Since the solution is unique by our assumption, the set $\Omega_i \oplus I_l$ is an over-approximation of the reachable set in $[0, \delta_i]$. On the other hand, if such an interval is not found, then the integration task fails. In that case, a remedy could be to decrease δ_i and try again. In the next section, we present a method to compute the flowpipe over-approximations which are represented by Taylor models.

3.3 Computing Taylor model flowpipes

3.3.1 Standard Taylor model integration method

We consider to compute the flowpipe over-approximations as Taylor Models (TMs) for a non-linear continuous system. They are also called *TM flowpipes*. Given an n-dimensional non-linear continuous system $\mathcal{S} :\ \dot{\vec{x}} = f(\vec{x}, t)$ and an interval or TM initial set $X_0 \subseteq \mathbb{R}^n$. Assume that the N time step-sizes are given by $\delta_1, \ldots, \delta_N$, then for $1 \le i \le N$, the i-th TM flowpipe is of the form $\mathcal{F}_i(\vec{x}_0, t) = (p_l(\vec{x}_0, t), I_l)$ such that $\vec{x}_0 \in X_0$ and $t \in [0, \delta_i]$. It is an over-approximation of the solution $\varphi_f(\vec{x}_0, \sum_{j=1}^{i-1} \delta_j + t)$, i.e.,

$$\varphi_f(\vec{x}_0, \sum_{j=1}^{i-1} \delta_j + t) \in p_l(\vec{x}_0, t) + I_l \quad \text{for all} \quad \vec{x}_0 \in X_0, t \in [0, \delta_i] \ .$$

The TM flowpipes may be of different orders. The method of computing TM flowpipes is called *TM integration*. We give a description of it as follows.

Given an integration task, the initial set X_l in the i-th time step is X_0 when $i = 1$, or given by the TM $\mathcal{F}_{i-1}(\vec{x}_0, \delta_{i-1})$ for $1 < i \le N$. We use the following two steps to compute the i-th TM flowpipe $\mathcal{F}_i$. We assume that the TM order is k.

Step 1. *Compute the order k Taylor polynomial of $\varphi_f(\vec{x}_l, t)$ at $t = 0$.* It can be obtained as

$$\begin{aligned} p_l(\vec{x}_l, t) \ &= \vec{x}_l + \mathrm{Trunc}_{k-1}(\mathcal{L}_f(\varphi_f(\vec{x}_l, t)))|_{t=0} \cdot t + \cdots \\ &\quad + \frac{1}{k!}\mathrm{Trunc}_0(\mathcal{L}_f^k(\varphi_f(\vec{x}_l, t)))|_{t=0} \cdot t^k \end{aligned} \tag{3.2}$$

wherein the operation $\mathrm{Trunc}_j(p)$ removes the terms of degrees $> j$ in the polynomial p. The truncated Lie derivatives in p_l can be computed iteratively, that is

$$\mathrm{Trunc}_{k-i}(\mathcal{L}_f^i(\varphi_f(\vec{x}_l, t))) = \mathrm{Trunc}_{k-i}(\mathcal{L}_f(\mathrm{Trunc}_{k-i+1}(\mathcal{L}_f^{i-1}(\varphi_f(\vec{x}_l, t)))))$$

for $i = 1, \ldots, k$, since the terms of degrees $\le (k - i)$ in $\mathcal{L}_f^i(\varphi_f(\vec{x}_l, t))$ are generated only by the terms of degrees $\le (k - i + 1)$ in $\mathcal{L}_f^{i-1}(\varphi_f(\vec{x}_l, t))$.

Definition 3.3.1 (Picard iteration)**.** *Given an ODE $\dot{\vec{x}} = f(\vec{x}, t)$ and an initial condition $\vec{x}(0) = \vec{x}_0$, the* Picard iteration *from a function $g_0(t)$ is defined by*

$$g_{i+1}(t) = \mathbb{P}_f(g_i) = \vec{x}_0 + \int_0^t f(g_i(s), s)ds \quad \textit{for } i \geq 0 \tag{3.3}$$

wherein $\mathbb{P}_f$ is called a Picard operator*.*

Besides the use of Lie derivatives, the polynomial p_l can also be generated by *Picard iteration* from $g_0(\vec{x}_l, t) = \vec{x}_l$. That is, we set an initial function $g_0(\vec{x}_l, t) = \vec{x}_l$, and apply the Picard operator as well as truncation operator to compute $p_l = g_k$:

$$g_{i+1}(\vec{x}_l, t) = \mathrm{Trunc}_i(\mathbb{P}_f(g_i)) \quad \text{for} \quad i = 0, \ldots, k-1$$

The application of the two methods on an ODE is given by Example 3.3.2.

Example 3.3.2. *Given a 2-dimensional ODE $\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = f(x, y) = \begin{pmatrix} 1+y \\ -x^2 \end{pmatrix}$. We want to compute the order 4 Taylor polynomial p_l for the solution. We first generate the Taylor polynomial up to order k by truncated Lie derivatives. For simplicity, we use x and y to denote the first and second components respectively for the function $\varphi_f(x_l, y_l, t)$.*

$$\begin{aligned}
&Trunc_3\left(\mathcal{L}_f\left(\begin{pmatrix} x \\ y \end{pmatrix}\right)\right) = \begin{pmatrix} 1+y \\ -x^2 \end{pmatrix} \\
&Trunc_2\left(\mathcal{L}_f^2\left(\begin{pmatrix} x \\ y \end{pmatrix}\right)\right) = Trunc_2\left(\mathcal{L}_f\left(\begin{pmatrix} 1+y \\ -x^2 \end{pmatrix}\right)\right) = \begin{pmatrix} -x^2 \\ -2x-2xy \end{pmatrix} \\
&Trunc_1\left(\mathcal{L}_f^3\left(\begin{pmatrix} x \\ y \end{pmatrix}\right)\right) = Trunc_1\left(\mathcal{L}_f\left(\begin{pmatrix} -x^2 \\ -2x-2xy \end{pmatrix}\right)\right) = \begin{pmatrix} -2x \\ -2-4y \end{pmatrix} \\
&Trunc_0\left(\mathcal{L}_f^4\left(\begin{pmatrix} x \\ y \end{pmatrix}\right)\right) = Trunc_0\left(\mathcal{L}_f\left(\begin{pmatrix} -2x \\ -2-4y \end{pmatrix}\right)\right) = \begin{pmatrix} -2 \\ 0 \end{pmatrix}
\end{aligned}$$

and then the order 4 Taylor expansion is given by

$$p_l(x_l, y_l, t) = \begin{pmatrix} x_l + t + y_l t - \frac{1}{2}x_l^2 t^2 - \frac{1}{3}x_l t^3 - \frac{1}{12}t^4 \\ y_l - x_l^2 t - x_l t^2 - \frac{1}{3}t^3 - x_l y_l t^2 - \frac{2}{3}y_l t^3 \end{pmatrix}$$

Then we turn to the second method. We start with the function $g_0(x_l, y_l, t)$ and apply *the Picard iteration with truncations.*

$$
\begin{aligned}
g_0(x_l, y_l, t) &= \begin{pmatrix} x_l \\ y_l \end{pmatrix} \\
g_1(x_l, y_l, t) &= Trunc_1\left(\begin{pmatrix} x_l \\ y_l \end{pmatrix} + \int_0^t \begin{pmatrix} 1 + y_l \\ -x_l^2 \end{pmatrix} ds\right) = \begin{pmatrix} x_l + t \\ y_l \end{pmatrix} \\
g_2(x_l, y_l, t) &= Trunc_2\left(\begin{pmatrix} x_l \\ y_l \end{pmatrix} + \int_0^t \begin{pmatrix} 1 + y_l \\ -(x_l + s)^2 \end{pmatrix} ds\right) = \begin{pmatrix} x_l + t + y_l t \\ y_l \end{pmatrix} \\
g_3(x_l, y_l, t) &= Trunc_3\left(\begin{pmatrix} x_l \\ y_l \end{pmatrix} + \int_0^t \begin{pmatrix} 1 + y_l \\ -(x_l + s + y_l s)^2 \end{pmatrix} ds\right) \\
&= \begin{pmatrix} x_l + t + y_l t \\ y_l - x_l^2 t - x_l t^2 - \frac{1}{3}t^3 \end{pmatrix} \\
g_4(x_l, y_l, t) &= Trunc_4\left(\begin{pmatrix} x_l \\ y_l \end{pmatrix} + \int_0^t \begin{pmatrix} 1 + y_l - x_l^2 s - x_l s^2 - \frac{1}{3}s^3 \\ -(x_l + s + y_l s)^2 \end{pmatrix} ds\right) \\
&= \begin{pmatrix} x_l + t + y_l t - \frac{1}{2}x_l^2 t^2 - \frac{1}{3}x_l t^3 - \frac{1}{12}t^4 \\ y_l - x_l^2 t - x_l t^2 - \frac{1}{3}t^3 - x_l y_l t^2 - \frac{2}{3}y_l t^3 \end{pmatrix}
\end{aligned}
$$

The function g_4 is the Taylor expansion p_l.

Step 2. *Evaluate a safe remainder interval I_l for the Taylor polynomial $p_l(\vec{x}_l, t)$ with $\vec{x}_l \in X_l$ and $t \in [0, \delta_i]$.* We try to find an interval I_l such that there exists a function $u(\vec{x}_l, t) \in p_l(\vec{x}_l, t) + I_l$ for all $\vec{x}_l \in X_l$ and $t \in [0, \delta_i]$, and

$$u(\vec{x}_l, t) = \vec{x}_l + \int_0^t f(u(\vec{x}_l, s), s) ds \tag{3.4}$$

i.e., $u(\vec{x}_l, t) = \varphi_f(\vec{x}_l, t)$. By our assumption, if the solution in $[0, \delta_i]$ exists then it is unique. Such a remainder interval could be verified by using Picard operator. Since the set of continuous functions $\{g(\vec{x}_l, t) \mid \vec{x}_l \in X_l, t \in [0, \delta_i]\}$ with the norm $|\cdot|$ defined by

$$|g| = \sup\{g(\vec{x}_l, t) \mid \vec{x}_l \in X_l, t \in [0, \delta_i]\}$$

forms a *Banach space*, and (p_l, I_l) for an interval I_l defines a convex and compact set of continuous functions, we can infer, by the *Schauder fixed point theorem*, that there is a function $u \in (p_l, I_l)$ which satisfies (3.4) when the Picard operator maps (p_l, I_l) to a subset of it. To detect this contractiveness, we may use TM arithmetic of order k. The result of a TM extension $\mathcal{P}_f((p_l, I_l))$ of $\mathbb{P}_f((p_l, I_l))$ also has the polynomial part p_l, therefore if its remainder is contained in I_l then the operator is contractive on (p_l, I_l).

Definition 3.3.3 (Convergence)**.** *Given an infinite sequence $s_1, s_2, s_3, \ldots$ in a normed vector space $(S, |\cdot|)$. We say that the sequence* converges *to a point $s^* \in S$ if $|s_i - s^*| \to 0$ when $i \to \infty$.*

Definition 3.3.4 (Cauchy sequence)**.** *An infinite sequence $s_1, s_2, s_3, \ldots$ is called a* Cauchy sequence *in a normed vector space $(S, |\cdot|)$, if for any $\varepsilon > 0$, there is a positive integer N such that for all positive integers $i, j \geq N$ we have that $|s_i - s_j| < \varepsilon$.*

Theorem 3.3.5 (Schauder fixed point theorem)**.** *Let U be a convex and compact set in a* Banach space $(S, |\cdot|)$ *which is a normed vector space whose Cauchy sequences are all convergent to an element in S. If $f : U \to U$ is* continuous*, then the mapping f has a fixed point in U, i.e., there is $u \in U$ such that $u = f(u)$.*

To find such an interval remainder I_l, we may first provide an estimation and successively enlarge it until we verify the contractiveness of the Picard operator. Afterwards, the interval may be refined by repeatedly applying the Picard operator to (p_l, I_l) using order k TM arithmetic. We show an example as below.

Example 3.3.6. *We consider the same ODE as that given by Example 3.3.2. Assume that the local initial set is given by the box $\{(x_l, y_l) \,|\, x_l \in [-1, 1] \wedge y_l \in [-0.5, 0.5]\}$, and the step-size is 0.02. We start with the remainder estimate $J_0 = \begin{pmatrix} [-0.1, 0.1] \\ [-0.1, 0.1] \end{pmatrix}$ for the order 3 Taylor expansion*

$$p_l(x_l, y_l, t) = \begin{pmatrix} x_l + t + y_l t \\ y_l - x_l^2 t - x_l t^2 - \frac{1}{3}t^3 \end{pmatrix}$$

We compute the following order 3 TM extension of the Picard operation,

$$\mathcal{P}_f((p_l, J_0)) = \begin{pmatrix} x_l \\ y_l \end{pmatrix} + \int_0^t \begin{pmatrix} 1 + (y_l - x_l^2 s - x_l s^2 - \frac{1}{3}s^3\,,\, [-0.1, 0.1]) \\ -((x_l + s + y_l s, [-0.1, 0.1]))^2 \end{pmatrix} ds$$

which yields the TM $\begin{pmatrix} x_l + t + y_l t & , & [-0.0023, 0.0023] \\ y_l - x_l^2 t - x_l t^2 - \frac{1}{3}t^3 & , & [-0.0060, 0.0060] \end{pmatrix}$ over $x_l \in [-1, 1]$, $y_l \in [-0.5, 0.5]$ and $t \in [0, 0.02]$. Then the remainder is refined to $J_1 = \begin{pmatrix} [-0.0023, 0.0023] \\ [-0.0060, 0.0060] \end{pmatrix}$.

For further refinement, we compute

$$\mathcal{P}_f((p_l, J_1)) = \begin{pmatrix} x_l \\ y_l \end{pmatrix} + \int_0^t \begin{pmatrix} 1 + (y_l - x_l^2 s - x_l s^2 - \frac{1}{3}s^3\,,\, [-0.0060, 0.0060]) \\ -((x_l + s + y_l s, [-0.0023, 0.0023]))^2 \end{pmatrix} ds$$

and it yields a smaller remainder interval $\begin{pmatrix} [-0.0002, 0.0002] \\ [-0.0003, 0.0003] \end{pmatrix}$.

At last, the i-th TM flowpipe $\mathcal{F}_i$ can be computed from evaluating $(p_l(X_l, t), I_l)$ by order k TM arithmetic. Notice that a safe remainder interval is not always guaranteed to be obtained, it might fail when either of the local initial set or the time step is too large.

In Figure 3.5 - 3.7, we give a visualization of the two steps as well as the computation of the local initial set in the next time step. The current local initial set is simply denoted by a point $\vec{x}_l$. We should mention that the size of the remainder interval in X_l does not necessarily increase along with the time steps. An example is given as follows.

Example 3.3.7. *We assume that*

$$\begin{aligned} (p_l(\vec{x}_l, t), I_l) &= (1 + 2x_l t^2 - 0.2x_l^2 t\,,\, [-0.1, 0.1]) \\ X_l &= (0.1 - x_0^2\,,\, [-0.5, 0.5]) \end{aligned}$$

wherein $x_0 \in [-1, 1]$ and $t \in [0, 0.2]$. To compute the order 2 TM for the local initial set in the next integration step, we first compute

$$(p_l'(\vec{x}_l), I_l) = (p_l(\vec{x}_l, 0.2), I_l) = (1 + 0.08x_l - 0.04x_l^2\,,\, [-0.1, 0.1])$$

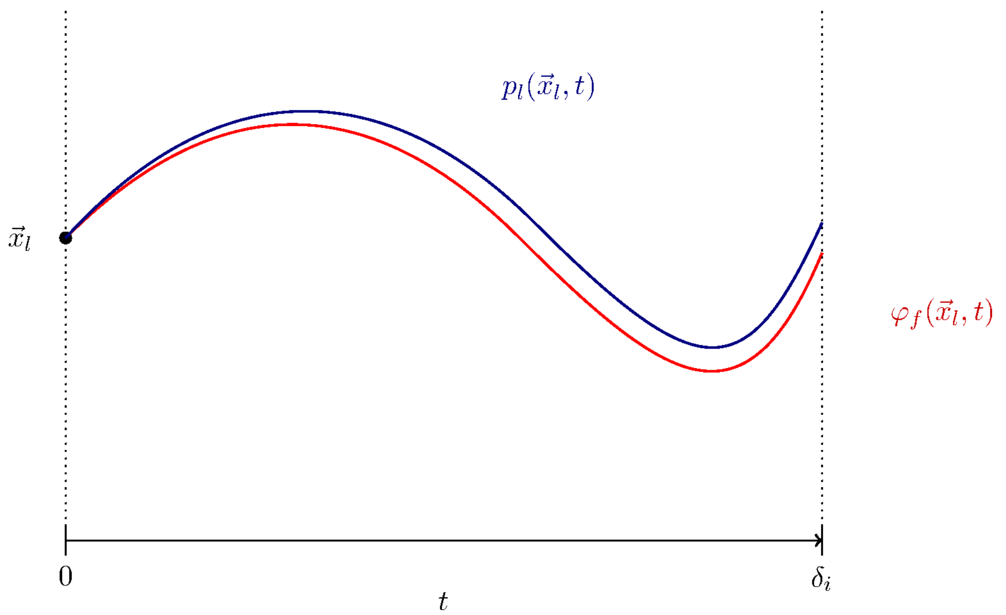


Figure 3.5: Step 1: Compute the order k Taylor approximation $p_l(\vec{x}_l, t)$

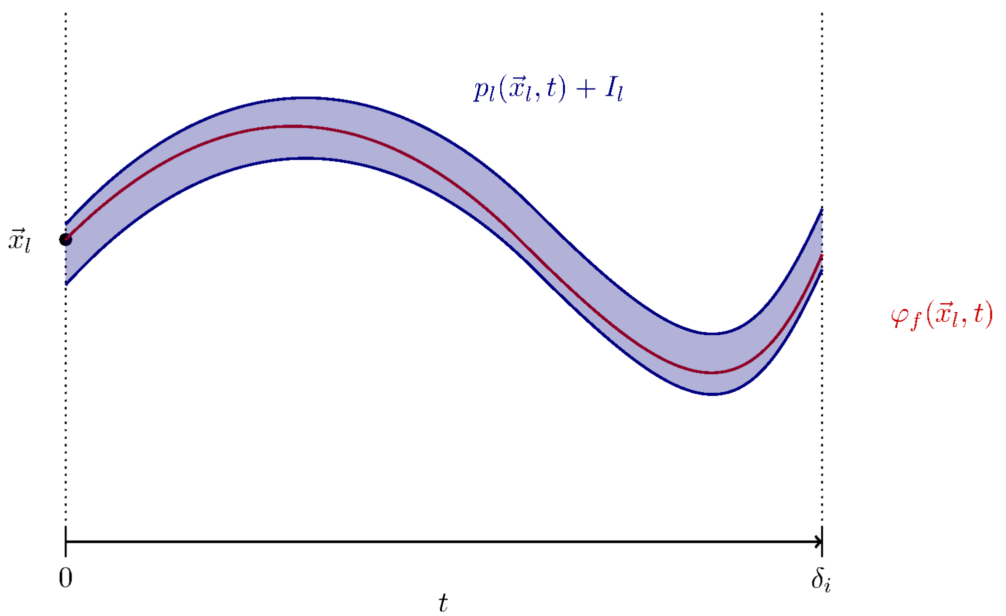


Figure 3.6: Step 2: Evaluate a proper remainder interval I_l

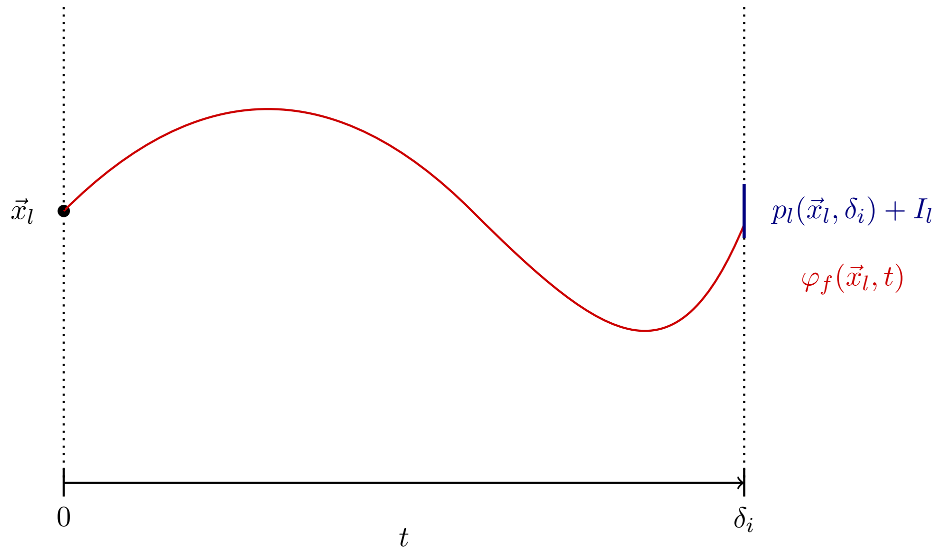


Figure 3.7: Compute the local initial set for the next time step

Then the substitution $(p'_l((0.1 - x_0^2, [-0.5, 0.5]))\,, I_l)$ *results in the TM*

$$(1.0076 - 0.072x_0^2\,,\ [-0.250, 0.250])$$

whose remainder size is smaller than that of $[-0.5, 0.5]$.

3.3.2 Preconditioned Taylor expansions

In a TM integration task, the remainders of the flowpipe over-approximations are often computed via lots of polynomial evaluations. As we pointed out in Chapter 2, the tightness of them is very sensitive to the evaluation schemes in use. In practice, for interval evaluating a polynomial p, it is often effective to limit the overestimation by first reformulating p over a set of variables each of which ranges in $[-1, 1]$ and then computing an interval for a Horner form of the result. The reason to do the reformulation is that $\mathrm{W}(I \cdot J) \leq \mathrm{W}(I' \cdot J)$ for any intervals I, I', J such that I is symmetric and $\mathrm{W}(I) = \mathrm{W}(I')$. We give an example as below.

Given two TMs $(-x+x^2, [-0.1, 0.1])$ and $(1+2x-x^2, [-0.1, 0.1])$, the domain of them is given by $x \in [2, 4]$. The order 2 TM of their product can be computed by

$$\begin{aligned}
&(-x + x^2, [-0.1, 0.1]) \cdot (1 + 2x - x^2, [-0.1, 0.1]) \\
=&\left(-x - x^2,\ \begin{array}{c} \mathtt{Int}(-x + x^2) \cdot [-0.1, 0.1] + [-0.1, 0.1] \cdot \mathtt{Int}(1 + 2 \cdot x - x^2) \\ +[-0.1, 0.1] \cdot [-0.1, 0.1] + \mathtt{Int}(3x^3 - x^4) \end{array}\right) \\
=&(-x - x^2, [-1.4, 1.4] + [-1.1, 1.1] + [-0.01, 0.01] + [-232, 176]) \\
=&(-x - x^2, [-234.51, 178.51])
\end{aligned}$$

which has a huge remainder interval whose width is 413.02. However, if we translate the two TMs equivalently to $(6+5y+y^2, [-0.1, 0.1])$ and $(-2-4y-y^2, [-0.1, 0.1])$ respectively over $y \in [-1, 1]$. The TM multiplication can be done by

$$
\begin{aligned}
&(6 + 5y + y^2, [-0.1, 0.1]) \cdot (-2 - 4y - y^2, [-0.1, 0.1]) \\
=&\left(12 - 14y - 24y^2, \quad \begin{array}{c} \mathtt{Int}(6 + 5y + y^2) \cdot [-0.1, 0.1] \\ +[-0.1, 0.1] \cdot \mathtt{Int}(-2 - 4y - y^2) \\ +[-0.1, 0.1] \cdot [-0.1, 0.1] + \mathtt{Int}(-9y^3 - y^4) \end{array} \right) \\
=&(-12 - 34y - 28y^2, [-1.2, 1.2] + [-0.7, 0.7] + [-0.01, 0.01] + [-10, 9]) \\
=&(-12 - 34y - 28y^2, [-11.91, 10.91])
\end{aligned}
$$

whose remainder width is only 22.82.

We consider to equivalently express the Taylor expansion p_l over another group of variables $\vec{y}_l$ which range in a subset of the unit box $[-1, 1]^n$. The methods to do that are called *preconditioning techniques*, and we only consider the translation form $\vec{x}_l = \vec{c}_l + A_l \cdot \vec{y}_l$ for $\vec{c}_l \in \mathbb{R}^n$ and $A_l \in \mathbb{R}^{n \times n}$ which is an invertible matrix. We call $\vec{c}_l$, A_l *preconditioning parameters*.

Many techniques are proposed to compute the preconditioning parameters. For example, the QR preconditioning technique for interval-based integration may also be applied to the TM case (see [Loh92, MB05]). When a preconditioning technique is used, we will meet the problem that is how to efficiently compute the preconditioned Taylor expansion for given $\vec{c}_l$ and A_l.

We introduce three approaches to compute a preconditioned Taylor expansion. The simplest one is to use Picard iteration from the initial function $g_0(\vec{y}_l, t) = \vec{c}_l + A_l \cdot \vec{y}_l$. We call it *Approach I*, it involves lots of variable substitutions which might be very time-consuming when either the vector field is a high-degree polynomial or the TM order is large. Hence, we also propose the use of the following two approaches for the situations in which Approach I does not work efficiently.

1. *Approach II: Based on a polynomial template.* Although the preconditioning parameters $\vec{c}_l$, A_l are different from step to step, the corresponding Taylor expansions can always be computed by

$$
\begin{aligned}
p_l(\vec{y}_l, t) =&\vec{c}_l + A_l \cdot \vec{y}_l + \mathrm{Trunc}_{k-1}((\mathcal{L}_f(\varphi_f(\vec{x}_l, t))|_{t=0})|_{\vec{x}_l = \vec{c}_l + A_l \cdot \vec{y}_l}) \cdot t + \cdots \\
&+ \frac{1}{k!} \mathrm{Trunc}_0((\mathcal{L}_f^k(\varphi_f(\vec{x}_l, t))|_{t=0})|_{\vec{x}_l = \vec{c}_l + A_l \cdot \vec{y}_l}) \cdot t^k
\end{aligned}
$$

   for which we may first derive a polynomial template over $\vec{x}_l$ and t,

$$
p_T(\vec{x}_l, t) = \vec{x}_l + \mathcal{L}_f(\varphi_f(\vec{x}_l, t))|_{t=0} \cdot t + \cdots + \frac{1}{k!} \mathcal{L}_f^k(\varphi_f(\vec{x}_l, t))|_{t=0} \cdot t^k
$$

   then replace $\vec{x}_l$ by $\vec{c}_l + A_l \cdot \vec{y}_l$ and truncate the terms of degrees $> k$. Note that the template p_T has nothing to do with $\vec{c}_l$, A_l, it then can be computed only once and reused in every time step.

   Unfortunately, such a scheme may also be time-costly, since no truncation is applied on the Lie derivatives in the template p_T. In the case that the dimension n is large and the vector field f has a high-degree polynomial part, the size of p_T could be prohibitively large.

Method	Degree of the vector field	TM order
Approach I	low	low
Approach II	low	high
Approach III	high	high

Table 3.1: Situations for applying different approaches to compute Taylor polynomials

2. *Approach III: Based on the preconditioned Taylor approximations.* The idea is similar to that described in [MB09]. By the precondition of the initial set $\vec{x}_l = \vec{c}_l + A_l \cdot \vec{y}_l$, we may derive a preconditioned solution

$$\varphi_f(\vec{x}_l, t) = \varphi_f(\vec{c}_l, t) + A_l \cdot \varphi_h(\vec{y}_l, t)$$

wherein the evolution from $\vec{y}_l$ is determined by another vector field h. It denotes the evolution of the preconditioned difference between the trajectories from $\vec{x}_l$ and $\vec{c}_l$. We give an intuitive explanation in Figure 3.8.

Then the Taylor expansion p_l can be derived as

$$p_l(\vec{y}_l, t) = p_{\vec{c}}(t) + A_l \cdot p_{\vec{y}}(\vec{y}_l, t)$$

wherein $p_{\vec{c}}$ is the order k Taylor polynomial of $\varphi_f(\vec{c}_l, t)$ and $p_{\vec{y}}$ is the order k Taylor polynomial of $\varphi_h(\vec{y}_l, t)$. Since $p_{\vec{c}}$ is univariate, we may compute it by mere Picard iteration and the cost is low. For $p_{\vec{y}}$, we may compute the following polynomial

$$p_{\vec{y}}(\vec{y}_l, t) = \vec{y}_l + \mathrm{Trunc}_{k-1}(\mathcal{L}_h(\varphi_h(\vec{y}_l, t)))|_{t=0} \cdot t + \cdots + \frac{1}{k!}\mathrm{Trunc}_0(\mathcal{L}_h^k(\varphi_h(\vec{y}_l, t)))|_{t=0} \cdot t^k$$

wherein h is unknown but can be replaced by an order $(k-1)$ Taylor polynomial p_h of it. That is,

$$\begin{aligned} p_h(\vec{y}, t) =& A_l^{-1} \cdot (\frac{d\varphi_f(\vec{x}_l, t)}{dt} - \frac{dp_{\vec{c}}}{dt}) \\ =& A_l^{-1} \cdot (f(\vec{x}, t) - \frac{dp_{\vec{c}}}{dt}) \\ =& A_l^{-1} \cdot (f(p_{\vec{c}} + A_l \cdot \vec{y}, t) - \frac{dp_{\vec{c}}}{dt}) \end{aligned}$$

As we mentioned before, the truncated Lie derivatives can be generated iteratively based on only lower-order terms, then we do not need to keep the whole part of a Lie derivative in the computation. Therefore, such an approach could have a lower time cost than either of the other two when the TM order is high.

This approach might not have a better performance than the other two when the TM order in use is low, since the preconditioned Taylor expansion could be computed more efficiently in a more direct way.

Table 3.1 provides a summary of the suitable situations for the three approaches. In Example 3.3.8, we show the computation of a preconditioned Taylor expansion by using all of the three approaches. To better understand their applicabilities, we provide Example 3.3.9 in which we evaluate their performance on a set of variants of a jet engine model. The experimental platform is a PC with an Intel Core i7-860 Processor (2.80 GHz), 4.0 Gigabyte memory and the operating system of Ubuntu Linux 12.04 LTS.

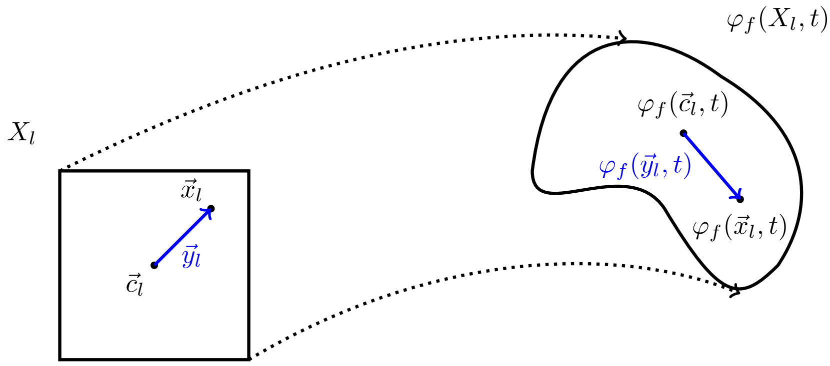


Figure 3.8: Example of the relationships among $\vec{x}_l$, $\vec{c}_l$ and $\vec{y}_l$. Here, the matrix A_l is identity.

Example 3.3.8. *We consider a 1-dimensional ODE $\dot{x} = f(x) = 2 + x^2$. The preconditioning parameters are given by $c_l = 1$ and $A_l = (0.5)$. We present the computation of the order 4 Taylor expansion p_l over y_l, t. We first follow the way of using Picard iteration, i.e., Approach I. The initial function is given by $g_0(y_l,t) = 1 + 0.5y_l$.*

$$\begin{aligned}
g_1(y_l,t) =& Trunc_1(1 + 0.5y_l + \int_0^t (2 + (1 + 0.5y_l)^2)ds) = 1 + 0.5y_l + 3t \\
g_2(y_l,t) =& Trunc_2(1 + 0.5y_l + \int_0^t (2 + (1 + 0.5y_l + 3t)^2)ds) = 1 + 0.5y_l + 3t + y_lt + 3t^2 \\
g_3(y_l,t) =& Trunc_3(1 + 0.5y_l + \int_0^t (2 + (1 + 0.5y_l + 3t + y_lt + 3t^2)^2)ds) \\
&= 1 + 0.5y_l + 3t + y_lt + 3t^2 + 0.25y_l^2t + 2.5y_lt^2 + 5t^3 \\
g_4(y_l,t) =& Trunc_4\left(\begin{array}{c} 1 + 0.5y_l + \\ \int_0^t (2 + (1 + 0.5y_l + 3t + y_lt + 3t^2 + 0.25y_l^2t + 2.5y_lt^2 + 5t^3)^2)ds \end{array}\right) \\
=& 1 + 0.5y_l + 3t + y_lt + 3t^2 + 0.25y_l^2t + 2.5y_lt^2 + 5t^3 + 0.75y_l^2t^2 \\
&+ 4.66667y_lt^3 + 7t^4
\end{aligned}$$

After 4 iterations we have the preconditioned Taylor expansion $p_l = g_4$.

When we use Approach II, the polynomial template is computed by

$$p_T = x_l + (2 + x_l^2)t + \frac{1}{2}(4x_l + 2x_l^3)t^2 + \frac{1}{6}(8 + 16x_l^2 + 6x_l^4)t^3 + \frac{1}{24}(64x_l + 80x_l^3 + 24x_l^5)t^4$$

By substituting $1 + 0.5y_l$ in the place of x_l and truncating the higher-order part, we obtain the same result that is

$$p_l = 1 + 0.5y_l + 3t + y_lt + 3t^2 + 0.25y_l^2t + 2.5y_lt^2 + 5t^3 + 0.75y_l^2t^2 + 4.6667y_lt^3 + 7t^4$$

Now we turn to Approach III. The order 4 Taylor polynomial for the solution $\varphi_f(1,t)$ is computed as $p_{\vec{c}} = 1 + 3t + 3t^2 + 5t^3 + 7t^4$ by Picard iteration. We compute the order 3 Taylor approximation for the vector field of $y(t) = \varphi_h(\vec{y}_l,t)$, that is

$$\begin{aligned}
p_h(y,t) =& Trunc_3(2 \cdot (2 + (1 + 3t + 3t^2 + 5t^3 + 7t^4 + 0.5y)^2 - 3 - 6t - 15t^2 - 28t^3)) \\
=& 2y + 0.5y^2 + 6yt + 6yt^2
\end{aligned}$$

Enhancement	TM order	Approach I (s)	Approach II (s)	Approach III (s)
+0	5	**2.2**	**2.3**	2.8
+0	8	22.2	**19.4**	23.7
+0	11	71.5	**47.3**	67.2
$+q_6$	5	**7.7**	11.8	**7.8**
$+q_6$	8	101.8	101.7	**86.4**
$+q_6$	11	512.9	480.7	**368.1**
$+q_6+q_9$	5	17.7	24.0	**14.1**
$+q_6+q_9$	8	234.1	226.5	**166.6**
$+q_6+q_9$	11	1531.9	1607.2	**914.4**

Table 3.2: Experiments on the jet engine model with different enhancements

Then the order 4 *Taylor approximation for* $\varphi(1+0.5y_l, t)$ *is computed by*

$$\begin{aligned}
p_{\vec{y}} =& y_l + Trunc_3(\mathcal{L}_h(y))|_{t=0} \cdot t + \frac{1}{2!} Trunc_2(\mathcal{L}_h^2(y))|_{t=0} \cdot t^2 \\
&+ \frac{1}{3!} Trunc_1(\mathcal{L}_h^3(y))|_{t=0} \cdot t^3 + \frac{1}{4!} Trunc_0(\mathcal{L}_h^4(y))|_{t=0} \cdot t^4 \\
=& y_l + (2y + 0.5y^2 + 6yt + 6yt^2)|_{t=0} \cdot t \\
&+ \frac{1}{2}(10y + 3y^2 + 36yt)|_{t=0} \cdot t^2 \\
&+ \frac{1}{6}(56y)|_{t=0} \cdot t^3 \\
&+ \frac{1}{24}(0)|_{t=0} \cdot t^4 \\
=& y_l + 2y_l t + 0.5y_l^2 t + 5y_l t^2 + 1.5y_l^2 t^2 + 9.3333y_l t^3
\end{aligned}$$

Hence, we have that

$$\begin{aligned}
p_l =& p_{\vec{c}} + (0.5) \cdot p_{\vec{y}} \\
=& 1 + 0.5y_l + 3t + y_l t + 3t^2 + 0.25y_l^2 t + 2.5y_l t^2 + 5t^3 + 0.75y_l^2 t^2 + 4.6667y_l t^3 + 7t^4
\end{aligned}$$

Example 3.3.9 (Jet engine)**.** *The Moore-Greitzer model of a jet engine is described in [APS08]. It is a* 2*-dimensional continuous system defined by*

$$\begin{cases} \dot{x} &= -y - 1.5x^2 - 0.5x^3 - 0.5 \\ \dot{y} &= 3x - y \end{cases}$$

The vector field is a (vector-valued) polynomial of degree 3*. We consider to enhance it by adding high-degree polynomial terms which are listed as follows.*

$$q_6(x,y) = \begin{pmatrix} 0.2x^3y^3 \\ 0.2x^3y^3 \end{pmatrix} \qquad q_9(x,y) = \begin{pmatrix} -0.1x^3y^6 - 0.1x^6y^3 \\ -0.1x^3y^6 - 0.1x^6y^3 \end{pmatrix}$$

In Table 3.2, we list the experimental results on those enhancements with different TM orders and approaches. The running time is obtained from computing 100 *flowpipes from the initial set* $x(0) \in [0.9, 1.1]$ *and* $y(0) \in [0.9, 1.1]$*.*

TM simplification. It is always necessary to simplify the representation of a TM. Since a polynomial of n variables and degree k may have as many as $\binom{n+k}{k}$ terms, we need to remove some terms in a polynomial part regularly during a computation but the resulting TM is still an over-approximation. To do so, we set a cutoff threshold $\varepsilon > 0$ such that if a term is verified to be contained in $[-\varepsilon, \varepsilon]$, we then remove it from the polynomial part and add its interval enclosure onto the remainder.

Non-polynomial terms in ODEs. If the vector field $f(\vec{x}, t)$ of the ODE contains non-polynomial terms, we recursively compute a TM (p_f, I_f) over the local initial set and time step on the structure of f, and use (p_f, I_f) instead of f in the subsequent computation.

We give Algorithm 6 as the framework of TM integration. In the algorithm, the operations of TM substitution and remainder refinement are considered the most time-consuming part. In the following content, we propose some methods to improve the efficiency.

Algorithm 6 TM integration for ODEs

Input: an ODE $\dot{\vec{x}} = f(\vec{x}, t)$, a TM initial set X_0, a time horizon $[0, \Delta]$, step-sizes $\delta_1, \ldots, \delta_N$ such that $\sum_{i=1}^{N} \delta_i = \Delta$, a positive integer k for the TM order in use
Output: an over-approximation $\mathcal{R}$ for the reachable set from X_0 in time $[0, \Delta]$

$\mathcal{R} \leftarrow \emptyset$;
$X_l \leftarrow X_0$;
for all $i = 1, \ldots, N$ **do**
  Compute the preconditioning parameters $\vec{c}_l$ and A_l;
  Derive an order k Taylor polynomial p_l according to the parameters;
  Come up with a remainder interval estimate I_l;
  while I_l is not a safe remainder interval for p_l in the time step $[0, \delta_i]$ **do**
    if $\mathrm{W}(I_l)$ exceeds a specified threshold **then**
      Terminate and return FAIL;
    else
      Enlarge I_l by multiplying a suitable scale factor;
    end if
  end while
  repeat
    $(p_l, I_l) \leftarrow \mathcal{P}_f((p_l, I_l))$; # by order k TM arithmetic
  until No big improvement on the last refinement of I_l
  $\mathcal{R} \leftarrow \mathcal{R} \cup \{((p_l, I_l), X_l)\}$;
  $X_l \leftarrow (p_l(X_l, \delta_i), I_l)$; # by order k TM arithmetic
end for
return $\mathcal{R}$;

Efficient TM substitutions. A TM substitution task is to replace the variables x_1, $\ldots$, x_n in a given TM (p, I) by a group of given TMs (q_1, I_1), $\ldots$, (q_n, I_n) respectively. In order to reduce the number of multiplications as well as avoid the dependency problem as much as possible, we first transform the polynomial p into a Horner form and then iteratively replace the variables by the corresponding TMs. The transformation procedure is presented in Algorithm 7 wherein the partial order of variables may be provided by a heuristic such as the one given in [CK04]. Note that the TMs (q_1, I_1), ..., (q_n, I_n) might be multiplied for several times, but it is only necessary to compute interval enclosures for the polynomials $q_1, \ldots, q_n$ in the first time. Therefore, before computing the substitution, we first scan the Horner form h, and then for $1 \le i \le n$, we compute an interval enclosure J_i for q_i such that x_i is multiplied by a clause for more than one time in h. Those computed intervals can be reused.

Algorithm 7 Function `Horner` of transforming a monomial form polynomial p into a Horner form

Input: a polynomial p in monomial form, a partial order $\prec$ on the variables
Output: a Horner form h of p

Set X as the set of variables in p;
Find $x \in X$ such that $x \prec y$ for all $y \in X \backslash \{x\}$;
Transform p into the form $q_0 + q_1 \cdot x + \cdots + q_k \cdot x^k$ such that $q_0, \ldots, q_k$ are polynomials not containing x;
$h \leftarrow \texttt{Horner}(q_0, \prec) + x \cdot (\texttt{Horner}(q_1, \prec) + \cdots + x \cdot (\texttt{Horner}(q_{k-1}, \prec) + x \cdot \texttt{Horner}(q_k, \prec)) \cdots);$
return h;

3.3.3 Fast remainder refinement

An iteration of the remainder refinement requires to compute a TM Picard operation of some order k. Such an operation involves TM substitution and is therefore time-consuming in general. However, as we mentioned before, the order k TM Picard operator on the solution over-approximation (p_l, I_l) results in a TM (p_l', I_l') such that $p_l = p_l'$ and $I_l' \subseteq I_l$. Then in all refinement iterations, we are dealing with the same polynomial input. It gives us the motivation to find a method that only deals with the remainder intervals.

Since the TM Picard operations $\mathcal{P}_f((p, I))$ and $\mathcal{P}_f((p, J))$ share the same interim interval results which are only related to p, we may keep those interim intervals by a queue in the first refinement iteration, and reuse them in the remaining ones. To do so, we give our method for each operator as follows.

- *Addition.* Since the remainder of the resulting TM is computed only from the remainders of the operands, there is no need to keep any interval.
- *Multiplication.* The multiplication of two TMs (p_1, I_1) and (p_2, I_2) is computed by
$$(p_1, I_1) \cdot (p_2, I_2) \;=\; (p_1 \cdot p_2 - p_e \;,\; \texttt{Int}(p_1) \cdot I_2 + I_1 \cdot \texttt{Int}(p_2) + I_1 \cdot I_2 + \texttt{Int}(p_e))$$
wherein p_e consists of the terms of degrees $> k$ in $p_1 \cdot p_2$. The interim intervals related to p_1, p_2 are $\texttt{Int}(p_1)$, $\texttt{Int}(p_2)$ and $\texttt{Int}(p_e)$, and they can be reused in computing the product of other two TMs (p_1, J_1), (p_2, J_2). Then we keep them in the queue.
- *Non-polynomial terms.* For a non-polynomial term $\phi(x)$ such that the variable x ranges in a TM (p, I), we first compute the Taylor expansion $q(x)$ of ϕ at $x = \mathrm{Mid}((p, I))$, and then evaluate a safe remainder interval J for it based on the Lagrange form. We keep all interim intervals that are related to p and used for computing J in the queue, and then, they can be reused to compute ϕ over another TM (p, I').

To better understand the method, we give Example 3.3.10.

Example 3.3.10. *We consider the non-polynomial ODE $\dot{x} = f(x) = \sin(x)$. The order 3 TM*

$$(x_0 + x_0 t + 0.5 x_0 t^2, [-0.1, 0.1])$$

is an over-approximation of the solution $x(t)$ over the time $[0, 0.02]$ w.r.t. the initial condition $x(0) = x_0$ wherein $x_0 \in [-1, 1]$. The order 3 approximation of $f(x)$ is $p_f(x) = x - \frac{1}{6}x^3$ whose Horner form is given by $x(1 - \frac{1}{6}x^2)$. We present the computation of the first refinement iteration along with the working queue content in Table 3.3. Then, based on the produced queue, we only need to deal with the remainder intervals in the remaining iterations, as the second one given by Table 3.4. Since the operations are handled in the same order, we only need to pick the elements from the queue according to that order. Note that we conservatively round every number to at most 5 decimal places.

In order to see the improvement of our fast remainder refinement method, we present Table 3.5 to give a comparison with the standard method from Berz and Makino. The system in the experiments is the jet engine model. We use a fixed step-size 0.02 in all time steps and compute 100 flowpipes from the initial set $x(0) \in [0.9, 1.1]$ and $y(0) \in [0.9, 1.1]$. All of the Taylor expansions are computed by Approach II. Except the experiments in Table 3.5, the fast remainder refinement method is used in all of the experiments in the thesis.

3.3.4 Case studies

In this section, we give two examples of continuous systems and present comparisons between the methods of interval-based integration and TM integration.

Example 3.3.11 (Lotka-Volterra system)**.** *The 2-dimensional Lotka-Volterra system depicts the populations change of a class of predators (wolves) and a class of preys (rabbits). The growth rate of preys' population over time is given by $\dot{x} = x(\alpha - \beta y)$ wherein α, β are constant parameters and y is the population of predators. It means that the number of preys grows exponentially without predation. The population growth of predators is governed by the differential equation $\dot{y} = -y(\gamma - \delta x)$ wherein γ, δ are constant parameters. We choose the parameters $\alpha = 1.5$, $\beta = 1$, $\gamma = 3$ and $\delta = 1$, then the ODE becomes*

$$\begin{cases} \dot{x} = 1.5x - xy \\ \dot{y} = -3y + xy \end{cases}$$

We choose the initial sets of different sizes around the point at $x = 5$, $y = 2$, and perform TM integration as well as the interval-based integration in VNODE-LP *respectively on them with similar settings for the time horizon $[0, 4]$. The results are given in Table 3.6 and 3.7. We compare the two integration methods only based on the solution enclosures at the end of the time horizon since the overestimation is eventually accumulated along with time steps. To intuitively compare the accuracies, we additionally over-approximate the TMs by intervals.*

Since the classical interval-based integration only deals with ITSs, its performance is much better than that of the TM integration when the initial set is small. It is also

Step	TM computation	Queue content
1: compute the order 2 TM over-approximation for $(x_0 + x_0t + 0.5x_0t^2,$ $[-0.1, 0.1])$	$(x_0 + x_0t, [-0.1, 0.1] + \texttt{Int}(\boldsymbol{0.5x_0t^2}))$ $=(x_0 + x_0t, [-0.1002, 0.1002])$	front $\mathbf{[-0.0002, 0.0002]}$ tail
2: replace x by $(x_0 + x_0t, [-0.1002, 0.1002])$ in $x(1 - \frac{1}{6}x^2)$	$-$	same as above
2.1: compute x^2 by order 2 TM arithmetic	$(x_0 + x_0t, [-0.1002, 0.1002])^2$ $=(x_0^2,$ $\texttt{Int}(\boldsymbol{x_0 + x_0t}) \cdot [-0.1002, 0.1002]$ $+ [-0.1002, 0.1002] \cdot \texttt{Int}(\boldsymbol{x_0 + x_0t})$ $+ ([-0.1002, 0.1002])^2$ $+ \texttt{Int}(\boldsymbol{2x_0^2t + x_0^2t^2}))$ $=(x_0^2,$ $\mathbf{[-1.02, 1.02]} \cdot [-0.1002, 0.1002]$ $+ [-0.1002, 0.1002] \cdot \mathbf{[-1.02, 1.02]}$ $+ [0, 0.01005]$ $+ \mathbf{[0, 0.0404]})$ $=(x_0^2, [-0.20441, 0.25486])$	front $[-0.0002, 0.0002]$ $\mathbf{[-1.02, 1.02]}$ $\mathbf{[-1.02, 1.02]}$ $\mathbf{[0, 0.0404]}$ tail
2.2: compute $(1 - \frac{1}{6}x^2)$ by order 2 TM arithmetic	$(1 - \frac{1}{6}x_0^2, [-0.04248, 0.03407])$	same as above
2.3: compute $x(1 - \frac{1}{6}x^2)$ by order 2 TM arithmetic	$(x_0 + x_0t, [-0.1002, 0.1002])$ $\cdot\, (1 - \frac{1}{6}x_0^2, [-0.04248, 0.03407])$ $=(x_0 + x_0t,$ $\mathbf{[-1.02, 1.02]} \cdot [-0.04248, 0.03407]$ $+ [-0.1002, 0.1002] \cdot \texttt{Int}(\boldsymbol{1 - \frac{1}{6}x_0^2})$ $+ [-0.1002, 0.1002] \cdot [-0.04248, 0.03407]$ $+ \texttt{Int}(\boldsymbol{-\frac{1}{6}x_0^3 - \frac{1}{6}x_0^3t})$ $=(x_0 + x_0t,$ $\mathbf{[-1.02, 1.02]} \cdot [-0.04248, 0.03407]$ $+ [-0.1002, 0.1002] \cdot \mathbf{[0.83333, 1.16667]}$ $+ [-0.1002, 0.1002] \cdot [-0.04248, 0.03407]$ $+ \mathbf{[-0.17, 0.17]}$ $=(x_0 + x_0t, [-0.33449, 0.33449])$	front $[-0.0002, 0.0002]$ $[-1.02, 1.02]$ $[-1.02, 1.02]$ $[0, 0.0404]$ $\mathbf{[-1.02, 1.02]}$ $\mathbf{[0.83333, 1.1667]}$ $\mathbf{[-0.17, 0.17]}$ tail
3: Evaluate a remainder interval J for $x(1 - \frac{1}{6}x)$ over $x \in (x_0 + x_0t,$ $[-0.1002, 0.1002])$	$\frac{1}{24}\sin([-1.1202, 1.1202])([-1.1202, 1.1202])^4$ $=\frac{1}{24} \cdot [-0.90019, 0.90019] \cdot [0, 1.57465]$ $=[-0.05907, 0.05907]$	same as above
4: Integrate $x(1 - \frac{1}{6}x) + J$ by order 3 TM arithmetic	$\int_0^t (x_0 + x_0s, [-0.39356, 0.39356])ds$ $=(x_0t + 0.5x_0t^2, [-0.00788, 0.00788])$	same as above
5: Add x_0 to derive the first iteration result	$(x_0 + x_0t + 0.5x_0t^2, [-0.00788, 0.00788])$	same as above

Table 3.3: First refinement iteration of Example 3.3.10

Step		Interval computation	Queue content
1:	compute the remainder interval for the order 2 TM over-approximation for $(x_0 + x_0 t + 0.5x_0 t^2,$ $[-0.00788, 0.00788])$	$(-, [-0.00788, 0.00788] + \mathbf{[-0.0002,\ 0.0002]})$ $=(-, [-0.00808, 0.00808])$	front **[-0.0002, 0.0002]** $[-1.02, 1.02]$ $[-1.02, 1.02]$ $[0, 0.0404]$ $[-1.02, 1.02]$ $[0.83333, 1.1667]$ $[-0.17, 0.17]$ tail
2:	replace x by $(x_0 + x_0 t,$ $[-0.00808, 0.00808])$ in $x(1 - \frac{1}{6}x^2)$	–	–
2.1:	compute a remainder interval for x^2	$(-, [-0.00808, 0.00808])^2$ $=(-,$ $\mathbf{[-1.02,\ 1.02]} \cdot [-0.00808, 0.00808]$ $+ [-0.00808, 0.00808] \cdot \mathbf{[-1.02,\ 1.02]}$ $+ [0, 0.00007]$ $+ \mathbf{[0, 0.0404]})$ $=(-, [-0.01649, 0.05696])$	front $[-0.0002, 0.0002]$ **[-1.02, 1.02]** **[-1.02, 1.02]** **[0,0.0404]** $[-1.02, 1.02]$ $[0.83333, 1.1667]$ $[-0.17, 0.17]$ tail
2.2:	compute a remainder interval for $(1 - \frac{1}{6}x^2)$	$(-, [-0.00950, 0.00275])$	–
2.3:	compute a remainder interval for $x(1 - \frac{1}{6}x^2)$	$(-, [-0.00808, 0.00808])$ $\cdot (-, [-0.00950, 0.00275])$ $=(-,$ $\mathbf{[-1.02, 1.02]} \cdot [-0.00950, 0.00275]$ $+ [-0.00808, 0.00808] \cdot \mathbf{[0.83333, 1.16667]}$ $+ [-0.00808, 0.00808] \cdot [-0.00950, 0.00275]$ $+ \mathbf{[-0.17, 0.17]}$ $=(-, [-0.1892, 0.1892])$	front $[-0.0002, 0.0002]$ $[-1.02, 1.02]$ $[-1.02, 1.02]$ $[0, 0.0404]$ **[-1.02,1.02]** **[0.83333,1.1667]** **[-0.17,0.17]** tail
3:	Evaluate a remainder interval J for $x(1 - \frac{1}{6}x)$ over $x \in (-,$ $[-0.00808, 0.00808])$	$\frac{1}{24}\sin([-1.02808, 1.02808])$ $\cdot ([-1.02808, 1.02808])^4$ $=\frac{1}{24} \cdot [-0.85631, 0.85631] \cdot [0, 1.11715]$ $=[-0.03986, 0.03986]$	–
4:	Integrate the remainder interval	$\int_0^t (-, [-0.22906, 0.22906])ds$ $=(-, [-0.00459, 0.00459])$	–

Table 3.4: Second refinement iteration of Example 3.3.10

TM order	Standard method	Our method
5	9.2	**2.3**
8	65.7	**19.4**
11	312.6	**47.3**

Table 3.5: Comparison of remainder refinement methods based on the jet engine model

Initial set	Step-size	ITS order	Time (s)	Enclosure
$x(0) \in [5, 5]$ $y(0) \in [2, 2]$	auto	5	**< 1**	$x(4) \in$ **1.64995003485[28990,97552]** $y(4) \in$ **2.0899878588[196711,331982]**
$x(0) \in [4.95, 5.05]$ $y(0) \in [1.95, 2.05]$	auto	5	**< 1**	$x(4) \in [1.5467170558689369, 1.7531830138436475]$ $y(4) \in [1.9672607341964982, 2.2127149834564204]$
$x(0) \in [4.9, 5.1]$ $y(0) \in [1.9, 2.1]$	auto	MAX_ORDER	Fail	–

Table 3.6: Interval-based integration in VNODE-LP on the Lotka-Volterra system

Initial set	Step-size	TM order	Time (s)	Enclosure
$x(0) \in [5, 5]$ $y(0) \in [2, 2]$	0.01	5	3.2	$x(4) \in 1.64995000[3203, 67277]$ $y(4) \in 2.089987[783478, 934250]$
$x(0) \in [4.95, 5.05]$ $y(0) \in [1.95, 2.05]$	0.01	5	8.9	$x(4) \in$ **[1.611473542017 , 1.688426503140]** $y(4) \in$ **[2.043880403362 , 2.136095251238]**
$x(0) \in [4.9, 5.1]$ $y(0) \in [1.9, 2.1]$	0.01	5	**8.9**	$x(4) \in$ **[1.571477158543 , 1.728422831796]** $y(4) \in$ **[1.994823280383 , 2.185152183021]**

Table 3.7: TM integration on the Lotka-Volterra system

no surprise to see that TM integration has a better accuracy when the initial set is not singleton, and such an advantage becomes more apparent when the size of the initial set is growing. For the singleton initial set $x(0) = 5$, $y(0) = 2$, the TM integration method even returns a worse over-approximation than the interval-based one. The reason could be that we symbolically represent the initial state by (x_0, y_0) in the TMs which is unnecessary and might cause additional running time and overestimation.

Example 3.3.12 (Spring-pendulum)**.** *The planar spring-pendulum described in [Mei07] is illustrated by Figure 3.10. It consists of a solid ball of mass m and a spring of natural length L and spring constant k. We study the evolutions of the length r of the spring and the angle θ between the spring and the vertical. They are modeled by the following ODE*

$$\begin{cases} m \cdot \ddot{r} &= m \cdot r \cdot \dot{\theta}^2 + m \cdot g \cdot \cos(\theta) - k \cdot (r - L) \\ r^2 \cdot \ddot{\theta} &= -2 \cdot r \cdot \dot{r} \cdot \dot{\theta} - g \cdot r \cdot \sin(\theta) \end{cases}$$

which can be equivalently reformulated as the following 4-dimensional first-order ODE

$$\begin{cases} \dot{r} &= v_r \\ \dot{\theta} &= v_\theta \\ \dot{v}_r &= r \cdot v_\theta^2 + g \cdot \cos(\theta) - k \cdot (r - L) \\ \dot{v}_\theta &= -\frac{(2 \cdot v_r \cdot v_\theta + g \cdot \sin(\theta))}{r} \end{cases}$$

We set the constant parameters $k = 2$, $L = 1$, $g = 9.8$ and do the comparisons on the initial sets of different sizes around the point

$$r(0) = 1.2,\ \theta(0) = 0.5,\ v_r(0) = 0,\ v_\theta(0) = 0$$

The results are listed in Table 3.8 and 3.9.

Accuracy improvement. The hardness of an integration task is very sensitive to the size of the initial set. That is, the lager the initial set, the harder the integration task. As shown by the above experiments, interval-based integration is more likely than TM integration to suffer from this problem. The reason is that interval arithmetic is more sensitive to the dependency problem than TM arithmetic. Although one may simply raise the approximation orders or reduce the time step-sizes to improve the overall accuracy,

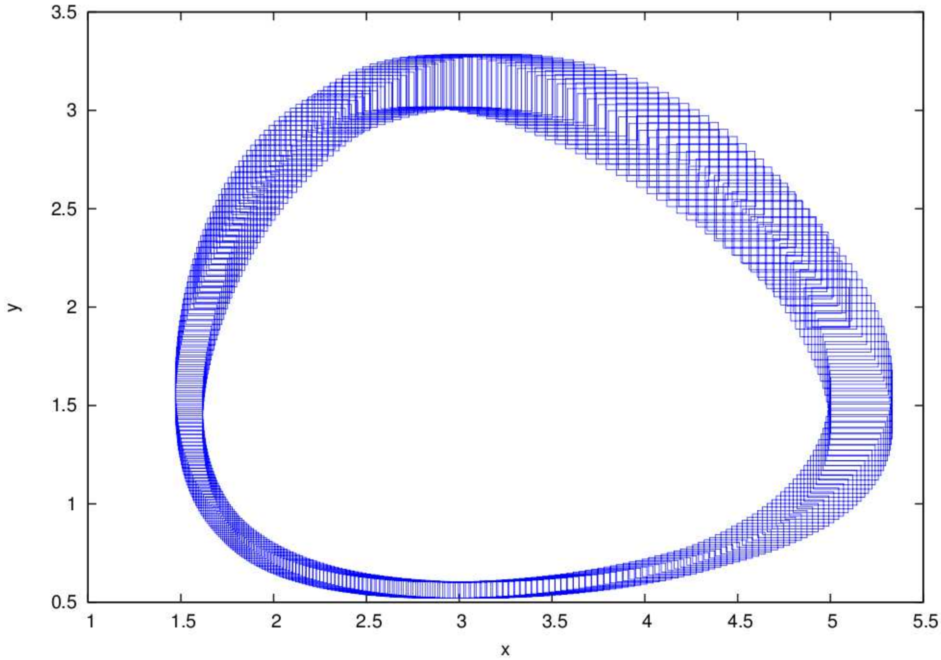


Figure 3.9: Interval enclosures of the TM flowpipes for the Lotka-Volterra system. They are computed from the initial set $x(0) \in [4.9, 5.1]$, $y(0) \in [1.9, 2.1]$ for the time horizon $[0, 4]$.

Initial set	Step-size	ITS order	Time (s)	Enclosure
$r(0) \in [1.2, 1.2]$ $\theta(0) \in [0.5, 0.5]$ $v_r(0) \in [0, 0]$ $v_\theta(0) \in [0, 0]$	auto	6	$< \mathbf{1}$	$r(10) \in$ **5.8090433374[042804,192374]** $\theta(10) \in$ **0.08409991010[42140,84239]** $v_r(10) \in$ **6.8272350580[110554,286352]** $v_\theta(10) \in$ **-0.02939916355[29386,83425]**
$r(0) \in [1.19, 1.21]$ $\theta(0) \in [0.49, 0.51]$ $v_r(0) \in [-0.01, 0.01]$ $v_\theta(0) \in [-0.01, 0.01]$	auto	6	$< \mathbf{1}$	$r(5) \in [2.4489430438636, 2.5787902600760]$ $\theta(5) \in 0.2[228203691481, 862895828842]$ $v_r(5) \in [4.6091859312751, 4.7582065882747]$ $v_\theta(5) \in [-0.3703928208511, -0.2471714144274]$
$r(0) \in [1.0, 1.6]$ $\theta(0) \in [0.4, 1]$ $v_r(0) \in [-0.3, 0.3]$ $v_\theta(0) \in [-0.3, 0.3]$	auto	MAX_ORDER	Fail	–

Table 3.8: Interval-based integration in VNODE-LP on the spring pendulum model

such improvements are always very limited due to round-off errors and the approximation methods in use. As we show in Table 3.6 and 3.8, interval-based method fails on the large initial sets even with the maximum order allowed in VNODE-LP. Then, as we pointed out in Chapter 2, a better accuracy can be obtained by perform a subdivision on the initial set, since most of the interval extensions in an integration task can be made inclusion isotonic and Lipschitz. However, the number of subdivisions grows exponentially with respect to the number of variables. Fortunately, by using TMs with proper selected orders, we can often avoid splitting an initial set but still obtain a good accuracy. As an example, we compute a 10-uniform subdivision of the following initial set for the spring pendulum model

$$r(0) \in [1, 1.6],\ \theta(0) = [0.4, 1],\ v_r(0) = [-0.3, 0.3],\ v_\theta(0) = [-0.3, 0.3]$$

and perform the interval-based integration implemented in VNODE-LP with ITS order 6. By collecting the computed intervals, we have the following interval enclosure for the

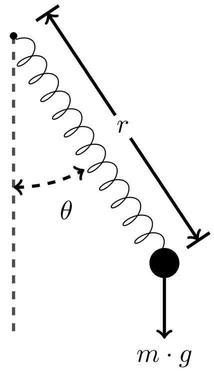


Figure 3.10: Spring-pendulum of Example 3.3.12

Initial set	Step-size	TM order	Time (s)	Enclosure
$r(0) \in [1.2, 1.2]$ $\theta(0) \in [0.5, 0.5]$ $v_r(0) \in [0, 0]$ $v_\theta(0) \in [0, 0]$	0.01	6	145.9	$r(10) \in [5.798127042349, 5.819959632464]$ $\theta(10) \in [0.077567535268, 0.090632284946]$ $v_r(10) \in [6.807928278833, 6.846541837222]$ $v_\theta(10) \in [-0.034224004444, -0.024574322666]$
$r(0) \in [1.19, 1.21]$ $\theta(0) \in [0.49, 0.51]$ $v_r(0) \in [-0.01, 0.01]$ $v_\theta(0) \in [-0.01, 0.01]$	0.01	6	389.8	$r(5) \in$ **[2.496405883474 , 2.531266738845]** $\theta(5) \in$ **[0.245863589457 , 0.263219159794]** $v_r(5) \in$ **[4.662911374025 , 4.704508620747]** $v_\theta(5) \in$ **[-0.324274826121 , -0.293190662999]**
$r(0) \in [1.0, 1.6]$ $\theta(0) \in [0.4, 1]$ $v_r(0) \in [-0.3, 0.3]$ $v_\theta(0) \in [-0.3, 0.3]$	0.01	6	**531.8**	$r(1) \in$ **[3.680305643636 , 6.429661766060]** $\theta(1) \in$ **[-0.324284379975 , 0.527654547935]** $v_r(1) \in$ **[2.549728783442 , 10.762474258784]** $v_\theta(1) \in$ **[-1.458244212326 , 0.890649195439]**

Table 3.9: TM integration on the spring pendulum model

solution at time 1

$$\begin{aligned} r(1) &\in [2.5481678021852265, 5.5600537403117008] \\ \theta(1) &\in [-0.3107606461333349, 0.6708186109998349] \\ v_r(1) &\in [3.3939038756048236, 10.7952720145605970] \\ v_\theta(1) &\in [-3.0506834955544365, 2.6157139253117295] \end{aligned}$$

which is comparable to the result of TM integration shown in Table 3.9, but the time cost is as high as 1218 seconds.

3.4 Adaptive techniques

In this section, we seek to improve the time cost of TM integration by allowing minor loss in accuracy. To do so, we consider to adaptively change the step-sizes or the TM orders in a TM integration task while the specified accuracy is still fulfilled. For interval-based integration, such ideas have been proposed and implemented in the tool VNODE as well as its successor VNODE-LP (see [Ned99, Ned11]). For TM integration, the tool COSY [MB06] from Berz et al. also supports varying step-sizes to improve the integration performance. Here, we present the methods which are different from the others.

Obviously, the running time of an integration task can be shortened by using larger step-sizes or lower TM orders, however those parameters can not be changed at will, since it may easily lead to an explosion of overestimation. Since the accuracy of a TM integration task is mainly reflected by the sizes of the flowpipe remainder intervals, we

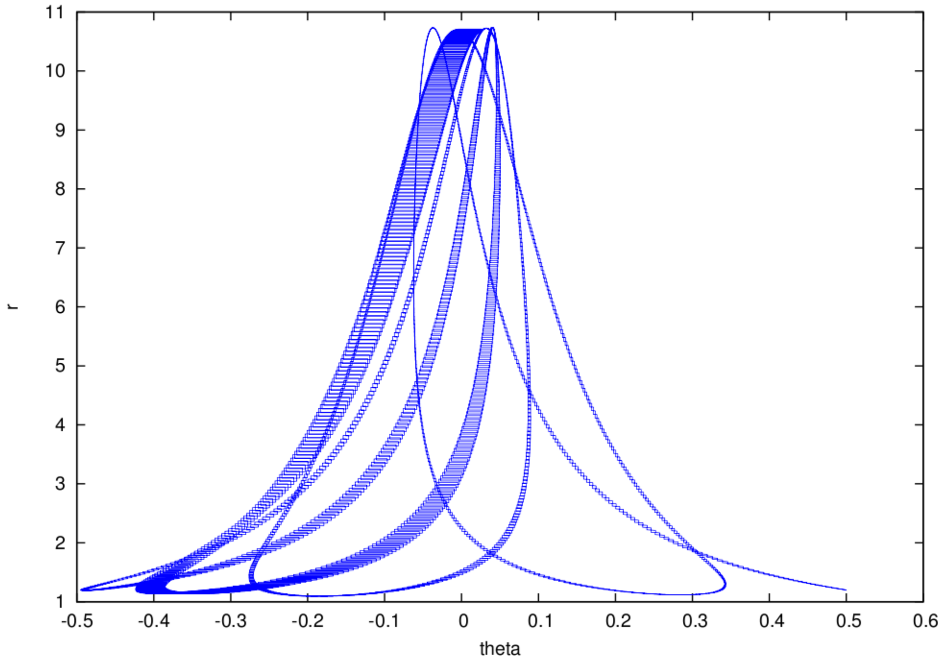


Figure 3.11: Interval enclosures of the TM flowpipes for the spring pendulum example. They are computed from the initial set $r(0) = 1.2, \theta(0) = 0.5, v_r(0) = 0, v_\theta(0) = 0$ for the time horizon $[0, 20]$.

may specify a maximum interval threshold $I_{\max}$ and slightly enlarge the step-size or lower the TM order while there is still a safe remainder interval contained in $I_{\max}$. When the system is multi-dimensional, the components of $I_{\max}$ are also allowed to have different sizes.

Our adaptive step-sizing scheme is presented by Algorithm 8. The step-size changes in a user-specified range $[\alpha, \beta]$ such that it is enlarged in every step by multiplying a scalar $\lambda_\uparrow$ onto it if the interval threshold $I_{\max}$ is not breached, otherwise, it is reduced by being multiplied by $\lambda_\downarrow$. If a step-size smaller than α is computed, then the integration task terminates and returns FAIL. In our implementation, the scalars are simply selected as $\lambda_\uparrow = 1.1$ and $\lambda_\downarrow = 0.5$.

The TM integration using adaptive orders can be done similarly. It is given by Algorithm 9. In every integration step, the TM order is increased by 1 if the remainder interval $I_{\max}$ is not breached, otherwise we lower it also by 1 to verify $I_{\max}$ again. If no proper TM order in the given range is found, then the integration job fails.

For a multi-dimensional system, a component of the TM (p_l, I_l) in the above algorithms represents an over-approximation of the solution in a dimension. However, for a complex system, different state variables often grow at varying rates and some of them might even be not correlated. Therefore it is clumsy to change the orders of the components uniformly. For instance, state variables that represent timers can be specified to have order 1 TMs whereas fast varying variables can be represented by higher-order TMs at the same time. Our *independent adaptive order* scheme is given as follows. We first concentrate on the dimensions in which the corresponding interval threshold in $I_{\max}$ is breached. The orders of those components are increased by 1. If the technique fails to

Algorithm 8 TM integration by using adaptive step-sizes

Input: an ODE $\dot{\vec{x}} = f(\vec{x}, t)$, a TM initial set X_0, a time horizon $[0, \Delta]$, the range $[\alpha, \beta]$ for a step-size, a positive integer k for the TM order, an interval threshold $I_{\max}$

Output: an over-approximation $\mathcal{R}$ for the reachable set from X_0 in time $[0, \Delta]$

$\mathcal{R} \leftarrow \emptyset$;
$t \leftarrow 0$;
$X_l \leftarrow X_0$;
$\delta \leftarrow \beta$;
while $t \leq \Delta$ **do**
    Compute the preconditioning parameters $\vec{c}_l$ and A_l;
    Derive an order k Taylor polynomial p_l according to the parameters;
    if $\delta < \beta$ **then**
        $\delta \leftarrow \min\{\lambda_\uparrow \cdot \delta, \beta\}$; # slightly enlarge the step-size
    end if
    while $I_{\max}$ is not a safe remainder interval for p_l over the time $[0, \delta]$ **do**
        $\delta \leftarrow \lambda_\downarrow \cdot \delta$; # slightly reduce the step-size
        if $\delta < \alpha$ **then**
            Terminate and return FAIL; # step-size is out of the range
        end if
    end while
    $I_l \leftarrow I_{\max}$;
    repeat
        $(p_l, I_l) \leftarrow \mathcal{P}_f((p_l, I_l))$; # by order k TM arithmetic
    until No big improvement on the last refinement of I_l
    $\mathcal{R} \leftarrow \mathcal{R} \cup \{((p_l, I_l), X_l)\}$;
    $X_l \leftarrow (p_l(X_l, \delta), I_l)$;
    $t \leftarrow t + \delta$; # by order k TM arithmetic
end while
return $\mathcal{R}$;

Algorithm 9 TM integration by using adaptive TM orders

Input: an ODE $\dot{\vec{x}} = f(\vec{x}, t)$, a TM initial set X_0, a time horizon $[0, \Delta]$, a step-size δ, a range $a \sim b$ for the TM orders, an interval threshold $I_{\max}$

Output: an over-approximation $\mathcal{R}$ for the reachable set from X_0 in time $[0, \Delta]$

$\mathcal{R} \leftarrow \emptyset$;
$N \leftarrow \lceil \frac{\Delta}{\delta} \rceil$;
$X_l \leftarrow X_0$;
$k \leftarrow a$; # specify the lowest order
for all $i = 1, \ldots, N$ **do**
  Compute the preconditioning parameters $\vec{c}_l$ and A_l;
  if $k > a$ **then**
    $k \leftarrow k - 1$; # decrease the order by 1
  end if
  Derive an order k Taylor polynomial p_l according to the parameters;
  while $I_{\max}$ is not a safe remainder interval for p_l **do**
    $k \leftarrow k + 1$; # increase the order by 1
    if $k > b$ **then**
      Terminate and return FAIL; # order is out of the range
    else
      Increase the order of p_l to k using Picard operator;
    end if
  end while
  $I_l \leftarrow I_{\max}$;
  repeat
    $(p_l, I_l) \leftarrow \mathcal{P}_f((p_l, I_l))$; # by order k TM arithmetic
  until No big improvement on the last refinement of I_l
  $\mathcal{R} \leftarrow \mathcal{R} \cup \{((p_l, I_l), X_l)\}$;
  $X_l \leftarrow (p_l(X_l, \delta), I_l)$; # by order k TM arithmetic
end for
return $\mathcal{R}$;

Test	Step-size	TM order	Time (s)	Solution enclosure
1	0.03	5	15.9	$X(15) \in [0.987576012185, 0.997127811231]$ $Y(15) \in [1.479230489975, 1.492068486468]$
2	$[0.01, 0.2]$	5	4.7	$X(15) \in [0.986712147280, 0.997991279644]$ $Y(15) \in [1.478023448254, 1.493275594069]$
3	0.03	$3 \sim 5$	5.4	$X(15) \in [0.984264065516, 1.000441898085]$ $Y(15) \in [1.474811213375, 1.496490371398]$
4	0.03	$X: 3 \sim 5$ $Y: 3 \sim 5$	5.1	$X(15) \in [0.983703965994, 1.001004863951]$ $Y(15) \in [1.474059852987, 1.497241858220]$

Table 3.10: Flowpipe construction for Brusselator using different settings

verify the safety of $I_{\max}$, the orders of all remaining components are increased by 1 as well, since those state variables might be correlated. This process continues until the upper order limit is breached. We present the whole scheme by Algorithm 10. In our implementation, the independent order increment is only performed by one time but it does not have to be.

In the rest of the section, we give several examples to show the advantages of the adaptive techniques.

Example 3.4.1 (Brusselator)**.** *A type of autocatalytic reaction can be modeled by* Brusselator *which is characterized by the reactions*

$$\begin{array}{lcl} A & \to & X \\ 2X+Y & \to & 3X \\ B+X & \to & Y+D \\ X & \to & E \end{array}$$

When A, B are considered as constants, we have the following rate equations for X and Y.

$$\begin{cases} \dot{X} &= A + X^2 \cdot Y - B \cdot X - X \\ \dot{Y} &= B \cdot X - X^2 \cdot Y \end{cases}$$

We set $A = 1$ and $B = 1.5$. Given the initial set $X(0) \in [0.8, 1]$, $Y(0) \in [0, 0.2]$. We present the results of computing the flowpipes for the time horizon $[0, 15]$ by different schemes in Table 3.10. We use the interval threshold $I_{\max} = [-10^{-5}, 10^{-5}]^2$ in all of the tests. In order to intuitively compare the accuracies, we give the interval enclosures of the TMs. In Test 1, we use a fixed step-size and a fixed uniform TM order. Not surprisingly, it consumes the greatest amount of time among all tests whereas the accuracy is the best. We use the adaptive step-size ranging in $[0.01, 0.2]$ in Test 2, the accuracy is comparable to that of Test 1 but the time cost is much less. We illustrate the step-size change during the computation in Figure 3.12. The results of using adaptive orders are given by Test 3 and 4. In the first one, we apply the scheme to uniformly change the TM orders in all dimensions. The order change is presented in Figure 3.13. Then in the second test, we allow the TM orders to change independently in different dimensions. The changes of the TM orders in the dimension X and Y are given in Figure 3.14. The loss of accuracy in either of the tests is also not big.

For this example, the adaptive step-sizing technique has an advantage in efficiency over the other schemes, nevertheless, one may later see that it is not always the case.

Algorithm 10 TM integration by using independent adaptive TM orders

Input: an n-dimensional ODE $\dot{\vec{x}} = f(\vec{x}, t)$, a TM initial set X_0, a time horizon $[0, \Delta]$, a step-size δ, the ranges $a_1 \sim b_1, \ldots, a_n \sim b_n$ for the TM orders, an interval threshold $I_{\max}$

Output: an over-approximation $\mathcal{R}$ for the reachable set from X_0 in time $[0, \Delta]$

$\mathcal{R} \leftarrow \emptyset$;
$N \leftarrow \lceil \frac{\Delta}{\delta} \rceil$;
$X_l \leftarrow X_0$;
for all $j = 1, \ldots, n$ **do**
  $k_j \leftarrow a_j$; # specify the lowest orders
end for
for all $i = 1, \ldots, N$ **do**
  Compute the preconditioning parameters $\vec{c}_l$ and A_l;
  for all $j = 1, \ldots, n$ **do**
    if $k_j > a_j$ **then**
      $k_j \leftarrow k_j - 1$;
    end if
  end for
  Derive a Taylor polynomial p_l according to the parameters such that for $1 \leq j \leq n$ the j-th component of p_l is of order k_j;
  while $I_{\max}$ is not a safe remainder interval for p_l **do**
    for all $j = 1, \ldots, n$ **do**
      if $k_j < b_j$ **then**
        Increase k_j by 1 if $I_{\max}[j]$ is breached;
        Increase the order of $p_l[j]$ to k_j using Picard operator;
      end if
    end for
    if $I_{\max}$ is not safe for p_l **then**
      for all k_j which is not increased in the previous step **do**
        Increase the order k_j by 1 if $k_j < b_j$;
        Increase the order of $p_l[j]$ to k_j using Picard operator;
      end for
    end if
    if no k_j is increased in the current iteration **then**
      Terminate and return FAIL;
    end if
  end while
  $I_l \leftarrow I_{\max}$;
  repeat
    $(p_l, I_l) \leftarrow \mathcal{P}_f((p_l, I_l))$; # by TM arithmetic
  until No big improvement on the last refinement of I_l
  $\mathcal{R} \leftarrow \mathcal{R} \cup \{((p_l, I_l), X_l)\}$;
  $X_l \leftarrow (p_l(X_l, \delta), I_l)$; # by TM arithmetic
end for
return $\mathcal{R}$;

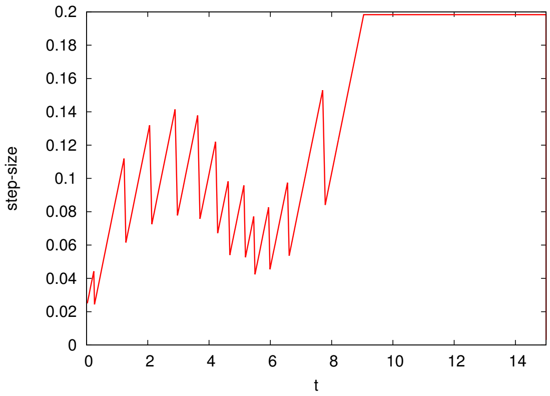


Figure 3.12: Change of the step-size in Test 2 on Brusselator

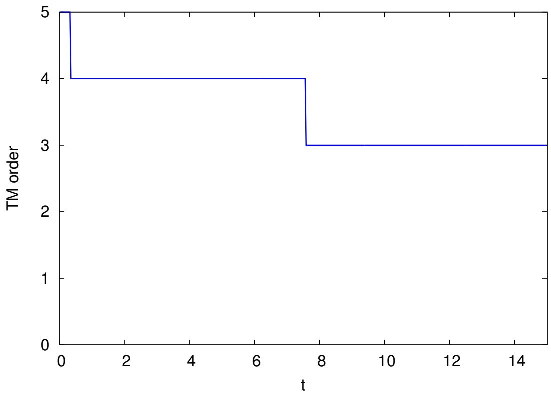
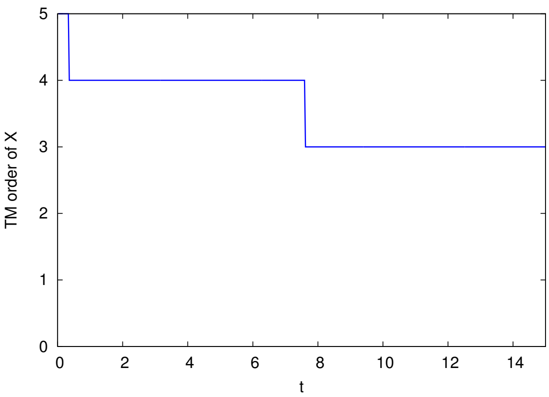


Figure 3.13: Change of the TM order in Test 3 on Brusselator



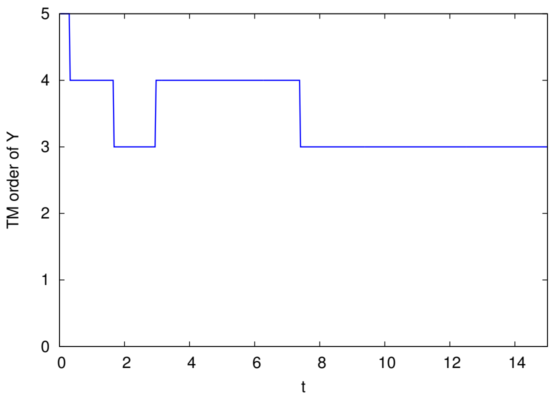


Figure 3.14: Change of the TM orders in Test 4 on Brusselator. Change of the order in the dimension X (left). Change of the order in the dimension Y (right).

Test	Step-size	TM order	Time (s)	Solution enclosure
1	0.003	8	767.5	$x(6.5) \in [0.484331866475, 1.379244709695]$ $y(6.5) \in [1.453031916529, 2.990964064560]$ $z(6.5) \in [18.124980017504, 21.162839708734]$
2	$[0.001, 0.05]$	8	522.5	$x(6.5) \in [0.219195452850, 1.618505334065]$ $y(6.5) \in [1.099867538815, 3.333154577724]$ $z(6.5) \in [17.762079569332, 21.626545865783]$
3	0.003	$6 \sim 10$	436.6	$x(6.5) \in [0.370398820486, 1.493166319960]$ $y(6.5) \in [1.296802176240, 3.147179628973]$ $z(6.5) \in [17.939699525684, 21.348120274208]$
4	0.003	$x: 6 \sim 10$ $y: 6 \sim 10$ $z: 6 \sim 10$	403.5	$x(6.5) \in [0.369340118569, 1.494225836838]$ $y(6.5) \in [1.295299703693, 3.148682034957]$ $z(6.5) \in [17.937821286088, 21.349993993524]$

Table 3.11: Flowpipe construction for Lorentz system using different settings

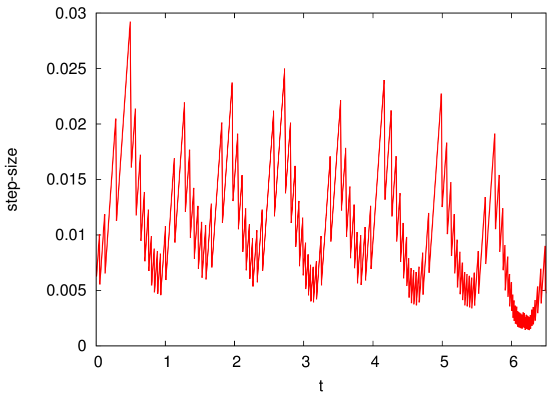


Figure 3.15: Change of the step-size in Test 2 on Lorentz system

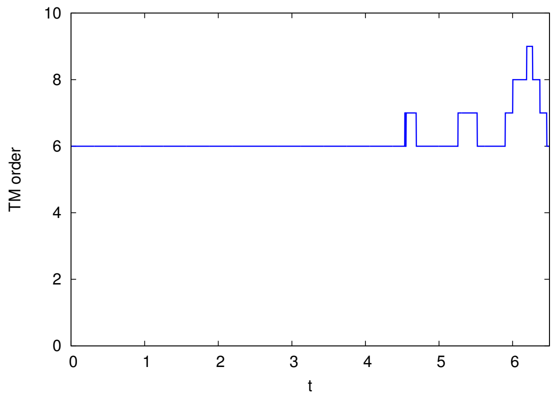


Figure 3.16: Change of the TM order in Test 3 on Lorentz system

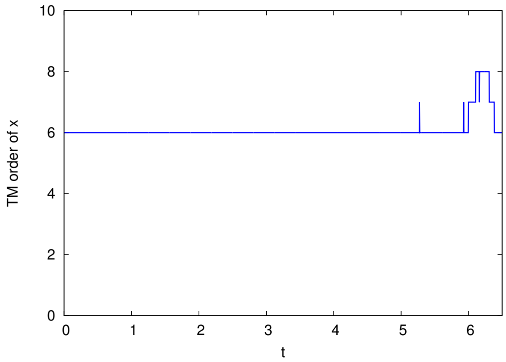
Example 3.4.2 (Lorentz system). *The Lorentz system is a simplified model which is developed by Edward Lorenz for atmospheric convection (see [Lor63]). It is described by a* 3*-dimensional ODE*

$$\begin{cases} \dot{x} &= \sigma \cdot (y - x) \\ \dot{y} &= x \cdot (\rho - z) - y \\ \dot{z} &= x \cdot y - \beta \cdot z \end{cases}$$

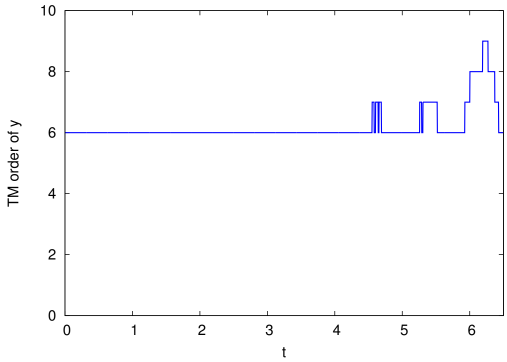
with the typical parameter values $\sigma = 10$, $\rho = \frac{8}{3}$ *and* $\beta = 28$. *Since it is a chaotic system, we study the short-term behaviors from the following small set of initial states*

$$x(0) \in [14.999, 15.001], \;\; y(0) \in [14.999, 15.001], \;\; z(0) \in [35.999, 36.001]$$

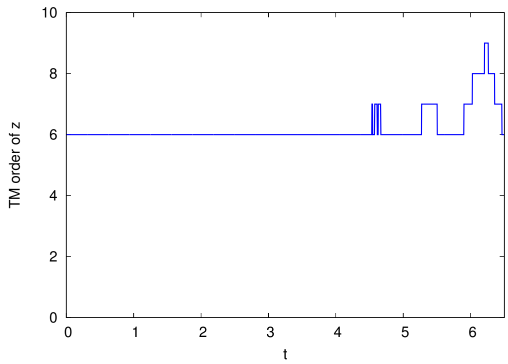
This time, we set the interval threshold as $I_{\max} = [-10^{-8}, 10^{-8}]^3$. *The computation results are given in Table 3.11. Similar to the previous example, we also illustrate the change of step-sizes and TM orders by Figure 3.15 to 3.17. Here, the independent adaptive order technique has the best performance. To visualize the TM flowpipes clearly, we plot a grid paving of them in Figure 3.18.*



(a) Change of the order in the dimension x



(b) Change of the order in the dimension y



(c) Change of the order in the dimension z

Figure 3.17: Change of the TM orders during the computation of Test 4 on Lorentz system

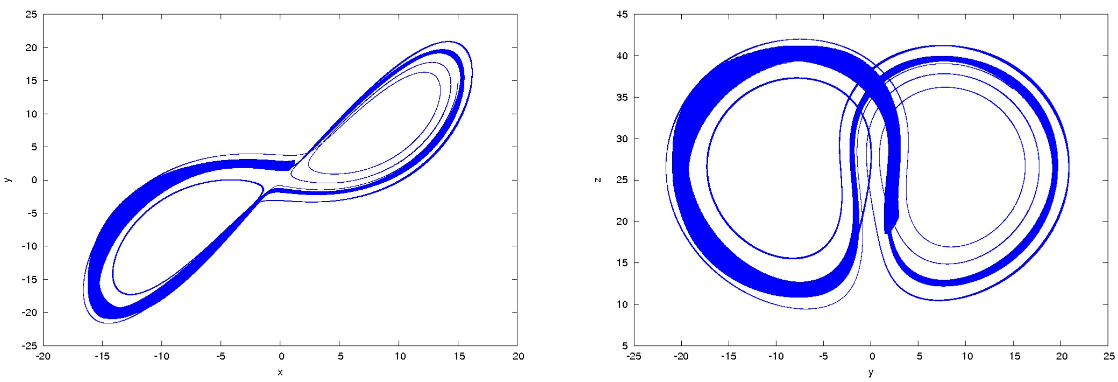


Figure 3.18: Grid pavings of the TM flowpipes computed in Test 1 on Lorentz system. Projection on the x-y plane (left). Projection on the y-z plane (right).

Test	Step-size	TM order	Time (s)	Solution enclosure
1	0.03	7	176.7	$x(6) \in [0.337524243821, 1.738995973771]$ $y(6) \in [-9.7180806640901, -7.360507441673]$ $z(6) \in [0.030072992385, 0.037971136986]$
2	$[0.005, 0.1]$	7	143.9	$x(6) \in [0.299242459359, 1.777276310591]$ $y(6) \in [-9.753524037789, -7.325072767832]$ $z(6) \in [0.029702744349, 0.038339574043]$
3	0.03	$5 \sim 8$	49.8	$x(6) \in [0.250145939269, 1.826296801216]$ $y(6) \in [-9.801939601429, -7.276761701073]$ $z(6) \in [0.029447163640, 0.038595468495]$
4	0.03	$x:\ 5 \sim 8$ $y:\ 5 \sim 8$ $z:\ 5 \sim 8$	44.5	$x(6) \in [0.234284317579, 1.842173097432]$ $y(6) \in [-9.817200175287, -7.261488616246]$ $z(6) \in [0.029335086590, 0.038707599492]$

Table 3.12: Flowpipe construction for Rössler attractor using different settings

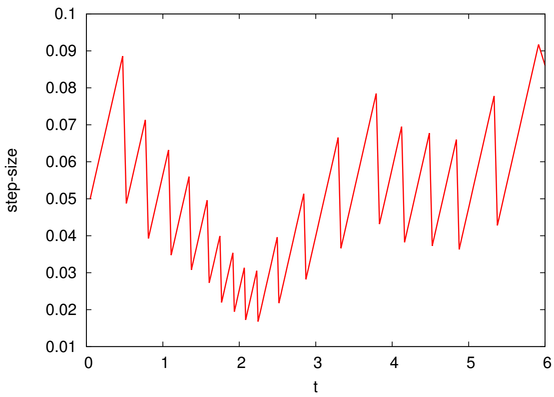


Figure 3.19: Change of the step-size in Test 2 on Rössler attractor

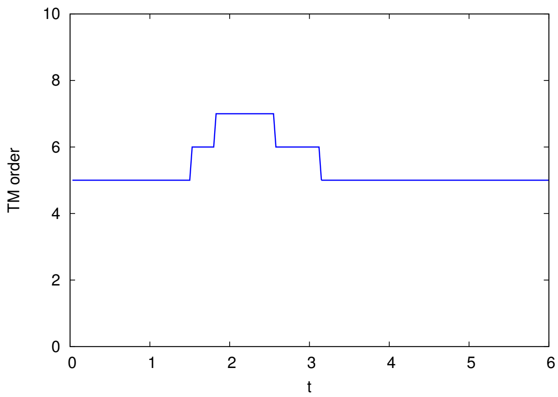


Figure 3.20: Change of the TM order in Test 3 on Rössler attractor

Example 3.4.3 (Rössler attractor). *We consider the chaotic attractor studied by Otto Rössler in [Rös76, Rös79]. It is characterized by the following ODE*

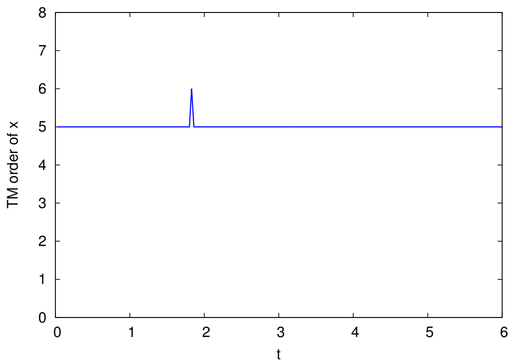
$$\begin{cases} \dot{x} &=& -y - z \\ \dot{y} &=& x + a \cdot y \\ \dot{z} &=& b + z \cdot (x - c) \end{cases}$$

with the parameter values $a = 0.2$, $b = 0.2$ and $c = 5.7$. We perform the TM integration with different computational schemes from the initial set

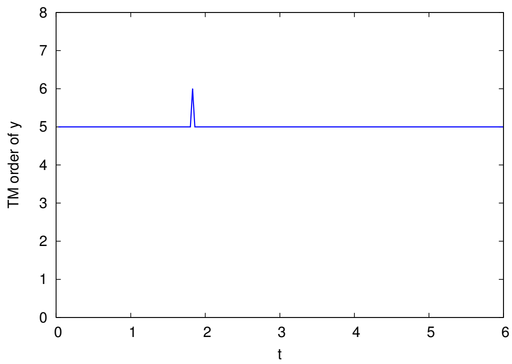
$$x(0) \in [-0.2, 0.2],\ \ y(0) \in [-8.6, -8.2],\ \ z(0) \in [-0.2, 0.2]$$

The interval threshold is set by $I_{\max} = [-10^{-4}, 10^{-4}]^3$. Again, we do four tests according to different integration schemes, the results are given by Table 3.12. In Figure 3.19 to 3.21, we also present the changes of the step-sizes and orders in those tests.

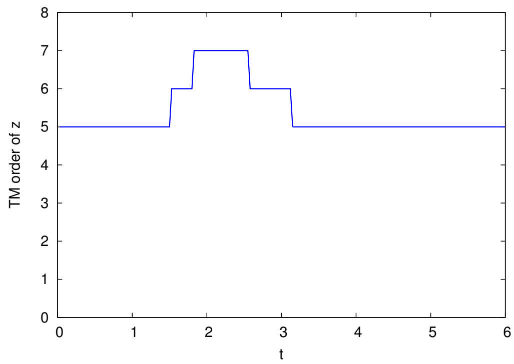
From the above experiments one may find that a great improvement in performance can be obtained by properly setting the adaptive parameters. We should also point out that the independent adaptive order method may perform worse than the uniform one when the state variables are highly correlated, since in that case it is often necessary to increase the order in different dimensions together to ensure the safety of $I_{\max}$.



(a) Change of the order in the dimension x



(b) Change of the order in the dimension y



(c) Change of the order in the dimension z

Figure 3.21: Change of the TM orders in Test 4 on Rössler attractor

3.5 Time-varying uncertainties

Up to now, all continuous systems considered by us have only deterministic bahavior, since the modeling ODEs are assumed to have unique solutions. In this section, we allow an ODE to have some parameters which change continuously along the time in interval ranges but with unknown rates. Those parameters are also called *time-varying uncertainties*. Such ODEs are very useful in modeling the systems with uncertain disturbances or inputs whose time derivatives often can not be given explicitly. In this section, we present a method to compute TM flowpipes for the ODEs with time-varying uncertainties.

An ODE with m time-varying uncertainties $\vec{u}(t)$ can be given in the form of

$$\dot{\vec{x}} = f(\vec{x}, t, \vec{u}(t)) \tag{3.5}$$

such that $\vec{u}(t) \in \mathcal{C}^0([-\infty, +\infty])$ and for all $t \geq 0$, $\vec{u}(t) \in \mathcal{U}$ for some $\mathcal{U} \in \mathbb{IR}^m$.

Then the IVP on an ODE with time-varying uncertainties becomes

$$\begin{cases} \dot{\vec{x}} & = & f(\vec{x}, t, \vec{u}(t)) \\ \vec{x}(0) & = & \vec{x}_0 \end{cases}$$

To give an enclosure of the exact solutions, we need to consider all possibilities of $\vec{u}$ in the integration task.

The integration of the ODEs with uncertainties has been studied elsewhere [SB06, LS07] whereas either the solutions are not validated, i.e., they are not over-approximations, or the uncertainties are assumed to be time-invariant. In [LS07], Lin and Stadtherr describe a method to replace those uncertain parameters by their interval enclosures in an integration job. The results are guaranteed to be over-approximations nevertheless the parameters should be time-invariant. We extend their method to take uncertainties in TM integration framework, we prove that the result generated by our method is always an over-approximation of the solutions of the given ODE with time-varying uncertainties.

Given $\dot{\vec{x}} = f(\vec{x}, t, \vec{u}(t))$ with $\vec{u} \in \mathcal{U}$ for some $\mathcal{U} \in \mathbb{IR}^m$, we assume that $f(\vec{x}, t, \vec{u}(t))$ is locally Lipschitz continuous w.r.t. $\vec{x}$ and continuous w.r.t. t. To compute the TM flowpipes over a bounded time horizon, we use the same integration step as that presented in Section 3.3 except that

(a) we allow polynomials to have interval coefficients, and

(b) the parameters $\vec{u}$ in every operation is replaced by their interval enclosures $\mathcal{U}$. That is, we consider $f(\vec{x}, t, \mathcal{U})$ instead of $f(\vec{x}, t, \vec{u}(t))$ in TM integration.

Theorem 3.5.1. *Given an ODE $\dot{\vec{x}} = f(\vec{x}, t, \vec{u}(t))$ such that $\vec{u}(t)$ are time-varying uncertainties bounded by some interval $\mathcal{U}$. If $f(\vec{x}, t, \vec{u}(t))$ is locally Lipschitz continuous w.r.t. $\vec{x}$ and continuous w.r.t. t, then a TM flowpipe computed by our method in a time step is an over-approximation of the solutions there.*

Proof. We only need to prove the conservativeness of the remainder evaluation. Assume that we have an estimation I_l such that the TM $(p_l(\vec{x}_l, t), I_l)$ is an estimation of the solution set from $\vec{x}_l \in X_l$ in the time interval $[0, \delta]$. We prove that if the following Picard operator

$$\mathbb{P}_f(g)(\vec{x}_l, t) = \vec{x}_l + \int_0^t f(g(\vec{x}_l, s), s, \vec{u}(s))ds$$

contracts on (p_l, I_l) for all $\vec{u} \in \mathcal{U}$ then the TM is an over-approximation of all solutions. Since $f(\vec{x}, t, \vec{u}(t))$ is locally Lipschitz continuous w.r.t. $\vec{x}$ and continuous w.r.t. t, for any $\vec{u}(t)$, if a solution $\vec{x}(\vec{x}_l, t)$ of $\dot{\vec{x}} = f(\vec{x}, t, \vec{u}(t))$ exists then it is unique and is a fixed point of the Picard operator, i.e.,

$$\vec{x}(\vec{x}_l, t) = \vec{x}_l + \int_0^t f(\vec{x}(\vec{x}_l, s), s, u(s))ds\,.$$

Since the Picard operator defines a continuous mapping, by Schauder fixed point theorem, if it maps (p_l, I_l) to a subset of it for some $\vec{u}(t)$, then the unique solution w.r.t. $\vec{u}(t)$ is included by the TM. Thereby, if it maps (p_l, I_l) to a subset of it for all $\vec{u}(t) \in \mathcal{U}$ then all solutions are included by (p_l, I_l).

To verify the contractiveness of the Picard operator on (p_l, I_l), we use TM arithmetic to evaluate

$$\mathcal{P}_f((p_l, I_l))(\vec{x}_l, t) \;=\; \vec{x}_l + \int_0^t f((p_l(\vec{x}_l, s), I_l), s, \mathcal{U})ds$$

If $\mathcal{P}_f((p_l, I_l))$ generates a subset of (p_l, I_l) then all solutions from X_l in the time interval $[0, \delta]$ are contained in (p_l, I_l). □

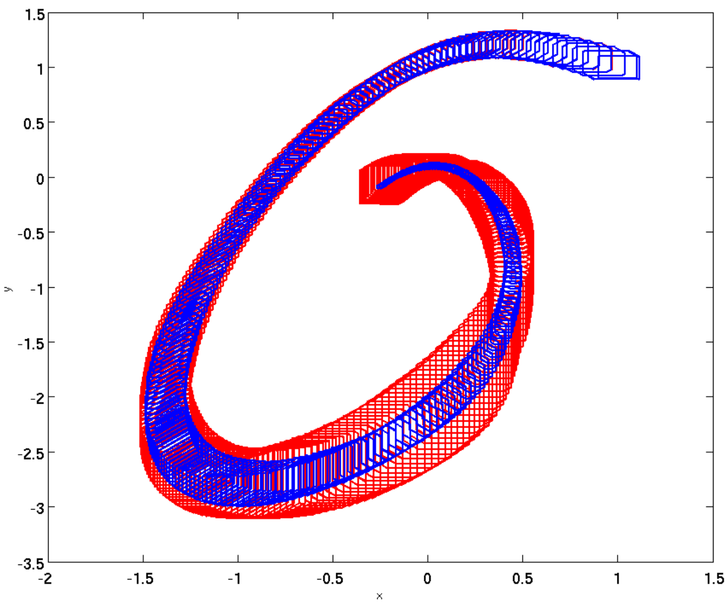


Figure 3.22: Flowpipe over-approximations with disturbances (in red) and without disturbances (in blue)

However, it is not easy to measure or control the overestimation in a TM when its polynomial part has non-degenerate interval coefficients. Therefore, we further over-approximate such a TM by another one whose polynomial part has only degenerate coefficients. For example, given a TM

$$(\ I_0 + I_1 \cdot \psi_1(\vec{x}) + \cdots + I_k \cdot \psi_k(\vec{x}) \ , \ I \ )$$

wherein $\psi_1(\vec{x}), \ldots, \psi_k(\vec{x})$ are the monomials and $I_0, \ldots, I_k$ are the interval coefficients. We over-approximate it by the following TM

$$\left(\begin{array}{c} \mathrm{Mid}(I_0) + \mathrm{Mid}(I_1) \cdot \psi_1(\vec{x}) + \cdots + \mathrm{Mid}(I_k) \cdot \psi_k(\vec{x}) \, , \\ I + \mathtt{Int}(I_0 - \{\mathrm{Mid}(I_0)\} + (I_1 - \{\mathrm{Mid}(I_1)\}) \cdot \psi_1(\vec{x}) + \cdots + (I_k - \{\mathrm{Mid}(I_k)\}) \cdot \psi_k(\vec{x})) \end{array} \right)$$

To achieve better approximation quality, it is also possible to treat the uncertainties by a new set of variables in part of the computation. However, we have to deal with the multivariate polynomials with more variables.

Example 3.5.2. *We introduce time-varying disturbances to the jet engine model, the ODE becomes*

$$\begin{cases} \dot{x} &= -y - 1.5x^2 - 0.5x^3 - 0.5 + u_1 \\ \dot{y} &= 3x - y + u_2 \end{cases}$$

wherein u_1, u_2 are the parameters for the disturbances, and we assume that $u_1, u_2 \in [-0.005, 0.005]$. We illustrate the TM flowpipes with as well as without the disturbances in Figure 3.22.

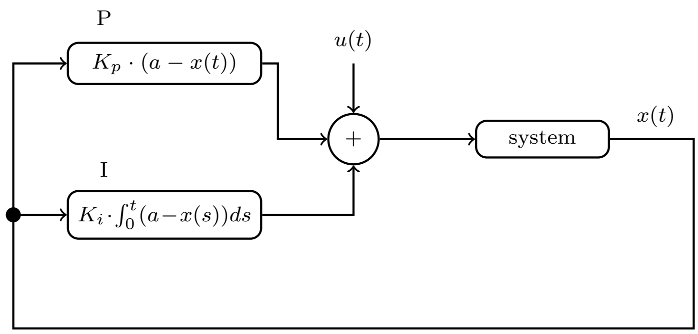


Figure 3.23: Block diagram of a PI controller with disturbance

Example 3.5.3 (PI controller with disturbance)**.** *Figure 3.23 shows the basic structure of a PI controller with a disturbance $u(t)$. The controller attempts to minimize the difference between the system output $x(t)$ and a desired constant value a. The proportion term P generates an output value that is proportional to the current difference, and the integral term I produces an output value which is the accumulation of the past differences since the system starting time. In practice, a PI controller is often disturbed by the working environment, we then introduce such a disturbance $u(t)$. Hence the system input $y(t)$ is governed by*

$$y(t) = K_p \cdot (a - x(t)) + K_i \cdot \int_0^t (a - x(s))ds + u(t)$$

As an example, we consider a cruise control system of a car which is similar to the one described in [ÅM11]. The velocity v of the car is governed by the differential equation

$$\dot{v} = -0.101 \cdot (v - v_e) + 1.3203 \cdot (x - x_e) - 0.01v^2$$

wherein x is the control input from the engine, v_e is the desired velocity, x_e is the desired input value and the term $0.01 \cdot v^2$ represents the friction. The cruise controller performs the PI control

$$x(t) = K_p \cdot (v_e - v(t)) + K_i \cdot \int_0^t (v_e - v(s))ds + u(t)$$

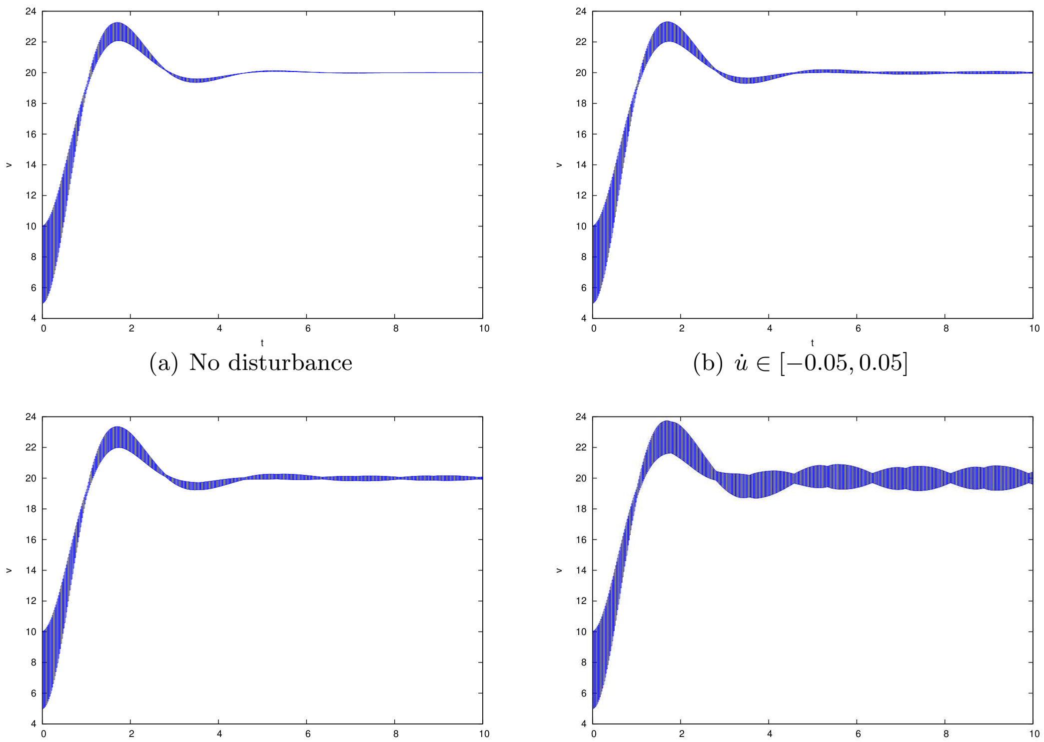
wherein we set $v_e = 20$, $x_e = 0.1616$, $K_p = 1$, $K_i = 3$ and assume that $\dot{u}$ is continuous over $t \in [0, +\infty)$. Hence the controlled system can be modeled by the following ODE

$$\begin{cases} \dot{v} &= -0.101 \cdot (v - 20) + 1.3203 \cdot (x - 0.1616) - 0.01v^2 \\ \dot{x} &= -(-0.101 \cdot (v - 20) + 1.3203 \cdot (x - 0.1616) - 0.01v^2) + 3 \cdot (20 - v) + \dot{u} \end{cases}$$

We consider the initial velocities in $[5, 10]$ and compute the TM flowpipes for the time horizon $[0, 10]$. In Figure 3.24, we illustrate the flowpipe over-approximations computed under different bounds on the disturbance rate $\dot{u}$.

3.6 Fast Taylor model flowpipe generation for linear ODEs

Although TM integration may also handle the systems defined by linear ODEs, its performance is not as good as that of the approaches which are specialized for those



(a) No disturbance (b) $\dot{u} \in [-0.05, 0.05]$

(c) $\dot{u} \in [-0.1, 0.1]$ (d) $\dot{u} \in [-0.5, 0.5]$

Figure 3.24: Flowpipe over-approximations under different bounds on the disturbance rate $\dot{u}$

systems. In this section, we present a method to efficiently compute TM flowpipes for linear ODEs, the main idea is to combine the use of TMs and support functions.

An uncertain linear system can be defined by an ODE of the following form

$$\dot{\vec{x}} = A \cdot \vec{x} + \vec{u} \tag{3.6}$$

such that A is a constant square matrix and the uncertainties $\vec{u}$ are bounded by an interval set $\mathcal{U}$. Since such an ODE has the following closed-form solution

$$\varphi(\vec{x}_0, t) \;=\; \exp(A \cdot t) \cdot \vec{x}_0 \;+\; \int_0^t \exp(A \cdot (t - s)) \cdot \vec{u}(s)\, ds \tag{3.7}$$

for the initial condition $\vec{x}(0) = \vec{x}_0$, there is no need to compute a preconditioned Taylor expansion, or apply remainder refinement iterations in computing a TM flowpipe. Instead, we may use a variant of Scheme II in Section 3.2.1. That is, we fixed a time step-size δ, over-approximate the solution in the first time step by a TM $\mathcal{F}_1$, then the TM flowpipes over the time horizon $[\delta, N \cdot \delta]$ can be recursively computed by

$$\mathcal{F}_i = \Phi \cdot \mathcal{F}_{i-1} \oplus \mathcal{B} \tag{3.8}$$

for $i = 2, 3, \ldots, N$, such that $\Phi = \exp(A{\cdot}\delta)$, and $\mathcal{B} = \{\int_0^\delta \exp(A{\cdot}(\delta{-}s)){\cdot}\vec{u}(s)\, ds \,|\, \vec{u}(s) \in \mathcal{U}\}$. Since Φ and $\mathcal{B}$ can hardly be computed exactly in general, over-approximations of them are often used in the iterations. Such a scheme is extensively used by many convex flowpipe over-approximations such as zonotopes [Gir05] and support functions [LG09]. Here, we present a method to compute $\mathcal{F}_1, \ldots, \mathcal{F}_N$ as TMs which often have better accuracy than convex over-approximations.

Compute $\mathcal{F}_1$. The purpose is to compute a TM $(p_1(\vec{x}_0, t), I_1)$ such that $\varphi(\vec{x}_0, t) \in p_1(\vec{x}_0, t) + I_1$ for all $t \in [0, \delta]$. To do so, we first compute an order k TM matrix over-approximation $(p_\Phi(t), I_\Phi)$ for $\exp(A \cdot t)$,

$$p_\Phi(t) \;=\; I_d + At + \frac{1}{2}A^2t^2 + \cdots + \frac{1}{k!}A^kt^k$$

wherein I_d is an identity matrix. The remainder interval matrix I_Φ can be evaluated base on the Lagrange remainder $\frac{A^{k+1} \cdot t^{k+1}}{(k+1)!} \exp(A \cdot \xi)$ with some $\xi \in [0, \delta]$. That is, we first compute the value of $\rho = \exp(\|A\|_\infty \cdot \delta)$ wherein $\|\cdot\|_\infty$ denotes the maximum norm. Then the matrix $\exp(A \cdot \xi)$ is contained in the interval matrix M_ρ whose entries are all defined by $[-\rho, \rho]$. Hence, $I_\Phi = \frac{A^{k+1} \cdot t^{k+1}}{(k+1)!} M_\rho$ contains the remainder for p_Φ for all $t \in [0, \delta]$. By choosing k sufficiently large, we are able to obtain arbitrarily good accuracy.

A TM for $\exp(A \cdot (t-s))$ can be computed similarly, and then by using TM arithmetic, we are able to obtain a TM $(p_\mathcal{B}(t), I_\mathcal{B})$ for $\int_0^t \exp(A \cdot (t-s)) \cdot \mathcal{U}\, ds$ with $t \in [0, \delta]$. Therefore, the first flowpipe over-approximation $\mathcal{F}_1$ is computed by

$$(p_\Phi(t), I_\Phi) \cdot \vec{x}_0 + (p_\mathcal{B}(t), I_\mathcal{B})\,.$$

Lemma 3.6.1. *For all $t \in [0, \delta]$, we have that $\varphi(\vec{x}_0, t) \in (p_\Phi(t), I_\Phi) \cdot \vec{x}_0 + (p_\mathcal{B}(t), I_\mathcal{B})$.*

Compute the remaining flowpipe over-approximations. By expanding the recurrence relation (3.8), the i-th TM flowpipe can be directly computed based on the first one.

$$\mathcal{F}_i = \Phi^{i-1} \cdot \mathcal{F}_1 \;\oplus\; \bigoplus_{j=0}^{i-2} \Phi^j \cdot \mathcal{B}$$

In our computation, we replace Φ and $\mathcal{B}$ by their over-approximations $(p_\Phi(\delta), I_\Phi)$ and $(p_\mathcal{B}(\delta), I_\mathcal{B})$ respectively. Hence, we have that

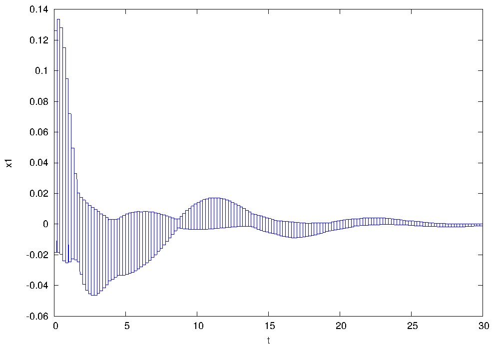
$$\begin{aligned} \mathcal{F}_i(t) \;&=\; (p_\Phi(\delta), I_\Phi)^{i-1} \cdot (p_\Phi(t), I_\Phi) \cdot \vec{x}_0 + (p_\Phi(\delta), I_\Phi)^{i-1} \cdot (p_\mathcal{B}(t), I_\mathcal{B}) \\ &\quad + \sum_{j=0}^{i-2} \big((p_\Phi(\delta), I_\Phi)^j \cdot (p_\mathcal{B}(\delta), I_\mathcal{B})\big) \end{aligned} \tag{3.9}$$

wherein $t \in [0, \delta]$. Since I_Φ can be made arbitrarily small when the order k is large enough, the main source of the overestimation in $\mathcal{F}_i$ is $I_\mathcal{B}$. To make $\mathcal{F}_i$ as tight as possible, we may represent $I_\mathcal{B}$ by its support function during the computation, since we only need to compute linear mappings and Minkowski sums.

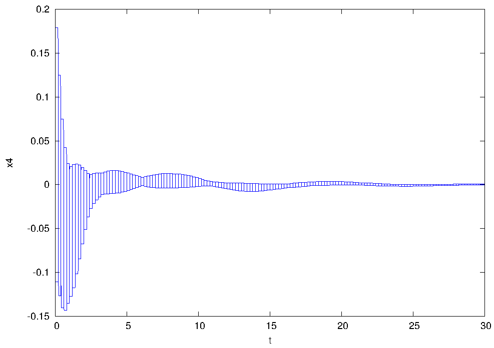
Theorem 3.6.2. *For all $1 \le i \le N$, we have that $\varphi(\vec{x}_0, (i-1)\delta + t) \in \mathcal{F}_i(t)$ for $t \in [0, \delta]$.*

We compare our method with the support function methods implemented in SpaceEx based on the helicopter example [FLD+11]. It is a continuous system defined by a linear ODE with 29 variables: $x_1, x_2, \ldots, x_{28}$ and t. The initial condition is defined by

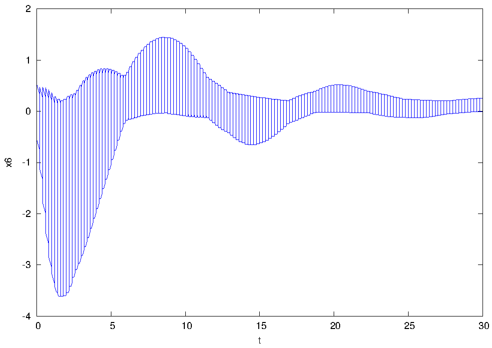
$$x_1, \ldots, x_8 \in [0, 0.1], \;\; x_9, \ldots, x_{28} = 0, \;\; t = 0$$



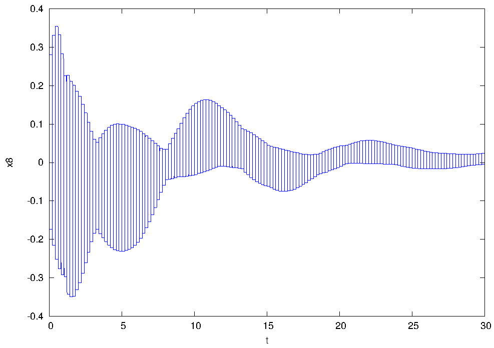
(a) Projection in t-x_1 plane



(b) Projection in t-x_4 plane



(c) Projection in t-x_6 plane



(d) Projection in t-x_8 plane

Figure 3.25: Flowpipe over-approximations of the helicopter example

and the time horizon under our consideration is $[0, 30]$.

We first apply the LGG algorithm [LG09] in SpaceEx with the step-size 0.1 and the tolerance 0.001. It costs around 35 seconds to compute box flowpipes, but more than 2000 seconds for the octagon ones. When we use the STC algorithm [FKL13] with the tolerance 0.05, only 12 seconds is needed for generating the box flowpipes, and the octagon ones cost 340 seconds.

When our method is used, the TM flowpipes with step-size 0.2 and order 70 are computed in 198 seconds. It is verified that the TM over-approximation at $t = 30$ is contained in the octagon over-approximation produced by SpaceEx at the same time. We illustrate the projections of our results in Figure 3.25. Moreover, we also tried the TM integration method on the example, but was not able to obtain any result in 2 hours.

We should mention that it is often not necessary for SpaceEx to compute octagon flowpipes in a safety verification task. Since the critical directions given by the unsafe set are usually not too many. However, octagon flowpipes are sometimes still needed for the reuse purpose. On the other hand, a TM flowpipe is already overall accurate when the order is high enough, and it can be directly reused with different unsafe specifications.

Unbounded initial sets. We investigate the possibility to deal with an unbounded initial set for a certain linear continuous system, that is, $\vec{u}$ is a constant vector. We assume that the initial set is defined by a polyhedra

$$X_0 = \{\vec{x} \in \mathbb{R}^n \mid \bigwedge_{1 \leq j \leq m} (\vec{a}_j^T \cdot \vec{x} \leq b_j)\}$$

which could be unbounded. Bases on the solution form (3.7), we can infer that the reachable set $\varphi(X_0, t)$ for all $t \in [0, +\infty)$ is defined by

$$\varphi(X_0, t) = \{\vec{x} \in \mathbb{R}^n \mid \bigwedge_{1 \leq j \leq m} (\vec{a}_j^T \cdot \varphi(\vec{x}, -t) \leq b_j)\}$$

since $\vec{x} \in \varphi(X_0, t)$ if and only if $\varphi(\vec{x}, -t) \in X_0$ for all $\vec{x} \in \mathbb{R}^n$. The backward flowmap $\varphi(\vec{x}, -t)$ may be over-approximated by a TM $(p_b(\vec{x}, -t), I_b)$ such that p_b may have interval coefficients. It can be done in a similar way that $\mathcal{F}_1$ is computed. Therefore, for all $1 \leq j \leq m$, we can compute a TM (with interval coefficients) $(p_j(\vec{x}, -t), [\alpha_j, \beta_j]) = \vec{a}_j^T \cdot (p_b(\vec{x}, -t), I_b)$ such that

$$p_j(\vec{x}, -t) + \alpha_j \ \leq \ \vec{a}_j^T \cdot \varphi(\vec{x}, -t) \ \leq \ p_j(\vec{x}, -t) + \beta_j$$

for all $\vec{x} \in \mathbb{R}^n$. Hence, an over-approximation of $\varphi(X_0, t)$ can be computed by

$$S_o = \{\vec{x} \in \mathbb{R}^n \mid \bigwedge_{1 \leq j \leq m} (p_j(\vec{x}, -t) + \alpha_j \leq b_j)\}$$

and an under-approximation of it can be obtained as

$$S_u = \{\vec{x} \in \mathbb{R}^n \mid \bigwedge_{1 \leq j \leq m} (p_j(\vec{x}, -t) + \beta_j \leq b_j)\}$$

If the TM $(p_b(\vec{x}, -t), I_b)$ can not be computed by one shot, we may also use flowpipe construction. Note that there is no need to truncation a polynomial term containing $\vec{x}$ in all computations. Further investigation on the applicability of the method will be done in the future.

Chapter 4

Taylor Model Flowpipes for Hybrid Systems

Hybrid systems are dynamical systems which are equipped with both continuous and discrete dynamics. The mathematical models for hybrid systems under our consideration are hybrid automata. To verify a safety property on a hybrid system, we compute the reachable set of its hybrid automaton and check whether an unsafe state is contained. However, such a task is can not be done explicitly in general since the reachability problem on hybrid automata is not decidable [ACH$^+$95, HKPV95] even in the case that all continuous dynamics are defined by linear ODEs. Therefore, similar to the reachability analysis on continuous systems, we seek to compute an over-approximation for the reachable set, and we show that it can also be done by computing flowpipe over-approximations.

Extending the use of Taylor Model (TM) arithmetic to generate reachable set over-approximations for hybrid automata is non-trivial. Besides the continuous dynamics defined by ODEs, we also need to handle mode invariants as well as the guards and reset mappings of discrete jumps. Given a computed TM flowpipe $\mathcal{F}$ under the continuous dynamics of a mode, we should exclude the states in $\mathcal{F}$ which stay outside of the mode invariant. To do so, we intersect $\mathcal{F}$ with the invariant. The situation is similar when handling a discrete jump. We intersect the TM flowpipes with the jump guard to obtain the set of states from which the jump can be executed. Therefore, it requires a method to compute flowpipe/invariant and flowpipe/guard intersections, and such a job is often not easy. In this chapter, we present two novel approaches to compute over-approximations for those intersections. They are *domain contraction* and *range over-approximation*. Since they are high-level techniques, we may apply them with various heuristics and other techniques in different situations.

In order to relieve the burden in the subsequent reachability computation, it is often necessary to aggregate several flowpipe over-approximations as one set. Since TMs are not closed under union, we need to compute a TM over-approximation for the result. In order not to introduce too much overestimation, we present several heuristics to compute a proper template for a parallelotopic aggregation. We show that the parallelotope can be efficiently translated to an order 1 TM. The possibilities of computing other classes of aggregations are also discussed.

4.1 Hybrid automata

We use *hybrid automata* as the formal models of hybrid systems. A hybrid automaton can be viewed as an extension of a *finite automaton* such that a finite set of real-valued variables are introduced. The variable values change continuously in a discrete state according to an ODE. They may also be updated by a discrete jump according to the reset mapping associated to it. We give the definition of a hybrid automaton as follows.

Definition 4.1.1 (Hybrid automaton)**.** *A* hybrid automaton *is denoted by a tuple*

$$\mathcal{A} = (\mathit{Loc}, \mathit{Var}, \mathit{Inv}, \mathit{Flow}, \mathit{Lab}, \mathit{Trans}, \mathit{Guard}, \mathit{Reset}, \mathit{Init})$$

wherein

- *Loc is finite set of discrete states which are also called* locations *or* modes*.*
- *Var consists n* real-valued variables *$\vec{x}$ for some integer $n > 0$.*
- *$\mathit{Inv} : \mathit{Loc} \to 2^{\mathbb{R}^n}$ associates a mode $\ell \in \mathit{Loc}$ an* invariant *$\mathcal{I}_\ell \subseteq \mathbb{R}^n$ such that the variables can only take the values in $\mathcal{I}_\ell$ when $\mathcal{A}$ is in the mode ℓ.*
- *Flow is a function that associates a mode $\ell \in \mathit{Loc}$ a* continuous dynamics *$\dot{\vec{x}} = f_\ell(\vec{x}, t)$ which governs the variable value change in the mode ℓ. We assume that f_ℓ is locally Lipschitz continuous w.r.t. $\vec{x}$ in the invariant of ℓ and continuous w.r.t. t in $\mathbb{R}$.*
- *Lab is the set of* labels *for the discrete transitions.*
- *$\mathit{Trans} \subseteq \mathit{Loc} \times \mathit{Lab} \times \mathit{Loc}$ is the set of* discrete transitions *or* jumps *associated with their labels among the modes.*
- *$\mathit{Guard} : \mathit{Trans} \to 2^{\mathbb{R}^n}$ assigns a jump $\alpha \in \mathit{Trans}$ a guard $\mathcal{G}_\alpha \subseteq \mathbb{R}^n$ such that the jump α can be executed if and only if the state variables are of the values in $\mathcal{G}$. Here, we only consider the jumps which are* not urgent*, that is, the execution of a jump is not mandatory when the guard is satisfied.*
- *$\mathit{Reset} : \mathit{Trans} \to (\mathbb{R}^n \to \mathbb{R}^n)$ associates a jump $\alpha \in \mathit{Trans}$ a reset mapping $\pi_\alpha : \mathbb{R}^n \to \mathbb{R}^n$ that updates the values of the variables after the execution of α.*
- *$\mathit{Init} \subset \mathit{Loc} \times \mathbb{R}^n$ is the initial state set of $\mathcal{A}$. Every state in Init also satisfies the mode invariants.*

A state *of $\mathcal{A}$ is a pair $\langle \ell, \vec{v} \rangle$ wherein ℓ denotes the current mode and $\vec{v} \in \mathbb{R}^n$ are the values of the variables. Additionally, we call ℓ the* discrete state *and $\vec{v}$ the* continuous state*.*

Intuitively, the state space of a hybrid automaton with n variables and l modes consists of l independent n-dimensional spaces defined by the mode invariants. The system state changes in one space according to a continuous dynamics and jump from one space to another by making a discrete transition.

Example 4.1.2. *We revisit the bouncing ball model presented in Example 1.1.1. We enhance the model by introducing an air friction which is represented by a quadratic expression in the velocity. We present the hybrid automaton of the example in Figure 4.1, the components are defined as follows.*

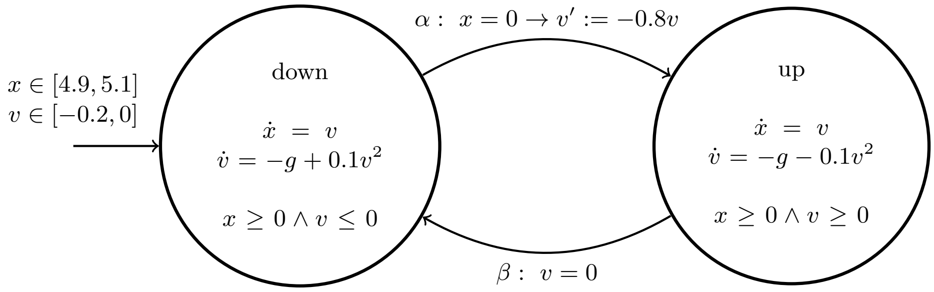


Figure 4.1: Hybrid automaton of the bouncing ball with air friction

- *Loc* $= \{down, up\}$ *in which the two modes correspond to the two directions of the velocity.*
- *Var* $= \{x, v\}$ *wherein x denotes the vertical distance from the ball to the ground, and v is the velocity of the ball.*
- *Inv*$(down) = \{(x, v) \in \mathbb{R}^2 \,|\, x \geq 0 \wedge v \leq 0\}$, *Inv*$(up) = \{(x, v) \in \mathbb{R}^2 \,|\, x \geq 0 \wedge v \geq 0\}$.
- *Flow*$(down) = \begin{cases} \dot{x} = v \\ \dot{v} = -g + 0.1v^2 \end{cases}$, *Flow*$(up) = \begin{cases} \dot{x} = v \\ \dot{v} = -g - 0.1v^2 \end{cases}$.
- *Lab* $= \{\alpha, \beta\}$.
- *Trans* $= \{\langle down, \alpha, up\rangle, \langle up, \beta, down\rangle\}$.
- *Guard*$(\langle down, \alpha, up\rangle) = \{(x, v) \in \mathbb{R}^2 \,|\, x = 0\}$,
  Guard$(\langle up, \beta, down\rangle) = \{(x, v) \in \mathbb{R}^2 \,|\, v = 0\}$.
- *Reset*$(\langle down, \alpha, up\rangle) = \begin{cases} x \mapsto x \\ v \mapsto -0.8v \end{cases}$, *Reset*$(\langle up, \beta, down\rangle) = \begin{cases} x \mapsto x \\ v \mapsto v \end{cases}$.
- *Init* $= \{\langle down, (x, v)\rangle \,|\, x \in [4.9, 5.1] \wedge v \in [-0.2, 0]\}$.

In Figure 4.1, the identity reset mapping is neglected. The state space is composed of two vector fields.

An *execution* (*run*) of a hybrid automaton $\mathcal{A}$ is a sequence of states

$$\langle \ell_0, \vec{v}_0\rangle, \langle \ell_1, \vec{v}_1\rangle, \langle \ell_2, \vec{v}_2\rangle, \ldots$$

which can be infinite such that $\langle \ell_0, \vec{v}_0\rangle \in$ *Init* and for two successive states $\langle \ell, \vec{v}\rangle$, $\langle \ell', \vec{v}'\rangle$ either of the following two evolutions holds.

- *Continuous evolution (time delay)* - We have that (i) $\ell = \ell'$, (ii) $\vec{v}' = \varphi_f(\vec{v}, t)$ for some $t \geq 0$ such that $\dot{\vec{x}} = f(\vec{x}, t)$ is the continuous dynamics of ℓ, and (iii) $\varphi_f(\vec{v}, t') \in$ *Inv*(ℓ) for all $t' \in [0, t]$. We denote it by $\langle \ell, \vec{v}\rangle \overset{t}{\leadsto} \langle \ell', \vec{v}'\rangle$.
- *Discrete evolution (jump)* - There exists a jump $\langle \ell, \alpha, \ell'\rangle \in$ *Trans* such that $\vec{v} \in$ *Guard*$(\langle \ell, \alpha, \ell'\rangle)$, $\vec{v}' \in$ *Inv*(ℓ'), and $\vec{v}' = \pi(\vec{v})$ wherein *Reset*$(\langle \ell, \alpha, \ell'\rangle) = \pi$. We denote it by $\langle \ell, \vec{v}\rangle \xrightarrow{\alpha} \langle \ell', \vec{v}'\rangle$.

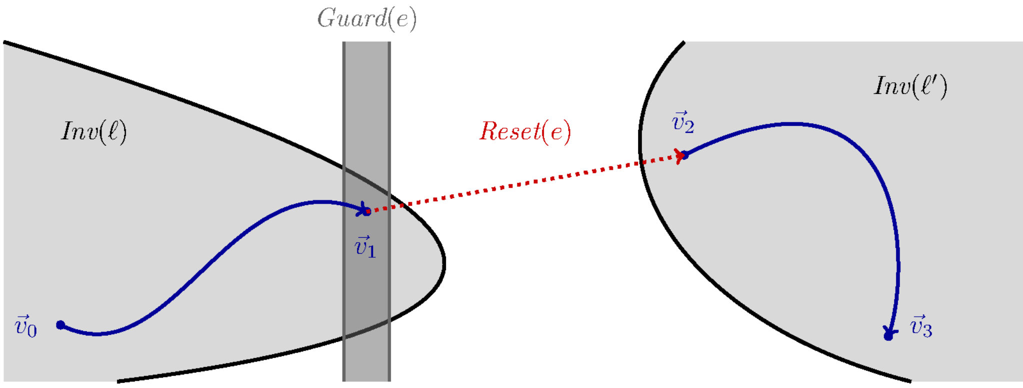


Figure 4.2: Example of an execution of a hybrid automaton

We illustrate an execution

$$\langle \ell, \vec{v}_0 \rangle,\ \langle \ell, \vec{v}_1 \rangle,\ \langle \ell', \vec{v}_2 \rangle,\ \langle \ell', \vec{v}_3 \rangle$$

in Figure 4.2 wherein e denotes a jump from ℓ to ℓ'. In the following content, we sometimes omit the labels in a hybrid automaton if there is at most one jump from ℓ to ℓ' for any two modes ℓ, ℓ'.

An execution is *zeno* if it has infinitely many jumps but the total amount of time delay is bounded. For instance, every execution which contains infinitely many jumps of the hybrid automaton in Example 4.1.2 is zeno. Zeno executions are not realistic, since no physical processor can do infinitely many actions in a finite amount of time. We call a hybrid automaton *non-zeno* if it has no zeno execution. A simple way to exclude zeno executions from a hybrid automaton is to force the system to stay in a mode for at least a specified amount of time before a jump is enabled.

We call an execution *blocking* if it can reach a state $\langle \ell, \vec{v} \rangle$ and there is no state $\langle \ell', \vec{v}' \rangle$ such that $\langle \ell, \vec{v} \rangle \overset{t}{\rightsquigarrow} \langle \ell', \vec{v}' \rangle$ for some $t > 0$, or $\langle \ell, \vec{v} \rangle \xrightarrow{\alpha} \langle \ell', \vec{v}' \rangle$ for a jump α. A hybrid automaton is *non-blocking* if it has no blocking execution. Since blocking executions represent behaviors in which time does not progress, they could be introduced by inconsistency in modeling. We may eliminate blocking executions from a hybrid automaton by adding new modes and jumps. In our analysis work, we do not require a hybrid automaton non-zeno or non-blocking.

Given a hybrid automaton $\mathcal{A}$, we say that a state $\langle \ell, \vec{v} \rangle$ is *reachable* if there exists a finite sequence which is an execution of $\mathcal{A}$ with the last state $\langle \ell, \vec{v} \rangle$. The set of reachable states are called *reachable set*. We also say that $\langle \ell, \vec{v} \rangle$ is reachable in a time interval T if the total amount of the time delay in an execution leading to $\langle \ell, \vec{v} \rangle$ is in T. Similarly, the state $\langle \ell, \vec{v} \rangle$ is reachable below j jumps if an execution leading to it involves at most j jumps.

Definition 4.1.3 (Hybrid reachability problem)**.** *Given a hybrid automaton $\mathcal{A}$, the* reachability problem *on $\mathcal{A}$ is to verify whether a given state is reachable. A* bounded *version of the problem is to determine whether the state is reachable in a bounded time interval and below a specified number of jumps.*

Subclass	Continuous dynamics	Guard/ invariant	Reset mapping	Bounded reachability	Unbounded reachability
TA	$\dot{\vec{x}} = 1$	$x \sim c$	$x \mapsto 0$	decidable	decidable
RHA	$\dot{\vec{x}} \in B$	$\vec{x} \in B$	$x \mapsto B$	decidable	undecidable
LHA I	$\dot{\vec{x}} = \vec{c}$	$p_L(\vec{x}) \leq 0$	$\vec{x} \mapsto p_L(\vec{x})$	decidable	undecidable
LHA II	$\dot{\vec{x}} = p_L(\vec{x})$	$p_L(\vec{x}) \leq 0$	$\vec{x} \mapsto p_L(\vec{x})$	undecidable	undecidable

Table 4.1: Decidability of the reachability problem on some subclasses of hybrid automata. Legends: TA: Timed Automata, RHA: Rectangular Hybrid Automata, LHA: Linear Hybrid Automata.

Decidability of the reachability problem. In Table 4.1, we give the decidability of the reachability problem on several subclasses of hybrid automata. The proofs are given elsewhere [AD94, ACH$^+$95, HKPV95]. The mode invariants and jumps guards under our consideration are defined by conjunctions of inequalities (constraints). In the table, we use $p_L(\vec{x})$ to denote a linear polynomial over $\vec{x}$, $\sim \in \{\leq, \geq, <, >\}$, and B denotes an interval. More precise complexities of the decidable problems can be found in the related work [AD94, ACH$^+$95, HKPV95]. For the decidable bounded reachability problems, it is not difficult to come up with a nondeterministic-polynomial algorithm for each of them.

In the thesis, we consider the hybrid automata with variables $\vec{x}$ such that (i) the continuous dynamics are defined by non-linear ODEs, (ii) the guards and invariants are defined by the inequalities of the form $g(\vec{x}) \leq 0$ wherein g is an analytic function, and (iii) the reset mappings are given by the form $\vec{x} \mapsto r(\vec{x})$ wherein r is also analytic.

We present a general procedure to compute the reachable set of a hybrid automaton by Algorithm 11. It is a fixed point computation whose termination is not guaranteed. Although the algorithm needs to handle infinitely (even uncountably) many hybrid states, we may collectively represent a set of states $\{\langle \ell, \vec{v} \rangle \mid \vec{v} \in V\}$ by $\langle \ell, V \rangle$ during the computation and the number of them can be reduced to be finite if the time horizon as well as the maximum number of jumps are bounded. Unfortunately, even the bounded reachable set can not be computed exactly in general, since those operations are not easy to handle. For example, to compute the exact reachable set under a continuous dynamics within an invariant can hardly be done in most cases.

We consider to use TM flowpipes to over-approximate the reachable set segments for a hybrid automaton. We may extend the flowpipe construction method for continuous systems to the hybrid case. In the following section, a general scheme is introduced.

4.2 Framework of the flowpipe construction

Given a hybrid automaton, we seek to over-approximate the set of the states which is reachable in a bounded time horizon $[0, \Delta]$ and with at most $\mathcal{J}$ jumps, such that Δ and $\mathcal{J}$ are user-specified. A general scheme to do it is described by Algorithm 12 wherein we assume that the initial set is collectively represented by the union of $\langle \ell_1, V_1 \rangle, \ldots, \langle \ell_m, V_m \rangle$ for some integer $m > 0$. The termination of the algorithm is ensured since we bound the time horizon as well as the jump depth.

The flowpipe representations should be proper with respect to the operations in Al-

Algorithm 11 Reachable set computation for hybrid automata

Input: a hybrid automaton $\mathcal{A}$

Output: the reachable set $\mathcal{R}$ of $\mathcal{A}$

$\mathcal{R} \leftarrow \emptyset$;
$\mathit{Queue} \leftarrow \emptyset$; # the queue to keep unvisited states
for all $\langle \ell, \vec{v} \rangle \in \mathit{Init}$ **do**
    $\mathit{Queue.enqueue}(\langle \ell, \vec{v} \rangle)$;
end for
while *Queue* is not empty **do**
    $\langle \ell, \vec{v} \rangle \leftarrow \mathit{Queue.dequeue}()$;
    $\mathcal{R}_\ell \leftarrow \mathit{Reach}_\ell(\langle \ell, \vec{v} \rangle)$; # compute the reachable set in ℓ
    $\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{R}_\ell$;
    for all $\langle \ell, \alpha, \ell' \rangle \in \mathit{Trans}$ **do**
        for all $\langle \ell, \vec{v} \rangle \in \mathcal{R}_\ell$ **do**
            if $\vec{v} \in \mathit{Guard}(\langle \ell, \alpha, \ell' \rangle)$ **then**
                $\vec{v}' \leftarrow \mathit{Reset}(\langle \ell, \alpha, \ell' \rangle)(\vec{v})$; # compute the reset mapping
                if $\langle \ell', \vec{v}' \rangle \notin \mathcal{R}$ **then**
                    $\mathit{Queue.enqueue}(\langle \ell', \vec{v}' \rangle)$; # $\langle \ell', \vec{v}' \rangle$ has not been visited
                end if
            end if
        end for
    end for
end while
return $\mathcal{R}$;

Algorithm 12 Flowpipe construction for hybrid automata

Input: a hybrid automaton $\mathcal{A}$, a bounded time horizon Δ, maximum jump depth $\mathcal{J}$
Output: over-approximation of the reachable set of $\mathcal{A}$ within time $[0, \Delta]$ and $\mathcal{J}$ jumps

$\mathcal{R} \leftarrow \emptyset$;
$\mathit{Queue} \leftarrow \emptyset$;
for all $i = 1, \ldots, m$ **do**
  $\mathit{Queue.enqueue}(\langle\langle \ell_i, V_i\rangle, 0, \mathcal{J}\rangle)$; # enqueue the initial sets
end for
while Queue is not empty **do**
  $\langle\langle \ell, V\rangle, t, j\rangle \leftarrow \mathit{Queue.dequeue}()$;
  $F \leftarrow \mathit{ComputeFlowpipes}_{\ell}(V, \Delta - t)$; # flowpipes in ℓ in time $\Delta - t$
  $\mathcal{R} \leftarrow \mathcal{R} \cup \{\langle \ell, \mathcal{F}\rangle \mid \mathcal{F} \in F\}$;
  for all $\langle \ell, \alpha, \ell'\rangle \in \mathit{Trans}$ **do**
    for all $\mathcal{F} \in F$ **do**
      if $j > 0$ **then**
        $\mathcal{F}_G \leftarrow \mathcal{F} \cap \mathit{Guard}(\langle \ell, \alpha, \ell'\rangle)$; # flowpipe/guard intersection
        if $\mathcal{F}_G \neq \emptyset$ **then**
          $\mathcal{F}_R \leftarrow \mathit{Reset}(\langle \ell, \alpha, \ell'\rangle)(\mathcal{F}_G)$; # compute the reset mapping
          if $\langle \ell', \mathcal{F}_R\rangle \not\subseteq \mathcal{R}$ **then**
            Compute the global starting time t_R of $\mathcal{F}_R$;
            $\mathit{Queue.enqueue}(\langle\langle \ell', \mathcal{F}_R\rangle, t_R, j-1\rangle)$;
          end if
        end if
      end if
    end for
  end for
end while
return $\mathcal{R}$;

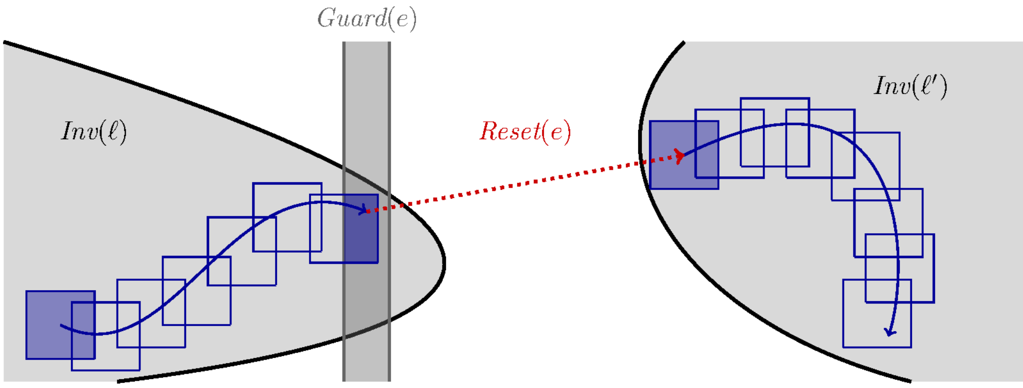


Figure 4.3: Flowpipe construction for a hybrid automaton

gorithm 12. As we said, the jump guards and mode invariants under our consideration are defined by inequalities. Hence an ideal class of representations should be closed under intersection with those sets. However, the goal can hardly be achieved. We consider to use TMs as the over-approximate representations for flowpipes. TMs are not closed under the intersection with a guard or an invariant, but are closed under polynomial mappings. The TM flowpipes in a mode can be computed by the TM integration technique except that in every integration step the computed TM flowpipe is intersected by the invariant. An example is illustrated in Figure 4.3, wherein the TM flowpipes are given by boxes. Besides, we also shade the initial flowpipe over-approximation in a mode and the flowpipe/guard intersection.

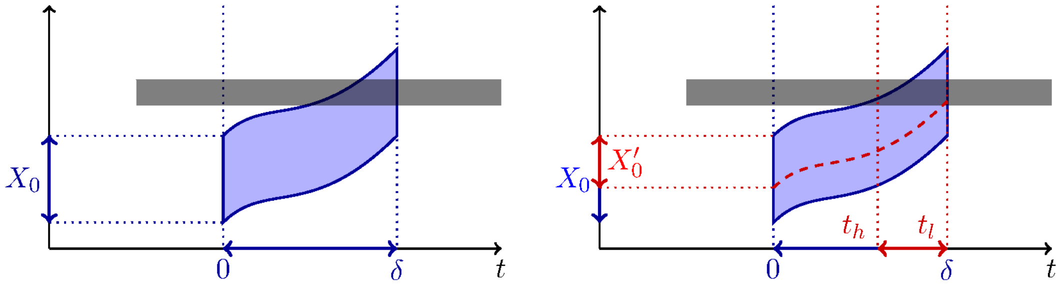
4.3 Flowpipe/guard intersections

We present two techniques to generate a TM for a flowpipe/guard intersection. A flowpipe/invariant intersection can be handled in a similar way. Since a TM flowpipe can always be equivalently or over-approximately transformed to a TM over an interval domain, we assume that the domains are intervals.

Given a jump guard $\mathcal{G} = \{\vec{x} \in \mathbb{R}^n \mid \bigwedge_{i=1}^{m} g_i(\vec{x}) \leq 0\}$ wherein $g_1, \ldots, g_m$ are analytic functions. The intersection of $\mathcal{G}$ and a TM flowpipe $\mathcal{F} = (p(\vec{x}_0, t), I)$ with $\vec{x}_0 \in X_0$, $t \in [0, \delta]$ for some $X_0 \in \mathbb{IR}^d$ is defined by

$$\mathcal{F} \cap \mathcal{G} \;=\; \{\vec{x} \mid \vec{x} = p(\vec{x}_0, t) + \vec{y} \wedge \vec{x}_0 \in X_0 \wedge t \in [0, \delta] \wedge \vec{y} \in I \wedge \bigwedge_{i=1}^{m} g_i(\vec{x}) \leq 0\} \tag{4.1}$$

which is often not a TM and can not be easily computed. However, we show that a TM over-approximation of it can be computed in the following two ways. (1) We contract the domain X_0 and $[0, \delta]$ to $X_0' \subseteq X_0$ and $[t_h, t_l] \subseteq [0, \delta]$ respectively, and then the TM $(p(\vec{x}_0, t), I)$ over the reduced domain is an over-approximation of the intersection. (2) We over-approximate $\mathcal{F}$ by a set $\overline{\mathcal{F}}$ whose intersection with $\mathcal{G}$ is easy to compute, and then we over-approximate the intersection $\overline{\mathcal{F}} \cap \mathcal{G}$ by a TM. We call the first method *domain contraction* and the second one *range over-approximation*.



(a) Flowpipe/guard intersection (b) After domain contraction

Figure 4.4: Example of domain contraction

4.3.1 Domain contraction

The purpose is to derive a *valid contraction* $X_0' \subseteq X_0$ and $[t_h, t_l] \subseteq [0, \delta]$ for the TM domain in (4.1). More precisely, the range of $(p(x_0, t), I)$ with $\vec{x}_0 \in X_0'$, $t \in [t_h, t_l]$ contains the intersection $\mathcal{F} \cap \mathcal{G}$.

Definition 4.3.1 (Valid contraction). *Given a TM $\mathcal{F}$ over a domain D and a guard set $\mathcal{G}$. We call $D' \subseteq D$ a* valid contraction *of D if and only if the set $\mathcal{F} \cap \mathcal{G}$ is contained in $\mathcal{F}$ over a domain D'.*

Example 4.3.2. *Given a TM flowpipe $x = 1 + x_0 \cdot t$ with $x_0 \in [-1, 1]$ and $t \in [0, 0.4]$. We denote the domain set by $[-1, 1] \times [0, 0.1]$. If the guard is defined by the set $\{x \in \mathbb{R} \,|\, x \geq 1.2 \wedge x \leq 1.4\}$, then $[0.5, 1] \times [0.2, 0.4]$ is a valid contraction of the domain. Note that a valid contraction is not necessarily the smallest one.*

The class of the contraction set X_0' is key to the overall performance. Here, we require that the representation of it is no more complex than that of X_0. Hence, we consider X_0' as an interval. The smallest valid interval contraction X_0^* can be obtained via optimization. The lower bounds of it in every dimension as well as the value t_h can be derived by solving the optimization problems

$$\inf\{z\} \quad \text{subject to} \quad \vec{x} = p(\vec{x}_0, t) + \vec{y} \wedge \vec{x}_0 \in X_0 \wedge t \in [0, \delta] \wedge \vec{y} \in I \wedge \bigwedge_{i=1}^{m} g_i(\vec{x}) \leq 0$$

for $z = \vec{x}_0[1], \ldots, \vec{x}_0[d]$ and $z = t$. For the upper bounds and the value t_l, we may solve the same problems except that inf is replaced by sup. We give an example in Figure 4.4. Unfortunately, such optimization problems are non-linear when either the order of the TM is more than 1, or at least one of the functions $g_1, \ldots, g_m$ is not a linear polynomial. Therefore, the computation of X_0^* is intractable in general. In the following content, we introduce some techniques to conservatively solve the optimization problems and yield thereby a superset of X_0^*.

Convex optimization. A convex optimization problem may be solved by an *interior-point method* [BV04] in polynomial-time complexity. Although the optimization problems on a flowpipe/guard intersection is usually not convex, we may perform a *convex relaxation* on it. That is, we compute a new polynomial p' from p and new functions $g'_1, \ldots, g'_m$ from $g_1, \ldots, g_m$ respectively such that the set of $\vec{x}$ subject to

$$\vec{x} = p'(\vec{x}_0, t) + \vec{y} \wedge \vec{x}_0 \in X_0 \wedge t \in [0, \delta] \wedge \vec{y} \in I \wedge \bigwedge_{i=1}^{m} g'_i(\vec{x}) \leq 0$$

is convex and contains the intersection. Techniques such as SemiDefinite Programming (SDP) relaxation [VB96, Pab03] and Sum-Of-Square (SOS) relaxation [Par00] could be applied. However, such methods still suffer from bad efficiency when d is large. Besides, it is also not easy to limit the overestimation introduced by a relaxation.

Interval Constraint Propagation (ICP). The technique of ICP is a combination of *interval arithmetic* and *constraint propagation* to conservatively solve a *Constraint Satisfiability Problem (CSP)*. That is to compute a superset of the exact solution set. It often costs much less time than finding the exact set, and returns a fairly tight enclosure.

Definition 4.3.3 (Constraint satisfiability problem)**.** *A Constraint Satisfiability Problem (CSP) is denoted by a triple $\langle X, D, Cons\rangle$ wherein X consists of finitely many variables which range in the set D, and Cons is a finite set of constraints over the variables in X. A* solution *to the CSP $\langle X, D, Cons\rangle$ is a valuation $\nu : X \to \mathbb{R}$ of the variables such that all constraints are satisfied. In addition, we call the set of all solutions the* solution set.

Then the domain contraction task becomes computing an interval enclosure of the solution set of the CSP $\langle X, D, Cons\rangle$ wherein

$$\begin{array}{rcl} X & = & \{\vec{x}_0[1], \ldots, \vec{x}_0[d], t\}, \\ D & = & X_0 \times [0, \delta], \\ Cons & = & \{\vec{x} = p(\vec{x}_0, t) + \vec{y},\ \vec{y} \in I,\ g_1(\vec{x}) \leq 0,\ \ldots,\ g_m(\vec{x}) \leq 0\} \end{array} \tag{4.2}$$

To derive an over-approximation of the solution set, we may use the *branch-and-prune* procedure presented in Algorithm 13. In each iteration, we check the inclusion of a solution in a subdivision B by interval arithmetic. If B possibly contains a solution, we then split it equally into two intervals in a dimension and put them back into the queue. Otherwise, B is discarded. The algorithm terminates and returns a set of intervals of widths $< \epsilon$, and the union of them defines an over-approximation of the exact solution set. If all functions are evaluated based on their Lipschitz and inclusion isotonic interval extensions, the resulting accuracy can always be improved by choosing smaller ϵ.

An interval valid contraction of $X_0 \times [0, \delta]$ can be obtained by branch-and-prune. We then compute an interval aggregation of the resulting set S. However, such an approach could also be computational expensive, since the number of intervals in S could be as many as $O((\lceil \frac{\mathrm{W}(D)}{\epsilon} \rceil)^{d+1})$. For the sake that we only compute interval contractions, it is more efficient to directly contract the original domain in every dimension without computing any subdivision of it. An efficient trade-off is described as follows.

An efficient approach. Our purpose is to compute a contraction for each component of $X_0 \times [0, \delta]$ without performing a subdivision on it. We follow a way to successively contract the original domain, as shown by Algorithm 14 wherein the input CSP encodes the domain contraction task. In each iteration of the main loop, we contract the current

Algorithm 13 Branch-and-prune algorithm

Input: a CSP $\langle X, D, \mathit{Cons}\rangle$, a threshold $\epsilon > 0$
Output: an over-approximation of the solution set

$S \leftarrow \emptyset$;
$\mathit{Queue.enqueue}(D)$; # queue for the solution set
while Queue is not empty **do**
  $B \leftarrow \mathit{Queue.dequeue}()$;
  if B possibly contains a solution **then**
    if the width of B in the i-th dimension for some i is no smaller than ϵ **then**
      Split B equally in the i-th dimension into B_1, B_2;
      $\mathit{Queue.enqueue}(B_1)$;
      $\mathit{Queue.enqueue}(B_2)$;
    else
      $S \leftarrow S \cup \{B\}$; # $\mathrm{W}(B) < \epsilon$
    end if
  end if
end while
return S;

Algorithm 14 Main procedure of the efficient approach

Input: a CSP $\langle \{\vec{x}_0[1], \ldots, \vec{x}_0[d], t\}, D, \mathit{Cons}\rangle$
Output: an interval enclosure of the solution set

$D_c \leftarrow D$;
repeat
  for all $z = \vec{x}_0[1], \ldots, \vec{x}_0[d], t$ **do**
    Compute a value $\texttt{lo}$ such that $\texttt{lo} \leq \nu(z)$ for any solution ν w.r.t. D_c.
    Compute a value $\texttt{up}$ such that $\texttt{up} \geq \nu(z)$ for any solution ν w.r.t. D_c.
    if $\texttt{lo} \leq \texttt{up}$ **then**
      Update the interval of D_c in the dimension of z to $[\texttt{lo}, \texttt{up}]$;
    else
      Set D_c empty and break; # the constraints are unsatisfiable
    end if
  end for # refine the domain D_c
until no great refinement on D_c
return D_c;

domain D_c one dimension after another, and the result is used in turn to improve the future contraction work.

The contraction in each dimension is done by searching approximations for the exact lower and upper bounds, and they are guaranteed to be conservative. To do so, we may still use the branch-and-prune framework however at most one subdivision is selected in each branching. The method to compute a lower bound approximation is given by Algorithm 15, wherein in each iteration we narrow down the interval $[\alpha, \beta]$ by the following steps.

1. Split $[\alpha, \beta]$ at its midpoint $\gamma = \frac{\alpha+\beta}{2}$.
2. Update the solution domain to D' which is same as D except that the interval in the dimension of z is $[\alpha, \gamma]$.
3. Check whether there is a solution in D'. The job can be conservatively done by interval arithmetic, that is, we use interval arithmetic to prove the nonexistence of a solution. If no solution is there, then we discard the branch $[\alpha, \gamma]$ and update the interval $[\alpha, \beta]$ to $[\gamma, \beta]$ in D'. Otherwise, $[\alpha, \beta]$ is updated to $[\alpha, \gamma]$.

Since we only discard the interval containing no solution, the exact lower bound of the solution set must lie in $[\alpha, \beta]$ in each iteration. Then the resulting value α is a conservative approximation. Unlike the general branch-and-prune algorithm, we only need to handle at most $O(\log(\lceil \frac{\mathrm{W}(D)}{\epsilon} \rceil))$ many branches.

Algorithm 15 Lower bound approximation search

Input: a CSP $\langle \{\vec{x}_0[1], \ldots, \vec{x}_0[d], t\}, D, \mathit{Cons} \rangle$, a threshold $\epsilon > 0$, a variable $z \in \{\vec{x}_0[1], \ldots, \vec{x}_0[d], t\}$

Output: a conservative approximation for the lower bound of the solution set in the dimension of z

Set α as the lower bound of D in the dimension of z;
Set β as the upper bound of D in the dimension of z;
$D' \leftarrow D$;
while $\beta - \alpha \geq \epsilon$ **do**
    $\gamma \leftarrow \frac{\alpha+\beta}{2}$;
    Update the interval of D' in the dimension of z to $[\alpha, \gamma]$;
    if D' possibly contains a solution **then**
        $\beta \leftarrow \gamma$;
    else
        $\alpha \leftarrow \gamma$;
    end if
end while
return α;

Lemma 4.3.4. *Algorithm 15 returns a conservative approximation of the lower bound of the solution set in the dimension of z.*

A conservative approximation for the upper bound in a dimension can be computed in a similar way except that we only consider the upper half in every iteration. We present the method by Algorithm 16. Hence, the total number of branches handled in our efficient approach is only $O(2 \cdot \eta \cdot (d+1) \cdot \log(\lceil \frac{\mathrm{W}(D)}{\epsilon} \rceil))$ wherein η is the iteration number of the main loop in Algorithm 14.

Algorithm 16 Upper bound approximation search

Input: a CSP $\langle \{\vec{x}_0[1], \ldots, \vec{x}_0[d], t\}, D, \mathit{Cons} \rangle$, a threshold $\epsilon > 0$, a variable $z \in \{\vec{x}_0[1], \ldots, \vec{x}_0[d], t\}$

Output: a conservative approximation for the lower bound of the solution set in the dimension of z

```
Set α as the lower bound of D in the dimension of z;
Set β as the upper bound of D in the dimension of z;
D' ← D;
while β − α ≥ ε do
   γ ← (α+β)/2;
   Update the interval of D' in the dimension of z to [γ, β];
   if D' possibly contains a solution then
      α ← γ;
   else
      β ← γ;
   end if
end while
return β;
```

Lemma 4.3.5. *Algorithm 16 returns a conservative approximation of the upper bound of the solution set in the dimension of z.*

Theorem 4.3.6. *Algorithm 14 computes a valid contraction of D.*

Example 4.3.7. *We revisit the Rössler attractor given in Example 3.4.3. This time we consider the intersection of the TM flowpipes in the time horizon $[0, 6]$ with two guard sets. The first one is given by*

$$G_1 = \{(x, y, z) \in \mathbb{R}^3 \mid z \geq 8\}$$

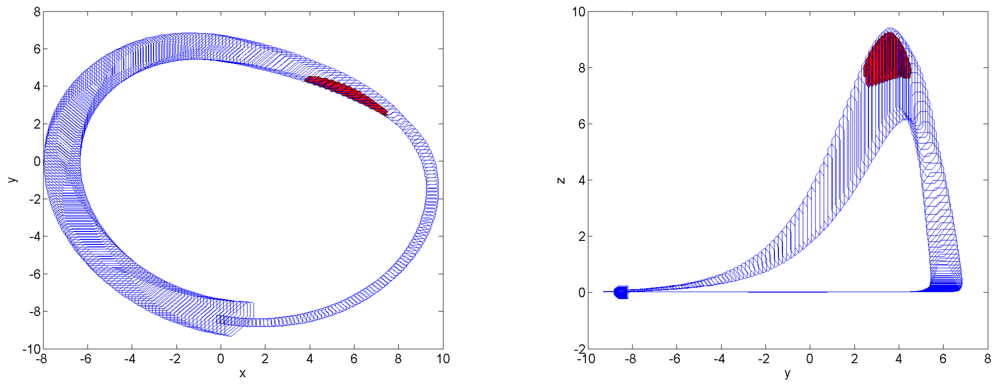
In Figure 4.5, we illustrate the flowpipe over-approximations as well as their contractions according to G_1. The other guard set is

$$G_2 = \{(x, y, z) \in \mathbb{R}^3 \mid -x + y - z \geq 6\}$$

We present the result in Figure 4.6. We also give an example of using domain contraction to deal with invariants. We associate the Rössler attractor example with the following invariant set

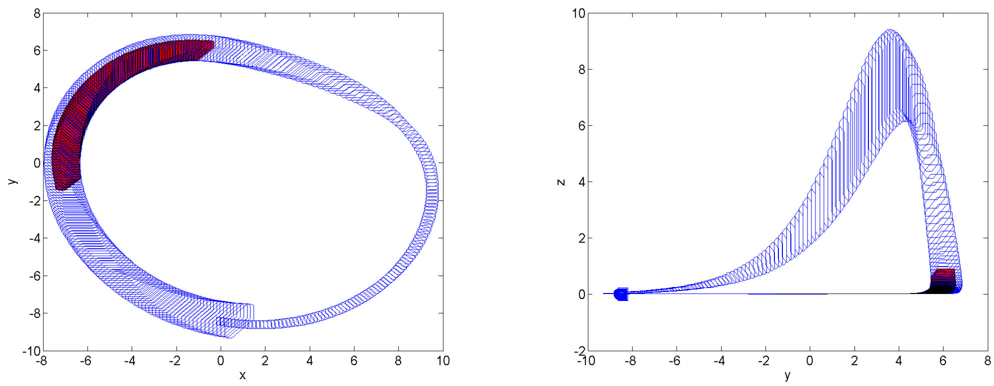
$$I = \{(x, y, z) \in \mathbb{R}^3 \mid x \in [-7, 9.5] \wedge y \in [-9, 6] \wedge z \in [0, 8]\}$$

In Figure 4.7, we present the invariant constrained flowpipe over-approximations as well as the unconstrained ones.



(a) Projections in the x-y plane (b) Projections in the y-z plane

Figure 4.5: Intersections computed by domain contraction with the guard G_1



(a) Projections in the x-y plane (b) Projections in the y-z plane

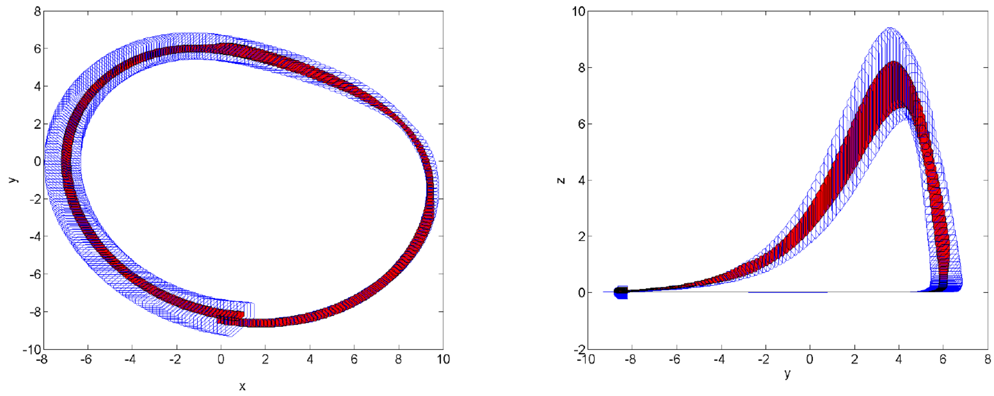
Figure 4.6: Intersections computed by domain contraction with the guard G_2

Remainder contraction. The algorithms for domain contraction may also be applied to reducing the TM remainder I, when the width of it is not small. Remainder contraction can be carried out together with domain contraction, we only need to introduce new variables to the CSP to include the remainder interval.

4.3.2 Range over-approximation

Other than reducing the domain of a TM flowpipe, an over-approximation of the flowpipe/guard intersection may also be derived by first computing an enclosure of the TM flowpipe range and then intersecting it with the guard, as illustrated by Figure 4.8. We call such a method *range over-approximation.* To do so, we need to over-approximate the range of a TM flowpipe $(p(\vec{x}_0, t), I)$ with $\vec{x}_0 \in X_0$ and $t \in [0, \delta]$, i.e., the set

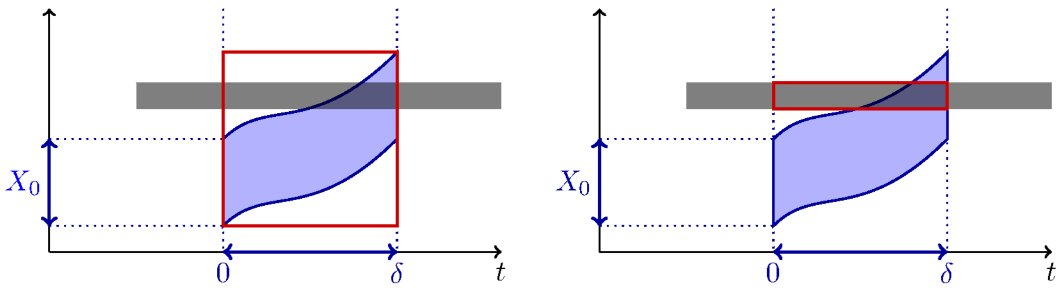
$$\mathtt{Rng}((p, I)) \;=\; \{\vec{x} \mid \vec{x} = p(\vec{x}_0, t) + \vec{y} \wedge \vec{x}_0 \in X_0 \wedge t \in [0, \delta] \wedge \vec{y} \in I\}$$



(a) Projections in the x-y plane

(b) Projections in the y-z plane

Figure 4.7: Invariant constrained and unconstrained flowpipe over-approximations.



(a) Over-approximate the flowpipe by a box

(b) Compute the intersection of the box and the guard

Figure 4.8: Example of range over-approximation

As we pointed out in Section 2.4, such an over-approximation can be computed as a polytope, a zonotope or an ellipsoid. We give some methods as below to compute an over-approximation for an n-dimensional TM flowpipe $\mathcal{F} = (p(\vec{x}_0, t), I)$ with $\vec{x}_0 \in X_0$ and $t \in [0, \delta]$.

Polytopic over-approximation. The range of $\mathcal{F}$ can be over-approximated by a $\mathcal{H}$-polytope P (a polytope in its $\mathcal{H}$-representation) via computing the support function values w.r.t. a template $\vec{l}_1, \ldots, \vec{l}_r \in \mathbb{R}^n$. That is, we solve the optimization problem

$$b_i = \sup\{\vec{l}_i^{\,T} \cdot \vec{x}\} \quad \text{subject to} \quad \vec{x} = p(\vec{x}_0, t) + \vec{y} \wedge \vec{x}_0 \in X_0 \wedge t \in [0, \delta] \wedge \vec{y} \in I$$

for $1 \leq i \leq r$, and the polytope P is computed as

$$P: \ (\vec{l}_1^{\,T}, \ldots, \vec{l}_r^{\,T}) \cdot \vec{x} \ \leq \ (b_1, \ldots, b_r)$$

Although those optimization problems might not be able to be solved efficiently, we may use the techniques of SDP relaxation, linear relaxation to compute a value that is larger than b_i for each $1 \leq i \leq r$, and the resulting polytope is still an over-approximation of the TM flowpipe.

Zonotopic over-approximation. The flowpipe $\mathcal{F}$ can be enclosed by a $\mathcal{G}$-zonotope $\mathcal{Z}$, i.e., $\mathcal{Z}$ is in a $\mathcal{G}$-representation. A simple but effective way to do so is to perform a conservative linearization on the TM,

$$(p', I') \;=\; (p - p_N \;,\; I + \mathtt{Int}(p_N)) \quad \text{wherein } p_N \text{ denotes the non-linear part of } p$$

and equivalently translating the TM (p', I') into a $\mathcal{G}$-zonotope.

Besides, a zonotopic over-approximation may also be obtained based on a template $\langle \vec{c}, \{\vec{l}_1, \ldots, \vec{l}_r\}\rangle$. The task can be done by finding r positive real values $a_1, \ldots, a_r$ such that the set

$$\mathcal{Z} = \{\vec{c} + \sum_{i=1}^{r} \lambda_i \cdot a_i \cdot \vec{l}_i \mid \lambda_1, \ldots, \lambda_r \in [0,1]\}$$

contains the TM flowpipe. The exact containment is hard to verify when the TM is of order > 1, but we may use interval arithmetic to efficiently find a more conservative over-approximation. As a feasible approach, we may initially choose $a_1, \ldots, a_r$ very close to zero and successively raise their values by small amounts until the zonotope contains the TM.

Definition 4.3.8 (Template of a zonotopic over-approximation)**.** *A* template *of a zonotope $\mathcal{Z}$ is a tuple $\langle \vec{c}, L\rangle$ wherein $\vec{c}$ is the center of $\mathcal{Z}$ and L is a set of nonzero vectors which determine the directions of the generators. If $\mathcal{Z}$ is n-dimensional, there should be at least n vectors in L linearly independent.*

Ellipsoidal over-approximation. An over-approximation may also be an ellipsoid $\mathcal{E} : (\vec{c}, Q)$ whose position and orientation can be given by a template. More precisely, the center $\vec{c}$ as well as the eigenvectors of Q can be defined by a template. Then the problem becomes to find proper eigenvalues for Q such that the set

$$\mathcal{E} = \{\vec{x} \in \mathbb{R}^n \mid (\vec{x} - \vec{c})^T \cdot Q \cdot (\vec{x} - \vec{c}) \leq 1\}$$

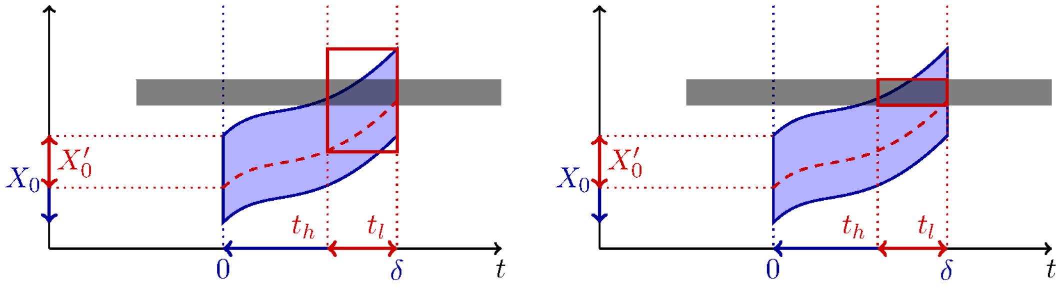
is a superset of the TM flowpipe. Since the eigenvalues of Q are the reciprocals of the squares of the semi-axes, we may initially choose large positive numbers, and then successively reduce them by small amounts until the ellipsoid defines an over-approximation.

Definition 4.3.9 (Template of an ellipsoidal over-approximation)**.** *A* template *of an n-dimensional ellipsoid $\mathcal{E}$ is a tuple $\langle \vec{c}, L\rangle$ wherein $\vec{c}$ is the center of $\mathcal{E}$ and L is a set of n linearly independent vectors which define the directions of the principal axes of $\mathcal{E}$.*

Support function over-approximation. Support functions, as a more general convex representation class, may also be used as range over-approximations. The TM flowpipe can be over-approximated by its convex hull whose support function $\rho_{\mathrm{Conv}(\mathcal{F})} : \mathbb{R}^n \to \mathbb{R}$ can be represented by the following optimization problem

$$\sup\{\vec{l}^{\,T} \cdot \vec{x}\} \quad \text{subject to} \quad \vec{x} = p(\vec{x}_0, t) + \vec{y} \wedge \vec{x}_0 \in X_0 \wedge t \in [0, \delta] \wedge \vec{y} \in I$$

Then the support function value $\rho_{\mathrm{Conv}(\mathcal{F})}(\vec{l})$ for some $\vec{l} \in \mathbb{R}^n$ can be obtained by solving the above problem. As we mentioned before, the solution is often hard to compute, however we may use interval arithmetic to derive a value which is larger than the exact one.



(a) Domain contraction along with range over-approximation

(b) Intersection of the range over-approximation and the guard

Figure 4.9: Combination of domain contraction and range over-approximation

We consider the following classes of the guard set. (i) When the guard is defined by only linear polynomial inequalities, then it is a polyhedron. We may apply the existing algorithms to derive a superset of the range over-approximation/guard intersection [BT00, GL08, Le 09, ASB10]. (ii) If the guard set is defined by non-linear inequalities, we may use ICP to contract the range over-approximation according to the guard. Since a range over-approximation has a simpler representation than the TM flowpipe, the task is easier than computing a domain contraction in general.

At last, we need to represent the intersection by a TM. To do so, we may first over-approximate the intersection by a zonotope and then translate it into an order 1 TM. An algorithm will be given in the next section.

The advantage of range over-approximation over domain contraction is that the final intersection over-approximation could be made entirely lie in the guard set based on a properly selected template.

Combine the use of domain contraction and range over-approximation. The techniques of domain contraction and range over-approximation can be used in a combination. We may first perform a domain contraction and then compute an over-approximation for the TM range over the contracted domain. If the domain is considerably contracted in the first step, we could cut off the overestimation that lies in the guard set but can not be excluded by only using the range over-approximation method. The main idea is illustrated in Figure 4.9.

4.4 Intersection aggregation

When a flowpipe/guard intersection is computed, we compute its image under the reset mapping of the jump, and the result is the initial set in the next mode. When there are several flowpipe over-approximations intersects the guard, then we have to compute an initial set in the next mode for each of them. If we do so, the number of the initial sets after several jumps may become prohibitively great, and it causes a huge computational burden. To avoid it, we consider to cluster those intersection pieces by one set and then transform it into a TM. Then, in the next mode, we only have one initial set. Hence, we use the following steps to over-approximate the reachable set under a jump.

(1) Use domain contraction as well as range over-approximation to compute TMs for flowpipe/guard intersections.

(2) Compute a TM over-approximation $\mathcal{T}$ for the union of those TMs.

(3) Compute the image of $\mathcal{T}$ under the reset mapping by TM arithmetic. The result is the initial set of the next mode.

Since TMs are not closed under union, it is hard to find the best TM for the union set. In the following content, we present several methods which try to obtain an accurate TM over-approximation. The main idea of them is to first derive a parallelotopic over-approximation and then equivalently translate it into a TM.

4.4.1 Aggregation by an oriented rectangular hull

The over-approximation is computed as a hyper-rectangle whose orientation is determined by means of *Principal Component Analysis (PCA)* [Jol02], and then translate the hyper-rectangle into a TM.

The technique of PCA is used to find uncorrelated variables which are called *principal components* to represent a set of data which are originally described by a larger number of variables. Here, we try to obtain an *orientation* which consists of n orthogonal vectors $\vec{l}_1, \ldots, \vec{l}_n \in \mathbb{R}^n$ for a finite set of samples (points, vectors) $\vec{s}_1, \ldots, \vec{s}_m$ in the space $\mathbb{R}^n$, such that an over-approximate hyper-rectangle Ω of the sample set can be computed as the following $\mathcal{H}$-polytope

$$\Omega : \ (\vec{l}_1^{\,T}, \ldots, \vec{l}_n^{\,T}, -\vec{l}_1^{\,T}, \ldots, -\vec{l}_n^{\,T}) \cdot \vec{x} \ \leq \ (a_1, \ldots, a_n, b_1, \ldots, b_n)$$

wherein

$$\begin{array}{rcll} a_i & = & \sup\{\vec{l}_i^{\,T} \cdot \vec{x}\} & \text{subject to} \quad \vec{x} \in \{\vec{s}_1, \ldots, \vec{s}_m\} \\ b_i & = & \sup\{-\vec{l}_i^{\,T} \cdot \vec{x}\} & \text{subject to} \quad \vec{x} \in \{\vec{s}_1, \ldots, \vec{s}_m\} \end{array}$$

for $1 \leq i \leq n$. By using PCA, the hyper-rectangle Ω is usually with a good accuracy. We introduce the basic steps to compute the orientation.

Step 1: Compute the sampling matrix. The samples $\vec{s}_1, \ldots, \vec{s}_m$ are organized as an $n \times m$ matrix

$$M_S \ = \ \begin{pmatrix} \vec{s}_1[1] & \vec{s}_2[1] & \cdots & \vec{s}_m[1] \\ \vec{s}_1[2] & \vec{s}_2[2] & \cdots & \vec{s}_m[2] \\ \vdots & \vdots & \vdots & \vdots \\ \vec{s}_1[n] & \vec{s}_2[n] & \cdots & \vec{s}_m[n] \end{pmatrix}$$

Then the *sampling matrix* $\overline{M}_S$ is computed by subtracting $\vec{s}$ which is the arithmetic mean of $\vec{s}_1, \ldots, \vec{s}_m$ from every column of M_S. That is, we shift those samples such that the arithmetic mean of them becomes zero.

Step 2: Compute the covariance matrix. The *covariance matrix* M_{Cov} of $\overline{M}_S$ is computed as

$$M_{Cov} \ = \ \frac{1}{m-1} \cdot \overline{M}_S \cdot \overline{M}_S^T$$

which is a symmetric matrix. The entry in the i-th row and j-th column of M_{Cov} is the *covariance* of the i-th and j-th variables. Intuitively, it tells how well the change in the two dimensions are correlated.

Step 3: Singular Value Decomposition of the covariance matrix. By computing the *Singular Value Decomposition (SVD)* of M_{Cov}, i.e.,

$$M_{Cov} \;=\; U \cdot \Sigma \cdot V^T$$

we obtain a matrix U whose columns provide an orientation.

Example 4.4.1. *We give an example of using PCA to compute a rectangular over-approximation for the sample set*

$$S = \{(1,3),(2.5,2.5),(1.6,2),(-1,-1.3),(-3.5,-4),(0,0.5),(-1.2,-0.3),(1,-0.3)\}$$

The sampling matrix as well as the covariance matrix are computed as

$$M_S \;=\; \begin{pmatrix} 0.95 & 2.45 & 1.55 & -1.05 & -3.55 & -0.05 & -1.25 & 0.95 \\ 2.7375 & 2.2375 & 1.7375 & -1.5625 & -4.2625 & 0.2375 & -0.5625 & -0.5625 \end{pmatrix}$$

and

$$M_{Cov} \;=\; \begin{pmatrix} 3.640000000000000 & 3.957857142857143 \\ 3.957857142857143 & 5.259821428571429 \end{pmatrix}$$

By computing the SVD of M_{Cov}, we obtain the orientation

$$\vec{l}_1 = (-0.632266098585395,\; -0.774751302405877)$$
$$\vec{l}_2 = (-0.774751302405877,\; 0.632266098585395)$$

The resulting rectangular over-approximation is shown in Figure 4.10. Obviously, it is much more accurate than an interval over-approximation, i.e., a box which is axis-aligned.

We give our method for intersection aggregation using PCA.

1. Compute a set of samples from the intersection over-approximations. The number of those samples is not necessarily exponential in the dimension number n but should be sufficient to reflect the distribution of the intersections.
2. Use PCA to find a proper orientation $\vec{l}_1, \ldots, \vec{l}_n \in \mathbb{R}^n$.
3. Compute a $\mathcal{H}$-polytope according to the template $\vec{l}_1, \ldots, \vec{l}_n, -\vec{l}_1, \ldots, -\vec{l}_n$ such that it contains all of the intersection over-approximations.

A key task here is the sample selection. Different from the relate work [SK03, Alt10, CÁ11], we consider to compute a smaller number of sample points in the following situations.

Intersection over-approximations are given by TMs. Given that an intersection over-approximation is of the form $(p(\vec{x}_0, t), I)$ wherein $\vec{x}_0 \in X_0'$, $t \in [t_h, t_l]$ such that X_0' is an interval. We may simply compute the point $\vec{s} = p(\vec{c}, \eta) + \vec{y}$ wherein $\vec{c} = \mathrm{Mid}(X_0')$, $\eta = \frac{t_h + t_l}{2}$ and $\vec{y} = \mathrm{Mid}(I)$ as the sample of the TM. Then we only need N samples for N intersections. Such a method may work fine if the over-approximation sets are of similar sizes. Otherwise we may select more samples near the surface of the TM. For example, we may first collect the facet centers of the box $X_0' \times [t_h, t_l]$ and then compute the samples as their images under the polynomial mapping p.

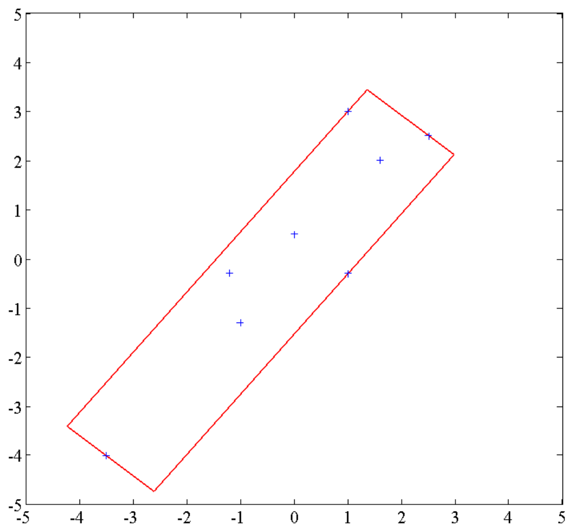


Figure 4.10: Over-approximating a sample set by a rectangular hull

Intersection over-approximations are given by convex geometric objects. For those geometric representations such as zonotopes, ellipsoids and polytopes, the samples can be generated in the following way. For an intersection over-approximation Ω, we select a set of samples $\vec{s}_1, \ldots, \vec{s}_m$ on the surface of Ω based on a given set of vectors $\vec{v}_1, \ldots, \vec{v}_m \in \mathbb{R}^n$ which are called *sampling directions*. That is, for each $1 \leq i \leq m$, we find a point $\vec{s}_i$ by linear programming such that

$$\vec{v}_i^T \cdot \vec{s}_i \;=\; \sup\{\vec{v}_i^T \cdot \vec{x} \mid \vec{x} \in \Omega\}$$

As a simpler way, we may also compute only one sample for each intersection, the point can be an approximation of its geometric center.

Example 4.4.2. *We consider a simple hybrid automaton which consists of three variables, two modes and one jump. The continuous dynamics in the first mode ℓ_1 is given by*

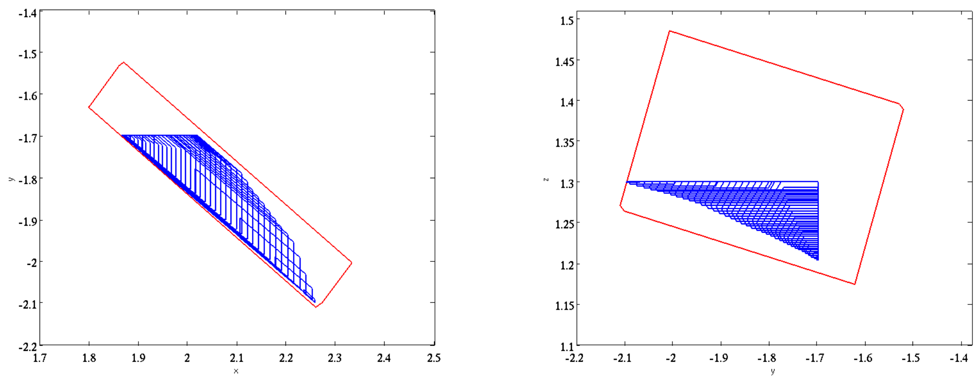
$$\begin{cases} \dot{x} &=& -9 \cdot (x-2) - 7 \cdot (y+2) + (z-1) + 0.2 \cdot (x-2) \cdot (y+2) \\ && +0.1 \cdot (y+2) \cdot (z-1) + 0.1 \cdot (x-2) \cdot (z-1) + 0.5 \cdot (z-1)^2 \\ \dot{y} &=& 6 \cdot (x-2) + 4 \cdot (y+2) + (z-1) \\ \dot{z} &=& 3 \cdot (x-2) + 2 \cdot (y+2) - 2.5 \cdot (z-1) \end{cases}$$

and the one in the second mode ℓ_2 is

$$\begin{cases} \dot{x} &=& 2.2x + 3.6y + 3.9z \\ \dot{y} &=& 3x + 2.4y + 3.4z - 0.01x^2 \\ \dot{z} &=& -5x - 5.4y - 6.7z \end{cases}$$

The only jump leads the system to transit from ℓ_1 to ℓ_2. The jump guard is defined by the box

$$\mathcal{G} \;=\; \{(x,y,z) \mid x \in [1.7, 2.3] \wedge y \in [-2.3, -1.7] \wedge z \in [0.7, 1.3]\}$$



(a) Projection in the x-y plane (b) Projection in the y-z plane

Figure 4.11: Rectangular aggregation for the flowpipe/guard intersections

and the reset mapping is identity. We consider the initial set $\langle \ell_1, X_0 \rangle$ *wherein*

$$X_0 \;=\; \{(x, y, z) \mid x \in [3, 3.2] \wedge y \in [-3, -2.8] \wedge z \in [1.3, 1.5]\}$$

and the time horizon $[0, 1]$*. We over-approximate the flowpipe/guard intersections by boxes, and by selecting the points according to the sampling directions*

$$(1, 0, 0),\ (0, 1, 0),\ (0, 0, 1),\ (-1, 0, 0),\ (0, -1, 0),\ (0, 0, -1)$$

we obtain the following orientation.

$$\begin{array}{rcl} \vec{l}_1 & = & (-0.688973, 0.712329, -0.133803) \\ \vec{l}_2 & = & (-0.259278, -0.414624, -0.872274) \\ \vec{l}_3 & = & (0.676824, 0.566281, -0.470356) \end{array}$$

The total time cost is nearly 7 *seconds. Figure 4.11 shows the rectangular aggregation.*

Unfortunately, the time cost of the PCA method can hardly be reduced. Firstly, the number of the samples should be enough to reflect the distribution of the points in the intersections. Secondly, we need to compute the SVD of an $n \times n$ matrix. On the other hand, an ill-oriented rectangular aggregation could also be generated by PCA when the intersections do not show a clear orientation. For example, the flowpipe/guard intersections in the above example have an apparent orientation in the subspace of x-y but not in the subspaces of y-z and x-z, then the aggregation is too conservative in some directions.

Translate a $\mathcal{H}$-represented hyper-rectangle into a TM. When a rectangular aggregation is computed, we show that its equivalent translation to an order 1 TM is not hard. Given a hyper-rectangle in $\mathbb{R}^n$,

$$\Omega :\ (\vec{l}_1^{\,T}, \ldots, \vec{l}_n^{\,T}, -\vec{l}_1^{\,T}, \ldots, -\vec{l}_n^{\,T}) \cdot \vec{x} \;\leq\; (a_1, \ldots, a_n, b_1, \ldots, b_n)$$

wherein $\vec{l}_1, \ldots, \vec{l}_n$ are orthogonal to each other. We want to compute a TM $(p(\vec{\sigma}), [0, 0]^n)$ wherein $\vec{\sigma} \in [-1, 1]^n$ and its range is Ω. The reason to use such a TM form is that every

TM over an interval domain can be equivalently transformed to it by a change of basis. Firstly, we assume that Ω is full-dimensional. The translation is no more than computing a $\mathcal{G}$-representation for Ω, i.e., finding the point $\vec{c}$ as well as n orthogonal vectors $\vec{g}_1, \ldots, \vec{g}_n$ such that

$$\Omega \;=\; \{\vec{c} + \sum_{i=1}^{n} \sigma_i \cdot \vec{g}_i \mid \vec{\sigma} \in [-1,1]^n\}$$

The above representation may be reformulated as

$$\Omega \;=\; \{\vec{c} + M_g \cdot \vec{\sigma} \mid \vec{\sigma} \in [-1,1]^n\}$$

wherein $M_g = (\vec{g}_1^{\,T}, \ldots, \vec{g}_n^{\,T})^T$.

The point $\vec{c}$ is the geometric center of Ω and it can be obtained by solving the following system of linear equations

$$\begin{cases} \vec{l}_1^{\,T} \cdot \vec{x} & = & \frac{a_1 - b_1}{2} \\ \vec{l}_2^{\,T} \cdot \vec{x} & = & \frac{a_2 - b_2}{2} \\ & \cdots & \\ \vec{l}_n^{\,T} \cdot \vec{x} & = & \frac{a_n - b_n}{2} \end{cases}$$

Then a $\mathcal{H}$-representation of the recentered hyper-rectangle $\Omega' = \{M_g \cdot \vec{\sigma} \mid \vec{\sigma} \in [-1,1]^n\}$ can be computed as

$$(\vec{l}_1^{\,T}, \ldots, \vec{l}_n^{\,T}, -\vec{l}_1^{\,T}, \ldots, -\vec{l}_n^{\,T}) \cdot \vec{x} \;\le\; (\lambda_1, \ldots, \lambda_n, \lambda_1, \ldots, \lambda_n) \tag{4.3}$$

wherein

$$(\lambda_1, \ldots, \lambda_n, \lambda_1, \ldots, \lambda_n) = (a_1, \ldots, a_n, b_1, \ldots, b_n) - (\vec{l}_1^{\,T}, \ldots, \vec{l}_n^{\,T}, -\vec{l}_1^{\,T}, \ldots, -\vec{l}_n^{\,T}) \cdot \vec{c}$$

If we reformulate the $\mathcal{H}$-representation (4.3) by

$$(\lambda_1^{-1} \cdot \vec{l}_1^{\,T}, \ldots, \lambda_n^{-1} \cdot \vec{l}_n^{\,T}, -\lambda_1^{-1} \cdot \vec{l}_1^{\,T}, \ldots, -\lambda_n^{-1} \cdot \vec{l}_n^{\,T}) \cdot \vec{x} \le (1, \ldots, 1)$$

the matrix $(\lambda_1^{-1} \cdot \vec{l}_1^{\,T}, \ldots, \lambda_n^{-1} \cdot \vec{l}_n^{\,T})$ defines a linear mapping from Ω' to $[-1,1]^n$, i.e.,

$$(\lambda_1^{-1} \cdot \vec{l}_1^{\,T}, \ldots, \lambda_n^{-1} \cdot \vec{l}_n^{\,T}) \cdot \Omega' \;=\; [-1,1]^n$$

Henceforth, the matrix M_g can be derived as the inverse of $(\lambda_1^{-1} \cdot \vec{l}_1^{\,T}, \ldots, \lambda_n^{-1} \cdot \vec{l}_n^{\,T})$.

When Ω is not full-dimensional, there will be k different indices $1 \le i_1, \ldots, i_k \le n$ such that $\lambda_{i_1} = \cdots = \lambda_{i_k} = 0$. Then we only need $n-k$ generators to represent Ω and they are given by the columns of M_g with the indices not in $\{i_1, \ldots, i_k\}$. We may still use the above method to compute the center $\vec{c}$ as well as the matrix M_g except that the values of $\lambda_{i_1}^{-1}, \ldots, \lambda_{i_k}^{-1}$ are set by 1 during the computation.

The translation method can be extended to deal with parallelotopes. In that case, the vectors $\vec{l}_1, \ldots, \vec{l}_n$ are not necessarily orthogonal but should be linearly independent.

Example 4.4.3. *Given a parallelotope*

$$P: \begin{pmatrix} 1 & 2 & 0 \\ 0 & 1 & 1 \\ 2 & 1 & 1 \\ -1 & -2 & 0 \\ 0 & -1 & -1 \\ -2 & -1 & -1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} \;\le\; \begin{pmatrix} 5 \\ 8 \\ 6 \\ -3 \\ -7 \\ -2 \end{pmatrix}$$

The center can be computed as $\vec{c} = (-1.75, 2.875, 4.625)$. *Then we obtain the values* $\lambda_1 = 1$, $\lambda_2 = 0.5$,$\lambda_3 = 2$, *and the matrix* M_g *is*

$$M_g = \begin{pmatrix} 0 & -0.25 & 1 \\ 0.5 & 0.125 & -0.5 \\ -0.5 & 0.375 & 0.5 \end{pmatrix}$$

Therefore, the TM representation for P *is obtained as*

$$\left(\begin{pmatrix} -0.25\sigma_2 + \sigma_3 \\ 0.5\sigma_1 + 0.125\sigma_2 - 0.5\sigma_3 \\ -0.5\sigma_1 + 0.375\sigma_2 + 0.5\sigma_3 \end{pmatrix}, \begin{pmatrix} [0,0] \\ [0,0] \\ [0,0] \end{pmatrix} \right) \quad \textit{wherein} \quad \sigma_1, \sigma_2, \sigma_3 \in [-1, 1]$$

On the other hand, for a parallelotope which is not full-dimensional,

$$Q : \begin{pmatrix} 1 & 2 & 0 \\ 0 & 1 & 1 \\ 2 & 1 & 1 \\ -1 & -2 & 0 \\ 0 & -1 & -1 \\ -2 & -1 & -1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} \leq \begin{pmatrix} 5 \\ 8 \\ 6 \\ -5 \\ -7 \\ -2 \end{pmatrix}$$

we may just take $\lambda_1^{-1} = 1$ *and obtain the same matrix* M_g *as the above one, nevertheless the first column of it will not be taken into account. Therefore the resulting TM is*

$$\left(\begin{pmatrix} -0.25\sigma_2 + \sigma_3 \\ 0.125\sigma_2 - 0.5\sigma_3 \\ 0.375\sigma_2 + 0.5\sigma_3 \end{pmatrix}, \begin{pmatrix} [0,0] \\ [0,0] \\ [0,0] \end{pmatrix} \right) \quad \textit{wherein} \quad \sigma_2, \sigma_3 \in [-1, 1]$$

In other words, the variable σ_1 *is a dummy variable.*

4.4.2 Aggregation by a parallelotope

As we pointed out previously, PCA might not work well when the intersections do not show a clear orientation. In that case, we consider to compute a general parallelotopic aggregation by determining the template based on a set of critical directions. The reason to take parallelotopes is twofold. Firstly, their translations to order 1 TMs are relatively easy. Secondly, we only need to solve $2n$ optimization problems to compute the $\mathcal{H}$-representation of a parallelotope.

In most safety verification tasks, we want the overestimation limited along some nonzero vectors. That is, we do not want the overestimation to grow towards the unsafe set. We call those vectors *critical directions*. For example, the boxes in Figure 4.12 are the set of intersection over-approximations to aggregate, and we want to limit the resulting overestimation in the direction given by the vector $\vec{l}$. An effective way to do so is taking $\vec{l}$ as well as $-\vec{l}$ in the template of the aggregation parallelotope. Since the lengths of critical directions are not concerned, we just assume that all critical directions are given by normalized vectors.

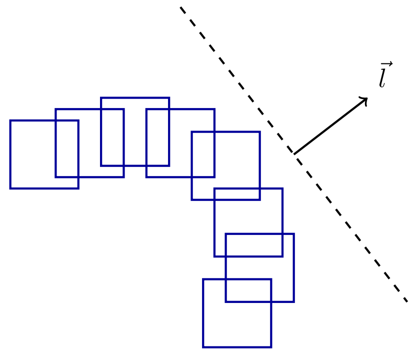


Figure 4.12: A critical direction for an intersection aggregation

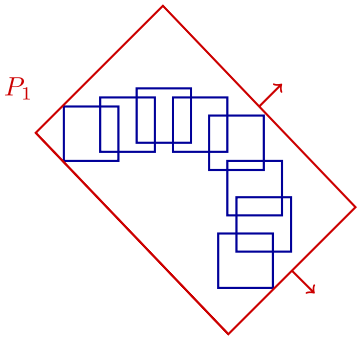
How to choose the critical directions? The critical directions could be obtained from several sources. For example, in order to make the aggregation set stay as much as possible in the guard and the mode invariant, we may pick $\frac{\vec{l}}{|\vec{l}|}$ as a critical direction for each linear constraint $\vec{l}^{\,T} \cdot \vec{x} \leq b$ in the definition of them. The critical directions related to an unsafe set can be selected in a similar way. Besides, we may also compute some directions according to the invariant set of the next mode. Some ideas of over-approximation refinement by using redundant constraints [ADMT10] could be adapted to finding critical directions.

Template determination. We present our algorithm to determine a parallelotopic template in the space $\mathbb{R}^n$. We seek to select n linearly independent vectors from a provided set $\mathcal{L}$ of critical directions. Assume that the intersection over-approximations form the set $S \subseteq \mathbb{R}^n$, the vectors are chosen based on the following criteria.

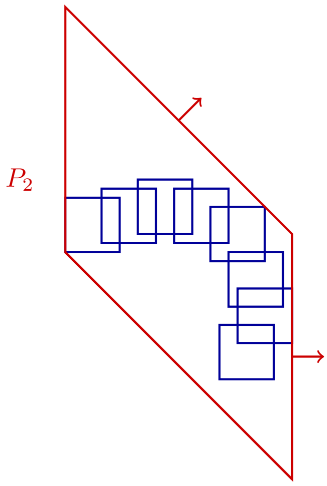
(a) The selected vectors should be as orthogonal as possible. More precisely, we want to minimize the absolute value of the cosine of the angle between any two selected vectors $\vec{l}, \vec{l}'$, that is $|\cos(\theta)| = |\frac{\vec{l}^{\,T} \cdot \vec{l}'}{\|\vec{l}\| \cdot \|\vec{l}'\|}|$. The purpose of it is to prevent the resulting parallelotope from having too sharp corners which often lead to large overestimation. We illustrate two examples in Figure 4.13. We use θ_1 and θ_2 to denote the angles between the selected vectors for P_1 and P_2 respectively, and have that $|\cos(\theta_1)| = 0$ and $|\cos(\theta_2)| = \frac{\sqrt{2}}{2}$.

(b) For a selected vector $\vec{l}$, we want to make the value $|\rho_S(\vec{l}) + \rho_S(-\vec{l})|$ as small as possible. More intuitively, the value gives the width of S measured along the direction $\vec{l}$ or $-\vec{l}$. The purpose of it is to wrap the set S in its thinnest orientation. As an example, consider the set S shown in Figure 4.13, we will choose the vector $(\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2})$ when the candidates are given by $(\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2})$,$(\frac{\sqrt{2}}{2}, -\frac{\sqrt{2}}{2})$,$(1, 0)$ and $(0, 1)$. Although such a width can not be computed exactly in some situations, we may compute an approximation of it.

Unfortunately, the conditions (a) and (b) can not be fulfilled at the same time in general, our algorithm iteratively choose n vectors from the critical direction set. In the i-th iteration, for $1 \leq i \leq n$, assume that $\vec{l}_1, \ldots, \vec{l}_{i-1}$ are the already selected vectors, and the remaining critical directions are given by the set $\mathcal{L}$, we do the following steps.



(a) Selected vectors: $(\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}), (\frac{\sqrt{2}}{2}, -\frac{\sqrt{2}}{2})$



(b) Selected vectors: $(1,0), (\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2})$

Figure 4.13: Parallelotopic aggregations computed based on different sets of vectors

1. For each vector $\vec{v} \in \mathcal{L}$, we compute the value
$$\mu(\vec{v}) \;=\; (1 - |\vec{l_1}^T \cdot \vec{v}|) \cdot (1 - |\vec{l_2}^T \cdot \vec{v}|) \cdot \cdots \cdot (1 - |\vec{l}_{i-1}^T \cdot \vec{v}|)$$
and then obtain the maximum value $\mu_{\max} = \{\mu(\vec{v}) \,|\, \vec{v} \in \mathcal{L}\}$. We collect the vectors $\vec{v} \in \mathcal{L}$ such that $\mu(\vec{v}) = \mu_{\max}$ by the set $\mathcal{L}_c$. Note that the value of $\mu(\vec{v})$ reflects the orthogonality of the vectors $\vec{l_1}, \ldots, \vec{l}_{i-1}$ and $\vec{v}$. If all of them are orthogonal to each other, then we have that $\mu(\vec{v}) = 1$. When any two vectors of them as well as their inverses form a sharp angle, then the value of $\mu(\vec{v})$ will not be high.

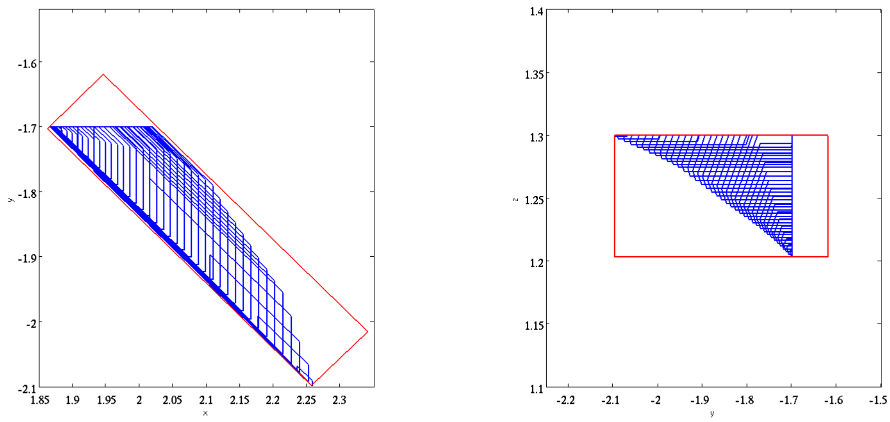
2. The i-th vector $\vec{l_i}$ is selected as the one $\vec{v} \in \mathcal{L}_c$ such that $|\rho_S(\vec{v}) + \rho_S(-\vec{v})|$ is minimum among all vectors in $\mathcal{L}_c$.

The above iterations can be carried out very efficiently. We may compute all of the *dot products* of two critical directions in advance, and keep them in a hash table. We may also associate priorities with the critical directions. For example, assume that the linear constraints $\vec{l_1}^T \cdot \vec{x} \le b_1, \ldots, \vec{l_m}^T \cdot \vec{x} \le b_m$ are defining the guard or the mode invariant. For two critical directions $\vec{l_i}, \vec{l_j} \in \{\vec{l_1}, \ldots, \vec{l_m}\}$, we consider the priority of $\vec{l_i}$ higher than that of $\vec{l_j}$ if the boundary $\vec{l_i}^T \cdot \vec{x} = b_i$ intersects the TM flowpipes but $\vec{l_j}^T \cdot \vec{x} = b_j$ does not. Such priorities may also be specified by users.

Example 4.4.4. *We consider the set of flowpipe/guard intersections in Example 4.4.2. This time we try to compute a parallelotopic aggregation based on the following critical direction set.*

$$\{\ (1,0,0), (0,1,0), (0,0,1), (\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}, 0), (\frac{\sqrt{2}}{2}, -\frac{\sqrt{2}}{2}, 0),$$
$$(0, \frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}), (0, \frac{\sqrt{2}}{2}, -\frac{\sqrt{2}}{2}), (\frac{\sqrt{2}}{2}, 0, \frac{\sqrt{2}}{2}), (\frac{\sqrt{2}}{2}, 0, -\frac{\sqrt{2}}{2})\ \}$$

The selected vectors are (in the order of their selection) $(\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}, 0)$, $(0,0,1)$ *and* $(\frac{\sqrt{2}}{2}, -\frac{\sqrt{2}}{2}, 0)$. *A visualization of the aggregation set as well as the intersection over-approximations is given by Figure 4.14. The total time cost is less than* 1 *second.*



(a) Projection in the x-y plane

(b) Projection in the y-z plane

Figure 4.14: Parallelotopic aggregation computed by selecting critical directions

Hence, a simple way to improve the overall accuracy could be providing sufficiently many critical directions. Besides the definition of guards, invariants and unsafe sets, those vectors may also be derived from the study of the flow direction in the current mode.

4.4.3 More representations for aggregation

Since an n-dimensional parallelotope only has $2n$ facets, it can only tightly enclose the over-approximated set in at most $2n$ directions. A better choice could be general zonotopes, since we are able to specify more generators. We may use the method described in Section 4.3.2.

Another feasible way to improve the accuracy could be using polytopes. Although we are not able to exactly represent a polytope by a TM over an interval domain, it can be viewed as an intersection of finitely many parallelotopes. Then we may keep the TM translations of those parallelotopes as a tuple, and apply flowpipe construction to each component individually. Such a flowpipe over-approximation intersects a set when every component has a non-empty intersection with the set. Although the method requires to keep a list of TMs for a flowpipe, the number of them is bounded by the number of halfspaces defining the polytope.

Besides, it is also possible to consider more general aggregation sets, for example, TMs. In order to achieve an acceptable efficiency, we need to provide a *template* for the TM aggregation. For example, we may bound the order of the TM by some integer $k > 0$, specify the remainder as $[0, 0]^n$ and the domain as the unit box $[-1, 1]^d$ for some $n, d > 0$. Then there could be $n \cdot \binom{d+k}{k}$ coefficients which may be expressed in terms of a set of unknown parameters $\lambda_1, \ldots, \lambda_q$ such that q could be much smaller than the number of the coefficients. Hence, we may formulate the TM aggregation problem as finding $\lambda_1 \in I_1, \ldots, \lambda_q \in I_q$ with user-specified sets $I_1, \ldots, I_q \subseteq \mathbb{R}$, such that the intersection over-approximations are contained in the range of the TM $(p(\vec{x}_0, \lambda_1, \ldots, \lambda_q), [0, 0]^n)$ with $\vec{x}_0 \in [-1, 1]^d$.

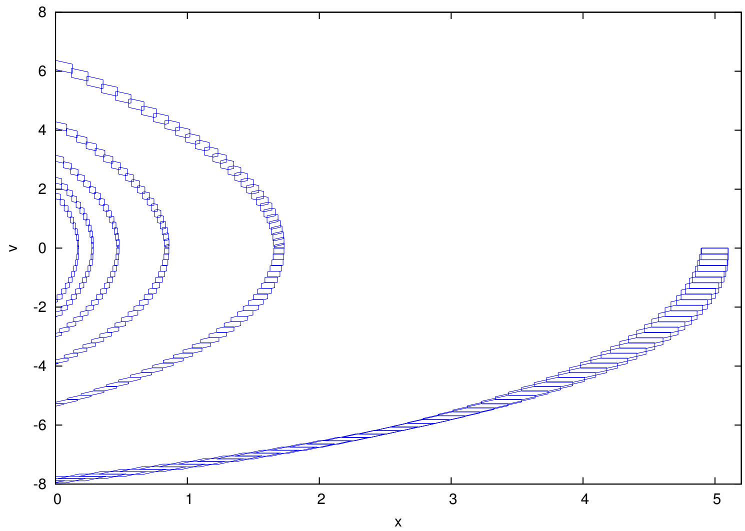


Figure 4.15: Flowpipe over-approximations of the bouncing ball with air friction

4.5 Applications

In this section, we apply our techniques for computing TM flowpipes to several hybrid system case studies. We start with some simple examples.

4.5.1 Simple examples

Bouncing ball with air friction. We revisit the model of a bouncing ball with air friction which is shown in Example 4.1.2. We compute the TM flowpipes for the model within the maximum jump depth 10, and the octagon enclosures of the TM flowpipes are given in Figure 4.15.

2-dimensional stable system. We consider an artificial system which is described by the hybrid automaton shown in Figure 4.16. The continuous dynamics in the location ℓ_1 is unstable, but whole system is stable. We want to study the behavior of the system in the time horizon $[0, 20]$ from the initial set $\langle \ell_1, X_0 \rangle$ wherein $X_0 = \{(x, y) \,|\, x \in [0.9, 1.1] \wedge y \in [-1.1, -0.9]\}$. After performing the flowpipe construction, we obtain the TM flowpipes shown in Figure 4.17.

3-dimensional stable system. We revisit the hybrid automaton given in Example 4.4.2. This time we perform the TM flowpipe construction from a larger initial set $\langle \ell_1, X_0' \rangle$ wherein

$$X_0' \;=\; \{(x, y, z) \mid x \in [3, 3.5] \wedge y \in [-3, -2.5] \wedge z \in [1, 1.5]\}$$

for the time horizon $[0, 10]$. The resulting TM flowpipes are presented in Figure 4.18.

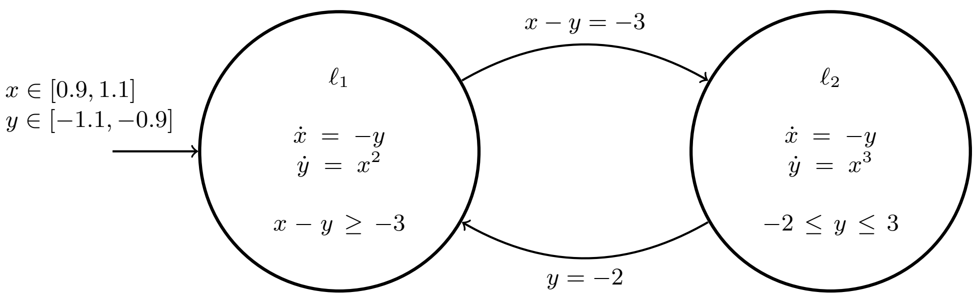


Figure 4.16: Hybrid automaton of the 2-dimensional stable system

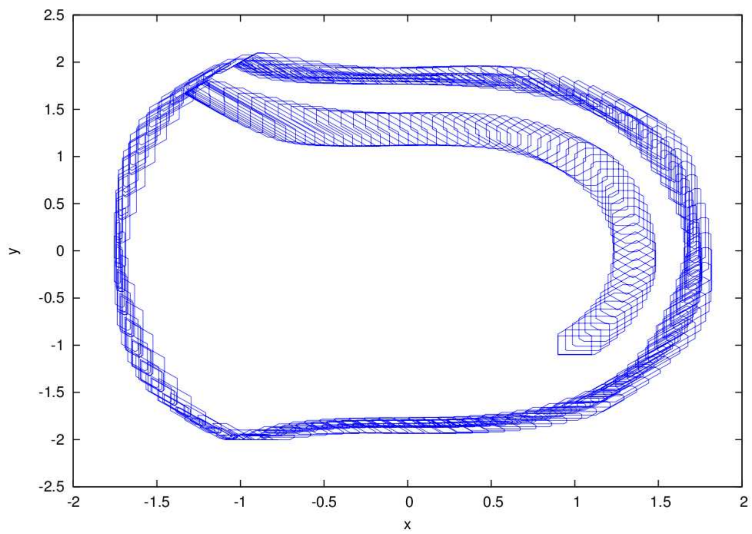


Figure 4.17: Flowpipe over-approximations of the 2-dimensional stable system

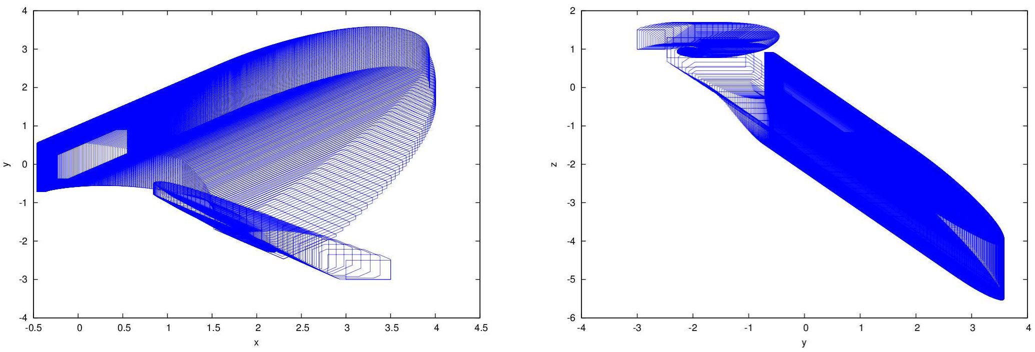
Non-holonomic integrator. We consider a simplified version of the hybrid control for *Brockett's non-holonomic integrator* [HM99, Lib03, GHT$^+$04]. It is a 3-dimensional system whose dynamics is given by

$$\begin{cases} \dot{x} &= u \\ \dot{y} &= v \\ \dot{z} &= x \cdot v - y \cdot u \end{cases}$$

wherein u, v are the control inputs

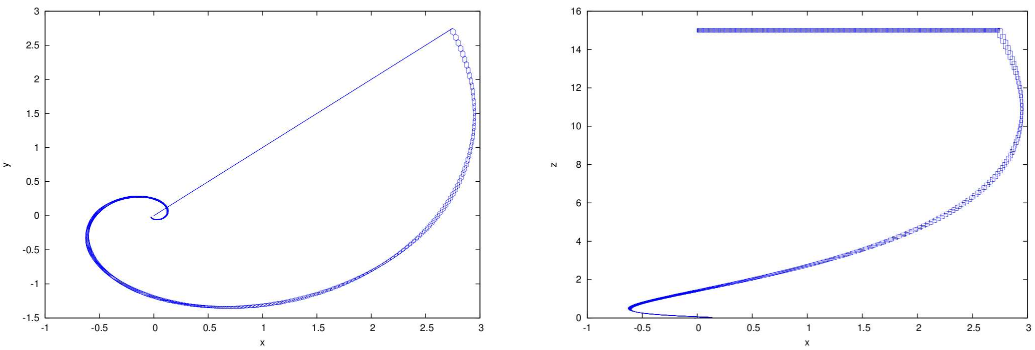
$$u = \begin{cases} 1, & x^2 + y^2 \leq |z| \\ -x + \frac{2 \cdot y \cdot z}{x^2+y^2}, & x^2 + y^2 > |z| \end{cases} \qquad v = \begin{cases} 1, & x^2 + y^2 \leq |z| \\ -y + \frac{2 \cdot x \cdot z}{x^2+y^2}, & x^2 + y^2 > |z| \end{cases}$$

which are designed to asymptotically stabilize the system. We start with the initial condition $x = 0$, $y = 0$ and $z \in [14.9, 15.1]$, and perform the flowpipe construction for the time horizon $[0, 7.5]$. The TM flowpipes are illustrated in Figure 4.19, where we can see that they are converging to the origin.



(a) Projection in the x-y plane (b) Projection in the y-z plane

Figure 4.18: Flowpipe over-approximations of the 3-dimensional stable system



(a) Projection in the x-y plane (b) Projection in the x-z plane

Figure 4.19: Flowpipe over-approximations of the non-holonomic integrator

4.5.2 Spiking neurons

We apply our reachability analysis techniques to two models of spiking neurons which are adapted from the ones presented in [Izh10].

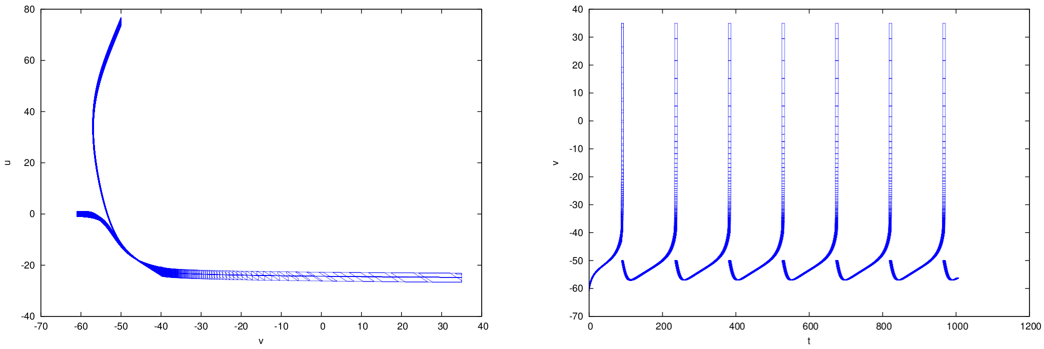
The general dynamics of spiking neurons is defined by

$$\begin{cases} C \cdot \dot{v} &= k \cdot (v - v_r) \cdot (v - v_t) - u + I \\ \dot{u} &= a \cdot (b \cdot (v - v_r) - u) \end{cases} \tag{4.4}$$

wherein the constant parameters are given by $C = 100$, $v_r = -60$, $v_t = -40$, $I = 70$, $a = 0.03$ and $b = -2$ for the first model. The value of k is 0.7 when $v \leq v_t$, otherwise it is 7. Whenever the value of v reaches 35, its value is reset to -50 and meanwhile u is updated to $u + 100$. Such a model can be formalized by a hybrid automaton consists of 2 modes and 2 variables.

We consider the initial condition $v(0) \in [-61, -59]$, $u(0) \in [-1, 1]$, and apply the TM flowpipe construction for the time horizon $[0, 1000]$. The result is presented in Figure 4.20.

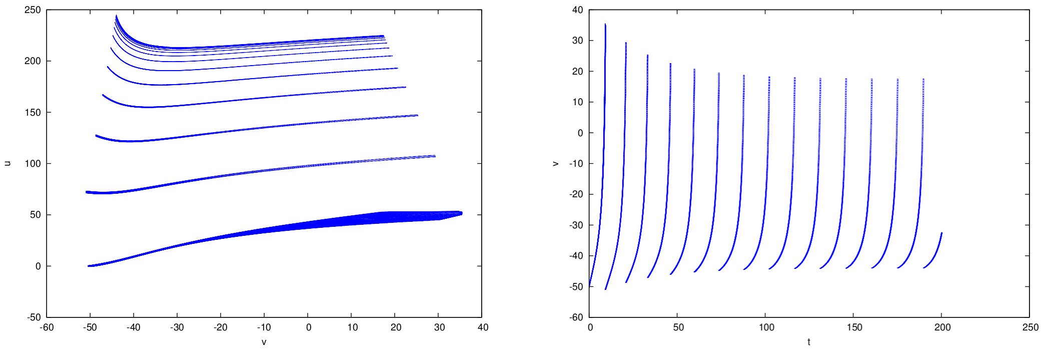
As another model, we take the constants $C = 100$, $k = 1$, $v_r = -56$, $v_t = -42$, $I = 300$, $a = 0.03$ and $b = 8$. The values of v, u are reset to $-53 + 0.04 \cdot u$ and $u + 20$ respectively when $v \geq 40 - 0.1 \cdot u$. This time we take the initial condition $v(0) \in [-50.5, -49.5]$,



(a) Projection in the v-u plane

(b) Projection in the t-v plane

Figure 4.20: Flowpipe over-approximations of the first spiking neuron model



(a) Projection in the v-u plane

(b) Projection in the t-v plane

Figure 4.21: Flowpipe over-approximations of the second spiking neuron model

$u(0) \in [-0.5, 0.5]$, and perform the TM flowpipe construction for the time horizon $[0, 200]$. The result is given in Figure 4.21.

4.5.3 Inverted pendulum

Stabilizing an inverted pendulum on a cart is a popular case study in controller synthesis [ÅF00]. Here we try to over-approximate the behavior of the model under one control strategy. As it is shown in Figure 4.22, the control input F is designed to regulate the pendulum to the upright position, i.e., $\theta = 0$ as well as $\dot{\theta} = 0$. The motion of the model can be described by

$$J \cdot \ddot{\theta} \;=\; m \cdot l \cdot g \cdot \sin(\theta) \;-\; m \cdot l \cdot \cos(\theta) \cdot F$$

wherein J is the moment of inertia with respect to the pivot point, m is the mass of the pendulum, l is the length of the rod, and g is the gravitational acceleration. For simplicity,

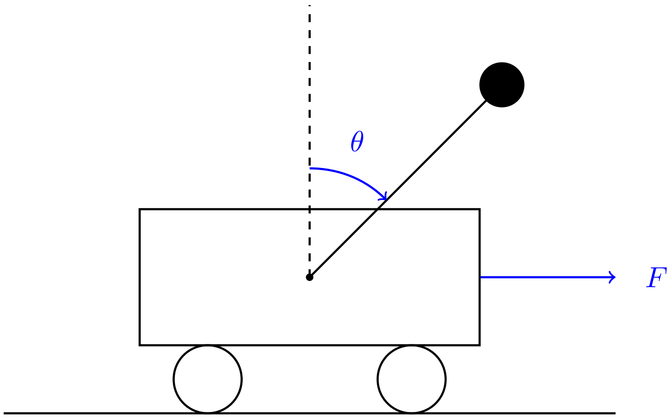
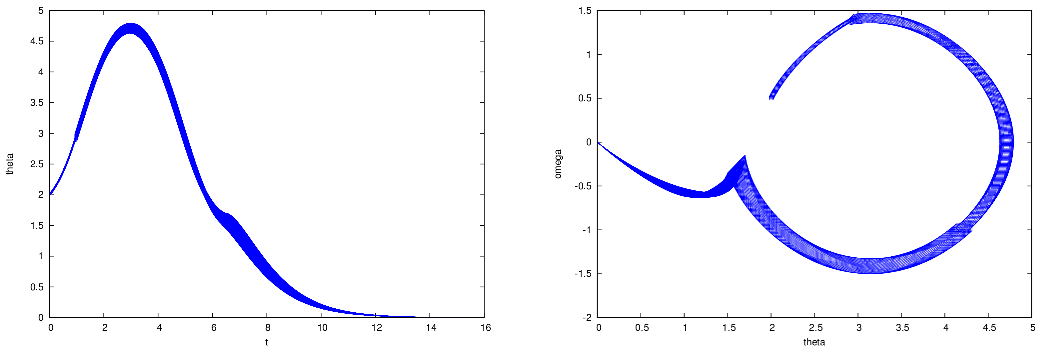


Figure 4.22: Inverted pendulum on a cart



(a) Projection in the θ-t plane

(b) Projection in the θ-ω plane

Figure 4.23: Flowpipe over-approximations of the inverted pendulum model

we set $J = 1$, $m = \frac{1}{g}$, $l = 1$, and denote $\frac{F}{g}$ by u. Then the following ODE is obtained

$$\begin{cases} \dot{\theta} &= \omega \\ \dot{\omega} &= \sin(\theta) - \cos(\theta)\cdot u \end{cases}$$

To stabilize the pendulum in the upright position, we apply the following switching strategy to u,

$$u = \begin{cases} \frac{2\cdot\omega+\theta+\sin(\theta)}{\cos(\theta)}, & E \in [-1,1], |\omega|+|\theta| \leq 1.85 \\ 0, & E \in [-1,1], |\omega|+|\theta| > 1.85 \\ \frac{\omega}{1+|\omega|}\cdot\cos(\theta), & E < -1 \\ \frac{-\omega}{1+|\omega|}\cdot\cos(\theta), & E > 1 \end{cases}$$

which is designed to stabilize the pendulum energy $E = \frac{1}{2}\cdot\omega + (\cos(\theta) - 1)$ at zero. Then the motion of the controlled pendulum can then be described by a hybrid automaton. We set the initial condition as $\theta \in [1.98, 2.02]$, $\omega \in [0.48, 0.52]$ and perform the TM flowpipe construction for the time horizon $[0, 15]$. The computed TM flowpipes are given in Figure 4.23.

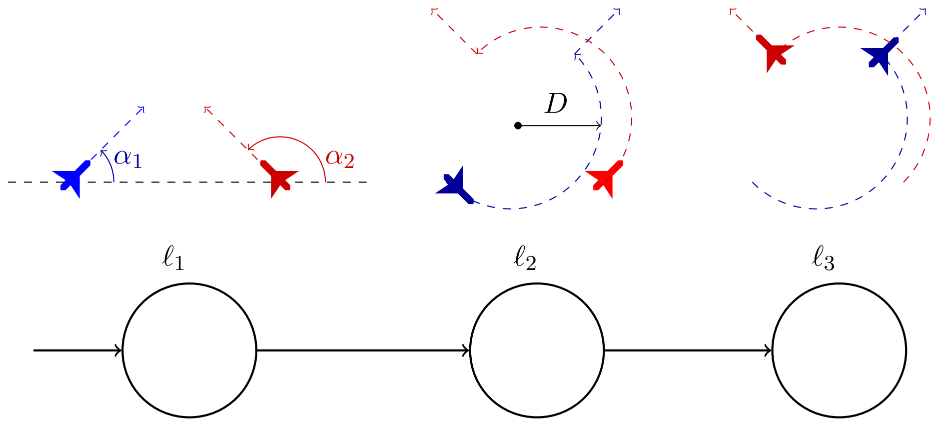


Figure 4.24: Hybrid automaton of the collision avoidance maneuver

4.5.4 Aircraft collision avoidance maneuver

We consider a collision avoidance maneuver of two aircraft at a fixed altitude, which is given in [MT00]. In the beginning, both of the aircraft are in straight flight with a relative heading. When they come too close, i.e., the distance of them is blow a specified value, the controller will make an instantaneous heading change of 90° on both of them, and then the two aircraft will complete a π-time semicircular arc flying. Afterwards, both aircraft make another 90° instantaneous heading change and resume their headings in the very beginning. Such a system can be modeled by a hybrid automaton consists of 3 modes, as it is given in Figure 4.24. The continuous variables are listed as below.

x_1, y_1 :	the coordinates of the first aircraft
α_1 :	the heading angle of the first aircraft
x_2, y_2 :	the coordinates of the second aircraft
α_2 :	the heading angle of the second aircraft
z :	timer for the semicircular arc flying

The continuous dynamics of the hybrid automaton is defined by

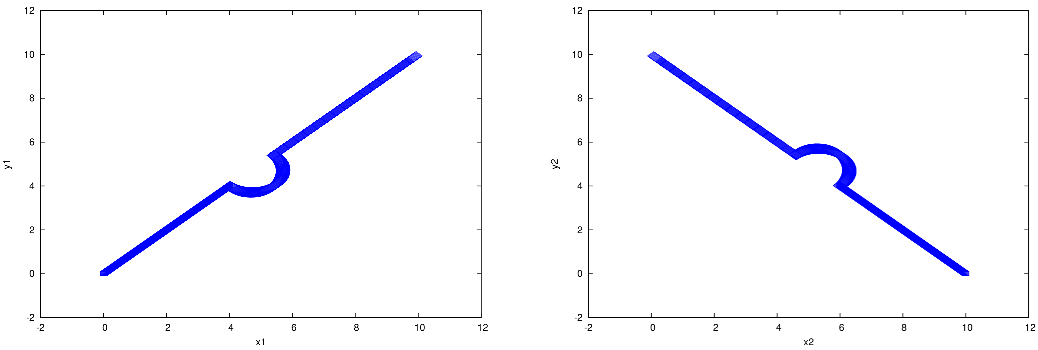
$$\dot{x}_1 = v_1 \cdot \cos(\alpha_1),\ \dot{y}_1 = v_1 \cdot \sin(\alpha_1),\ \dot{x}_2 = v_2 \cdot \cos(\alpha_2),\ \dot{y}_2 = v_2 \cdot \sin(\alpha_2),$$
$$\dot{\alpha}_1 = \begin{cases} 1, & \text{in } \ell_2 \\ 0, & \text{otherwise} \end{cases},\ \dot{\alpha}_2 = \begin{cases} 1, & \text{in } \ell_2 \\ 0, & \text{otherwise} \end{cases},\ \dot{z} = \begin{cases} 1, & \text{in } \ell_2 \\ 0, & \text{otherwise} \end{cases}$$

The angles α_1, α_2 as well as the timer z only change in ℓ_2. The jump from ℓ_1 to ℓ_2 is executed when the distance between the aircraft is below D^2. It gives that the invariant of ℓ_1 is defined by the constraint $(x_1 - x_2)^2 + (y_1 - y_2)^2 \geq D^2$, and the jump guard is defined by $(x_1 - x_2)^2 + (y_1 - y_2)^2 = D^2$. When the semicircular arc flying is complete, i.e., $z = \pi$, the jump from ℓ_2 to ℓ_3 is executed and the heading angles are updated by

$$\alpha_1' := \alpha_1 - \frac{\pi}{2}, \qquad \alpha_2' := \alpha_2 - \frac{\pi}{2}$$

We set the constant parameters as $v_1 = 1$, $v_2 = 1$, $D = 2$, and consider the initial condition

$$\begin{array}{llll} x_1 \in [-0.1, 0.1], & y_1 \in [-0.1, 0.1], & x_2 \in [9.9, 10.1], & y_2 \in [-0.1, 0.1], \\ \alpha_1 = \frac{1}{4}\pi, & \alpha_2 = \frac{3}{4}\pi, & z = 0 & \end{array}$$



(a) Projection in the x_1-y_1 plane (b) Projection in the x_2-y_2 plane

Figure 4.25: Flowpipe over-approximations of the collision avoidance maneuver

The flowpipe construction result over the time interval $[0, 15]$ is presented in Figure 4.25.

4.5.5 Glycemic Control in Diabetic Patients

We study the "minimal model" defined in [BIBC79, BPC81, BFA85] for the dynamics of glucose and insulin interaction in the blood system. It is described by the following ODE.

$$\begin{cases} \dot{G} &= -p_1 \cdot G - X \cdot (G + G_B) + P \\ \dot{X} &= -p_2 \cdot X + p_3 \cdot I \\ \dot{I} &= -n \cdot (I + I_B) + \frac{u}{V_I} \end{cases}$$

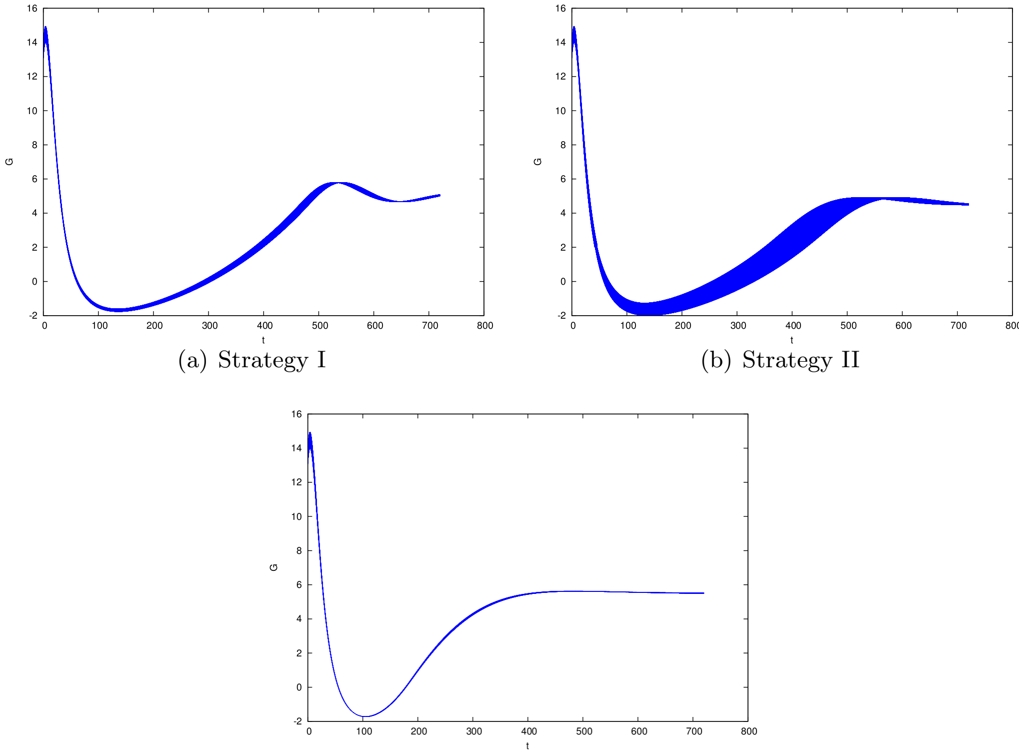
wherein p_1, p_2, p_3 are constant parameters whose typical values are $p_1 = 0$ min^{-1}, $p_2 = 0.025$ min^{-1} and $p_3 = 0.013$ min^{-2} U^{-1} L. The values of G_B and I_B are the basal values of plasma glucose concentration and free plasma insulin concentration respectively, and they are given by $G_B = 4.5$ mmol L^{-1}, $I_B = 0.015$ U L^{-1}. The constant V_I is the insulin distribution volume, and the value of n denotes the fractional disappearance rate of insulin. We take their values as $V_I = 12$ L and $n = \frac{5}{54}$ min^{-1}. The remaining parameters are variables, their meanings are given as below.

G : the difference of plasma glucose concentration
I : the free plasma insulin concentration
X : the insulin concentration in an interstitial chamber
P : the rate of infusion of exogeneous glucose
u : the rate of infusion of exogeneous insulin

We use the definition $P = 0.5 \cdot \exp(-0.05 \cdot t)$ given in [Fis91] for the rate of insulin infusion, and consider three different strategies for the insulin delivery rate u.

Strategy I is the glucose control described in [CKBC84]. The rate u is 0.5 U h^{-1} when $G < 4$ mmol L^{-1}, and it is 2.5 U h^{-1} when $G > 8$ mmol L^{-1}. If G is between 4 and 8 mmol L^{-1}, we use $u = (0.5 \cdot G - 1.5)$ U h^{-1}.

Strategy II is taken from [FKSC85], it is similar to the first strategy but considers the glucose rates 2 and 12 mmol L^{-1}. That is, we set $u = 0.5$ U h^{-1} when $G < 2$ mmol L^{-1},



(a) Strategy I

(b) Strategy II

(c) Strategy III

Figure 4.26: Flowpipe over-approximations of the glycemic control model

and $u = 2.5$ U h^{-1} when $G > 12$ mmol L^{-1}. For that G is between 2 and 12 mmol L^{-1}, u is defined by $(0.2 \cdot G + 0.1)$ U h^{-1}.

In order to better stabilize the glucose level, a better designed control strategy is presented in [FKSC85]. We call it *Strategy III*. The controller reads the value of G in the beginning of every 3 hours and do the following job. If $G \geq 6$ mmol L^{-1}, then we use the input rate $u = G \cdot (0.41 - 0.0094 \cdot G)$ U min^{-1}. Otherwise u is set to be the linear form $0.007533 \cdot (1 + 0.22 \cdot G)$ U min^{-1}.

The initial condition under our consideration is given by

$$G(0) \in [13, 14], \quad X(0) = 0, \quad I(0) = 0.5$$

Figure 4.26 illustrate the TM flowpipes computed in the time horizon $[0, 720]$ for the model with the three control strategies respectively.

4.5.6 Non-linear transmission line circuits

We study a non-linear resistor circuit which is presented in Figure 4.27. It is originally considered by Chen et al. [CW00], and then adapted to be interesting hybrid case studies [RW03, Gu11]. The circuit is composed of $(n + 1)$ non-linear resistors and the same number of capacitors. Each non-linear resistor consists of a diode and a unit resistor $(r = 1)$. For simplicity, we assume that all capacitors have unit capacitance $C = 1$. For

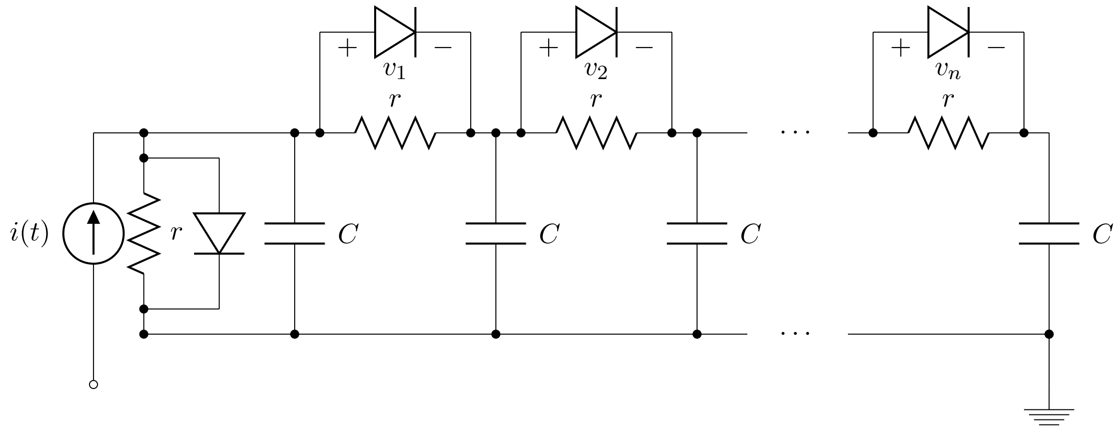


Figure 4.27: Transmission line circuit

each diode, the I-V characteristic is given by $I = \exp(\alpha \cdot V) - 1$. The current source $i(t)$ in the figure is the input, and v_1 is the single output of the circuit. Therefore, the whole circuit system can be described by the following ODE.

$$\begin{cases} \dot{v}_1 & = & -2 \cdot v_1 + v_2 + 2 - \exp(\alpha \cdot v_1) - \exp(\alpha \cdot (v_1 - v_2)) + i(t) \\ \dot{v}_2 & = & -2 \cdot v_2 + v_1 + v_3 + \exp(\alpha \cdot (v_1 - v_2)) - \exp(\alpha \cdot (v_2 - v_3)) \\ & \cdots & \\ \dot{v}_{n-1} & = & -2 \cdot v_{n-1} + v_{n-2} + v_n + \exp(\alpha \cdot (v_{n-2} - v_{n-1})) - \exp(\alpha \cdot (v_{n-1} - v_n)) \\ \dot{v}_n & = & -v_n + v_{n-1} - 1 + \exp(\alpha \cdot (v_{n-1} - v_n)) \end{cases}$$

In order to reduce the stiffness of the dynamics, we reduce the value of α from 40 to 5. Scalable continuous and hybrid benchmarks can be built based on various types of inputs. Here, we consider a discontinuous input $i(t)$ which is adapted from the one presented in [RW03].

$$i(t) = \begin{cases} 2, & t \leq 1 \\ 3 - t, & 1 < t \leq 2 \\ 1, & t > 2 \end{cases}$$

Then the whole system can be modeled by a hybrid automaton consists of 3 modes and n variables. We consider the initial value of v_i ranging in $[0, 0.02]$ for all $1 \leq i \leq n$. Figure 4.28 shows the flowpipe over-approximations computed by our techniques over the time horizon $[0, 3]$ for $n = 6$.

4.6 Summary

We introduce the use of TM flowpipes to over-approximate the reachable set for a non-linear hybrid automaton. Similar work can be found in [RN11] which uses Interval Taylor Series (ITS). By using TMs, we are able to avoid splitting any set in reachability analysis, and therefore can deal with some case studies with more than 5 variables. Although it requires to compute multivariate polynomials, under the help of proper simplification methods, our method can still efficiently produce an accurate result in most cases.

The applicability of a representation class could be investigated based on a large number of examples which should cover most well-known difficulties in hybrid system

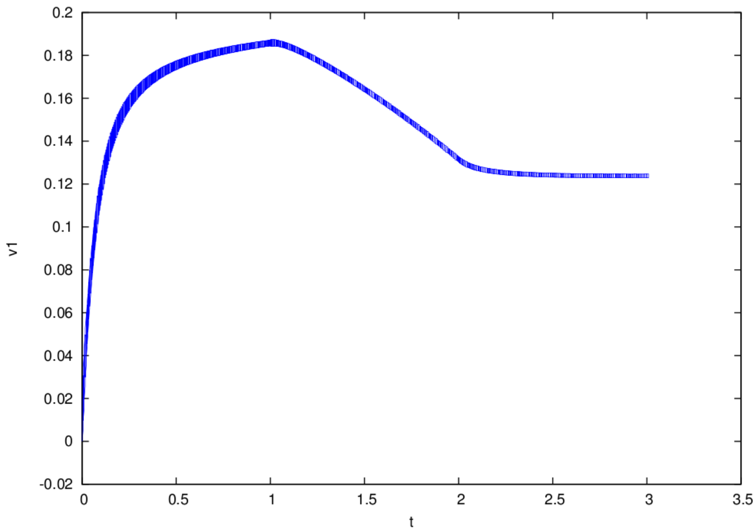


Figure 4.28: Flowpipe over-approximations of the line circuit model with $n = 6$

reachability analysis. For example, large initial sets, high nondeterminism on jumps and so on. Since hybrid systems do not have standard benchmarks, it is also significant to propose new case studies which are adapted from industrial applications. In this chapter, we present a considerable number of examples which are taken from biology, medicine, mechanics and electricity. It can be seen that our TM flowpipe construction method has good performance over all of them. Further investigations on the applicability of TM flowpipes will be given in the next chapter.

Chapter 5

The Tool Flow*

It is important to make the Taylor model (TM) techniques presented in the thesis accessible to other people. To do so, we implement most of the introduced techniques and heuristics in a tool named FLOW*. As we found few tools which can effectively deal with non-linear hybrid systems from different areas, it is also urgent for us to release the tool.

This chapter devotes a detailed description of the tool FLOW* which is a reachability analyzer for non-linear hybrid systems. It is implemented in `C++` based on some open source libraries such as the GNU MPFR Library and the GNU Scientific Library (GSL). Besides the techniques introduced previously, we also implement some algorithms to improve the efficiency of TM computation. The performance of FLOW* is shown via comparisons with VNODE-LP, dReach and SpaceEx. Moreover, we also provide a scalability evaluation based on the non-linear transmission line circuit benchmark.

5.1 Overview

Figure 5.1 provides a bird's view of the modules in FLOW* which consists of two main parts:

- *TM related modules* - the basic computational libraries of FLOW*. It includes a library of interval arithmetic, a library of TM arithmetic, and a library for computing intersections of TMs with other sets which are defined by systems of polynomial constraints.

- *Reachability related modules* - the high-level algorithms for computing flowpipe over-approximations under continuous dynamics. It also includes a parser for continuous and hybrid reachability problems as well as a parser for TM flowpipes.

The main functionality of the tool is to solve a hybrid reachability problem. However, it may also be used as a validated ODE solver, or to conservatively check the safety of given TMs. As a reachability problem solver, the tool accepts a file in which the following content should be specified.

- A *hybrid automaton* which is described by the language introduced in Section 5.3.

- An *initial set* which can be an interval or a TM in a mode of the hybrid automaton.

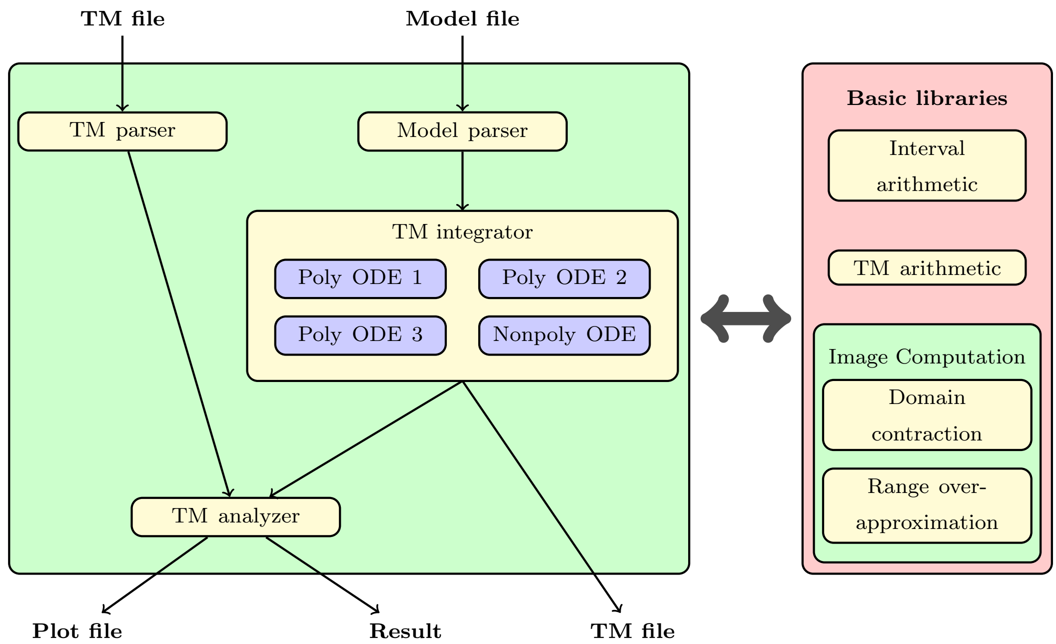


Figure 5.1: Structure of FLOW* v1.2.1

- An *unsafe set* which is defined by a system of polynomial constraints. If no unsafe set is given, the tool will not do a safety checking after the computation of TM flowpipes.

- A *reachability setting* which includes the time step-sizes, TM orders, remainder estimation, cutoff threshold, floating-point precision and etc.. They will be applied to the consequent reachability computation. The details will be addressed in the following sections.

After parsing the model file, the tool calls Algorithm 12, in the main loop of which the flowpipes in a location is generated by a TM integrator. As we stated in Section 3.3, a preconditioned Taylor expansion for a polynomial ODE can be generated by different schemes which are suitable to different situations. Here, FLOW* provides three options, i.e., Poly ODE 1-3 which are implemented according to Approach I-III respectively presented in Section 3.3. Non-polynomial ODEs are treated separately. In each time step, we first compute a TM for the vector field, and then compute a preconditioned Taylor expansion by Approach III. After the reachability computation is accomplished, the following results will be returned by FLOW*.

- *A TM file.* All computed TM flowpipes along with the state space specification as well as the plot setting will be stored in an output file. Such a file could be used by other tools for further analyzing the TMs. FLOW* can also be used to work on those TMs again while the unsafe set or plot setting is changed.

- *Result of safety checking.* If an unsafe set is specified in the model file, the tool will conservatively check the safety of the computed TM flowpipes. If no TM "intersects" the unsafe set, the tool returns `SAFE`, otherwise the result is `UNKNOWN`.

- *A plot file.* FLOW* supports computing an interval or octagon enclosure for a 2-dimensional projection of a TM flowpipe. The plot setting can be specified in the model file such. The tool will produce a GNUPLOT or MATLAB file for generating the figure.

To let FLOW* work on a model file `problem.model`, one may simply execute
`$./flowstar < problem.model`
After the tool terminates, a plot file as well as a TM file are generated and placed in the subdirectory named *outputs*. If the plot file is for GNUPLOT, then it ends with `.plt`, otherwise it is an m-file and is for MATLAB. If the tool is not able to complete the computation, it will still output the TMs which has already been computed. Such a failure is often caused by that the tool can not find a safe remainder in an integration task.

5.2 Basic computational libraries

In FLOW*, the library of interval arithmetic is implemented based on the GNU MPFR Library. The upper and lower bound of an interval is in type `mpfr_t`, and the precision can be specified by users. To take all round-off errors into account, we sometimes have to do one real operation for two times with different rounding modes. For example, in handling the interval multiplication

$$[a, b] \cdot [c, d] \;=\; [\min\{a \cdot c, a \cdot d, b \cdot c, b \cdot d\}\,,\, \max\{a \cdot c, a \cdot d, b \cdot c, b \cdot d\}]$$

we should round the results of all real multiplications for the low bound towards $-\infty$, and round those for the upper bound towards $+\infty$. In order to ensure the conservativeness of the results, we always treat real numbers as intervals. Then a polynomial in FLOW* is always with interval coefficients.

A TM then is represented by a polynomial along with an interval. The TM arithmetic is no more than a combination of polynomial and interval arithmetic. As we introduced previously, the representation of a TM can be simplified by moving the "small" terms in the polynomial part into the remainder interval. FLOW* accepts a user-specified value $\varepsilon \geq 0$ which is called a *cutoff threshold* such that for each TM multiplication result, a polynomial term contained in $[-\varepsilon, \varepsilon]$ is removed and an interval enclosure of it is added to the remainder. Besides, to measure and control the overestimation, we also regularly narrow the coefficients in a TM to degenerate intervals.

We consider two methods to compute an interval enclosure for a polynomial p over an interval domain D. In the first method, we compute a Horner form h of p by Algorithm 7 with the variables' declaration order, and then evaluate h by interval arithmetic. However, we should not do such operation too often, since the Horner form transformation usually takes quite a while when the variables are not few. As the second method, we may equivalently transform the polynomial p into q whose domain is a unit box $[-1, 1]^n$ for some positive integer n. Then the range of a monomial in q is $[-1, 1]$ if a variable in it has an odd degree, or $[0, 1]$ when any variable in it has an even degree. Then, to compute an interval for q, we may just scan the degrees of the variables, compute interval enclosures for the terms, and then sum up the results. Table 5.1 shows that the second method has

Benchmark	Var	Initial set	T	δ	k	ε	t_1 (s)	W_1	t_2 (s)	W_2
jet engine	2	$x \in [0.9, 1.1]$ $y \in [0.9, 1.1]$	10	0.02	4	10^{-12}	11.5	0.0364	**7.1**	**0.0340**
Brusselator	2	$x \in [0.8, 1]$ $y \in [0, 0.2]$	10	0.02	4	10^{-12}	13.1	0.0249	**8.9**	**0.0247**
Van der Pol	2	$x \in [1.25, 1.55]$ $y \in [2.25, 2.35]$	7	0.02	5	10^{-12}	14.1	0.6308	**8.7**	**0.6120**
Lorentz	3	$x \in [14.99, 15.01]$ $y \in [14.99, 15.01]$ $z \in [35.99, 36.01]$	2	0.01	5	10^{-12}	29.5	0.3822	**19.8**	**0.3751**
Rössler	3	$x \in [-0.2, 0.2]$ $y \in [-8.6, -8.2]$ $z \in [-0.2, 0.2]$	6	0.02	5	10^{-12}	19.5	1.8781	**12.4**	**1.8713**
coupled Van der Pol	4	$x_1 \in [0.9, 1.1]$ $y_1 \in [0.9, 1.1]$ $x_2 \in [0.9, 1.1]$ $y_2 \in [0.9, 1.1]$	5	0.02	5	10^{-10}	266	0.6135	**178**	**0.5776**
Lotka-Volterra	5	$x_i \in [0.9, 1]$, $1 \le i \le 5$	5	0.02	4	10^{-10}	349	0.07506	**246**	**0.07502**
biological model I	7	$x_i \in [0.99, 1.01]$, $1 \le i \le 7$	2	0.01	4	10^{-9}	110	0.1077	**72**	**0.1070**
biological model II	9	$x_i \in [0.99, 1.01]$, $1 \le i \le 9$	2	0.01	4	10^{-9}	305	2.0188	**223**	**1.9416**

Table 5.1: Comparison of the two interval evaluation methods. Legends: **Var**: number of variables, $\boldsymbol{T}$: $[0, T]$ is the time horizon, $\boldsymbol{\delta}$: time step-size, $\boldsymbol{k}$: TM order, $\boldsymbol{\varepsilon}$: cutoff threshold, $\boldsymbol{t_1}$: time cost of using the first method, $\boldsymbol{W_1}$: width of the interval enclosure computed using the first method for the solution at T, $\boldsymbol{t_2}$: time cost of using the second method, $\boldsymbol{W_2}$: width of the interval enclosure computed using the second method for the solution at T.

a better performance than the first one in TM integration. The benchmarks which are not introduced before are given as follows.

Example 5.2.1 (Van der Pol oscillator)**.** *A model of the Van der Pol oscillator is given by the following ODE.*

$$\begin{cases} \dot{x} &= y \\ \dot{y} &= y - x - x^2 y \end{cases}$$

The same model is also studied elsewhere [ASB08, Alt10].

Example 5.2.2 (Coupled Van der Pol oscillator)**.** *Two Van der Pol oscillators of the type introduced above can be coupled directly [RH80]. The composed model is described by the following* 4*-dimensional ODE.*

$$\begin{cases} \dot{x}_1 &= y_1 \\ \dot{y}_1 &= (1 - x_1^2) \cdot y_1 - x_1 + (x_2 - x_1) \\ \dot{x}_2 &= y_2 \\ \dot{y}_2 &= (1 - x_2^2) \cdot y_2 - x_2 + (x_1 - x_2) \end{cases}$$

Example 5.2.3 (Lotka-Volterra model of 5 variables)**.** *The* 5*-dimensional Lotka-Volterra model is an adaptation of the system presented in [WVS06].*

$$\begin{cases} \dot{x}_1 &= x_1 \cdot (1 - (x_1 + 0.85x_2 + 0.5x_5)) \\ \dot{x}_2 &= x_2 \cdot (1 - (x_2 + 0.85x_3 + 0.5x_1)) \\ \dot{x}_3 &= x_3 \cdot (1 - (x_3 + 0.85x_4 + 0.5x_2)) \\ \dot{x}_4 &= x_4 \cdot (1 - (x_4 + 0.85x_5 + 0.5x_3)) \\ \dot{x}_5 &= x_5 \cdot (1 - (x_5 + 0.85x_1 + 0.5x_4)) \end{cases}$$

Example 5.2.4 (Biological models). *Many systems in biology can be characterized by ODEs. We propose two high-dimensional models which are adapted from the biological systems presented in [KHK+ 05]. The first one is of* 7 *variables and is denoted by* biological model I *in Table 5.1. Its modeling ODE is given as below.*

$$\begin{cases} \dot{x}_1 &= -0.4x_1 + 5x_3x_4 \\ \dot{x}_2 &= 0.4x_1 - x_2 \\ \dot{x}_3 &= x2 - 5x_3x_4 \\ \dot{x}_4 &= 5x_5x_6 - 5x_3x_4 \\ \dot{x}_5 &= -5x_5x_6 + 5x_3x_4 \\ \dot{x}_6 &= 0.5x_7 - 5x_5x_6 \\ \dot{x}_7 &= -0.5x_7 + 5x_5x_6 \end{cases}$$

The second one is even more complex. It consists of 9 *variables. We denote it* biological model II*, the ODE is given as follows.*

$$\begin{cases} \dot{x}_1 &= 3x_3 - x_1x_6 \\ \dot{x}_2 &= x_4 - x_2x_6 \\ \dot{x}_3 &= x_1x_6 - 3x_3 \\ \dot{x}_4 &= x_2x_6 - x_4 \\ \dot{x}_5 &= 3x_3 + 5x_1 - x_5 \\ \dot{x}_6 &= 5x_5 + 3x_3 + x_4 - x_6 \cdot (x_1 + x_2 + 2x_8 + 1) \\ \dot{x}_7 &= 5x_4 + x_2 - 0.5x_7 \\ \dot{x}_8 &= 5x_7 - 2x_6x_8 + x_9 - 0.2x_8 \\ \dot{x}_9 &= 2x_6x_8 - x_9 \end{cases}$$

It is not difficult to find the reason. In a TM integration task, if the initial set is represented by a TM over a unit box domain, then all variables except the time variable t in the computed TM flowpipes are ranging in $[-1, 1]$. Therefore, when we treat t as an coefficient, there is no need to do the transformation in the second method.

5.3 Input language

To explicitly describe a reachability problem, the following content should be given.

5.3.1 Definition of the system

If the system is pure continuous, it is just defined by an ODE. In FLOW*, an ODE should be specified in the following form.

```
<integration_scheme>
{
 <ODE>
}
```

such that an n-dimensional ODE is given by n equations of the form `x' =` φ wherein `x` is a state variable, and φ is an expression defined by the following syntax

$$\varphi \ \texttt{::=}\ \varphi \texttt{ + } \varphi \ \texttt{|}\ \varphi \texttt{ - } \varphi \ \texttt{|}\ \varphi \texttt{ * } \varphi \ \texttt{|}\ \texttt{-}\varphi \ \texttt{|}\ (\varphi) \ \texttt{|}\ \varphi\texttt{\^{}n} \ \texttt{|}\ \texttt{x} \ \texttt{|}\ \texttt{r}$$
$$\texttt{|}\ \sin(\varphi) \ \texttt{|}\ \cos(\varphi) \ \texttt{|}\ \exp(\varphi) \ \texttt{|}\ \varphi \texttt{ / } \varphi \ \texttt{|}\ \texttt{sqrt}(\varphi)$$

wherein `n` is a non-negative integer and `r` is a rational number. The integration scheme can be any of the approaches for computing preconditioned Taylor expansions described before as well as the one for non-polynomial ODEs. As an example, the following specification tells the tool to integrate the ODE $\dot{x} = 1 + x^2$ using the scheme `Poly ODE 2`.

```
poly ode 2
{
 x' = 1 + x^2
}
```

Besides, it is also possible to associate interval time-varying uncertainties with the vector field. For example, `x' = 1 + x^2 + [0.001,0.1]`.

When the system is a hybrid automaton, the definition is of the following form,

```
modes
{
 <mode(s)>
}
jumps
{
 <jump(s)>
}
```

such that a mode is defined in the form of

```
<mode_name>
{
 <integration_scheme>
 {
  <ODE>    # continuous dynamics in the mode
 }
 inv       # definition of the mode invariant
 {
  <polynomial_inequalities>
 }
}
```

wherein a polynomial inequality should be over the state variables. The definition of a jump should be given in the following form,

```
<start_mode_name> -> <end_mode_name>
guard      # definition of the jump guard
{
 <polynomial_inequalities>
}
reset      # definition of the reset mapping
{
 <expressions>
}
<aggregation_scheme>
```

wherein the reset mapping is polynomial but allowed to be associated with an interval uncertainty. For example, `x' := 1 - x + [-0.05,0.002]` defines that the value of `x` is updated to the value of `1 - x + [-0.05,0.002]` after the jump. If a variable is not reset by the jump, we may just neglect it.

FLOW* provides two options to aggregate a set of intersection over-approximations. The option

```
interval aggregation
```

tells the tool to use an interval aggregation for the jump. It simply computes an interval enclosure for all intersections. The second option is implemented based on the idea described in Section 4.4.2, it is specified by the form

```
parallelotope aggregation { <critical_directions> }
```

wherein the the critical directions can be specified by users, and the tool will also consider the vectors from the following set

$$L_{\text{def}} = \left\{ (a_1, \ldots, a_n) \;\middle|\; \begin{array}{c} \exists i.((1 \leq i \leq n) \wedge (a_i = 1) \wedge \forall j.((j \neq i) \rightarrow (a_j = 0))) \\ \bigvee \exists i.\exists j.[(1 \leq i < j \leq n) \wedge (a_i = 1) \wedge ((a_j = 1) \vee (a_j = -1)) \\ \wedge \forall k.(((1 \leq k \leq n) \wedge (k \neq i) \wedge (k \neq j)) \rightarrow (a_k = 0)))] \end{array} \right\}$$

wherein n is the number of the state variables.

5.3.2 Initial and unsafe set

An initial set can be given by an interval or a TM. If the system is hybrid, we also need to specify an initial mode. For an interval initial set, we only need to specify the interval range for each state variable. On the other hand, for a TM one, we should first declare the variables in the polynomial part and they should be distinguished from the state variables. Then we give the TM expression. An example is given as below,

```
tm var x0,x1,x2
x = 1 + x1^2 - x2 + [-0.02,0.01]
y = x0^3 - x1 + [0,0.1]
x0 in [-0.2,0.2]
x1 in [-0.2,0.2]
x2 in [-0.1,0.1]
```

wherein `x`, `y` are the state variables.

The unsafe set for a reachability problem is optional in FLOW*. It can be defined by a system of polynomial inequalities. If the system is hybrid, we may define such a set for each mode.

5.3.3 Reachability setting

Figure 5.2 shows a typical reachability setting in FLOW*. The detailed explanations are given as below.

Time step-size. The time step-size is fixed at 0.02. For any positive numbers a, b with $a \leq b$, users may specify

```
setting
{
  fixed steps 0.02
  time 10                   # the time horizon is [0,10]
  remainder estimation 1e-3
  identity precondition
  gnuplot octagon x,y       # produce a gnuplot file
  fixed orders 8
  cutoff 1e-15              # the cutoff threshold
  precision 53              # the precision used by MPFR library
  output result             # name of the output files
  max jumps 10              # the bound on the jump depth
  print on                  # print out the computation steps
}
```

Figure 5.2: Example of reachability setting

```
adaptive steps { min a , max b }
```

to use an adaptive step-size in $[a, b]$.

Remainder estimation. The remainder estimation in each integration step and in each dimension is $[-0.001, 0.001]$. Since a remainder estimation is not necessarily symmetric or has the same single interval in each dimension, we may also provide an arbitrary interval, for example,

```
remainder estimation { x:[0.1,0.101], y:[-0.01,0.06] }
```

It tells the tool to use $[0.1, 0.101]$ as the estimation in the dimension of x, and $[-0.01, 0.06]$ as the estimation in the dimension of y.

Preconditioning setting. We implemented two preconditioning techniques introduced in [MB05]. They are *QR preconditioning* and *identity preconditioning*. In the example, identity preconditioning will be used in the TM integration. To apply QR preconditioning, one may use

```
QR precondition
```

Plot setting. To visualize a computed TM flowpipe, Flow* can generate interval or octagon over-approximations for its 2-dimensional projections. In the example, the projection is in the x-y plane, and the TM flowpipes will be wrapped by octagons. One may replace `gnuplot` by `matlab` to obtain a Matlab file. To obtain a better visualization of the TMs, Flow* may also produce a grid paving on them, for example,

```
gnuplot grid 10 x,y
```

tells the tool to produce a 10×10 grid paving for each TM flowpipe projection.

TM order. The order of all TM flowpipes is 8 in the example. To perform an adaptive order, for example, ranges from 5 to 10, one may specify

```
adaptive orders { min 5 , max 10 }
```

We may also let the tool adapt the orders in different dimensions independently, for example,

```
adaptive orders { min { x:5 , y:5 } , max { x:10 , y:10 } }
```

5.3.4 Examples

We present example files for both continuous and hybrid reachability problems. Firstly, we consider the lac operon model given in [KHK+05]. The behavior of the system is characterized by the following ODE.

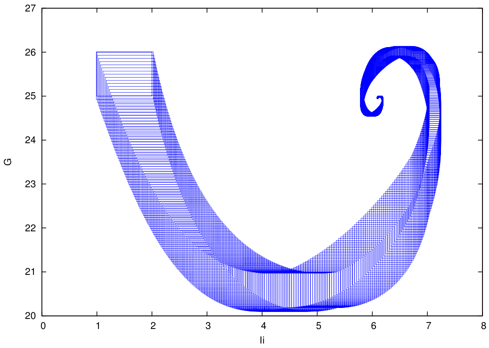
$$\begin{cases} \dot{I}_i &= -2 \cdot k_3 \cdot I_i^2 \cdot \frac{k_8 \cdot R_i \cdot G^2 + \tau}{k_3 \cdot I_i^2 + \mu} + 2 \cdot k_{-3} \cdot F_1 + \frac{(k_5 \cdot I_e - (k_9 + k_{-5}) \cdot I_i) \cdot k_{-2} \cdot \chi \cdot k_4 \cdot \eta \cdot (k_3 \cdot I_i^2 + \mu)}{k_7 \cdot (k_2 \cdot (k_8 \cdot R_i \cdot G^2 + \tau) + k_{-2} \cdot (k_3 \cdot I_i^2 + \mu))} \\ \dot{G} &= -2 \cdot k_8 \cdot R_i \cdot G^2 + \frac{2 \cdot k_{-8} \cdot (k_8 \cdot R_i \cdot G^2 + \tau)}{k_3 \cdot I_i^2 + \mu} + \frac{k_9 \cdot I_i \cdot k_{-2} \cdot \chi \cdot k_4 \cdot \eta \cdot (k_3 \cdot I_i^2 + \mu)}{k_7 \cdot (k_2 \cdot (k_8 \cdot R_i \cdot G^2 + \tau) + k_{-2} \cdot (k_3 \cdot I_i^2 + \mu))} \end{cases}$$

wherein I_i is the internal inducer and G is the glucose concentration. The constants are given as below.

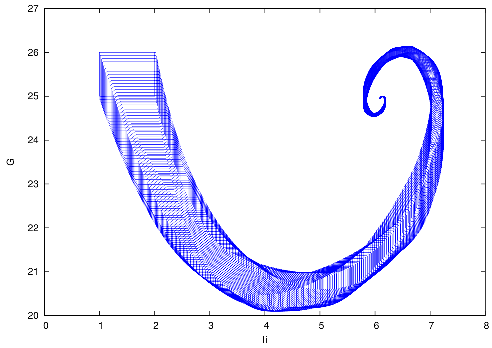
$$\begin{array}{lllll} k_2 = 4 \cdot 10^5, & k_{-2} = 0.03, & k_3 = 0.2, & k_{-3} = 60, & k_4 = 1, \\ k_5 = 0.6, & k_{-5} = 0.006, & k_6 = 3 \cdot 10^{-6}, & k_7 = 3 \cdot 10^{-6}, & k_8 = 0.03, \\ k_{-8} = 1 \cdot 10^{-5}, & k_9 = 5000, & R_i = 0.01, & \chi = 0.002002, & \eta = 0.005, \\ F_1 = 0.0001, & I_E = 91100, & \tau = 0.008. & & \end{array}$$

The reachability problem on the lac operon model w.r.t. the initial set $I_i(0) \in [1, 2]$, $G(0) \in [25, 26]$ and the time horizon $[0, 150]$ can be described by the input file below. Since the ODE contains non-polynomial expressions, we apply the integration scheme `nonpoly ode`. We allow the TM orders in different dimensions change independently. Figure 5.3 shows the plot results of three different settings. They are computed based on the same set of TM flowpipes.

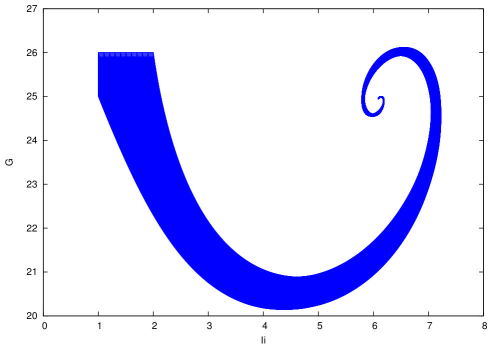
```
continuous reachability
{
 state var Ii , G

 setting
 {
  fixed steps 0.2
  time 150
  remainder estimation 1e-4
  QR precondition
  gnuplot octagon Ii,G
  adaptive orders { min {Ii:4, G:4} , max {Ii:6, G:6} }
  cutoff 1e-20
  precision 53
  output LacOperon
  print on
 }

 nonpoly ode
 {
```



(a) interval



(b) octagon



(c) grid 10

Figure 5.3: Flowpipe over-approximations of the lac operon model

```
   Ii' = - 0.4 * Ii^2 * ((0.0003*G^2 + 0.008)/(0.2*Ii^2 + 2.00001))
         + 0.012 + (0.0000003*(54660 - 5000.006*Ii)
         * (0.2*Ii^2 + 2.00001))/(0.00036*G^2 + 0.00960018
         + 0.000000018*Ii^2)
   G'  = - 0.0006*G^2 + (0.000000006*G^2 + 0.00000016)/(0.2*Ii^2
         + 2.00001) + (0.0015015*Ii*(0.2*Ii^2 + 2.00001))
         / (0.00036*G^2 + 0.00960018 + 0.000000018*Ii^2)
  }

  init
  {
   Ii in [1,2]
   G  in [25,26]
  }
 }
```

Now we turn to the hybrid example defined in [Gu11]. Its dynamics is defined as follows,

$$\begin{cases} \dot{x} &= -10 \cdot x + y + z + u \\ \dot{y} &= x - y \\ \dot{z} &= x - z - x^2 \end{cases}$$

wherein u is the input given as below.

$$u = \begin{cases} 1, & t < 5 \\ 5, & 5 \le t < 10 \\ 2, & 10 \le t < 15 \\ 4, & 15 \le t \le 20 \end{cases}$$

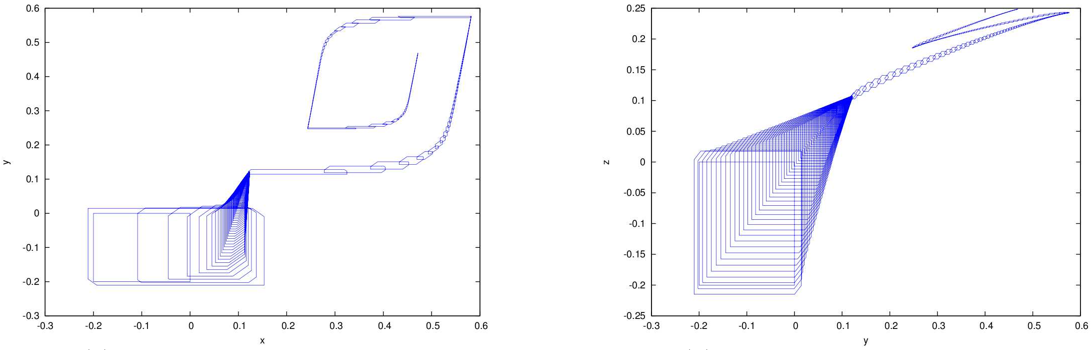
The initial condition under our consideration is

$$x(0) \in [-0.2, 0], \qquad y(0) \in [-0.2, 0], \qquad z(0) \in [-0.2, 0], \qquad t = 0$$

We show the modeling file of the reachability problem for the time horizon $[0, 20]$ as below. Figure 5.4 illustrates the octagon over-approximations of the computed TM flowpipes.

```
hybrid reachability
{
 state var x, y, z, t
 setting
 {
  fixed steps 0.05
  time 20
  remainder estimation 1e-2
  identity precondition
  gnuplot octagon x,y
  fixed orders 4
  cutoff 1e-12
  precision 53
  output nonlinear
  max jumps 3
  print on
 }

 modes
 {
  l1
  {
   poly ode 2
   {
    x' = -10*x + y + z + 1     y' = x - y
    z' = x - z - x^2           t' = 1
   }
   inv
   {
    t <= 5
   }
  }

  l2
  {
   poly ode 2
   {
    x' = -10*x + y + z + 5     y' = x - y
    z' = x - z - x^2           t' = 1
   }
   inv
   {
    t >= 5  t <= 10
```

```
    }
   }

   l3
   {
    poly ode 2
    {
     x' = -10*x + y + z + 2     y' = x - y
     z' = x - z - x^2           t' = 1
    }
    inv
    {
     t >= 10  t <= 15
    }
   }

   l4
   {
    poly ode 2
    {
     x' = -10*x + y + z + 4     y' = x - y
     z' = x - z - x^2           t' = 1
    }
    inv
    {
     t >= 15  t <= 20
    }
   }
  }

  jumps
  {
   l1 -> l2
   guard { t = 5 }
   reset { }
   parallelotope aggregation { }

   l2 -> l3
   guard { t = 10 }
   reset { }
   parallelotope aggregation { }

   l3 -> l4
   guard { t = 15 }
   reset { }
   parallelotope aggregation { }
  }

  init
  {
   l1
   {
    x in [-0.2,0]   y in [-0.2,0]   z in [-0.2,0]   t in [0,0]
   }
  }
 }
```



(a) Projection in the x-y plane

(b) Projection in the y-z plane

Figure 5.4: Flowpipe over-approximations of the non-linear hybrid system

```
<state_variable_declaration>
<location_invariants>          # only for hybrid systems
<computation_paths>            # only for hybrid systems
<plot_setting>
<output_name>
<unsafe_set>                   # optional
<flowpipe_expressions>
```

Figure 5.5: Format of the TM files

5.4 Format of Taylor model files

Flow* stores all computed TM flowpipes along with the state space specification in a file whose format is given by Figure 5.5. The reason to generate such a file is twofold. Firstly, one may change any of the invariant definition, plot setting, output name and the unsafe set, and let the tool verify the new safety property, or produce a new plot file. Secondly, the computed TM flowpipes can be input to other tools for further analysis.

Assume that `x1`, ..., `xn` are the state variables, a TM flowpipe is kept in the following form,

```
{
 x1 = p1 + [a1,b1]
 ...
 xn = pn + [an,bn]

 y1 in [c1,d1]
 ...
 ym in [cm,dm]
}
```

such that `p1`, ..., `pn` are polynomials over the TM variables `y1`, ..., `ym`.

For hybrid automata, a TM flowpipe should be associated with a mode. To do so, we keep those computed flowpipes by blocks each of which is associated with a mode.

```
{
 <mode_name>
 {
```

```
  <tm_flowpipes>
 }
 ...
 <mode_name>
 {
  <tm_flowpipes>
 }
}
```

The modes are arranged in the order they are visited in the reachability computation. Besides, the tool also outputs the computation paths (mode sequences) along with the intervals containing the time points when jumps are made. Such information could be useful to find counterexamples.

5.5 Performance evaluation

We evaluate the performance of Flow* v1.2.1 in the following aspects. For continuous reachability problems, we make a comparison with the validated ODE solver VNODE-LP [Ned11] based on a group of continuous systems whose dimensions range from 2 to 9. For hybrid systems, the comparison is done with a SMT-based tool named dReach [Gao12] based on the benchmarks presented in Section 4.5. Both of the tools have good performance on some challenging benchmarks. We also implemented a prototype of the fast integration method described in Section 3.6, a comparison of it with SpaceEx based on the filtered oscillator benchmarks is given. Besides, we also present a scalability evaluation of Flow* based on the non-linear line circuit benchmarks. Unlike the experiments in related work, we always consider relatively large initial sets, since they are challenging and very often in applications.

Our experimental platform is a computer equipped with a processor of Intel Core i7-860 (8M Cache, 2.80 GHz) and 4096 MB RAM. The operating system is Ubuntu 12.04 LTS.

5.5.1 Comparison with VNODE-LP

It is hard to exactly compare the accuracy between Flow* and VNODE-LP, since Flow* uses TM over-approximations for solutions whereas VNODE-LP uses intervals. To give a reasonable and clear comparison, we use the following method. Given a benchmark which consists of an ODE, an interval initial set and a bounded time horizon $[0, T]$, we evaluate an interval enclosure I_T of the TM flowpipe produced by Flow* at time T, and then compare $\mathrm{W}(I_T)$ with the width of the interval over-approximation computed by VNODE-LP. If $W(I_T)$ is smaller, then Flow* has a better accuracy than VNODE-LP on the benchmark. The reason to choose the end time point T is that the overestimation is eventually accumulated in both of the TM and interval integration method.

We present the comparison of the tools based on 9 benchmarks by Table 5.2. The initial sets and time horizons are same as those in Table 5.1. For each benchmark, we perform an N-subdivision on the initial set, since it is too large for VNODE-LP to integrate. We also tried to make N as small as possible.

		FLOW*						VNODE-LP		
Benchmark	Var	δ	k	I_e	ε	t (s)	W	N	t (s)	W
jet engine	2	0.04	$4 \sim 8$	$[-10^{-5}, 10^{-5}]^2$	10^{-12}	4.7	0.0281	8	**1.0**	**0.0207**
Brusselator	2	0.04	$3 \sim 6$	$[-10^{-5}, 10^{-5}]^2$	10^{-12}	4.9	**0.0264**	10	**1.9**	0.0335
Van der Pol	2	0.03	$5 \sim 8$	$[-10^{-5}, 10^{-5}]^2$	10^{-12}	7.6	0.5792	10	**0.8**	**0.5153**
Lorentz	3	0.01	$4 \sim 7$	$[-10^{-5}, 10^{-5}]^3$	10^{-12}	18	0.2939	10	**14**	**0.2631**
Rössler	3	0.02	$4 \sim 6$	$[-10^{-4}, 10^{-4}]^3$	10^{-12}	**6.6**	**1.9704**	20	54	1.9799
coupled Van der Pol	4	0.02	$5 \sim 8$	$[-10^{-5}, 10^{-5}]^4$	10^{-10}	**226**	**0.5465**	12	325	6.2236
Lotka-Volterra	5	0.02	$4 \sim 6$	$[-10^{-3}, 10^{-3}]^5$	10^{-10}	**176**	**0.0752**	8	333	0.0772
biological model I	7	0.01	$3 \sim 5$	$[-10^{-5}, 10^{-5}]^7$	10^{-9}	**72**	**0.1070**	4	487	0.1198
biological model II	9	0.01	4	$[-10^{-3}, 10^{-3}]^9$	10^{-9}	**298**	**1.5143**	4	39924	2.7952

Table 5.2: Comparison between FLOW* and VNODE-LP. Legends: **Var**: number of variables, $\boldsymbol{\delta}$: time step-size, $\boldsymbol{k}$: TM order, $\boldsymbol{I_e}$: remainder estimation, $\boldsymbol{\varepsilon}$: cutoff threshold, $\boldsymbol{t}$: time cost, $\boldsymbol{W}$: width of the interval enclosure computed for the solution at T, $\boldsymbol{N}$: number for subdivision on the initial set.

From the experimental results, it can be seen that the scalability of FLOW* is better than that of VNODE-LP. The reason is that we are able to accurately handle a large initial set as one piece.

5.5.2 Comparison with dReach

We consider the 11 hybrid system benchmarks described in Section 4.5 for the comparison with dReach. Note that they are not artificial, since each benchmark is a concrete application. The reason to choose dReach is that (a) the tool integrates a set of advanced techniques and shows a good ability to handle some non-trivial case studies, (b) it has a user-friendly interface and a hybrid reachability problem can be easily encoded by its language, and (c) compare with the other tools mentioned in Section 1.1.2, it works on the most number of the benchmarks with acceptable running times.

Our experimental scenarios are set as follows. Since dReach is not a tool to compute flowpipe over-approximations, our goal here is to prove safety properties. In the following content, we give a detailed description for all experimental results given in Table 5.3.

Non-holonomic integrator. The unsafe condition is given by $x \geq 3$ in both of the locations. FLOW* spends 201 seconds to compute the TM flowpipes in the time horizon $[0, 7.5]$, and proves the safety. However, for any $1 \leq N \leq 10$, by applying an N-subdivision on the initial set, dReach does not terminate in 1 hour for any path with 1 jump.

Spiking neurons. The unsafe condition for the first neuron model is $u \leq -25$, and that of the second one is $u \geq 250$. FLOW* proves the safety for both of the models with the time costs listed in Table 5.3. On the other hand, dReach only spends 0.3 second to prove the safety for the paths with less than 2 jumps of the first model, but not able to handle more jumps even the initial set is 1000-subdivided. For the second neuron model, dReach is not able to prove the safety in 1 hour for any path with the subdivision number 1 to 10.

		FLOW*					dReach			
Benchmark	Var	δ	k	I_e	ε	t (s)	N	k	p	t (s)
non-holonomic integrator	3	0.01	$5 \sim 8$	$[-10^{-5}, 10^{-5}]$	10^{-12}	**201**	$1 \sim 10$	≤ 1	0.001	T.O.
neuron I	2	0.02	$4 \sim 6$	$[-10^{-2}, 10^{-2}]$	10^{-12}	**367**	≥ 100	≤ 15	0.0001	fail
neuron II	2	0.02	$4 \sim 6$	$[-10^{-2}, 10^{-2}]$	10^{-12}	**70**	$1 \sim 10$	≤ 15	0.001	T.O.
inverted pendulum	3	0.01	$5 \sim 7$	$[-10^{-3}, 10^{-3}]$	10^{-12}	**52**	$1 \sim 10$	≤ 5	0.001	fail
aircraft	7	0.02	$2 \sim 5$	$[-10^{-6}, 10^{-6}]$	10^{-12}	**6.9**	10	≤ 2	0.01	278
glycemic control I	3	0.05	$2 \sim 5$	$[-10^{-1}, 10^{-1}]$	10^{-12}	64	5	≤ 2	0.01	**1.1**
glycemic control II	3	0.05	$2 \sim 5$	$[-10^{-1}, 10^{-1}]$	10^{-12}	**95**	$1 \sim 5$	≤ 2	0.01	T.O.
glycemic control III	3	0.05	$2 \sim 5$	$[-10^{-2}, 10^{-2}]$	10^{-12}	**46**	$1 \sim 5$	≤ 1	0.01	T.O.
line circuit $n = 2$	2	0.01	$3 \sim 6$	$[-10^{-3}, 10^{-3}]$	10^{-12}	2.3	1	≤ 2	0.01	**0.2**
line circuit $n = 4$	4	0.01	$3 \sim 6$	$[-10^{-4}, 10^{-4}]$	10^{-10}	48	4	≤ 2	0.01	**9.6**
line circuit $n = 6$	6	0.0002 $\sim$ 0.02	4	$[-10^{-6}, 10^{-6}]$	10^{-9}	**243**	4	≤ 2	0.01	T.O.

Table 5.3: Comparison between FLOW* and dReach. Legends: **Var**: number of variables, $\boldsymbol{\delta}$: time step-size, ***k* in Flow*** : TM order, $\boldsymbol{I_e}$: remainder estimation, $\boldsymbol{\varepsilon}$: cutoff threshold, $\boldsymbol{t}$: time cost, $\boldsymbol{N}$: number for subdivision on the initial set, ***k* in dReach** : unrolling depth of bounded model checking, $\boldsymbol{p}$: value of numerical perturbation, **T.O.**: time out, i.e., > 3600.

Inverted pendulum. We consider the unsafe condition $\theta \geq 5$. The tool dReach only costs 3 seconds to prove the safety for all paths with less than 2 jumps, but fail to deal with more jumps even the initial set is 10-subdivided. However, FLOW* spends 52 seconds to compute all TM flowpipes and even can prove the safety w.r.t. the less restrictive unsafe condition $\theta \geq 4.8$.

Aircraft collision avoidance. We want to ensure that the two aircraft can never be located in a box with radius 0.1. We have to perform a 10-subdivision on the initial set to make dReach prove the safety in a reasonably short time, whereas FLOW* completes the flowpipe construction as well as the safety verification in 7 seconds.

Glycemic control. We use the same unsafe condition $G \leq -2$ for all of the three glycemic control models. dReach outperforms FLOW* on the first control model when the initial set is 5-subdivided. However, it is not able to prove the safety in 1 hour for any of the rest two models.

Line circuit. The unsafe condition is set to be $v_1 \geq 0.21$ for the line circuit model of any scale. dReach has a much better performance than FLOW* when $n = 2$ and $n = 4$. However, for the larger scale $n = 6$, because of the hardness of the continuous dynamics, dReach fails to work on the 3-subdivision of the initial set, and can not give a result in 1 hour for the 4-subdivision case.

By our observation, dReach and FLOW* have advantages over different case studies. The former one works more efficient on the systems with moderate dynamics or tiny initial sets. On the other hand, by using TMs, FLOW* is able to handle quite difficult continuous

			Flow*				SpaceEx (LGG)			SpaceEx (STC)	
Benchmark	Var	T	δ	k	P	time	δ	box	octagon	box	octagon
filtered oscillator 6	6	$[0,4]$	0.05	8	128	2.1	0.05	0.3	3.4	0.2	1.9
filtered oscillator 10	10	$[0,4]$	0.05	8	128	5.9	0.05	0.4	31	0.5	21
filtered oscillator 18	18	$[0,4]$	0.05	8	128	20	0.05	0.6	617	1.2	425
filtered oscillator 34	34	$[0,4]$	0.05	8	128	98	0.05	12	T.O.	5.1	T.O.

Table 5.4: Comparison between Flow* and SpaceEx. Legends: **Var**: number of variables, $\boldsymbol{T}$: time horizon, $\boldsymbol{\delta}$: time step-size, $\boldsymbol{k}$: TM order, **P**: precision, **box**: box over-approximation, **octagon**: octagon over-approximation, **T.O.**: time out, i.e., > 1800.

dynamics and relatively large initial sets. A large number of jumps may degenerate the performance of both tools.

5.5.3 Comparison with SpaceEx

We embedded the fast integration method presented in Section 3.6 into the framework of computing TM flowpipes for hybrid automata, and make a comparison with SpaceEx on the filtered oscillator benchmarks [FLD+11]. The experimental results are given in Table 5.4. For each benchmark, the TM flowpipe at time $t = 4$ is ensured to be included in the box flowpipe at $t = 4$ computed by SpaceEx.

As we discussed in Section 3.6, although it is often not necessary for SpaceEx to compute octagon flowpipes, we may need them for the reuse purpose. On the other hand, TM flowpipes can always be directly reused with different unsafe specifications. It can be seen that our method is very competitive to the support function methods.

5.5.4 Scalability evaluation

It is also significant to investigate the performance of Flow* on the examples of different scales. Here, we present a scalability evaluation of the tool based on the non-linear line circuit benchmarks. The number of variables under our consideration are the even numbers from 2 to 12. For each benchmarks, the reachability setting is chosen to achieve a good performance.

As we said, the line circuit benchmarks can be either continuous or hybrid based on the definition of the input $i(t)$. Thus, we use the input defined in Section 4.5.6 for the hybrid case, and use $i(t) = \sin(5 \cdot t)$ for the continuous case. Since the continuous dynamics are not easy to handle, we have to use small time step-sizes. Due to the limit of the memory size, we set the time horizon for all continuous benchmarks by $[0, 2]$, and for all hybrid benchmarks by $[0, 2.5]$.

We present the experimental results in Table 5.5. It can be seen that Flow* is able to deal with the benchmarks with even 12 variables.

	continuous					hybrid				
n	δ	k	I_e	ε	t (s)	δ	k	I_e	ε	t (s)
2	0.03	$3 \sim 6$	$[-10^{-3}, 10^{-3}]$	10^{-12}	1.4	0.01	$3 \sim 6$	$[-10^{-3}, 10^{-3}]$	10^{-12}	2.3
4	0.01	$3 \sim 6$	$[-10^{-5}, 10^{-5}]$	10^{-10}	56	0.01	$3 \sim 6$	$[-10^{-4}, 10^{-4}]$	10^{-10}	48
6	0.0002 $\sim$ 0.02	4	$[-10^{-5}, 10^{-5}]$	10^{-8}	73	0.0002 $\sim$ 0.02	4	$[-10^{-5}, 10^{-5}]$	10^{-8}	243
8	0.0002 $\sim$ 0.01	4	$[-10^{-5}, 10^{-5}]$	10^{-8}	176	0.0002 $\sim$ 0.01	4	$[-10^{-5}, 10^{-5}]$	10^{-8}	851
10	0.0002 $\sim$ 0.005	4	$[-10^{-5}, 10^{-5}]$	10^{-7}	205	0.0002 $\sim$ 0.005	4	$[-10^{-5}, 10^{-5}]$	10^{-7}	904
12	0.0002 $\sim$ 0.005	4	$[-10^{-5}, 10^{-5}]$	10^{-7}	402	0.0002 $\sim$ 0.005	4	$[-10^{-5}, 10^{-5}]$	10^{-7}	1933

Table 5.5: Scalability evaluation of FLOW* on the non-linear line circuit benchmarks. Legends: **Var**: number of variables, $\boldsymbol{\delta}$: time step-size, $\boldsymbol{k}$: TM order, $\boldsymbol{I_e}$: remainder estimation, $\boldsymbol{\varepsilon}$: cutoff threshold, $\boldsymbol{t}$: time cost.

5.6 Future work

Although we are able to deal with relatively large initial sets and some high dimensional systems using TMs, it is still time-costly to compute TM flowpipes in general when the system has a large number of variables. Therefore, we plan to improve the tool FLOW* in the following aspects.

- *Composition of hybrid automata.* Currently, FLOW* only accepts one hybrid automaton in a reachability problem. Then, we have to compose a hybrid automaton manually, when it is given by its components. In the future, FLOW* will have a function to compute a composition of finitely many hybrid automata.

- *Combination of adaptive step-sizes and orders.* FLOW* does not support using both adaptive step-sizes and orders in an integration task currently. However, we are planing to design a strategy to combine the two adaptive techniques.

- *More sophisticated approach for TM simplification and aggregation.* As we mentioned, manipulating high-order TMs is expensive when the number of variables is not small. Currently, FLOW* simply moves the "small" terms of the polynomial part of a TM into the remainder interval. Such a method, however, could be too conservative in some case. It could be more effective to do the simplification with regard to some specified variables. For intersection aggregation, the current version of FLOW* only supports intervals or parallelotopes. We seek to include more aggregation schemes, such as using zonotopes or higher-order TMs.

- *More accurate intersection over-approximation.* FLOW* uses the standard interval arithmetic to evaluate the value of a function or check the satisfiability of a constraint. Therefore, the intersection over-approximations could be too coarse in some situations. In the future, we plan to use one or several SMT solvers to handle the satisfiability checking in domain contraction.

- *Better memory management.* Currently, FLOW* keeps all computed TM flowpipes in memory and it often consumes all usable memory in a system when the flowpipe number is large. In the next version, the tool will periodically dump the computed TM flowpipes into a file and releases the memory they use.

Chapter 6

Conclusion

In the thesis, we introduce the use of Taylor Models (TMs) in the reachability analysis of non-linear hybrid systems. The method provides a new perspective of using high-order over-approximations for the bounded reachable sets of non-linear hybrid systems. It shows by the experiments that our techniques may handle hybrid systems with even more than 10 variables. The main contribution of the thesis is briefly reviewed as below.

- *TM integration with adaptive techniques and efficient TM computation.* It is an extension of the work from Berz and Makino. The new techniques improve the performance of computing TM flowpipes with acceptable loss of accuracy. Such a result is further extended to deal with ODEs with time-varying parameters. Besides, we also present an efficient method to handle linear ODEs.

- *Efficient flowpipe/guard intersection methods.* We present two techniques: domain contraction and range over-approximation for enclosing the intersection of a TM flowpipe and a jump guard. Unlike the interval-based methods in the related work, our approaches never require to split a given set and hence show a low computational complexity. Moreover, in order to further improve the performance during the reachability computation, we also present different heuristics for aggregating intersection over-approximations.

- *The tool* FLOW* *for the reachability analysis of non-linear hybrid systems.* We implemented most of our techniques in the tool FLOW*. It deals with two classes of tasks: (a) TM flowpipe computation for continuous and hybrid systems, and (b) safety analysis on TMs. As it is shown by the experiments, FLOW* is very competitive to the other tools, and its advantage is apparent on medium- and large-scale systems.

The work in the thesis can be extended in several directions.

(1) *Better flowpipe over-approximations.* There are several possibilities to compute a flowpipe over-approximation which is more accurate than a TM. One of them could be combining the use of TM arithmetic and the interval Hermite-Obreschkoff method [Ned99]. The latter one has been successfully applied to integrating stiff or chaotic ODEs for long time horizons. As another possibility, one may use better approximation forms than Taylor expansions for ODE solutions. The candidates could be Bernstein polynomials and Chebyshev polynomials. However, neither of the forms

could be calculated based on the monomial basis, and therefore to efficient manipulate their multivariate versions is challenging.

(2) *Use of TMs in statistical model checking.* In [ZSS+13], we investigate the use of TMs to enclose a set of simulation trajectories of analog circuits. The experiments show that our methods can greatly improve the performance on producing results with high confidence values. An extension of the work is described in [ZSS14].

(3) *Use of TMs to generate flowpipe under-approximations.* Unlike over-approximations, very little work has been focused on computing reachable set under-approximations. Since TMs can provide high-order over-approximations for the forward flowmaps of continuous systems, it is also possible to use a TM to over-approximate a backward flowmap. Based on it, we are able to track the evolution of the constraints defining the initial set. In [CSÁ14], an approach to compute flowpipe under-approximations is presented. Although it requires to check the connectedness of an approximation set when the system is non-linear, the method is still able to handle a non-trivial system with 7 variables. It is promising to make an improvement on it such that the connectedness verification is not needed any more. We also plan to extend this approach to produce guaranteed counterexamples for non-linear hybrid systems.

Besides the above ones, it is also significant to extend the TM techniques to do Signal Temporal Logic (STL) or Metric Temporal Logic (MTL) model checking.

Bibliography

[ACH$^+$95] 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. *Theor. Comput. Sci.*, 138(1):3–34, 1995.

[AD94] R. Alur and D. L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.

[ADE$^+$01] R. Alur, T. Dang, J. M. Esposito, R. B. Fierro, Y. Hur, F. Ivancic, V. Kumar, I. Lee, P. Mishra, G. J. Pappas, and O. Sokolsky. Hierarchical hybrid modeling of embedded systems. In *Proceedings of the 1st International Workshop on Embedded Software (EMSOFT'01)*, volume 2211 of *Lecture Notes in Computer Science*, pages 14–31. Springer, 2001.

[ADG07] E. Asarin, T. Dang, and A. Girard. Hybridization methods for the analysis of nonlinear systems. *Acta Inf.*, 43(7):451–476, 2007.

[ADI03] R. Alur, T. Dang, and F. Ivancic. Progress on reachability analysis of hybrid systems using predicate abstraction. In *Proceedings of the 6th International Workshop on Hybrid Systems: Computation and Control (HSCC'03)*, volume 2623 of *Lecture Notes in Computer Science*, pages 4–19. Springer, 2003.

[ADM02] E. Asarin, T. Dang, and O. Maler. The d/dt tool for verification of hybrid systems. In *Proceedings of the 14th International Conference on Computer Aided Verification (CAV'02)*, volume 2404 of *Lecture Notes in Computer Science*, pages 365–370. Springer, 2002.

[ADMT10] E. Asarin, T. Dang, O. Maler, and R. Testylier. Using redundant constraints for refinement. In *Proceedings of the 8th International Symposium on Automated Technology for Verification and Analysis (ATVA'10)*, volume 6252 of *Lecture Notes in Computer Science*, pages 37–51. Springer, 2010.

[AF92] D. Avis and K. Fukuda. A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. *Discrete & Computational Geometry*, 8(1):295–313, 1992.

[ÅF00] K. J. Åström and K. Furuta. Swinging up a pendulum by energy control. *Automatica*, 36(2):287–295, 2000.

[AGH$^+$00] R. Alur, R. Grosu, Y. Hur, V. Kumar, and I. Lee. Modular specification of hybrid systems in charon. In *Proceedings of the 3rd workshop on Hybrid Systems: Computation and Control (HSCC'00)*, volume 1790 of *Lecture Notes in Computer Science*, pages 6–19. Springer, 2000.

[AHS09] K. Atkinson, W. Han, and D. E. Stewart. *Numerical Solution of Ordinary Differential Equations*. John Wiley & Sons, 2009.

[Alt10] M. Althoff. *Reachability Analysis and its Application to the Safety Assessment of Autonomous Cars*. PhD thesis, Technischen Universität München, 2010.

[ÅM11] K. J. Åström and R. M. Murray. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press, 2011.

[Apo67] T. M. Apostol. *Calculus, Vol. 1, One-Variable Calculus with an Introduction to Linear Algebra*. Wiley, 1967.

[Apo69] T. M. Apostol. *Calculus, Vol. 2, Multi-Variable Calculus and Linear Algebra with Applications*. Wiley, 1969.

[APS08] 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*, 44(8):2163–2170, 2008.

[ASB08] M. Althoff, O. Stursberg, and M. Buss. Reachability analysis of nonlinear systems with uncertain parameters using conservative linearization. In *Proceedings of the 47th IEEE Conference on Decision and Control (CDC'08)*, pages 4042–4048. IEEE, 2008.

[ASB10] M. Althoff, O. Stursberg, and M. Buss. Computing reachable sets of hybrid systems using a combination of zonotopes and polytopes. *Nonlinear Analysis: Hybrid Systems*, 4(2):233–249, 2010.

[BBC+08] L. Benvenuti, D. Bresolin, A. Casagrande, P. Collins, A. Ferrari, E. Mazzi, A. Sangiovanni-Vincentelli, and R. Villa. Reachability computation for hybrid systems with Ariadne. In *Proceedings of the 17th IFAC World Congress*. IFAC Papers-OnLine, 2008.

[Bem04] A. Bemporad. Efficient conversion of mixed logical dynamical systems into an equivalent piecewise affine form. *IEEE Transactions on Automatic Control*, 49(5):832–838, 2004.

[Ber99] M. Berz. *Modern Map Methods in Particle Beam Physics*, volume 108 of *Advances in Imaging and Electron Physics*. Academic Press, 1999.

[BFA85] R. N. Bergman, D. T. Finegood, and M. Ader. Assessment of insulin sensitivity in vivo. *Endocrine Reviews*, 6:45–86, 1985.

[BG06] F. Benhamou and L. Granvilliers. Continuous and interval constraints. In F. Rossi et al., editor, *Handbook of Constraint Programming*, pages 571–590. Elsevier, 2006.

[BIBC79] R. N. Bergman, Y. Z. Ider, C. R. Bowden, and C. Cobelli. Quantitative estimation of insulin sensitivity. *The American Journal of Physiology*, 236:E667–677, 1979.

[BJ10] N. Brisebarre and M. Joldeş. Chebyshev interpolation polynomial-based tools for rigorous computing. In *Proceedings of the 2010 International Symposium on Symbolic and Algebraic Computation (ISSAC'10)*, pages 147–154. ACM, 2010.

[BJMD+12] N. Brisebarre, M. Joldes, É. Martin-Dorel, M. Mayero, J.-M. Muller, I. Pasca, L. Rideau, and L. Théry. Rigorous polynomial approximation using taylor models in coq. In *Proceedings of the 4th International Symposium on NASA Formal Methods (NFM'12)*, volume 7226 of *Lecture Notes in Computer Science*, pages 85–99. Springer, 2012.

[BM98] M. Berz and K. Makino. Verified integration of ODEs and flows using differential algebraic methods on high-order Taylor models. *Reliable Computing*, 4:361–369, 1998.

[BM09] M. Berz and K. Makino. Rigorous global search using taylor models. In *Proceedings of the 2009 Conference on Symbolic Numeric Computation (SNC'09)*, pages 11–20. ACM, 2009.

[BMP99] O. Bournez, O. Maler, and A. Pnueli. Orthogonal polyhedra: Representation and computation. In *Proceedings of the 2nd workshop on Hybrid Systems: Computation and Control (HSCC'99)*, volume 1569 of *Lecture Notes in Computer Science*, pages 46–60. Springer, 1999.

[BPC81] R. N. Bergman, L. S. Phillips, and C. Cobelli. Physiologic evaluation of factors controlling glucose tolerance in man: measurement of insulin sensitivity and β-cell glucose sensitivity from the response to intravenous glucose. *The Journal of Clinical Investigation*, 68:1456–1467, 1981.

[BT00] O. Botchkarev and S. Tripakis. Verification of hybrid systems with linear differential inclusions using ellipsoidal approximations. In *Proceedings of the 3rd workshop on Hybrid Systems: Computation and Control (HSCC'00)*, volume 1790 of *Lecture Notes in Computer Science*, pages 73–88. Springer, 2000.

[BV04] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[CÁ11] X. Chen and E. Ábrahám. Choice of directions for the approximation of reachable sets for hybrid systems. In *Proceedings of the 13th International Conference on Computer Aided Systems Theory (EUROCAST'11)*, volume 6927 of *Lecture Notes in Computer Science*, pages 535–542. Springer, 2011.

[CÁF11] X. Chen, E. Ábrahám, and G. Frehse. Efficient bounded reachability computation for rectangular automata. In *Proceedings of the 5th International Workshop on Reachability Problems (RP'11)*, volume 6945 of *Lecture Notes in Computer Science*, pages 139–152. Springer, 2011.

[CÁS12] X. Chen, E. Ábrahám, and S. Sankaranarayanan. Taylor model flowpipe construction for non-linear hybrid systems. In *Proceedings of the 33rd IEEE Real-Time Systems Symposium (RTSS'12)*, pages 183–192. IEEE Computer Society, 2012.

[CC77] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM Symposium on Principles of Programming Languages (POPL'77)*, pages 238–252. ACM, 1977.

[CG02] M. Ceberio and L. Granvilliers. Horner's rule for interval evaluation revisited. *Computing*, 69(1):51–81, 2002.

[CGP99] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.

[CK98] A. Chutinan and B. H. Krogh. Computing polyhedral approximations to flow pipes for dynamic systems. In *Proceedings of the 37th IEEE Conference on Decision and Control (CDC'98)*, volume 2, pages 2089–2094, 1998.

[CK04] M. Ceberio and V. Kreinovich. Greedy algorithms for optimizing multivariate horner schemes. *SIGSAM Bull.*, 38(1):8–15, 2004.

[CKBC84] D. J. Chisholm, E. W. Kraegen, D. J. Bell, and D. R. Chipps. A semi-closed loop computer-assisted insulin infusion system. *The Medical journal of Australia*, 141:784–789, 1984.

[CSÁ14] X. Chen, S. Sankaranarayanan, and E. Ábrahám. Under-approximate flowpipes for non-linear continuous systems. In *Proceedings of the 14th Conference on Formal Methods in Computer-Aided Design (FMCAD'14)*, pages 59–66. IEEE, 2014.

[CSS03] M. Colón, S. Sankaranarayanan, and H. Sipma. Linear invariant generation using non-linear constraint solving. In *Proceedings of 15th International Conference on Computer Aided Verification (CAV'03)*, volume 2725 of *Lecture Notes in Computer Science*, pages 420–432. Springer, 2003.

[CW00] Y. Chen and J. White. A quadratic method for nonlinear model order reduction. In *Proceedings of the 2000 International Conference on Modeling and Simulation of Microsystems*, pages 477–480, 2000.

[Dan00] T. Dang. *Verification and Synthesis of Hybrid Systems*. PhD thesis, Institut National Polytechnique de Grenoble, 2000.

[DLM09] T. Dang, C. Le Guernic, and O. Maler. Computing reachable states for non-linear biological models. In *Proceedings of the 7th International Conference on Computational Methods in Systems Biology (CMSB'09)*, volume 5688 of *Lecture Notes in Computer Science*, pages 126–141. Springer, 2009.

[DM07] A. Donzé and O. Maler. Systematic simulation using sensitivity analysis. In *Proceedings of the 10th International Workshop on Hybrid Systems: Computation and Control (HSCC'07)*, volume 4416 of *Lecture Notes in Computer Science*, pages 174–189. Springer, 2007.

[DMT10] T. Dang, O. Maler, and R. Testylier. Accurate hybridization of nonlinear systems. In *Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control (HSCC'10)*, pages 11–20. ACM, 2010.

[Don07] A. Donzé. *Trajectory-based verification and controller Synthesis for continuous and hybrid systems.* PhD thesis, University Joseph Fourier, 2007.

[Edg08] G. Edgar. *Measure, Topology, and Fractal Geometry (2nd edition).* Springer, 2008.

[ERNF11] A. Eggers, N. Ramdani, N. Nedialkov, and M. Fränzle. Improving sat modulo ode for hybrid systems analysis by combining different enclosure methods. In *Proceedings of the 9th International Conference on Software Engineering and Formal Methods (SEFM'11)*, volume 7041 of *Lecture Notes in Computer Science*, pages 172–187. Springer, 2011.

[FH07] M. Fränzle and C. Herde. Hysat: An efficient proof engine for bounded model checking of hybrid systems. *Formal Methods in System Design*, 30(3):179–198, 2007.

[FHT+07] 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*, 1(3-4):209–236, 2007.

[Fis91] M. E. Fisher. A semiclosed-loop algorithm for the control of blood glucose levels in diabetics. *IEEE transactions on biomedical engineering*, 38(1):57–61, 1991.

[FKL13] G. Frehse, R. Kateja, and C. Le Guernic. Flowpipe approximation and clustering in space-time. In *Proceedings of the 16th international conference on Hybrid systems: Computation and Control (HSCC'13)*, pages 203–212. ACM, 2013.

[FKSC85] S. M. Furler, E. W. Kraegen, R. H. Smallwood, and D. J. Chisholm. Blood glucose control by intermittent loop closure in the basal mode: computer simulation studies with a diabetic model. *Diabetes Care*, 8:553–561, 1985.

[FLD+11] 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 *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *Lecture Notes in Computer Science*, pages 379–395. Springer, 2011.

[Fre05a] G. Frehse. *Compositional Verification of Hybrid Systems using Simulation Relations.* PhD thesis, Radboud Universiteit Nijmegen, 2005.

[Fre05b] G. Frehse. Phaver: Algorithmic verification of hybrid systems past hytech. In *Proceedings of the 8th International Workshop on Hybrid Systems: Computation and Control (HSCC'05)*, volume 3414 of *Lecture Notes in Computer Science*, pages 258–273. Springer, 2005.

[Gao12] S. Gao. *Computable Analysis, Decision Procedures, and Hybrid Automata: A New Framework for the Formal Verification of Cyber-Physical Systems.* PhD thesis, Carnegie Mellon University, 2012.

[GHT$^+$04] R. Goebel, J. Hespanha, A. R. Teel, C. Cai, and R. Sanfelice. Hybrid systems: Generalized solutions and robust stability. In *IFAC Symposium on Nonlinear Control Systems (NOLCOS'04)*, pages 1–12, 2004.

[Gir05] A. Girard. Reachability of uncertain linear systems using zonotopes. In *Proceedings of the 8th International Workshop on Hybrid Systems: Computation and Control (HSCC'05)*, volume 3414 of *Lecture Notes in Computer Science*, pages 291–305. Springer, 2005.

[GKC13] S. Gao, S. Kong, and E. M. Clarke. Satisfiability modulo odes. In *Proceedings of the 13th International Conference on Formal Methods in Computer-Aided Design (FMCAD'13)*, pages 105–112. IEEE, 2013.

[GL08] A. Girard and C. Le Guernic. Zonotope/hyperplane intersection for hybrid systems reachability analysis. In *Proceedings of the 11th International Workshop on Hybrid Systems: Computation and Control*, volume 4981 of *Lecture Notes in Computer Science*, pages 215–228. Springer, 2008.

[GP05] A. Girard and G. J. Pappas. Approximate bisimulations for constrained linear systems. In *Proceedings of the 44th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC'05)*, pages 4700–4705, 2005.

[GP06] A. Girard and G. J. Pappas. Verification using simulation. In *Proceedings of the 9th International Workshop on Hybrid Systems: Computation and Control (HSCC'06)*, volume 3927 of *Lecture Notes in Computer Science*, pages 272–286. Springer, 2006.

[GT08] S. Gulwani and A. Tiwari. Constraint-based approach for analysis of hybrid systems. In *Proceedings of the 20th International Conference on Computer Aided Verification (CAV'08)*, volume 5123 of *Lecture Notes in Computer Science*, pages 190–203. Springer, 2008.

[Gu11] C. Gu. *Model Order Reduction of Nonlinear Dynamical Systems*. PhD thesis, University of California, Berkeley, 2011.

[HHWT95] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. Hytech: the next generation. In *Proceedings of the 16th IEEE Real-Time Systems Symposium (RTSS'95)*, pages 56–65. IEEE Computer Society, 1995.

[HHWT97] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. Hytech: A model checker for hybrid systems. In *Proceedings of the 9th International Conference on Computer Aided Verification (CAV'97)*, volume 1254 of *Lecture Notes in Computer Science*, pages 460–463. Springer, 1997.

[HKPV95] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing (STOC'95)*, pages 373–382. ACM, 1995.

[HM99] J. P. Hespanha and A. S. Morse. Stabilization of nonholonomic integrators via logic-based switching. *Automatica*, 35(3):385–393, 1999.

[HRGZ97] M. Henk, J. Richter-Gebert, and G. M. Ziegler. Basic properties of convex polytopes. In J. E. Goodman et al., editor, *Handbook of Discrete and Computational Geometry*, pages 243–270. CRC Press, Inc., 1997.

[HTP05] E. Haghverdi, P. Tabuada, and G. J. Pappas. Bisimulation relations for dynamical, control, and hybrid systems. *Theor. Comput. Sci.*, 342(2-3):229–261, 2005.

[IF79] K. Ichida and Y. Fujii. An interval arithmetic method for global optimization. *Computing*, 23(1):85–97, 1979.

[Izh10] E. M. Izhikevich. *Dynamical Systems in Neuroscience: The Geometry of Excitability and Bursting*. The MIT Press, 2010.

[JKDW01] L. Jaulin, M. Kieffer, O. Didrit, and E. Walter. *Applied Interval Analysis*. Springer, 2001.

[Jol02] I. T. Jolliffe. *Principal Component Analysis, 2nd edition*. Springer, 2002.

[KC91] D. R. Kincaid and E. W. Cheney. *Numerical Analysis: Mathematics of Scientific Computing*. Brooks Cole, 1991.

[KHK+05] E. Klipp, R. Herwig, A. Kowald, C. Wierling, and H. Lehrach. *Systems Biology in Practice: Concepts, Implementation and Application*. Wiley-Blackwell, 2005.

[KV00] A. B. Kurzhanski and P. Varaiya. Ellipsoidal techniques for reachability analysis. In *Proceedings of the 3rd workshop on Hybrid Systems: Computation and Control (HSCC'00)*, volume 1790 of *Lecture Notes in Computer Science*, pages 202–214. Springer, 2000.

[KV06] A. A. Kurzhanskiy and P. Varaiya. Ellipsoidal toolbox. Technical Report UCB/EECS-2006-46, EECS Department, University of California, Berkeley, May 2006.

[Kva08] M. Kvasnica. *Efficient software tools for control and analysis of hybrid systems*. PhD thesis, ETH Zürich, 2008.

[Le 09] C. Le Guernic. *Reachability Analysis of Hybrid Systems with Linear Continuous Dynamics*. PhD thesis, Université Joseph Fourier, 2009.

[LG09] C. Le Guernic and A. Girard. Reachability analysis of hybrid systems using support functions. In *Proceedings of the 21st International Conference on Computer Aided Verification (CAV'09)*, volume 5643 of *Lecture Notes in Computer Science*, pages 540–554. Springer, 2009.

[Lib03] D. Liberzon. *Switching in Systems and Control*. Springer, 2003.

[Loh92] R. J. Lohner. Computation of guaranteed enclosures for the solutions of ordinary initial and boundary value problems. In J. R. Cash et al., editor, *Computational ordinary differential equations*, pages 425–435. Clarendon Press, 1992.

[Lor63] E. N. Lorentz. Deterministic nonperiodic flow. *Journal of Atmospheric Sciences*, 20:130–141, 1963.

[LS07] Y. Lin and M. A. Stadtherr. Validated solutions of initial value problems for parametric odes. *Appl. Numer. Math.*, 57(10):1145–1162, 2007.

[Mak98] K. Makino. *Rigorous analysis of nonlinear motion in particle accelerators.* PhD thesis, Michigan State University, 1998.

[MB96] K. Makino and M. Berz. Remainder differential algebras and their applications. In M. Berz et al., editor, *Computational Differentiation: Techniques, Applications, and Tools*, pages 63–75. SIAM, 1996.

[MB03] K. Makino and M. Berz. Taylor models and other validated functional inclusion methods. *J. Pure and Applied Mathematics*, 4(4):379–456, 2003.

[MB05] K. Makino and M. Berz. Suppression of the wrapping effect by taylor model-based verified integrators: Long-term stabilization by preconditioning. *International Journal of Differential Equations and Applications*, 10(4):353–384, 2005.

[MB06] K. Makino and M. Berz. COSY INFINITY version 9. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 558(1):346–350, 2006.

[MB09] K. Makino and M. Berz. Rigorous integration of flows and ODEs using Taylor models. In *Proceedings of the 2009 conference on Symbolic numeric computation (SNC'09)*, pages 79–84. ACM, 2009.

[Mei07] J. D. Meiss. *Differential Dynamical Systems.* SIAM publishers, 2007.

[MH02] J. C. Mason and D. C. Handscomb. *Chebyshev Polynomials.* Chapman and Hall/CRC, 2002.

[MKC09] R. E. Moore, R. B. Kearfott, and M. J. Cloud. *Introduction to Interval Analysis.* SIAM, 2009.

[MT00] I. Mitchell and C. Tomlin. Level set methods for computation in hybrid systems. In *Proceedings of the 3rd International Workshop on Hybrid Systems: Computation and Control (HSCC'00)*, volume 1790 of *Lecture Notes in Computer Science*, pages 310–323. Springer, 2000.

[MT05] I. Mitchell and J. A. Templeton. A toolbox of hamilton-jacobi solvers for analysis of nondeterministic continuous and hybrid systems. In *Proceedings of the 8th International Workshop on Hybrid Systems: Computation and Control (HSCC'05)*, volume 3414 of *Lecture Notes in Computer Science*, pages 480–494. Springer, 2005.

[MV03] C. Moler and C. Van Loan. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM Review*, 45(1):3–49, 2003.

[Ned99] N. S. Nedialkov. *Computing Rigorous Bounds on the Solution of an Initial Value Problem for an Ordinary Differential Equation*. PhD thesis, University of Toronto, 1999.

[Ned11] N. S. Nedialkov. Implementing a rigorous ode solver through literate programming. In A. Rauh and E. Auer, editors, *Modeling, Design, and Simulation of Systems with Uncertainties*, volume 3 of *Mathematical Engineering*, chapter Mathematical Engineering, pages 3–19. Springer Berlin Heidelberg, 2011.

[NJC99] N. S. Nedialkov, K. R. Jackson, and G. F. Corliss. Validated solutions of initial value problems for ordinary differential equations. *Applied Mathematics and Computation*, 105(1):21–68, 1999.

[NJN06] M. Neher, K. R. Jackson, and N. S. Nedialkov. On Taylor model based integration of ODEs. *SIAM Journal on Numerical Analysis*, 45:236–262, 2006.

[Pab03] P. A. Pablo. Semidefinite programming relaxations for semialgebraic problems. *Mathematical Programming*, 96(2):293–320, 2003.

[Par00] P. A. Parrilo. *Structured Semidefinite Programs and Semialgebraic Geometry Methods in Robustness and Optimization*. PhD thesis, California Institute of Technology, 2000.

[Pen00] J. M. Pena. On the multivariate horner scheme. *SIAM Journal on Numerical Analysis*, 37(4):1186–1197, 2000.

[Phi03] G. M. Phillips. *Interpolation and Approximation by Polynomials*. Springer, 2003.

[Pla10] A. Platzer. *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Springer, 2010.

[PQ08] A. Platzer and J.-D. Quesel. Keymaera: A hybrid theorem prover for hybrid systems (system description). In *Proceedings of the 4th International Joint Conference on Automated Reasoning (IJCAR'08)*, volume 5195 of *Lecture Notes in Computer Science*, pages 171–178. Springer, 2008.

[RH80] R. H. Rand and P.J. Holmes. Bifurcation of periodic motions in two weakly coupled Van der Pol oscillators. *International Journal of Non-Linear Mechanics*, 15(4-5):387–399, 1980.

[RN11] N. Ramdani and N. S. Nedialkov. Computing reachable sets for uncertain nonlinear hybrid systems using interval constraint-propagation techniques. *Nonlinear Analysis: Hybrid Systems*, 5(2):149–162, 2011.

[Rös76] O. E. Rössler. An equation for continuous chaos. *Physics Letters A*, 57(5):397–398, 1976.

[Rös79] O. E. Rössler. An equation for hyperchaos. *Physics Letters A*, 71(2-3):155–157, 1979.

[RS05] S. Ratschan and Z. She. Safety verification of hybrid systems by constraint propagation based abstraction refinement. In *Proceedings of the 8th International Workshop on Hybrid Systems: Computation and Control (HSCC'05)*, volume 3414 of *Lecture Notes in Computer Science*, pages 573–589. Springer, 2005.

[RST02] L. Ros, A. Sabater, and F. Thomas. An ellipsoidal calculus based on propagation and fusion. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 32(4):430–442, 2002.

[RW03] M. Rewienski and J. White. A trajectory piecewise-linear approach to model order reduction and fast simulation of nonlinear circuits and micromachined devices. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(2):155–170, 2003.

[SB06] A. B. Singer and P. I. Barton. Bounding the solutions of parameter dependent nonlinear ordinary differential equations. *SIAM Journal on Scientific Computing*, 27:2167–2182, 2006.

[SDI08] S. Sankaranarayanan, T. Dang, and F. Ivancic. Symbolic model checking of hybrid systems using template polyhedra. In *Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'08)*, volume 4963 of *Lecture Notes in Computer Science*, pages 188–202. Springer, 2008.

[SK03] O. Stursberg and B. H. Krogh. Efficient representation and computation of reachable sets for hybrid systems. In *Proceedings of the 6th International Workshop on Hybrid Systems: Computation and Control (HSCC'03)*, volume 2623 of *Lecture Notes in Computer Science*, pages 482–497. Springer, 2003.

[SRKC00] B. Silva, K. Richeson, B. Krogh, and A. Chutinan. Modeling and verification of hybrid dynamical system using checkmate. In *ADPM 2000*. Shaker, 2000.

[SSM04] S. Sankaranarayanan, H. Sipma, and Z. Manna. Constructing invariants for hybrid systems. In *Proceedings of the 7th International Workshop on Hybrid Systems: Computation and Control (HSCC'04)*, volume 2993 of *Lecture Notes in Computer Science*, pages 539–554. Springer, 2004.

[TD13] R. Testylier and T. Dang. Nltoolbox: A library for reachability computation of nonlinear dynamical systems. In *Proceedings of the 11th International Symposium on Automated Technology for Verification and Analysis (ATVA'13)*, volume 8172 of *Lecture Notes in Computer Science*, pages 469–473. Springer, 2013.

[Tiw08] H. R. Tiwary. *Complexity of Some Polyhedral Enumeration Problems*. PhD thesis, Saarland University, 2008.

[Tre13] N. Trefethen. *Approximation Theory and Approximation Practice*. Society for Industrial and Applied Mathematics, 2013.

[VB96] L. Vandenberghe and S. Boyd. Semidefinite programming. *SIAM Review*, 38(1):49–95, 1996.

[WVS06] J. C. Wildenberg, J. A. Vano, and J. C. Sprott. Complex spatiotemporal dynamics in Lotka-Volterra ring systems. *Ecological Complexity*, 3(2):140–147, 2006.

[Zha92] F. Zhao. *Automatic Analysis and Synthesis of Controllers for Dynamical Systems Based on Phase-Space Knowledge.* PhD thesis, Massachusetts Institute of Technology, 1992.

[Zie95] G. M. Ziegler. *Lectures on Polytopes*, volume 152 of *Graduate Texts in Mathematics*. Springer, 1995.

[ZSS$^+$13] Y. Zhang, S. Sankaranarayanan, F. Somenzi, X. Chen, and E. Ábrahám. From statistical model checking to statistical model inference: characterizing the effect of process variations in analog circuits. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD) 2013,*, pages 662–669. IEEE/ACM, 2013.

[ZSS14] Y. Zhang, S. Sankaranarayanan, and F. Somenzi. Sparse statistical model inference for analog circuits under process variations. In *Proceedings of the 19th Asia and South Pacific Design Automation Conference (ASP-DAC'14)*, pages 449–454. IEEE, 2014.

Aachener Informatik-Berichte

This list contains all technical reports published during the past three years. A complete list of reports dating back to 1987 is available from:

`http://aib.informatik.rwth-aachen.de/`

To obtain copies please consult the above URL or send your request to:

Informatik-Bibliothek, RWTH Aachen, Ahornstr. 55, 52056 Aachen,
Email: `biblio@informatik.rwth-aachen.de`

2012-01	Fachgruppe Informatik: Annual Report 2012
2012-02	Thomas Heer: Controlling Development Processes
2012-03	Arne Haber, Jan Oliver Ringert, Bernhard Rumpe: MontiArc - Architectural Modeling of Interactive Distributed and Cyber-Physical Systems
2012-04	Marcus Gelderie: Strategy Machines and their Complexity
2012-05	Thomas Ströder, Fabian Emmes, Jürgen Giesl, Peter Schneider-Kamp, and Carsten Fuhs: Automated Complexity Analysis for Prolog by Term Rewriting
2012-06	Marc Brockschmidt, Richard Musiol, Carsten Otto, Jürgen Giesl: Automated Termination Proofs for Java Programs with Cyclic Data
2012-07	André Egners, Björn Marschollek, and Ulrike Meyer: Hackers in Your Pocket: A Survey of Smartphone Security Across Platforms
2012-08	Hongfei Fu: Computing Game Metrics on Markov Decision Processes
2012-09	Dennis Guck, Tingting Han, Joost-Pieter Katoen, and Martin R. Neuhäußer: Quantitative Timed Analysis of Interactive Markov Chains
2012-10	Uwe Naumann and Johannes Lotz: Algorithmic Differentiation of Numerical Methods: Tangent-Linear and Adjoint Direct Solvers for Systems of Linear Equations
2012-12	Jürgen Giesl, Thomas Ströder, Peter Schneider-Kamp, Fabian Emmes, and Carsten Fuhs: Symbolic Evaluation Graphs and Term Rewriting — A General Methodology for Analyzing Logic Programs
2012-15	Uwe Naumann, Johannes Lotz, Klaus Leppkes, and Markus Towara: Algorithmic Differentiation of Numerical Methods: Tangent-Linear and Adjoint Solvers for Systems of Nonlinear Equations
2012-16	Georg Neugebauer and Ulrike Meyer: SMC-MuSe: A Framework for Secure Multi-Party Computation on MultiSets
2012-17	Viet Yen Nguyen: Trustworthy Spacecraft Design Using Formal Methods
2013-01 *	Fachgruppe Informatik: Annual Report 2013
2013-02	Michael Reke: Modellbasierte Entwicklung automobiler Steuerungssysteme in Klein- und mittelständischen Unternehmen
2013-03	Markus Towara and Uwe Naumann: A Discrete Adjoint Model for OpenFOAM
2013-04	Max Sagebaum, Nicolas R. Gauger, Uwe Naumann, Johannes Lotz, and Klaus Leppkes: Algorithmic Differentiation of a Complex C++ Code with Underlying Libraries

2013-05 Andreas Rausch and Marc Sihling: Software & Systems Engineering Essentials 2013

2013-06 Marc Brockschmidt, Byron Cook, and Carsten Fuhs: Better termination proving through cooperation

2013-07 André Stollenwerk: Ein modellbasiertes Sicherheitskonzept für die extrakorporale Lungenunterstützung

2013-08 Sebastian Junges, Ulrich Loup, Florian Corzilius and Erika brahám: On Gröbner Bases in the Context of Satisfiability-Modulo-Theories Solving over the Real Numbers

2013-10 Joost-Pieter Katoen, Thomas Noll, Thomas Santen, Dirk Seifert, and Hao Wu: Performance Analysis of Computing Servers using Stochastic Petri Nets and Markov Automata

2013-12 Marc Brockschmidt, Fabian Emmes, Stephan Falke, Carsten Fuhs, and Jürgen Giesl: Alternating Runtime and Size Complexity Analysis of Integer Programs

2013-13 Michael Eggert, Roger Häußling, Martin Henze, Lars Hermerschmidt, René Hummen, Daniel Kerpen, Antonio Navarro Pérez, Bernhard Rumpe, Dirk Thißen, and Klaus Wehrle: SensorCloud: Towards the Interdisciplinary Development of a Trustworthy Platform for Globally Interconnected Sensors and Actuators

2013-14 Jörg Brauer: Automatic Abstraction for Bit-Vectors using Decision Procedures

2013-16 Carsten Otto: Java Program Analysis by Symbolic Execution

2013-19 Florian Schmidt, David Orlea, and Klaus Wehrle: Support for error tolerance in the Real-Time Transport Protocol

2013-20 Jacob Palczynski: Time-Continuous Behaviour Comparison Based on Abstract Models

2014-01 * Fachgruppe Informatik: Annual Report 2014

2014-02 Daniel Merschen: Integration und Analyse von Artefakten in der modellbasierten Entwicklung eingebetteter Software

2014-03 Uwe Naumann, Klaus Leppkes, and Johannes Lotz: dco/c++ User Guide

2014-04 Namit Chaturvedi: Languages of Infinite Traces and Deterministic Asynchronous Automata

2014-05 Thomas Ströder, Jürgen Giesl, Marc Brockschmidt, Florian Frohn, Carsten Fuhs, Jera Hensel, and Peter Schneider-Kamp: Automated Termination Analysis for Programs with Pointer Arithmetic

2014-06 Esther Horbert, Germán Martín García, Simone Frintrop, and Bastian Leibe: Sequence Level Salient Object Proposals for Generic Object Detection in Video

2014-07 Niloofar Safiran, Johannes Lotz, and Uwe Naumann: Algorithmic Differentiation of Numerical Methods: Second-Order Tangent and Adjoint Solvers for Systems of Parametrized Nonlinear Equations

2014-08 Christina Jansen, Florian Göbe, and Thomas Noll: Generating Inductive Predicates for Symbolic Execution of Pointer-Manipulating Programs

2014-09 Thomas Ströder and Terrance Swift (Editors): Proceedings of the International Joint Workshop on Implementation of Constraint and Logic Programming Systems and Logic-based Methods in Programming Environments 2014

2014-14 Florian Schmidt, Matteo Ceriotti, Niklas Hauser, and Klaus Wehrle: HotBox: Testing Temperature Effects in Sensor Networks

2014-15 Dominique Gückel: Synthesis of State Space Generators for Model Checking Microcontroller Code

2014-16 Hongfei Fu: Verifying Probabilistic Systems: New Algorithms and Complexity Results

2015-01 * Fachgruppe Informatik: Annual Report 2015

2015-05 Florian Frohn, Jürgen Giesl, Jera Hensel, Cornelius Aschermann, and Thomas Ströder: Inferring Lower Bounds for Runtime Complexity

2015-06 Thomas Ströder and Wolfgang Thomas (Editors): Proceedings of the Young Researchers' Conference "Frontiers of Formal Methods"

* These reports are only available as a printed version.
Please contact `biblio@informatik.rwth-aachen.de` to obtain copies.