

Optimization of Lyapunov Invariants in Analysis and Implementation of Safety-Critical Software Systems

by

Mardavij Roozbehani

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy


at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2008


© Massachusetts Institute of Technology 2008. All rights reserved.

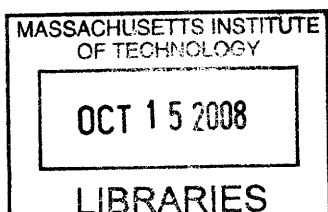
Author.....  Department of Aeronautics and Astronautics
25 July 2008

Certified by.....  Alexandre Megretski
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Certified by..... Eric Feron
Dutton Ducoffe Professor of Aerospace Software Engineering
Thesis Supervisor

Certified by..... Emilio Frazzoli
Associate Professor of Aeronautics and Astronautics
Thesis Supervisor

Accepted by.....  Prof. David L. Darmofal
Associate Department Head
Chair, Department Committee on Graduate Students



ARCHIVES

Optimization of Lyapunov Invariants in Analysis and Implementation of Safety-Critical Software Systems

by

Mardavij Roozbehani

Submitted to the Department of Aeronautics and Astronautics
on 25 July 2008, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

This dissertation contributes to two major research areas in safety-critical software systems, namely, *software analysis*, and *software implementation*. In reference to the software analysis problem, the main contribution of the dissertation is the development of a novel framework, based on Lyapunov invariants and convex optimization, for verification of various safety and performance specifications for software systems. The enabling elements of the framework for software analysis are: (i) dynamical system interpretation and modeling of computer programs, (ii) Lyapunov invariants as behavior certificates for computer programs, and (iii) a computational procedure for finding the Lyapunov invariants.

- (i) The view in this dissertation is that software defines a rule for iterative modification of the operating memory at discrete instances of time. Hence, it can be modeled as a discrete-time dynamical system with the program variables as the state variables, and the operating memory as the state space. Three specific modeling languages are introduced which can represent a broad range of computer programs of interest to the control community. These are: Mixed Integer-Linear Models, Graph Models, and Linear Models with Conditional Switching.
- (ii) Inspired by the concept of Lyapunov functions in stability analysis of nonlinear dynamical systems, Lyapunov invariants are introduced and proposed for analysis of behavioral properties, and verification of various safety and performance specifications for computer programs. In the same spirit as standard Lyapunov functions, a Lyapunov invariant is an appropriately defined function of the state which satisfies a *difference inequality* along the trajectories. It is shown that variations of Lyapunov invariants satisfying certain technical conditions can be formulated for verification of several common specifications. These include but are not limited to: absence of overflow, absence of division-by-zero, termination in finite time, and certain user-specified program assertions.
- (iii) A computational procedure based on convex relaxation techniques and numerical optimization is proposed for finding the Lyapunov invariants that prove the specifications.

The framework is complemented by the introduction of a notion of optimality for the graph models. This notion can be used for constructing efficient graph models that improve the

analysis in a systematic way. It is observed that the application of the framework to (graph models of) programs that are semantically identical but syntactically different does not produce identical results. This suggests that the success or failure of the method is contingent on the choice of the graph model. Based on this observation, the concepts of graph reduction, irreducible graphs, and minimal and maximal realizations of graph models are introduced. Several new theorems that compare the performance of the original graph model of a computer program and its reduced offsprings are presented.

In reference to the software implementation problem for safety-critical systems, the main contribution of the dissertation is the introduction of an algorithm, based on optimization of quadratic Lyapunov functions and semidefinite programming, for computing optimal state space implementations for digital filters. The particular implementation that is considered is a finite word-length implementation on a fixed-point processor with quantization before or after multiplication. The objective is to minimize the effects of finite word-length constraints on performance deviation while respecting the overflow limits. The problem is first formulated as a special case of controller synthesis where the controller has a specific structure, which is known to be a hard non-convex problem in general. It is then shown that this special case can be convexified exactly and the optimal implementation can be computed by solving a semidefinite optimization problem. It is observed that the optimal state space implementation of a digital filter on a machine with finite memory, does not necessarily define the same transfer function as that of an ideal implementation.

Thesis Supervisor: Alexandre Megretski

Title: Professor of Electrical Engineering and Computer Science

Thesis Supervisor: Eric Feron

Title: Dutton Ducoffe Professor of Aerospace Software Engineering

Thesis Supervisor: Emilio Frazzoli

Title: Associate Professor of Aeronautics and Astronautics

To Mitra

Acknowledgements

I would like to take this opportunity to express my deepest appreciations to my advisors Sasha Megretski, Eric Feron, and Emilio Frazzoli. Sasha generously spent countless hours teaching me some of his profound technical expertise and provided me with an invaluable training that I could receive nowhere else. I will be forever grateful for this opportunity. I cannot thank Eric enough for his constant support, advice, and encouragement, and for teaching me how to evaluate and pursue great research ideas. I am grateful for all the important skills that I acquired through him during these years. I am truly thankful to Emilio for his invaluable support and guidance over the past year.

I am grateful to Prof. John Deyst, and Prof. Hamsa Balakrishnan for kindly serving on my thesis committee and for providing valuable feedback which improved this work in many ways. I would like to thank Prof. Pablo Parrilo for many insightful discussions and for his generosity with his time. I would also like to thank Prof. Munther Dahleh and Prof. Sanjoy Mitter for their constructive feedback and encouragement, particularly about the chapter on implementation (chapter 6).

Many thanks to Mehrdad Pakmehr at Georgia Tech who generously read the entire thesis and his comments and questions helped me improve the clarity of the presentation. I thank my friends and colleagues at MIT who made these years memorable: Amirali Ahmadi, Ola Ayaso, Amit Bhatia, Animesh Chakravarthy, Peyman Faratin, Ather Gattami, Lisa Gaumond, Brian Haines, Sertac Karaman, Patrick Kreidel, Jerome Le Ny, Rodin Lyasof, Mike Rinehart, Philip Root, Navid Sabbaghi, Keith Santarelli, Sridevi Sarma, Christian Schunk, Tom Schouwenaars, Danielle Tarraf, and Peng Yu.

A special word of thanks goes to the staff members of the Department of Aeronautics and Astronautics and of LIDS: Marie Stuppard, Lisa Gaumond, Brian Jones, Doris Inslee, Angela Olsen, and Jennifer Donovan who were always supportive and helpful beyond expectations.

I am grateful to my mother Manijeh and my father Houshang for their unconditional love, dedication, and support. I owe everything I have ever achieved to them. I sincerely thank my sisters Mojgan and Marjan, and my brother Hajir who have always been there for me. Lastly, I would like to wholeheartedly thank my wife, Mitra for her loving devotion, and her mother, Mahnaz for her enduring love and support.

Funding Acknowledgement

Funding for this research has been provided in parts by the National Science Foundation, Awards NSF-0451865-CNS/EHS, NSF-0715025-CNS/EHS, and NSF-0615025-CSR/EHS - Certification of Safety-Critical Software.

Contents

1	Introduction	10
1.1	Motivation	10
1.2	Literature Review	11
1.2.1	Formal Methods	11
1.2.2	System Theoretic Methods	14
1.3	Statement of Contributions and Thesis Outline	15
1.3.1	Software Analysis	15
1.3.2	Software Implementation	18
1.4	Notations	19
2	Dynamical System Interpretation and Modeling of Computer Programs	20
2.1	Generic Representations	21
2.1.1	Concrete Representation of Computer Programs	21
2.1.2	Abstract Representation of Computer Programs	27
2.2	Specific Models of Computer Programs	36
2.2.1	Mixed-Integer Linear Models	37
2.2.2	Graph models	45
2.3	Specifications	55
2.3.1	Safety	55
2.3.2	Program Termination in Finite Time	59
2.4	The Implications of Abstractions	59
2.5	Summary	60

3	Lyapunov Invariants as Behavior Certificates	62
3.1	Preliminaries	62
3.1.1	Lyapunov Invariants for MILMs	65
3.1.2	Lyapunov Invariants for Graph Models	65
3.2	Behavior Certificates	68
3.2.1	Liveness	68
3.2.2	Safety	72
3.3	Summary	81
4	Computation of Lyapunov Invariants	82
4.1	Preliminaries	83
4.1.1	Convex Parameterization of Lyapunov Invariants	83
4.1.2	Convex Relaxation Techniques	85
4.2	Optimization of Lyapunov Invariants for Mixed-Integer Linear Models	89
4.2.1	Quadratic Invariants	89
4.2.2	Linear Invariants	92
4.3	Optimization of Lyapunov Invariants for Graph Models	96
4.3.1	Node-wise Polynomial Invariants	97
4.3.2	Node-wise Quadratic Invariants for Linear Graphs	98
4.4	Case Study	100
4.5	Summary	103
4.6	Appendix	104
5	Optimal Graph Models	106
5.1	Motivation	106
5.2	Graph Reduction and Irreducible Graph Models	110
5.3	Comparison of Irreducible Graph Models	117
5.3.1	Comparison of Maximal and Minimal Realizations of K1 Graphs	120
5.3.2	Comparison of Maximal and Minimal Realizations of Kn Graphs	125
5.4	Summary	135

6	Optimal Implementation of Linear Time-Invariant Systems for Safety-Critical Applications	136
6.1	Introduction	137
6.2	Problem Formulation	142
6.2.1	Linearization via signal + noise model	143
6.2.2	Structured Linear Controller Synthesis Formulation	145
6.3	Optimal State Space Implementation via \mathcal{H}^∞ Optimization	147
6.3.1	Nonconvex Matrix Inequality Criterion	147
6.3.2	Convexification of the Matrix Inequality Criterion	149
6.4	Numerical Simulations	152
6.5	Summary	154
6.6	Appendix	156
7	Conclusions and Future work	161
7.1	Summary	161
7.2	Future Work	163

List of Figures

2-1	Conceptual diagram of evolution of the trajectories of a computer program and its abstraction.	28
2-2	Graph model of a code fragment (Program 2-5) with time varying arc labels. The transition labels are shown in boxes and the passport labels are in brackets. For simplicity, only the non-identity transition labels are shown.	52
3-1	A graph model. There is an invariant set X_i assigned to each node. A transition label T_{ji} and a passport label Π_{ji} is assigned to each arc (i, j) from node i to node j	67
3-2	The graph model of an abstraction of Program 3-1.	80
4-1	The graph of Program 4-4.	102
5-1	Graph Models of Programs \mathcal{P}_1 (left) and \mathcal{P}_2 (right).	109
5-2	Minimal (left) and Maximal (right) realizations for program \mathcal{P}_1	112
5-3	A minimal realization for program \mathcal{P}_2	113
5-4	A Maximal realization for program \mathcal{P}_2	114
5-5	With proper labeling of this graph model, a counterexample can be constructed to prove that an irreducible realization of higher order does not always outperform an irreducible realization of lower order.	118
5-6	For this graph, it is possible to choose the state transition operators A_i, B_i, C_i, D_i such that the minimal realization outperforms the maximal realization.	119
5-7	A K3 graph with $\mathcal{N}^* = \{2, 4, 6\}$. The minimal order is 3, and the maximal order is 6.	126

6-1	The error system	142
6-2	The Quantizer $\Gamma(\cdot)$: Two's complement rounding with saturation	143
6-3	The error system corresponding to a particular finite-state implementation with quantization after multiplication $x_c[k + 1] = \Gamma(A_c x_c[k] + B_c w[k])$. Inside the dashed box is the quantizer. Given $H(z)$, the objective is to find (A_c, B_c, C_c, D_c) such that the error $\ e\ $ is small for some appropriately defined norm.	145
6-4	Numerical simulations: comparison of our results with [83].	153
6-5	Numerical simulations: comparison of our results with [83].	155

Chapter 1

Introduction

1.1 Motivation

Software in safety-critical systems is designed to implement intelligent algorithms that control the interaction of physical devices with their environments, often through sophisticated feedback laws. Examples of such systems can be found in aerospace, automotive, and medical applications, as well as many other real-time embedded control systems. Failure of these safety-critical systems often leads to loss of human life or a huge loss in capital. To guarantee safety, functionality, and performance of these systems, correctness and reliability of the embedded software must be established.

While real-time safety-critical software must satisfy various resource allocation, timing, scheduling, fault tolerance, and performance constraints, the very least to require is that the software must execute safely, free of run-time errors. For a comprehensive discussion of the theoretical and practical issues that arise in analysis, design and implementation of real-time software systems see for instance [49, 87, 69, 42], and the references therein. One of the objectives of this document is to develop a systematic framework for verification of certain safety and liveness properties of computer programs to rule out run-time errors and guarantee termination in finite time. Although this was the motivation, the framework is applicable to verification of a broader range of specifications concerning functionality and performance of numerical software systems. More details will be provided later in this chapter.

According to Boeing Co. and Honeywell Inc., software development accounts for sixty to

eighty percent of the effort spent on the development of complex control systems, with much of the effort expended on validation and verification of the software after or during its development [42]. While extensive simulation and testing account for a large portion of this effort, they can only help in detecting potential programming or design errors, but they cannot prove safety or performance properties of these complex systems. In safety-critical applications, it is necessary to generate and document mathematical proofs of safety and performance of the software. Formal verification methods aim at generating such proofs by reasoning on mathematical models of computer programs. Verification by reasoning on a mathematical model of software (or hardware) is sometimes referred to as model-based verification [68]. An extensive collection of model-based verification methods developed by computer scientists, as well as several new results are presented in [75, 72, 2]. The approach adapted in this document falls under the category of model-based verification methods.

1.2 Literature Review

1.2.1 Formal Methods

In the computer science literature, formal verification methods are described as techniques for proving (or disproving) that a mathematical model of the software satisfies a given *specification*. What is meant by *specification* is a set of behavioral properties that need to be proven to guarantee safety, good performance, or functionality. The specifications might be defined informally, though they must be expressed in mathematical terms before they can be verified formally. The underlying mathematical model is often a discrete transition system which can be deterministic or non-deterministic. The choice of the model, however, is usually not an independent process and depends on the specifications, as well as the available proof methods. Hence, iterative refinement of the model and the proof technique may become necessary to successfully prove the required specifications. In a verification process, this often entails going from coarse abstractions to refined models that emulate the behavior of the software more accurately. On the other hand, the complexity of the proof method grows with the levels of details in both the model and the specifications. Therefore, in practice, a compromise must be reached between the specifications and the complexity of the verification method. The tradeoff between

complexity of the proof methods and level of detail in the specifications/mathematical model evidently draws the contrast between two well-known formal methods for software verification: *model checking* and *abstract interpretation*.

Model Checking

Formal verification methods have gone under significant development in the past few decades. Pioneered by Clarke, Emerson and Sifakis, *model checking* [21, 23] emerged as a means to deal with the problems of specification, development and verification of sequential or concurrent finite-state systems. The properties are expressed in terms of temporal logic formulae and the system is modeled as a state transition system. Symbolic algorithms are then used to perform an exhaustive exploration of all possible states and check whether or not the specifications satisfy the properties. Model checking has proven to be a powerful technique for verification of circuits [22], security and communication protocols [62, 70], and stochastic processes [24, 8]. Several software tools such as SPIN [98], BLAST [95], and NuSMV [96] have been developed and widely used. For software systems, when applicable, model checking techniques result in strong statements about the behavior of the system. The trade-off, however, is that verification of strong properties and increased accuracy is achieved at the cost of increased computational requirements and limited scalability to large systems. The introduction of Binary Decision Diagrams (BDDs) [20], which are efficient data structures for representing boolean functions has improved the scalability of these techniques and model checking of systems with very large number of states is now possible. Nevertheless, when the number of possible states is astronomical, such as in programs with non-integer variables, or when the number of possible states is infinite, such as when the state space is continuous, model checking in its pure form is not directly applicable. In such cases, combinations of various abstraction techniques and model checking have been considered for verification [4, 38, 30, 89]; scalability, however, remains a challenge.

Alternative formal methods can be found in the computer science literature mostly under *deductive verification* [60, 61], *type inference* [76, 65], *data flow analysis* [43], and *abstract interpretation* [26, 27]. Despite their differences, these methods share extensive similarities. In particular, a notion of program abstraction and symbolic program execution or constraint

propagation is present in all of them. A comparison of advantages and disadvantages of these methods, as well as a discussion of the challenges that they each face can be found in [28], and [72]. Here, we review *abstract interpretation* as it appears to have better scalability properties and has been used in verification of safety-critical systems [14]. More detailed comparisons with our work will be provided in the upcoming chapters, as relevant results are presented.

Abstract Interpretation

Initiated by the work of P. Cousot and R. Cousot in the 1970s [26, 27], *abstract interpretation* was developed as a theory for formal approximation of the operational semantics of computer programs in a way that would facilitate systematic reasoning about the behavior of programs. The operational semantics of a computer program refers to a mathematical characterization of all possible sequences that can be generated by the program. In verification by abstract interpretation, first, an abstract model is built by replacing the domain of concrete operational semantics by a domain of abstract operational semantics defined over semilattices. Construction of abstract models has two components: abstraction of domains (sets of numbers), and abstraction of functions. The domain abstractions are typically in the form of sign, interval, polyhedral, and congruence abstractions of sets of data, or a combination of these. The function abstractions are highly influenced by the domain abstractions. For instance, for a monotonic function $f : X \mapsto X$, its abstraction $f^\# : X^\# \mapsto X^\#$ is defined by $f^\#(x) := (\alpha \circ f \circ \gamma)(x)$, where $\alpha : X \mapsto X^\#$ is an abstraction map, and $\gamma : X^\# \mapsto X$ is a concretization map. If $\alpha \circ f \circ \gamma$ is not easily computable, which is often the case, further simplification becomes necessary.

In verification by abstract interpretation, the program analyzer reads the program text and the specifications and generates a system of fixpoint equations and constraints. Abstraction of the program semantics and the specifications, along with symbolic forward and/or backward executions of the abstract model are the enabling components in constructing the system of fixpoint equations and constraints. The solution to the constrained system of fixpoint equations results in an inductive assertion which is invariant under all possible executions. The program invariants are then used by the analyzer for checking the specifications.

A critical phase in this process is solving the constrained system of fixpoint equations. An iterative equation solving procedure is often used at this phase. However, finite convergence

of the iterates can be guaranteed only for very simple abstractions, e.g. sign and simple congruence abstractions. In practice, to guarantee convergence of the iterates, narrowing (outer approximation) operators are used to estimate the solution in a finite number of steps, followed by widening (inner approximation) to improve the estimate [28]. This compromise, often causes the method to generate weak invariants, resulting in considerable conservatism in analysis [25]. Nevertheless, these methods have shown to be practical for verification of limited properties of real-time, embedded software of commercial aircraft [14, 94].

1.2.2 System Theoretic Methods

While software analysis has been the subject of a great volume of literature in computer science, little has appeared on this subject in the systems and control literature. Much of the relevant results in systems and control literature can be found in the field of hybrid systems [5]. Many of the proposed techniques for verification of hybrid systems are based on explicit computation of the reachable sets, either exactly or approximately. These include but are not limited to techniques based on quantifier elimination [54, 88], Hamilton Jacobi equations [67], ellipsoidal calculus [51], and mathematical programming [12, 93, 10]. Alternative approaches aim at establishing properties of hybrid systems by the combined use of bisimulation mechanisms and Lyapunov techniques. Bisimulations (approximate bisimulations) of a Hybrid system are finite-state quotients whose reachability properties are equivalent to (over-approximate) those of the original infinite-state system. A so-called bisimulation function is a function bounding the distance between the observations of two systems and is non-increasing under their parallel evolutions. Approximate bisimulation relations can therefore, be characterized by the level sets of a (bisimulation) function which satisfies Lyapunov-like differential inequalities [35]. The bisimulation relations can then be used for constructing a finite-state approximation of the hybrid system which can be subsequently verified via model checking techniques [36, 37, 53, 52, 89, 4]. This approach has particularly had success in reachability analysis of timed automata and linear hybrid automata.

In principle, many of the methods developed in system and control theory for systems governed by differential equations, particularly Lyapunov theoretic techniques, have been shown to be applicable to hybrid systems. Examples include optimal control theory for hybrid systems

[58, 44, 18], computation of Lyapunov functions for hybrid systems [17, 46, 47], reachability analysis of hybrid systems using bisimulations [53, 36], or verification of hybrid systems using barrier certificates [78, 77]. While Lyapunov functions and similar concepts have been used in verification of stability and/or temporal properties of system level descriptions of hybrid systems, to the best of our knowledge, this dissertation is the first document to present a systematic framework based on Lyapunov functions and convex optimization for verification of a broad range of specifications for computer programs. The novelty of our approach is in the transfer of Lyapunov functions and the associated computational techniques from control systems analysis to software analysis. As we will see later in the document, our framework applies to a broad class of numerical programs and is not limited to applications in hybrid systems or safety-critical control systems. However, this appears to be an area where the framework is readily applicable. The rationale is that since the embedded control software essentially implements a control law that is designed via system theoretic tools, such tools are most viable for verification at the implementation level.

1.3 Statement of Contributions and Thesis Outline

In this dissertation we consider two important problems concerning safety-critical software systems: software analysis and software implementation.

1.3.1 Software Analysis

In reference to the software analysis problem, the main contribution of the dissertation is the development of a systematic framework based on Lyapunov invariants and convex optimization for verification of various safety and performance specifications. Our framework, however, is not restricted to software applications in safety-critical systems. It is shown by means of a myriad of examples throughout the thesis, that many numerical computer programs that may not necessarily appear in safety-critical applications can be modeled and verified in this framework. The enabling elements of the framework for software analysis are dynamical system interpretation and modeling of computer programs, Lyapunov invariants as certificates for the behavior of the programs, and a computational procedure for finding the Lyapunov invariants.

The computational procedure consists of linear parametrization of the search space, convex relaxation techniques, and convex optimization.

- **Dynamical system interpretation and modeling of computer programs:** This is the topic of Chapter 2. The view in this document is that software defines a rule for iterative modification of the operating memory at discrete instances of time. Hence, it can be modeled as a discrete-time dynamical system with the program variables as the state variables, and the operating memory as the state space. We introduce generic dynamical system representations of computer programs, which can be concrete or abstract. Beyond the generic representations, we also introduce specific modeling languages. These include:
 - Mixed-Integer Linear Models.
 - Graph Models.
 - Linear Models with Conditional Switching (LMwCS).

While the generic dynamical system representations are suitable for establishing and presenting fundamental results on analysis of software via Lyapunov invariants, the specific modeling languages are more suitable for explicit computation of the Lyapunov invariants in an optimization-based framework. It is shown through several examples throughout the thesis that these models can represent a broad range of computer programs of interest to the control community, e.g. safety-critical control software of embedded systems.

- **Lyapunov invariants as certificates for the behavior of programs:** This is the topic of Chapter 3. Inspired by the concept of Lyapunov¹ functions in stability analysis of nonlinear dynamical systems, we propose using Lyapunov invariants for analysis of behavioral properties and verification of safety and performance specifications of computer programs. In the same spirit as standard Lyapunov functions, a Lyapunov invariant is an appropriately-defined, real-valued function of the state (the program variables) which satisfies a *difference inequality* along the execution trace of a computer program. Hence, depending on the difference inequality that must be satisfied, a Lyapunov invariant may

¹Named after the Russian mathematician Aleksandr Mikhailovich Lyapunov.

or may not monotonically decrease along the execution trace of a computer program. However, at each increment of time, the value of a Lyapunov invariant cannot increase by more than a constant multiple of its current value. This notion is formalized and presented in mathematical terms in Chapter 3. We show that different variations of Lyapunov invariants satisfying certain technical conditions can be formulated for verification of several safety and performance specifications of computer programs. A specification can be verified² via our framework if it can be interpreted and expressed in one of the following terms:

- Safety: The property that a certain subset of the state space will never be reached.
- Liveness: The property that all of the trajectories will enter a certain subset of the state space in finite-time.

We will show in Chapter 3, that the framework can be conveniently used for (but is not restricted to) ruling out the following unsafe situations in computer programs:

- Infinite loops.
- Variable overflow.
- Division-by-zero.
- Out-of-bounds array indexing.
- Taking the square root (even root), or real logarithm of a negative number.

Additional properties that do not necessarily lead to run time errors but can be verified in this framework are:

- Verification of user-specified program assertions.
- Verification of user-specified program invariants.

Other verification problems such as pointer tracking, and race conditions do not fall within the scope of this manuscript.

²We would like to stress that the criteria that we present are in general sufficient and not necessary. By “a specification can be verified” we mean that “sufficient criteria for the specification to hold can be formulated.”

- **Computational procedure:** This is the topic of Chapter 4. Similar to the difficulties in analysis of nonlinear systems via Lyapunov functions, the main challenge in analysis of computer programs via Lyapunov invariants is finding them. The procedure that we use for finding the Lyapunov invariants is standard and consists of the following steps:
 - 1. Restricting the search space to a linear subspace.
 - 2. Using convex relaxation techniques such as the \mathcal{S} -Procedure, or sum-of-squares relaxation to formulate the search problem as a convex optimization problem.
 - 3. Using convex optimization tools to numerically compute the behavior certificates. Depending on the mathematical model and the convex relaxation method, the search problem will be formulated as a linear program [13], semidefinite program [16, 91], or a sum-of-squares program [73]. This is the final stage of the verification process. If the convex optimization problem has a feasible solution, a certificate for the specification has been found, otherwise, the result is inconclusive.

- **Optimal Graph Models:** This is the topic of Chapter 5. The framework is complemented by the introduction of a notion of optimality for the graph models. This notion can be used for constructing efficient graph models that improve the analysis in a systematic way. It is observed that the application of the framework to (graph models of) programs that are semantically identical but syntactically different does not produce identical results. This suggests that the success or failure of the method is contingent on the choice of the graph model. Based on this observation, the concepts of graph reduction, irreducible graphs, and minimal and maximal realizations of graph models are introduced. Several new theorems that compare the performance of the original graph model of a computer program and its reduced offsprings are presented.

1.3.2 Software Implementation

Software implementation is discussed in Chapter 6. In reference to the software implementation problem for safety-critical systems, the main contribution of the dissertation is the introduction of an algorithm, based on optimization of quadratic Lyapunov functions and semidefinite programming, for computation of optimal state space implementations for digital filters and linear

controllers. The particular implementation that is considered is a finite word-length implementation on a fixed-point processor with quantization after multiplication. The objective is to minimize the effects of finite word-length constraints on performance deviation, while respecting the overflow limits. The problem is first formulated as a special case of the linear controller synthesis problem where the controller has a specific structure. This problem is known to be a hard non-convex problem in general. It is then shown that this special case can be convexified exactly, and the optimal implementation can be computed by solving a semidefinite optimization problem. It is observed that the optimal state space implementation of a digital filter on a machine with finite memory does not necessarily define the same transfer function as that of an ideal implementation.

1.4 Notations

In this document, \mathbb{R} denotes the set of real numbers, \mathbb{R}_+ the set of positive real numbers, $\overline{\mathbb{R}}_+$ the set of nonnegative real numbers, and $\mathbb{R}^{n \times m}$ the set of $n \times m$ real matrices. Similarly, \mathbb{Z} denotes the set of integers, \mathbb{Z}_+ (or \mathbb{N}) the set of positive integers and $\overline{\mathbb{Z}}_+$ the set of nonnegative integers: $\mathbb{N} \cup \{0\}$. The notation $\mathbb{Z}(n, m)$ is used to denote the set of integers: $\{n, n+1, \dots, m\}$. The $n \times n$ Identity matrix is denoted by I_n and the $n \times n$ Zero matrix is denoted by 0_n . The transposed of a real matrix $P \in \mathbb{R}^{n \times m}$ is denoted by P^T , and for a square matrix $Q \in \mathbb{R}^{n \times n}$, we use $\text{He}(Q)$ to denote $Q + Q^T$, and $\text{Trace}(Q)$ to denote the sum of the diagonal elements of Q . The set of all real symmetric $n \times n$ matrices is denoted by \mathbb{S}^n , and the subset of \mathbb{S}^n consisting of all real diagonal matrices of size n is denoted by \mathbb{D}^n . For $P \in \mathbb{S}^n$, $P \succ 0$ means that P is a positive definite matrix and $P \succeq 0$ means that P is a positive semidefinite matrix. A directed graph with a set of nodes \mathcal{N} and set of arcs \mathcal{E} is denoted by $G(\mathcal{N}, \mathcal{E})$. The set of incoming nodes of node $i \in \mathcal{N}$ is denoted by $\mathcal{I}(i)$ and the set of outgoing nodes by $\mathcal{O}(i)$. For a subset of nodes $\underline{\mathcal{N}} \subset \mathcal{N}$, the set $\bigcup_{i \in \underline{\mathcal{N}}} \mathcal{I}(i)$ is denoted by $\mathcal{I}(\underline{\mathcal{N}})$. The set $\mathcal{O}(\underline{\mathcal{N}})$ is defined analogously. A simple cycle of length m on a directed graph $G(\mathcal{N}, \mathcal{E})$ is denoted by \mathcal{C}_m and sometimes by $\mathcal{C}_m \in G$. the subscript m is dropped whenever the length of the cycle is immaterial. The initial or start node on a graph $G(\mathcal{N}, \mathcal{E})$ is denoted by \emptyset and the terminal node by ∞ . For a vector $v \in \mathbb{R}^n$ and $q \in \mathbb{Z}_+$, the q norm is denoted by $\|v\|_q$ and is defined as $\|v\|_q := \left(\sum_{i=1}^n |v_i|^q \right)^{\frac{1}{q}}$. The infinity norm of a vector $v \in \mathbb{R}^n$ is defined as $\|v\|_\infty := \max_i |v_i|$.

Chapter 2

Dynamical System Interpretation and Modeling of Computer Programs

In this chapter, we develop the first component of our framework for analysis of software systems, namely, dynamical system interpretation and modeling. We interpret computer programs as discrete-time dynamical systems and introduce generic dynamical system representations which formalize this interpretation. We also introduce specific modeling languages as special cases of the generic representations¹. These include *Mixed-Integer Linear Models (MILM)*, *Graph Models*, and *Linear Models with Conditional Switching (LMwCS)*. The generic representations will be used throughout the document, particularly in Chapter 3, to establish fundamental results on analysis of computer programs via Lyapunov invariants. These results are independent of the specific choice of a modeling language. The specific modeling languages are used in the document (cf. Chapter 4) for computation of the Lyapunov invariants in a systematic framework.

The models, whether generic or specific, can be *concrete* or *abstract*. Intuitively, a concrete model is an accurate model of the behavior of a program at the implementation level; while an

¹In this document, the terms *representation* and *model* have identical meanings and can be used interchangeably.

abstract model is an over-approximation of a concrete model in the sense that every trajectory of a concrete model is also a trajectory of the corresponding abstract model. The rationale for exploiting abstract models is clear: we would like to perform analysis on models that formally carry their properties to the actual programs, yet are easier to analyze than the concrete models. We will also discuss some minor technical conditions which must hold for an abstract model to remain faithful to the actual program; meaning that the behavioral properties of the abstract model must imply those of the concrete model.

2.1 Generic Representations

2.1.1 Concrete Representation of Computer Programs

A computer program can be viewed as a rule for iterative modification of the operating memory, possibly in response to real-time inputs. Since computers are inherently constrained with finite memory, computer programs can be accurately modeled as finite-state machines with inputs drawn from a finite alphabet source. In particular, we will consider generic models defined by a finite state space set X with selected subsets $X_0 \subseteq X$ of initial states and $X_\infty \subset X$ of terminal states, and by a set-valued function $f : X \mapsto 2^X$, such that $f(x) \subseteq X_\infty, \forall x \in X_\infty$.

Definition 2.1 *The dynamical system $\mathcal{S}(X, f, X_0, X_\infty)$ is a concrete representation of a computer program \mathcal{P} , if there exists a one-to-one map between the set of all sequences that can be generated by \mathcal{P} , and the set of all sequences $\mathcal{X} := (x(0), x(1), \dots, x(t), \dots)$ of elements from X , satisfying*

$$x(0) \in X_0 \subseteq X, \quad x(t+1) \in f(x(t)) \quad \forall t \in \mathbb{Z}_+, \quad (2.1)$$

where

$$f : X \mapsto 2^X, \text{ s.t. } f(x) \subseteq X_\infty, \forall x \in X_\infty \subset X.$$

Note that the uncertainty in the definition of $x(0)$ allows for dependence of the program on different initial conditions, and the uncertainty in the definition of $x(t+1)$ represents dependence on different parameters as well as the program's ability to respond to real-time inputs. From a dynamical systems perspective, analysis of software is defined as the task of verifying certain properties of system (2.1). This is the view adopted in this document.

Remark 2.1 Throughout this document we use the terms “trace” and “trajectory” of a computer program \mathcal{P} interchangeably to refer to a sequence $\mathcal{X} \in \mathcal{P}$, which is understood in the same sense as Definition 2.1. Also, we do not differentiate between a computer program \mathcal{P} and its concrete dynamical system representation $\mathcal{S}(X, f, X_0, X_\infty)$.

Example 2.1 Integer Division²: Consider the following program written in the standard C Language. Its functionality is to compute the result of the integer division of the value of dd (dividend) by dr (divisor). The quotient is returned in q and the remainder is stored in r .

```

int IntegerDivision ( int dd,int dr )
{int q = {0}; int r = {dd};
while (r >= dr)
{ q = q + 1;
  r = r - dr; }
return q; }
```

Program 2-1: The Integer Division Program

Denote by $\underline{\mathbb{Z}}$ the subset of integers that can be represented by 16 bits: $\underline{\mathbb{Z}} = \mathbb{Z} \cap [-32768, 32767]$. The state variables of this program are dd , dr , q , and r , and they are all elements of $\underline{\mathbb{Z}}$. A concrete representation of this program is defined via the following elements:

$$X = \underline{\mathbb{Z}}^4, X_0 = \{(dd, dr, q, r) \in X \mid q = 0, r = dd\}$$

$$X_\infty = \{(dd, dr, q, r) \in X \mid r < dr\}$$

$$f : (dd, dr, q, r) \mapsto \begin{cases} (dd, dr, q + 1, r - dr), & (dd, dr, q, r) \in X \setminus X_\infty \\ (dd, dr, q, r), & (dd, dr, q, r) \in X_\infty \end{cases}$$

For instance, the sequence \mathcal{X} is an element of the program *IntegerDivision*, where:

$$\mathcal{X} := \left(\left(\begin{bmatrix} 10 \\ 3 \\ 0 \\ 10 \end{bmatrix}, \begin{bmatrix} 10 \\ 3 \\ 1 \\ 7 \end{bmatrix}, \begin{bmatrix} 10 \\ 3 \\ 2 \\ 4 \end{bmatrix}, \begin{bmatrix} 10 \\ 3 \\ 3 \\ 1 \end{bmatrix}, \begin{bmatrix} 10 \\ 3 \\ 3 \\ 1 \end{bmatrix}, \begin{bmatrix} 10 \\ 3 \\ 3 \\ 1 \end{bmatrix}, \dots \right).$$

²Example adopted from [75]

In this example, f is deterministic and is not set-valued. Note that this program is correct only if the values of dd and dr are positive. If $dd \geq 0$, and $dr \leq 0$, then the program never exits the “while” loop and the value of q keeps increasing, potentially leading to an overflow.

An alternative approach to constructing a dynamical system model of Program 2-1 is to treat the input variables (dd and dr), which remain constant in the course of an execution, as symbolic parameters. The result is a model with the following elements:

$$X = \mathbb{Z}^2, X_0 = \{(q, r) \in X \mid q = 0, r = dd\}$$

$$X_\infty = \{(q, r) \in X \mid r < dr\}$$

$$f : (q, r) \mapsto \begin{cases} (q + 1, r - dr), & (q, r) \in X \setminus X_\infty \\ (q, r), & (q, r) \in X_\infty \end{cases}$$

In this case, X_0 and X_∞ are parametric subsets of X , and f is also a parametric (not set-valued) function. We will come back to this issue and compare these modeling choices in the upcoming chapters when we introduce Lyapunov invariants as behavior certificates for computer programs. For the time being, we just mention that in the latter case, one has to resort to parameter-dependent Lyapunov invariants for proving behavioral properties of Program 2-1. It is important to also mention that for the purpose of verification via the framework that is developed in this document the two models are practically equivalent and neither one presents particular advantages or disadvantages from a feasibility or computational cost of analysis point of view.

In Example 2.1, the choice of the state space as $X = \mathbb{Z}^4$ as opposed to $X = \mathbb{Z}^4$ is not free of subtleties. Strictly speaking, when we define $X = \mathbb{Z}^4$ we must also prove that the program variables do not assume any values outside of \mathbb{Z}^4 . If the operations are done in modulo arithmetic, then the result of an overflow in \mathbb{Z}^4 (a variable exceeding 32767 or dropping off below -32768) is a rollover to the same set \mathbb{Z}^4 . Hence, the choice is correct. However, this complicates the definition of the update rule, and an exception must be added in the definition of $f(\cdot)$ to reflect these possibilities. Furthermore, if a rollover occurs, extreme deviations from the desired trajectories will follow and the program will return erroneous results. An alternative is to assume that the variables do remain within the interval $[-32768, 32767]$, and the event

that a variable leaves this interval is characterized by an overflow and the program terminates with a runtime error. If this can be established, then the choice of the state space as $X = \underline{\mathbb{Z}}^4$ is also justified. A third alternative would be to define $X = \mathbb{Z}^4$, which removes the minor technicality associated with the definition of the state space. However, over the set $\mathbb{Z}^4 \setminus \underline{\mathbb{Z}}^4$, f is undefined, which requires us again to prove that the variables do not leave $\underline{\mathbb{Z}}^4$, and characterize the event that the variables leave $\underline{\mathbb{Z}}^4$ (the safe subset of the state space) as an unsafe event which leads to a runtime error. As it can be observed the latter two approaches are practically equivalent. In this document, whenever we define a state space set for a computer program, it is with the understanding that it is either proven or assumed that the definition is correct, in the sense that the variables cannot leave the state space. The event that the variables leave the state space is then considered an unsafe event, leading to a runtime error.

In a concrete representation, the elements of the state space X belong to a certain finite subset of the rational numbers, that is, rational numbers that can be represented by a fixed number of bits in a specific arithmetic framework. For instance, on a 16-bit processor, these subsets may consist of unsigned integers between -2^{15} and $2^{15} - 1$, or all the rational numbers that can be represented with 16 bits in fixed-point or floating point arithmetic. Naturally, the same is true for the subsets X_0 and X_∞ . When the elements of X are non-integers, due to the quantization effects, the set-valued map f often defines very complicated dependencies between the elements of X , even for simple programs involving only elementary arithmetic operations. We present an example.

Example 2.2 Square Root. *Consider the following program written in the standard C Language. Its functionality is to compute the square root of a bounded positive number y up to a predefined precision e . This value is stored in the variable x and returned. In addition, the number of iterations for this procedure to be completed is computed and stored in the integer variable *Counter*. The program will continue to improve its current estimate of the square root of y only until this value is needed. It may be the case that the square root of y is no longer needed because a parallel processor has already computed it, or the feedback control law has changed, or an external process has determined that the current value of y has become obsolete and the program *ComputeSqrt()* must be recalled to compute the square root of the new estimate of y . At each iteration, the program will determine if the square root of y is still needed by checking the*

boolean variable *NeedSqrtY* which is updated in real-time. This real time input is accessed via a pointer variable that points to the memory address of *RealTimeInput*. If the boolean value at the memory address of *RealTimeInput* and subsequently *NeedSqrtY* becomes *False*, then the program terminates to avoid expending the resources unnecessarily.³

```

double ComputeSqrt ( double y )
//y in the interval [1e - 4, 1e4]
{  double x = {1}; double z = {1};
    const double e = {0.0001};
    int Counter = {0};
    bool *PtrToInput = &RealTimeInput;
    bool NeedSqrtY = *PtrToInput;
    while (fabs (z) >= e && NeedSqrtY)
    {   x = 0.5 * (x + y/x);
        z = x * x - y;
        Counter = Counter + 1;
        NeedSqrtY = *PtrToInput; }
    return x; }

```

Program 2-2: Computation of the Square root

Here, we present one possible way to construct a concrete dynamical system representation of this program. Denote by \mathbb{F} , the subset of rational numbers that can be represented in double precision format on the corresponding processor. Let \mathbb{Z} denote the set of integers and \mathbb{B} the set of boolean variables $\{\text{True}, \text{False}\}$. The state variables of this program are $(x, y, z) \in \mathbb{F}^3$, $\text{Counter} \in \mathbb{Z}$, and $\text{NeedSqrtY} \in \mathbb{B}$. We can define a binary variable $v \in \{-1, 1\}$ to represent $\text{NeedSqrtY} \in \mathbb{B}$, and rename the integer variable $\text{Counter} \in \mathbb{Z}$ as $c \in \mathbb{Z}$. The state space can thus, be defined as: $X := \mathbb{F}^3 \times \mathbb{Z} \times \{-1, 1\}$. The set of initial states $X_0 \subset X$ is defined as: $X_0 :=$

³This program is constructed for educational purposes and is meant to represent several real-life scenarios in one small academic example. In practice, such programs would not necessarily include all the features of *ComputeSqrt* as presented above. For instance, there is probably little incentive in keeping track of the number of iterations in this particular case.

$\{(1, y, 1) \mid y \in \mathbb{F} \cap [10^{-4}, 10^4]\} \times \{0\} \times \{-1, 1\}$. The set of terminal states $X_\infty \subset X$ is given by $X_\infty := X_{1\infty} \cup X_{2\infty}$ where $X_{1\infty} := \mathbb{F}^3 \times \mathbb{Z} \times \{-1\}$ and $X_{2\infty} := \{(x, y, z) \in \mathbb{F}^3 \mid |z| < 10^{-4}\} \times \mathbb{Z} \times \{-1, 1\}$. Over the set X_∞ , the map f is simply defined as the identity map, and over the set $X \setminus X_\infty$, the set-valued map f is defined in the following way:

$$f : \begin{cases} x \mapsto \Gamma(0.5(x + yx^{-1})) \\ y \mapsto y \\ z \mapsto \Gamma([\Gamma(0.5(x + yx^{-1}))]^2 - y) \\ c \mapsto c + 1 \\ v \mapsto \{-1, 1\} \end{cases}$$

where $\Gamma : \mathbb{R} \rightarrow \mathbb{F}$ is the quantization function in double precision format (more about computations with floating point numbers and the quantization function will be presented in Section 2.1.2). Note that in a similar fashion to Example 2.1, it is possible to treat y as a parameter and construct a parameter-dependent model of the program `ComputeSqrt` with x, z, c, v as the variables and y as the symbolic parameter of the model⁴.

In Section 2.3, we will present mathematical definitions of several common specifications for safety-critical software. While defining safety specifications in mathematical terms is necessary for formalizing the proofs of correctness, the definitions are very intuitive and logical. At this point, we would like to put this chapter in perspective by engaging in informal discussions about a few of these specifications in the context of Example 2.2. In Example 2.2, the program `ComputeSqrt` can generate different trajectories that depend on the initial value of y , and the real-time input `*PtrToInput`. It is obvious that the program terminates when the “while” loop terminates, which happens when the condition of the “while” loop is violated. Therefore, the program `ComputeSqrt` can be guaranteed to terminate in finite time if it can be shown that every such trajectory satisfies either $v(t) \in \{-1\}$ (equivalently $\text{NeedSqrtY}(t) \in \{\text{False}\}$), or that $(x(t), y(t), z(t)) \in \{(x, y, z) \in \mathbb{F}^3 \mid |z| < 10^{-4}\}$ for some positive integer t . On the other hand, to prove that an overflow runtime error does not occur during an execution of `ComputeSqrt`, we must prove that the variables do not grow in magnitude beyond a pre-specified safe limit. When

⁴See [82] for a detailed analysis of a class of programs similar to Program 2-2.

considering overflow runtime errors, boolean variables need not be verified as the only possible values that they can assume are in $\{\text{True}, \text{False}\}$ (i.e. in $\{-1, 1\}$). Programming errors arising from mishandling of boolean variables typically correspond to type mismatching, which can usually be identified by a regular compiler at compile time. On a 16-bit machine, variables of the type “double” are stored in 64-bit registers and integers of the type “int” are stored in 16-bit registers. The overflow region of the program `ComputeSqrt` can therefore, be characterized by:

$$X_- := \{b \in \mathbb{F}^3 \mid \|b\|_\infty \geq 1.7 \times 10^{308}\} \times \{c \in \mathbb{Z} \mid |c| \geq 32767\}.$$

The program `ComputeSqrt` can be guaranteed to run without an overflow runtime error if it can be shown that no trajectory can reach the set X_- . Finally, a division-by-zero runtime error does not occur if it can be proven that the value of x never becomes zero. We will show in Chapter 3, that each of these properties holds if a Lyapunov invariant satisfying certain technical conditions (adapted to the particular property) exists, and we will show in Chapter 4, how to find such functions.

2.1.2 Abstract Representation of Computer Programs

As we discussed in the previous section, the true state space of a computer program is a discrete finite subset of the rational numbers. This subset consists of all the rational numbers that can be represented by a finite number of bits in binary form, and depends on the operational arithmetic, e.g. fixed-point or floating-point arithmetic. The finiteness property of the state space presents advantages and challenges. The advantage is that computer programs can be accurately modeled as finite-state machines with inputs drawn from a finite alphabet set. Hence, strictly speaking, verification (e.g. proving or disproving finite-time termination) of programs running on computers with finite memory (i.e. finite-state machines) is not an undecidable problem. At least in theory, it can be performed by exploring and verifying all possible state trajectories by either numeric or symbolic simulation (e.g. in model checking). The challenge, however, is that the complexity of the finite-state models grow exponentially in the available number of bits, which renders exact verification of the finite-state models often impractical. Moreover, when performing calculations with non-integer numbers, a processor represents them

in an approximate binary form, which complicates the definitions of even simple operations such as addition and scaling (cf. Example 2.2).

In order to overcome these challenges, one often has to resort to a real-valued abstraction whose set of behavior properties (equivalently all possible trajectories) contains that of the actual program as a subset. In an abstract model, the state space is not constrained to be a finite set. An abstract model which deals with non-integer arithmetic can be defined in terms of real numbers, which has the potential to simplify the analysis dramatically. The drawbacks are twofold: the first is the obvious conservatism that is introduced by over-approximation of the set of possible behaviors; the second is undecidability. Nevertheless, abstract models simplify the task of program analysis and often make it possible to formulate computationally tractable (sufficient) conditions for a verification problem which would otherwise be intractable.

Definition 2.2 *Given a program \mathcal{P} and its dynamical system model $\mathcal{S}(X, f, X_0, X_\infty)$, we call the model $\mathcal{S}(\bar{X}, \bar{f}, \bar{X}_0, \bar{X}_\infty)$ an abstraction of \mathcal{P} , if $X \subseteq \bar{X}$, $X_0 \subseteq \bar{X}_0$, $f(x) \subseteq \bar{f}(x) : \forall x \in X$, and the following condition holds:*

$$\bar{X}_\infty \cap X \subseteq X_\infty \tag{2.2}$$

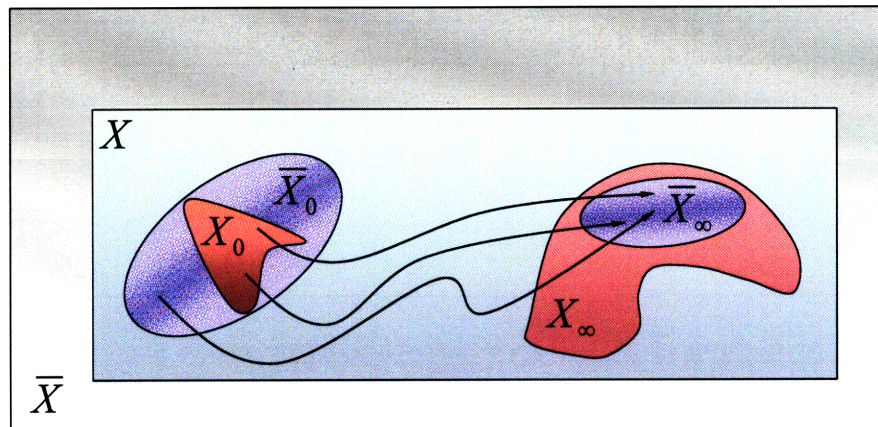


Figure 2-1: Conceptual diagram of evolution of the trajectories of a computer program and its abstraction.

An abstract representation can be interpreted as a formal over-approximation of the corresponding concrete representation. It follows from the definitions of \bar{X}_0 and $\bar{f}(x)$ as supersets of

X_0 and $f(x)$ that every trajectory of the actual program is also a trajectory of the abstract model, which is convenient for proving certain safety specifications such as absence of overflow (cf. Section 2.3). The definition of \overline{X}_∞ is slightly more subtle. We require \overline{X}_∞ to satisfy (2.2), so that the finite-time termination property of the concrete representation (equivalently the actual program) can be inferred from the finite-time termination of the abstract representation. This issue is discussed in more detail in Section 2.4, where a formal proof is also given. For the time being, we provide an intuitive justification for (2.2). We would like to be able to infer that if all the trajectories of the abstract model eventually enter the terminal set \overline{X}_∞ , then all the trajectories of the actual program will eventually enter the set X_∞ . It is tempting to require that $\overline{X}_\infty \subseteq X_\infty$, however, this may not be possible as X_∞ is often a discrete set of measure zero in the reals and \overline{X}_∞ is dense in the reals. The definition of \overline{X}_∞ as in (2.2) resolves this issue, while maintaining that the finite-time termination property can be carried over to the actual program.

Construction of an abstract representation $\mathcal{S}(\overline{X}, \overline{f}, \overline{X}_0, \overline{X}_\infty)$ from a concrete representation $\mathcal{S}(X, f, X_0, X_\infty)$ involves abstraction of each of the elements X , f , X_0 , X_∞ in a way that is consistent with Definition 2.4. Towards this end, abstraction of two types of objects must be constructed: sets and functions. Abstraction of the state space X is usually the trivial task. It often involves replacing the domain of floats by reals, or replacing the domain of integers by reals, or a combination of these. Abstraction of the other sets X_0 and X_∞ often involves a combination of replacement of the domain of floats or integers by reals and abstraction of functions (or functional relations) that define these sets. This will become clearer after we discuss abstraction of the set-valued function f . The function f usually consists of the composition of several simpler functions. Let $f_1 : X_1 \rightarrow Y_1$ and $f_2 : X_2 \rightarrow Y_2$ be set-valued functions such that $Y_1 \subseteq X_2$. Let $\overline{f}_1 : \overline{X}_1 \rightarrow \overline{Y}_1$ be an abstraction of f_1 , and $\overline{f}_2 : \overline{X}_2 \rightarrow \overline{Y}_2$ be an abstraction of f_2 in the sense that $f_1(x) \subseteq \overline{f}_1(x)$, $\forall x \in X_1$, and $f_2(x) \subseteq \overline{f}_2(x)$, $\forall x \in X_2$. Further assume that $\overline{Y}_1 \subseteq \overline{X}_2$. Then $\overline{f}_2 \circ \overline{f}_1$ is an abstraction of $f_2 \circ f_1$. This process can be repeated for construction of abstraction of a complicated function which can be expressed as the composition of several simpler functions. In particular when the domains of \overline{f}_i , $i = 1, \dots, m$ are the whole state space (e.g. the entire \mathbb{R}^n) then the conditions $\overline{Y}_i \subseteq \overline{X}_{i+1}$ are automatically satisfied. The implication of this simple observation is that an abstraction \overline{f} of the function f

can be constructed by simply replacing every subfunction in the composition of f by its abstract version.

Abstraction of Common Nonlinearities

In this section, we briefly review abstractions of some frequently-used nonlinear functions. This section is included to emphasize that our approach to developing abstract models is through construction of semialgebraic abstractions of nonlinear functions via uncertainty sets.

Trigonometric Functions Abstraction of trigonometric functions can be obtained by first approximating the function by its Taylor series expansion and then representing the absolute error by a static bounded uncertainty. For instance, an abstraction of the function $\sin(\cdot)$ can be defined in the following way:

Abstraction of $\sin(x)$:	$x \in [-\frac{\pi}{2}, \frac{\pi}{2}]$	$x \in [-\pi, \pi]$
$\overline{\sin}(x) \in \{x + aw \mid w \in [-1, 1]\}$	$a = 0.571$	$a = 3.142$
$\overline{\sin}(x) \in \{x - \frac{1}{6}x^3 + aw \mid w \in [-1, 1]\}$	$a = 0.076$	$a = 2.027$

Abstraction of $\cos(\cdot)$ can be done in a similar fashion. It is also possible to obtain piecewise linear abstractions by first approximating the function by a piecewise linear function and then representing the absolute error by a bounded uncertainty. For instance, if $x \in [0, \pi/2]$ then a reasonable piecewise linear approximation can be given by:

$$s(x) = \begin{cases} 0.9x & \text{if } x \in [0, 0.8] \\ 0.4x + 0.4 & \text{if } x \in [0.8, 1.6] \end{cases}$$

It can be verified that $|\sin(x) - s(x)| < 0.06, \forall x \in [0, \pi/2]$. Hence, an abstraction of $\sin(\cdot)$ can be constructed in the following way:

$$\overline{\sin}(x) \in \{T(x, v, w) \mid (x, v, w) \in S\}, \text{ where: } w = (w_1, w_2) \text{ and} \quad (2.3)$$

$$T : (x, v, w) \rightarrow 0.45(1 + v)x + (1 - v)(0.2x + 0.2) + 0.06w_1$$

$$S \triangleq \{(x, v, w) \mid x = 0.2[(1 + v)(1 + w_2) + (1 - v)(3 + w_2)], (w, v) \in [-1, 1]^2 \times \{-1, 1\}\}$$

We refer the reader to Section 2.2 (Mixed-Integer Linear Models) for algorithmic representation of piecewise linear functions using binary and continuous variables.

The Sign Function (sgn) and the Absolute Value Function (abs) The $\text{sgn}(\cdot)$ function:

$$\text{sgn}(x) = \begin{cases} 1, & x \geq 0 \\ -1, & x < 0 \end{cases}$$

appears commonly in computer programs, either explicitly or as an interpretation of *if-then-else* commands. A semialgebraic abstraction of the $\text{sgn}(\cdot)$ function can be constructed in the following fashion:

$$\overline{\text{sgn}}(x) \in \{v \mid xv \geq 0, v \in \{-1, 1\}\}$$

Note that $\text{sgn}(0) = 1$, while its abstraction is ambiguous at zero: $\overline{\text{sgn}}(0) \in \{-1, 1\}$.

The absolute value of a bounded variable $x \in [-1, 1]$ can be represented (precisely) in the following way:

$$\text{abs}(x) = \left\{ xv \mid x = \frac{1}{2}(v + w), w \in [-1, 1], v \in \{-1, 1\} \right\}$$

Floating-Point or Fixed-Point Arithmetic⁵ For computations with floating-point numbers, the IEEE 754-1985 norm has become the hardware standard in many processors such as Intel and PowerPC, and is supported by most popular programming languages such as C. In this standard, a *float* number is represented by a triplet (s, f, e) , where:

- $s \in \{0, 1\}$ is the sign bit.
- f is the fractional part, represented by a p -bit unsigned integer: $f_1 \dots f_p, f_i \in \{0, 1\}$.
- e is the biased exponent, represented by a q -bit unsigned integer: $e_1 \dots e_q, e_i \in \{0, 1\}$.

⁵This subsection is based on [66], Chapter 7. We present the material here for completeness. The reader is referred to [66] for a more comprehensive discussion of computations with floats.

A floating point number $z = (s, f, e)$ is then in one of the following forms:

- $z = (-1)^s \times 2^{e-bias} \times 1.f$, when $1 \leq e \leq 2^q - 2$.
- $z = (-1)^s \times 2^{1-bias} \times 0.f$, when $e = 0$, and $f \neq 0$.
- $z = +0$ (when $s = 1$) and $z = -0$ (when $s = 0$) and $e = f = 0$.
- $z = +\infty$ (when $s = 1$) and $z = -\infty$ (when $s = 0$) and $e = 2^q - 1$, $f \neq 0$.
- $z = NaN$ when $e = 2^q - 1$, $f = 0$.

The values of p , q , and $bias$ depend on the specific format:

- If format is 32-bit single precision (f=32), then $bias = 127$, $q = 8$, and $p = 23$.
- If format is 64-bit double precision (f=64), then $bias = 1023$, $q = 11$, and $p = 52$.

Other formatting standards such as *long double* or *quadruple* precision also exist. In floating point computations, the result of performing the arithmetic operation $\otimes \in \{+, -, \times, \div\}$ on two *float* variables x and y is stored in a *float* variable $z := \text{float}(x \otimes y, f)$ which is a complicated function of x , y , and the format f . Examples of the format f include the IEEE 754 with 32-bit single precision, or IEEE 754 with 64-bit double precision. A floating-point operation is equivalent to performing the operation on the reals followed by rounding the result to a *float* [66]. In IEEE 754 the possible rounding modes are rounding towards 0, towards $+\infty$, towards $-\infty$, and to the nearest (n). The rounding function $\Gamma_{f,m} : \mathbb{R} \rightarrow \mathbb{F} \cup \{\Omega\}$ maps a real number to a float number or to runtime error, depending on the format ‘f’ and the rounding mode ‘m’. We refer the reader to [66] for more details on the rounding function. For our purposes, it is sufficient to say that for all rounding modes, the following relation holds:

$$\forall x \in [-\alpha_f, \alpha_f] : |\Gamma_{f,m}(x) - x| \leq \gamma_f |x| + \beta_f$$

where $\gamma_f := 2^{-p}$, and $\alpha_f := (2 - 2^{-p}) 2^{2^q - bias - 2}$ is the largest non-infinite number, and $\beta_f := 2^{1 - bias - p}$ is the smallest non-zero positive number.

Based on the above discussions, an abstraction of the floating-point arithmetic operators can be constructed in the following way:

$$x + y \in [-\alpha_f, \alpha_f] \Rightarrow \text{float}(x + y) = z \in \{x + y + \delta w \mid w \in [-1, 1], \delta = \gamma_f(|x| + |y|) + \beta_f\}$$

$$x - y \in [-\alpha_f, \alpha_f] \Rightarrow \text{float}(x - y) = z \in \{x - y + \delta w \mid w \in [-1, 1], \delta = \gamma_f(|x| + |y|) + \beta_f\}$$

$$x \times y \in [-\alpha_f, \alpha_f] \Rightarrow \text{float}(x \times y) = z \in \{x \times y + \delta w \mid w \in [-1, 1], \delta = \gamma_f(|x| \times |y|) + \beta_f\}$$

$$x \div y \in [-\alpha_f, \alpha_f] \Rightarrow \text{float}(x \div y) = z \in \{x \div y + \delta w \mid w \in [-1, 1], \delta = \gamma_f(|x| \div |y|) + \beta_f\}$$

where the constants α_f , β_f and γ_f are defined as before. The above abstractions can still be complicated as the magnitude of δ depends on the operands x and y . In practice, for most computer programs of safety critical systems, the values of the program variables are expected to be much smaller in magnitude than the very large number α_f . Assuming that all the program variables (including the result of the arithmetic operation) reside in $[-\alpha, \alpha]$ where $\alpha \ll \alpha_f$ then a simpler but more conservative abstraction can be constructed in the following way:

$$x \otimes y \in [-\alpha, \alpha] \Rightarrow \text{float}(x \otimes y) = z \in \{x \otimes y + \delta w \mid w \in [-1, 1], \delta = \alpha \gamma_f + \beta_f\}$$

For instance, if $f=32$, $\alpha = 10^6$, then $\delta = 0.12$, and if $f=64$, $\alpha = 10^{10}$, then $\delta = 2.3 \times 10^{-6}$.

Abstractions of arithmetic operations in fixed-point computations is similar to the above. The magnitude of δ will depend on the number of bits and the dynamic range. For instance, in the two's complement format we have: $\delta = \rho (2^b - 1)^{-1}$, where ρ is the dynamic range and b is the number of bits.

Modulo Arithmetic Consider the function $\text{mod} : \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z}$ defined in the following way:

$$\text{mod}(t, s) = t - ns, \text{ where } n = \lfloor \frac{t}{s} \rfloor.$$

Abstraction of $\text{mod}(\cdot, \cdot)$ for the general case is complicated. However, the following scenario is not uncommon: assume that it is known that $t_1 < s$ and $t_2 < s$. Then:

$$\text{mod}(t_1 + t_2, s) \in \left\{ t_1 + t_2 - \frac{s}{2}(1 + v) \mid (t_1 + t_2 - s)v \geq 0, v \in \{-1, 1\} \right\}.$$

A common instance of the above scenario is when $t_2 < s$ is a constant and t_1 is a variable that is initialized to zero and updated according to $t_1 \rightarrow \text{mod}(t_1 + t_2, s)$. It is possible to construct similar abstractions for more complicated scenarios by including more binary variables.

Example 2.3 Abstract model of the program *ComputeSqrt*: Consider the C program presented in Example 2.2. The elements of an abstract model can be defined in the following way:

$$\bar{X} : = \mathbb{R}^3 \times \mathbb{Z} \times \{-1, 1\}$$

$$\bar{X}_0 : = \{(1, y, 1) \mid y \in [10^{-4}, 10^4]\} \times \{0\} \times \{-1, 1\}$$

$$\bar{X}_\infty : = \bar{X}_{1\infty} \cup \bar{X}_{2\infty} \text{ where}$$

$$\bar{X}_{1\infty} : = \mathbb{R}^3 \times \mathbb{Z} \times \{-1\}, \bar{X}_{2\infty} := \{(x, y, z) \in \mathbb{R}^3 \mid |z| < e\} \times \mathbb{Z} \times \{-1, 1\}$$

The set valued map \bar{f} is defined in the following way:

$$\bar{f} : \begin{cases} x \mapsto \{0.5(x + yx^{-1}) + \delta_1 w_1 \mid w_1 \in [-1, 1]\} \\ y \mapsto y \\ z \mapsto \{0.5(x + yx^{-1})^2 - y + \delta_2 w_2 \mid w_2 \in [-1, 1]\} \\ c \mapsto c + 1 \\ v \mapsto \{-1, 1\} \end{cases}$$

where δ_1 and δ_2 represent the magnitude of the uncertainties that are introduced by floating-point roundoff errors. It can be verified that with 64-bit double precision format, assuming that all the variables remain within $[-10^{10}, 10^{10}]$, then $\delta_1, \delta_2 \leq \delta := 8 \times 10^{-6}$.

Example 2.4 Consider the following program:

```

void Accelerated Turn (double x, double y, double h)
//y coordinate initially in the interval [0, 1],
//h initially in the interval [0, 1], is the upper bound for y
//x coordinate initially in {-1, 1}
{ double t = {0}; //t is the turn angle
  while (y < h)
  {t = mod (t + 0.001, 1);
   if x > 0   {
     x = x - cos(t); y = y + sin(t);
   }
   else
     x = x + cos(t); y = y + sin(t);
  }}

```

Program 2-3.

A concrete representation can be defined by $\mathcal{S}(X, f, X_0, X_\infty)$ where

$$X = \mathbb{F}^4, X_0 = \mathbb{F}^4 \cap (\{-1, 1\} \times [0, 1] \times \{0\} \times [0, 1]), X_\infty = \{(x, y, t, h) \in \mathbb{F}^4 \mid y \geq h\}$$

Over the set X_∞ , f is the identity map and over $X \setminus X_\infty$ it can be defined in the following way:

$$f : (x, y, t, h) \mapsto \Gamma(x + \text{sgn}(x) \times \cos(t), y + \sin(t), \text{mod}(t + 0.001, 1), h)$$

Various levels of abstraction can be applied to $\mathcal{S}(X, f, X_0, X_\infty)$ to obtain an abstract model. For the moment, let us assume that the net effect of round-off errors are such that the absolute error in the computation of $x \pm \sin(t)$ (or $x \pm \cos(t)$) is never larger than a small positive number δ . We make similar assumptions about y .

$$\bar{X} = \mathbb{R}^4, \bar{X}_0 = \{-1, 1\} \times [0, 1] \times \{0\} \times [0, 1], \bar{X}_\infty = \{(x, y, t, h) \in \mathbb{R}^4 \mid y \geq h\}$$

$$\bar{f} : \begin{cases} x \mapsto \{x + v_1 \times \cos(t) + \delta w_1 \mid xv_1 \geq 0, w_1 \in [-1, 1], v_1 \in \{-1, 1\}\} \\ y \mapsto \{y + \sin(t) + \delta w_2, \mid w_2 \in [-1, 1]\} \\ t \mapsto \{t + 0.001 - 0.5 - 0.5v_2 \mid (t + 0.001 - 1)v_2 \geq 0, v_2 \in \{-1, 1\}\} \\ h \mapsto h \end{cases}$$

Further abstractions are possible by defining:

$$\bar{X} = \mathbb{R}^4, \bar{X}_0 = \{-1, 1\} \times [0, 1] \times \{0\} \times [0, 1], \bar{X}_\infty = \{(x, y, t, h) \in \mathbb{R}^4 \mid y \geq h\}$$

$$\bar{f} : \begin{cases} x \mapsto \{x + v_1 \times (1 - 0.5t^2) + \delta w_1 + 0.01w_3 \mid xv_1 \geq 0, w_1, w_3 \in [-1, 1], v_1 \in \{-1, 1\}\} \\ y \mapsto \{y + t - t^3/6 + \delta w_2 + 0.05w_4, \mid w_2, w_4 \in [-1, 1]\} \\ t \mapsto \{t + 0.001 - 0.5 - 0.5v_2 \mid (t + 0.001 - 1)v_2 \geq 0, v_2 \in \{-1, 1\}\} \\ h \mapsto h \end{cases}$$

2.2 Specific Models of Computer Programs

In a verification framework, specific modeling languages are particularly necessary for automating the proof process. In this section, we propose three specific modeling languages for dynamical system representation of computer programs: *Mixed-Integer Linear Models (MILM)*, *Graph Models*, and *Linear Models with Conditional Switching (LMwCS)*. We believe that these models can represent a broad range of computer programs of interest to the control community. In comparison with the generic dynamical system representation $\mathcal{S}(X, f, X_0, X_\infty)$, in the specific models, we specify the state space X , and the structure of the corresponding subsets $X_0 \subseteq X$, and $X_\infty \subseteq X$. The same is true for the set-valued map f which is restricted to be a piecewise affine or piecewise polynomial set-valued map represented in a specific format.

We use the generic representations whenever the details of the model is irrelevant to the discussion and/or the result. This includes some of the fundamental results in Chapter 3 on analysis of software via Lyapunov invariants. We also use the generic representations in Section 2.4 to study the consequences of abstractions on proofs of correctness. On the other hand, the specific models are very convenient models for computation of the Lyapunov invariants via

convex optimization (cf. Chapter 4). They can be conveniently included in a fully automated or semi-automated verification framework. The choice of the modeling language is influenced by the specifications and by practical considerations such as availability of an automated parsing tool to translate the computer code into a particular modeling language, existence of an efficient convex relaxation technique, and compatibility with a particular numerical engine for convex optimization. We will discuss some of the advantages and disadvantages that each of these models offer as we present them in this section.

2.2.1 Mixed-Integer Linear Models

Using mixed-integer linear models for software analysis is motivated by the observation that these models can provide a relatively compact description of the behavior of programs with arbitrary piecewise affine dynamics defined over bounded polytopic subsets of the Euclidean space. In addition, generalization of the model to a specific class of programs with piecewise affine dynamics defined over parabolic subsets (sets with a second order description) of the Euclidean space is relatively straightforward. Further generalization to programs with piecewise polynomial dynamics is also possible. We will discuss these generalizations briefly in the remarks section after Proposition 2.1. Proposition 2.1 establishes the universality of mixed-integer linear models, in the sense that they can represent arbitrary piecewise affine functions with closed graphs. The statement of the proposition was formulated in [41]. Mixed Logical Dynamical Systems (MLDS) with very similar structure to the models presented in Proposition 2.1 were considered in [11] for analysis of a class of hybrid systems. Although there are some minor differences between the MILMs introduced in this section and the MLDS in [11], the main contributions here are the application of the model to software analysis and presenting a proof for the statement on the universality of MILMs, which was first formulated in [41].

Proposition 2.1 *Universality of Mixed-Integer Linear Models.* *Let $f : X \rightarrow \mathbb{R}^n$ be a piecewise affine function with a closed graph, defined on a compact state space $X \subseteq [-1, 1]^n$, which consists of a finite union of compact polytopes. That is:*

$$f(x) = 2A_i x + 2B_i, \quad \text{subject to} \quad x \in X_i, \quad i \in \mathbb{Z}(1, N),$$

where, each X_i is a compact polytopic set. Then, f can be defined precisely, by imposing linear equality constraints on a finite number of binary variables and a finite number of continuous variables ranging over compact intervals. More specifically, there exist matrices F and H , such that the following two sets are equal:

$$G_1 \triangleq \{(x, f(x)) \mid x \in X\}$$

$$G_2 \triangleq \left\{ (x, y) \mid F[x \ w \ v \ 1]^T = y, H[x \ w \ v \ 1]^T = 0, (w, v) \in [-1, 1]^q \times \{-1, 1\}^r \right\}$$

Proof. The proof is by construction. Let $X = \bigcup_{i=1}^N X_i$, where the X_i 's are compact polytopic sets. Further, assume that each X_i is characterized by a finite set of linear inequality constraints:

$$X_i := \{x \mid \mathbf{S}_i x \leq \mathbf{s}_i, \mathbf{S}_i \in \mathbb{R}^{N_i \times n}, \mathbf{s}_i \in \mathbb{R}^{N_i}\}$$

Let $v := [v_1 \ \dots \ v_N] \in \{-1, 1\}^N$, and consider the following sets:

$$G_{xv} \triangleq \left\{ (x, v) \mid \sum_{i=1}^N v_i = -N + 2, (\mathbf{S}_i x - \mathbf{s}_i)(v_i + 1) \leq 0, v_i \in \{-1, 1\} : i \in \mathbb{Z}(1, N) \right\},$$

$$G_{xy} \triangleq \left\{ (x, y) \mid \sum_{i=1}^N (1 + v_i)(A_i x + B_i) = y, (x, v) \in G_{xv} \right\}.$$

First, we prove that $G_{xy} = G_1$. Define N binary vectors $\eta^j \in \{-1, 1\}^N$, $j \in \mathbb{Z}(1, N)$ according to the following rule: $\eta_i^j = 1 \Leftrightarrow i = j$. Also define $I(x) := \{j \in \mathbb{Z}(1, N) \mid x \in X_j\}$. Now, let $(x_0, f(x_0)) \in G_1$. Then:

$$x_0 \in \bigcup_{i=1}^{i=N} X_i, \quad (x_0, \eta^j) \in G_{xv} : \forall j \in I(x_0).$$

Therefore, $(x_0, 2A_j x_0 + 2B_j) \in G_{xy} : \forall j \in I(x_0)$, which by supposition, implies that:

$$(x_0, f(x_0)) \in G_{xy}.$$

This proves that $G_1 \subseteq G_{xy}$. Now, let $(x_0, y_0) \in G_{xy}$. Then, there exists $\hat{v} \in \{-1, 1\}^N$, such that $(x_0, \hat{v}) \in G_{xv}$. Hence, there exists $j \in \mathbb{Z}(1, N)$, such that $x_0 \in X_j$, and $y_0 = 2A_jx + 2B_j$. It follows from the definition of f that $(x_0, y_0) \in G_1$. This proves that $G_{xy} \subseteq G_1$. We have shown that $G_1 = G_{xy}$. It remains to show that G_{xy} has an equivalent description to G_2 .

Define new variables $u_i := xv_i \in [-1, 1]^n$. Then:

$$G_{xy} \triangleq \left\{ (x, y) \mid y = \sum_{i=1}^N (A_i x + A_i u_i + B_i + B_i v_i), \mathbf{S}_i u_i + \mathbf{S}_i x - \mathbf{s}_i v_i - \mathbf{s}_i \leq 0 \right. \\ \left. \sum_{i=1}^N v_i = -N + 2, u_i = xv_i, v_i \in \{-1, 1\} : i \in \mathbb{Z}(1, N) \right\}. \quad (2.4)$$

By definition, $u_i \in [-1, 1]^n$ is the multiplication of a bounded continuous variable $x \in [-1, 1]^n$, and a binary scalar variable v_i . This (nonlinear) transformation can be represented by an affine transformation involving auxiliary variables $\underline{z}_i \in [-1, 1]^n$, and $\bar{z}_i \in [-1, 1]^n$, subject to a set of linear constraints, in the following way:

$$u_i = 2\underline{z}_i - x - v_i \mathbf{1}_n + \mathbf{1}_n, \quad \underline{z}_i \leq v_i \mathbf{1}_n, \quad \bar{z}_i \leq -v_i \mathbf{1}_n, \quad \underline{z}_i = x - \bar{z}_i - \mathbf{1}_n \quad (2.5)$$

equivalently:

$$u_i = 2\underline{z}_i - x - (v_i - 1) \mathbf{1}_n, \quad \underline{z}_i = x - \bar{z}_i - \mathbf{1}_n. \\ \underline{z}_i = (v_i - 1) \mathbf{1}_n + \underline{w}_i, \quad \bar{z}_i = (-v_i - 1) \mathbf{1}_n + \bar{w}_i.$$

where $\underline{w}_i, \bar{w}_i \in [-1, 1]^n$. Since by assumption each X_i is bounded, for all $i \in \mathbb{Z}(1, N)$ and all $j \in \mathbb{Z}(1, N_i)$, the quantities

$$\underline{R}_{ij} := \min_{x \in X_i} \mathbf{S}_{ij} x - \mathbf{s}_{ij} \quad (2.6)$$

exist, are finite and can be computed by solving $N_i \times N$ linear programs given in (2.6) (\mathbf{S}_{ij} and \mathbf{s}_{ij} denote the j -th row of \mathbf{S}_i and \mathbf{s}_i respectively). Define $\underline{R}_i := \text{diag}_{j \in \mathbb{Z}(1, N_i)} \{\underline{R}_{ij}\}$. Then G_{xy} as defined in (2.4) is equivalent to:

$$G_{xy} \triangleq \left\{ (x, y) \mid y = \sum_{i=1}^N (A_i x + A_i u_i + B_i + B_i v_i), (x, u_i, v_i) \in \mathbf{H} \right\} \quad (2.7)$$

where,

$$\mathbf{H}=\{(x, u_i, v_i)\} : \left\{ \begin{array}{l} 0 = \mathbf{S}_i u_i + \mathbf{S}_i x - \mathbf{s}_i v_i - \mathbf{s}_i - \underline{R}_i (w_i + \mathbf{1}_{N_i}) \\ u_i = 2z_i - x - (v_i - 1) \mathbf{1}_n \\ z_i = (v_i - 1) \mathbf{1}_n + \underline{w}_i \\ z_i = x - \bar{z}_i - \mathbf{1}_n, \quad z_i, \bar{z}_i \in [-1, 1]^n \\ \bar{z}_i = (-v_i - 1) \mathbf{1}_n + \bar{w}_i, \\ \sum_{i=1}^N v_i = -N + 2, \quad v_i \in \{-1, 1\} \\ [-1, 1]^n \ni z_i, \bar{z}_i, \underline{w}_i, \bar{w}_i, u_i, [-1, 1]^{N_i} \ni w_i \end{array} \right. \quad (2.8)$$

The equality constraints that define the Matrix H are precisely the equalities in (2.8), while the equality constraint in (2.7) defines the Matrix F . This completes the proof. ■

So far, we have established that by imposing linear equality constraints on boolean and continuous variables defined over a quasicube⁶, one can define arbitrary piecewise affine functions on finite unions of polytopes. This observation serves as the basis for considering the mixed-integer linear models for software analysis. The motivation is that these models are capable of providing relatively compact descriptions of complicated dependencies between the program variables.

A mixed-integer linear model of a computer program is defined via the following elements:

1. The state space $X \subset [-1, 1]^n$.
2. The state transition function $f : X \mapsto 2^X$ is defined by two matrices F , and H of dimensions n -by- $(n + q + r + 1)$ and p -by- $(n + q + r + 1)$ respectively, according to:

$$f(x) \in \left\{ F[x \ w \ v \ 1]^T \mid H[x \ w \ v \ 1]^T = 0, \ (w, v) \in [-1, 1]^q \times \{-1, 1\}^r \right\}. \quad (2.9)$$

3. There are two possible ways to define the set of initial conditions for mixed-integer linear models.

- (a) If X_0 is finite and its cardinality is not too large, then one can conveniently specify X_0 by its elements. We will see in Chapter 4 that in this case, per each element

⁶ A quasicube is defined as the Cartesian product of a hypercube and the set of vertices of another hypercube.

of X_0 , one additional constraint needs to be added to the set of constraints of an optimization problem that will be set up for verification of certain properties of the model.

- (b) If X_0 is not finite, or $|X_0|$ is too large, X_0 or an abstraction of it can be specified by a matrix $H_0 \in \mathbb{R}^{N_0 \times n_e}$ which specifies a union of compact polytopes in the state space in the following way:

$$X_0 = \{x \in X : H_0[x \ w \ v \ 1]^T = 0, (w, v) \in [-1, 1]^q \times \{-1, 1\}^r\}. \quad (2.10)$$

Note that it is possible to choose the matrix H_0 such that X_0 is finite. In other words, case (b) covers case (a) as a special case.

4. Finally, the set of terminal states X_∞ is defined by

$$X_\infty = \{x \in X : H[x \ w \ v \ 1]^T \neq 0, \forall w \in [-1, 1]^q, \forall v \in \{-1, 1\}^r\}. \quad (2.11)$$

Therefore, $\mathcal{S}(X, f, X_0, X_\infty)$ is well defined. A compact description of a mixed integer linear model of a program \mathcal{P} is either of the form $\mathcal{S}(F, H, H_0, n, q, r)$, or of the form $\mathcal{S}(F, H, X_0, n, q, r)$ whenever X_0 is finite and does not have too many elements.

MILMs can represent a broad range of computer programs of interest to the control community. These include but are not limited to single flow programs, and control programs of gain scheduled linear systems in embedded applications. As we will discuss in Chapter 3, natural Lyapunov invariant candidates for MILMs are quadratic functionals. Within this class, the traditional quadratic relaxation techniques, e.g. the \mathcal{S} -Procedure, can be used to formulate the search for the Lyapunov invariants as a semidefinite optimization problem.

Remarks

1. We have defined the MILMs over a quasicube of unit length. In practice, to represent a computer program in this format, often an appropriate scaling of the variables is needed. Alternatively, one can consider MILMs over a quasicube of length α . That is, the state space can be defined as $X := [-\alpha, \alpha]^n$, and $(w, v) \in [-\alpha, \alpha]^q \times \{-\alpha, \alpha\}^r$.

2. It is not uncommon to encounter the following programming situation (displayed on the left):

$$\begin{array}{lll}
 \text{if } g(x) \geq 0 & & \\
 \quad x = f_1(x); & & x \mapsto f_1(x) \quad \text{if } g(x) > 0 \\
 \text{else} & \text{Abstraction} & x \mapsto f_2(x) \quad \text{if } g(x) < 0 \\
 \quad x = f_2(x); & \Rightarrow & x \mapsto \{f_1(x), f_2(x)\} \quad \text{if } g(x) = 0 \\
 \text{end} & &
 \end{array}$$

where $f_1(x)$ and $f_2(x)$ are affine expressions such that $f_1(x_0) \neq f_2(x_0)$ for some x_0 satisfying $g(x_0) = 0$. In such situations, the graph of the function f (cf. Proposition 2.1) is not closed. Nevertheless, a MILM can be constructed using the same procedure in Proposition 2.1 by considering an appropriate abstraction of the program (displayed above on the right). If the correctness of the program does depend on the accurate definition of the map at x_0 (where $g(x_0) = 0$) then the above abstraction is inadequate⁷.

3. It can be observed from the proof of Proposition 2.1 that it is possible to consider MILMs with both equality and inequality constraints. In other words, converting all the inequality constraints in (2.4) and (2.5) to equality constraints is not essential. The advantage in converting the inequalities to equalities is that the overall representation is more compact. It is far more convenient to work with models that include only equality constraints rather than both equalities and inequalities. The drawback is that more auxiliary slack variables must be introduced.
4. It is particularly appealing to consider models which include both linear and quadratic equality constraints (namely, the mixed-integer linear-quadratic models (MILQM)). If quadratic equality constraints are allowed, then the constraints $u_i = xv_i$ can be readily included in the quadratic constraints without the need to introduce the auxiliary variables

⁷In this case, a remedy exists based on using open (or semi-open) intervals (e.g. $[-1, 1)$) for defining the slack variables followed by sum-of-squares relaxation as the convex relaxation technique in the optimization phase.

$\bar{z}_i, \underline{z}_i$. If semidefinite optimization is the method of choice in the search for (quadratic) Lyapunov invariants as behavior certificates, then including quadratic constraints in the model does not complicate the process, as efficient techniques exist for semidefinite programming relaxation of quadratic constraints (cf. Chapter 4). However, it is well known that applying different convex relaxations to equivalent constraints (described in alternative but equivalent forms) does not in general, lead to identical results. Hence, it is hard to predict which model would lead to better results in analysis of the behavioral properties of the software via semidefinite optimization.

5. It follows from the discussion in item (3), that the MILMs can be conveniently generalized to MILQMs without making the analysis process more complicated, particularly if semidefinite optimization is used in the search for behavior certificates. If quadratic constraints (both equalities and inequalities) are allowed, they are not restricted to the representation of the constraints $u_i = xv_i$. More complicated dependencies can be modeled with MILQMs, particularly when the sets X_i are not polytopic, but they have a second order description, e.g. $X_1 := \{(x_1, x_2) \mid x_1^2 + x_2^2 - x_1x_2 \leq 0\}$.
6. Further generalizations of the MILMs to models with polynomial dynamics is also possible. The same approach that is used in the proof of Proposition 2.1 (i.e. assigning a binary variable v_i to each X_i) can be used for modeling programs with polynomial or polynomial fractional dynamics. For instance, consider the following pseudocode for computing the square root of a positive number:

```

 $x, y, e$  : real
while  $|x^2 - y| > e$ 
 $x = 0.5(x + y/x)$ ;
end

```

The program can be modeled as: $x_+ = f(x_e)$, subject to $h(x_e) = 0, g(x_e) > 0$; where $x_e = (x, y, e, z, v)$, and $f : x_e \rightarrow 0.5(x + z)$, $h : x_e \rightarrow zx - y$, $g : x_e \rightarrow vx^2 - vy - e$, $v \in \{-1, 1\}$.

Example 2.5 A MILM of an abstraction of Program 2-1 (Section 2.1), the *IntegerDivision* program, with all the integer variables replaced with real variables, is defined by $\mathcal{S}(F, H, H_0, 4, 3, 0)$, where:

$$H_0 = \begin{bmatrix} 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 1 & 0 & 1 \\ -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \quad F = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1/\alpha \\ 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$H = \begin{bmatrix} 0 & 1 & 0 & -1 & 1 & 0 & 0 & 1 \\ 0 & -1 & 0 & 0 & 0 & 1 & 0 & 1+1/\alpha \\ -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Here, α is a parameter used for scaling all the variables within the interval $[-1, 1]$.

Example 2.6 Consider the following program:

```

//x1, x2 initially zero
while x2 < 100
  if x1 > 0
    x1 = x1 - a * (w1 + 2);
  else
    x1 = x1 + b * (w1 + 2);
  end
x2 = x2 + 1;
end
```

Program 2-4.

where a and b are fixed parameters and $w_1 \in [-1, 1]$ represents real-time input. The MILM of this program is defined by $\mathcal{S}(F, H, \{(0, 0)\}, 2, 8, 1)$, where

$$F = \begin{bmatrix} 1 & 0 & c_2 & c_1 & 0 & 0 & 0 & 0 & 0 & 0 & 2c_1 & 2c_2 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/\alpha \end{bmatrix}$$

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & -0.5 & 0 & 0 & 0 & 0 & 0 & -0.5 & 0 \\ 0 & 1 & 0 & 0 & 0 & r_1 & 0 & 0 & 0 & 0 & 0 & r_2 \\ 0 & 0 & -1 & -1 & 0 & 0 & 2 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & -1 & -1 \end{bmatrix}$$

$$c_2 = (b - a)/2\alpha, \quad c_1 = -(b + a)/2\alpha, \quad r_1 = 0.5 + 50/\alpha, \quad r_2 = 0.5 - 50/\alpha.$$

Again α is a scaling factor.

2.2.2 Graph models

In this section we introduce the so-called *graph models* for analysis of computer programs. Practical considerations such as universality, expressivity, and strong resemblance to the natural flow of computer code, which is attractive for automated parsing, render graph models a convenient and efficient model for analysis of software. Before we proceed to defining graph models, for convenience, we introduce the following notations:

Notation 2.1 We denote by s_i the projection operator $s_i : (\mathbb{Z} \cup \{\bowtie\}) \times \mathbb{R}^n \rightarrow \mathbb{R}^n$, defined according to:

$$s_i(i, x) = x : i \in \mathbb{Z} \cup \{\bowtie\}, \quad x \in \mathbb{R}^n.$$

Notation 2.2 Let Π denote a semialgebraic set, defined according to:

$$\Pi := \{x \mid H(x) = 0, F(x) \geq 0, G(x) \neq 0\}, \quad (2.12)$$

where H , F , and G , are multivalued polynomial functions mapping \mathbb{R}^n to \mathbb{R}^{n_H} , \mathbb{R}^{n_F} , and \mathbb{R}^{n_G}

respectively. Equivalently, Π can be understood as an operator mapping the set of all polynomial functions over \mathbb{R}^n to a semialgebraic set $\Pi : \mathcal{F} \rightarrow \mathcal{S}$, in the following way:

$$\Pi(\tau) := \left\{ x \mid H_{ji}^k(\tau(x)) = 0, F_{ji}^k(\tau(x)) \geq 0, G_{ji}^k(\tau(x)) \neq 0 \right\}$$

where $\tau : \mathbb{R}^n \rightarrow \mathbb{R}^n$ is polynomial. Whenever the argument of $\Pi(\cdot)$ is the identity map, we simply write Π instead of $\Pi(I)$, which is consistent with (2.12).

The definition that we present here for graph models is very intuitive, although the notation may appear somewhat cumbersome. A graph model⁸ is defined on a directed graph $G(\mathcal{N}, \mathcal{E})$. The elements of this model are:

1. A set of nodes $\mathcal{N} := \{\emptyset\} \cup \{1, \dots, m\} \cup \{\infty\}$. These nodes are effectively the line numbers or locations of lines of code. Node \emptyset is the the starting node and node ∞ is the terminal node. The only possible transition from node ∞ is the identity transition to node ∞ .
2. A set of arcs $\mathcal{E} := \{(i, j, k) \mid i \in \mathcal{N}, j \in \mathcal{O}(i)\}$, where $\mathcal{O}(i)$ is the set of all nodes to which transition from node i is possible in one step. Similarly, $\mathcal{I}(i)$ denotes the set of all nodes from which transitions to node i is possible in one step. Multiple arcs between nodes are allowed and the third element in the triplet (i, j, k) is the index of the k -th arc between nodes i and j . The set $\mathcal{A}_{ji} := \{1, \dots, \bar{\kappa}_{ji}\}$ denotes the set of all indices of the arcs from node i to node j , where $\bar{\kappa}_{ji}$ is the total number of arcs starting from node i , ending at node j .
3. A set of program variables $x_q \in \mathbb{R}$, $q \in \mathbb{Z}(1, n)$. Given \mathcal{N} and n , we define the state space of a graph model as $X := \mathcal{N} \times \mathbb{R}^n$. The state (i, x) of a graph model has therefore, two components: The discrete component $i \in \mathcal{N}$ is the node number (location or line number) and the continuous component $x \in \mathbb{R}^n$ corresponds to the program variables. We sometimes use the notation $\tilde{x} := (i, x)$ to refer to the state of a graph model.
4. A set of *transition* labels (T_{ji}^k, S_{ji}^k) assigned to every arc $(i, j, k) \in \mathcal{E}$, where $T_{ji}^k : \mathbb{R}^n \rightarrow 2^{\mathbb{R}^n}$ represents a set-valued function mapping x to the set $T_{ji}^k x := \{\bar{T}_{ji}^k(x, w) \mid (x, w) \in S_{ji}^k\}$,

⁸Models conceptually similar to the graph models proposed in this document have been reported in [75] for software verification, and in [3, 19] for modeling and verification of hybrid systems.

where $\overline{T}_{ji}^k : \mathbb{R}^{n+n_w} \rightarrow \mathbb{R}^n$ is a polynomial function of x and w , and S_{ji}^k is a semialgebraic set defined in terms of $x \in \mathbb{R}^n$ and $w \in \mathbb{R}^{n_w}$. If T_{ji}^k is a deterministic function, we drop S_{ji}^k from the transition label and simply write T_{ji}^k as the *transition* label.

5. A set of *passport* labels Π_{ji}^k assigned to all the arcs $(i, j, k) \in \mathcal{E}$, where Π_{ji}^k represents a semialgebraic set. State transition along the arc (i, j, k) is possible if and only if $x \in \Pi_{ji}^k$. In Chapter 5, where we introduce the concept of graph reduction, we may use the notation $\Pi_{ji}^k(\tau)$ for the passport label which is understood in the same sense as Definition 2.2.
6. A set of semialgebraic invariant sets $X_i \subset \mathbb{R}^n$, $i \in \mathcal{N}$ are assigned to every node on the graph, such that $s_l(i, x) \in X_i$. In other words, every time that location i is reached, the program variable x is known to be in the set X_i . Equivalently, a state $\tilde{x} := (i, x)$ such that $x \notin X_i$ is unreachable. Sometimes, we may use the notation $X_i(\tau)$ for the invariant sets which is again understood in the same sense as Definition 2.2.

Therefore, a graph model defines a dynamical system model $\mathcal{S}(X, f, X_0, X_\infty)$ of a computer program \mathcal{P} , where the state space X is given by:

$$X := \mathcal{N} \times \mathbb{R}^n,$$

the set of initial and terminal states are given by:

$$X_0 := \{\emptyset\} \times X_\emptyset, \quad X_\infty := \{\bowtie\} \times X_\bowtie,$$

and the state transition map $f : X \rightarrow 2^X$ is given by:

$$f(\tilde{x}) \equiv f(i, x) = \left\{ (j, T_{ji}^k x) \mid j \in \mathcal{O}(i), x \in \Pi_{ji}^k \right\}. \quad (2.13)$$

According to (2.13), the uncertainty in the definition of $\tilde{x}(t+1)$ enters the model in two ways: one is the uncertainty in the discrete state transition: $i \rightarrow j \in \mathcal{O}(i)$, and the other is the uncertainty associated with the continuous state transition $x \rightarrow T_{ji}^k x$ which is by definition a set-valued map.

Remarks

1. In a graph model, node \emptyset represents a (perhaps fictitious) line of code containing all the available information about the initial conditions of the continuous variables. This information is included in the model via the invariant set of node \emptyset : $s_i(\emptyset, x) \in X_\emptyset$.
2. Multiple arcs between nodes enable modeling of "or" or "xor" type conditional transitions in computer programs. The passport labels associated with multiple arcs between two nodes are not necessarily exclusive, that is multiple transitions along different arcs may be possible at each instant of time. This allows for nondeterministic modeling.
3. The transition label (T_{ji}^k, S_{ji}^k) may represent a simple update rule which depends on the real-time input. For instance, if $\bar{T} = Ax + Bw$, and $S = \mathbb{R}^n \times [-1, 1]$, then $x \xrightarrow{(T,S)} \{Ax + Bw \mid w \in [-1, 1]\}$. In other cases, (T_{ji}^k, S_{ji}^k) may represent an abstraction of a nonlinear transformation (cf. Section 2.1.2). For instance, an abstraction of the assignment $x \rightarrow \sin(x)$ can be represented by $x \xrightarrow{(T,S)} \{T(x, w) \mid (x, w) \in S\}$, where T and S are given in Equation 2.3.
4. We have defined the second component of the state space to be a continuous variable in \mathbb{R}^n . It may be the case that some of these variables are binary variables residing in $\{-1, 1\}^q$, or bounded continuous variables residing in $[-1, 1]^r$, $r + q \leq n$. In such cases, this information can be included in the model by adding the set

$$X_{\mathcal{N}} := \{x \in \mathbb{R}^n \mid 1 - x_i^2 = 0, 1 - x_j^2 \geq 0, i \in \mathbb{Z}(1, q), j \in \mathbb{Z}(q + 1, q + r)\},$$

to the invariant set of every node $i \in \mathcal{N}$. We will reserve the term "discrete component" of the state space for the first element of the state (i, x) which refers to a line number or a node on the graph. With a slight abuse of terminology, we will refer to the second component of the state variable in a graph model as the continuous component even if some of these variables are known to be binary variables. Alternatively one may choose to define the state space of a graph model as $X := \mathcal{N} \times \mathbb{R}^n \times \{-1, 1\}^q$ and refer to the binary variables as the third element of the state (i, x, v) , while the second component represents a truly continuous variable in \mathbb{R}^n . The two approaches are practically equivalent and the

choice does not change the outcome or the complexity of the verification process presented in this thesis.

5. The invariant sets X_i are meaningful elements of the model only if they are strict subsets of \mathbb{R}^n , otherwise they provide no nontrivial information about the behavior of the program. Note that the invariant sets are not essential elements of the model. They can be included if they are readily available or easily computable. For instance, the invariant sets may be generated in the following ways:

- (a) They are provided by the programmer and they represent his or her knowledge about the behavior of the program. The programmer provides these invariants to the program analyzer to include them in the graph model.
- (b) The invariant sets can be constructed from the passport and transition labels of the graph in the following way: Assume that T_{ij}^k is affine, deterministic, invertible, and the inverse is easily computable, then

$$X_i := \bigcup_{j \in \mathcal{I}(i), k \in \mathcal{A}_{ij}} \Pi_{ij}^k([T_{ij}^k]^{-1}) \quad (2.14)$$

is an invariant set for node i . Furthermore, assuming that for $j \in \mathcal{I}(i)$, the invariant sets X_j are available, then

$$X_i := \bigcup_{j \in \mathcal{I}(i), k \in \mathcal{A}_{ij}} \Pi_{ij}^k([T_{ij}^k]^{-1}) \cap X_j([T_{ij}^k]^{-1}). \quad (2.15)$$

is an invariant set for node i .

- (c) An alternative approach which can be helpful when T_{ij}^k is not invertible is to replace $\Pi_{ij}^k([T_{ij}^k]^{-1})$ and $X_j([T_{ij}^k]^{-1})$ in (2.14) and (2.15) by $\mathcal{R}(T_{ij}^k)$ (the range of T_{ij}^k). A very common case in programming is when T_{ij}^k maps certain variables to constants. When T_{ij}^k is neither invertible, nor a constant map, a possible approach would be to exploit convex optimization for computation of (an over-approximation of) $\mathcal{R}(T_{ij}^k)$ over $X_i \cap \Pi_{ij}^k$.

Graph models with non-smooth state-dependent or time-varying arcs.

In the graph model description that we introduced above, the transition and passport labels associated with the arcs can be time-varying or non-smooth functions of the state variables. A very common case is when some parameters (e.g. coefficients) of the transition functions are drawn from a finite set (e.g. a multidimensional array or data structure). The index variable that refers to an element from the array can be random (defining a time-varying transitions), or it can be a function of other state variables (defining a non-smooth state-dependent transition). For instance, consider the following fragment from a program with two variables: $x \in \mathbb{R}$, and $i \in \{1, 2\}$:

L1 : $x = A[i] x$;
 L2 : expression

Then, the state transition from node 1 to 2 is defined by:

$$T_{21} : (x, i) \rightarrow (A[i] x, i)$$

where A is an array of size 2. The map T_{21} as presented above is not readily in semialgebraic form. However, a procedure similar to the one used in construction of MILMs (cf. Proposition 2.1) can be used to construct a valid transition label which conforms with our framework. For the above example, this can be done in the following way:

$$T_{21}(x, i) = \{\bar{T}_{21}(x, i, v_1, v_2) \mid (x, i, v_1, v_2) \in S_{21}\}.$$

$$\bar{T}_{21}(x, i, v_1, v_2) \triangleq A[1] v_1 x + A[2] v_2 x.$$

$$S_{21} \triangleq \{(x, i, v_1, v_2) \mid v_1 + v_2 = 1, v_1, v_2 \in \{0, 1\}, v_1(i-1) + v_2(i-2) = 0\}.$$

In light of Proposition 2.1, generalization of the above technique to multidimensional arrays of arbitrary finite size can be done in a systematic way. However, the number of binary decision variables grows (linearly) with the number of elements in the array, which can be undesirable

for very large arrays. A conservative alternative would be to abstract the array by the upper and lower bounds on the magnitude of its elements.

$$T_{21}(x, i) : = \{\bar{T}_{21}(x, i, w) \mid (x, i, w) \in S_{21}\}.$$

$$\bar{T}_{21}(x, i, v_1, v_2) : = wx.$$

$$S_{21} : = \{(x, i, w) \mid \min_i A[i] \leq w \leq \max_i A[i]\}.$$

This approach provides a relatively inexpensive abstraction of complicated dependencies between variables. However, it can be very conservative in some applications. A third approach to constructing graph models with fixed (time-invariant) labels is to attempt to derive a fixed map by computing the net effect of several lines of code [31]. When applicable, the result is a fixed map which is obtained by taking the composition of several functions. This is an instance of a more general concept in computer science, namely, extracting the higher level semantics. Higher levels of semantic collection allow one to define more compact models of the software. However, this task is often very complex when it goes beyond pattern matching as information must be collected over several lines of code and then linked into a compact model.

For instance, it is not uncommon to encounter coding scenarios like the following:

L1 :	for (k = 1 ; k == N ; k++)	{
L2 :	y[k] = x[k]; x[k] = 0; }	
L3 :	for (i = 1 ; i == N ; i++)	{
L4 :	for (j = 1 ; j == N ; j++)	{
L5 :	x[i] = x[i] + y[j] * A[i][j];	
L6 :	}}	

Program 2-5.

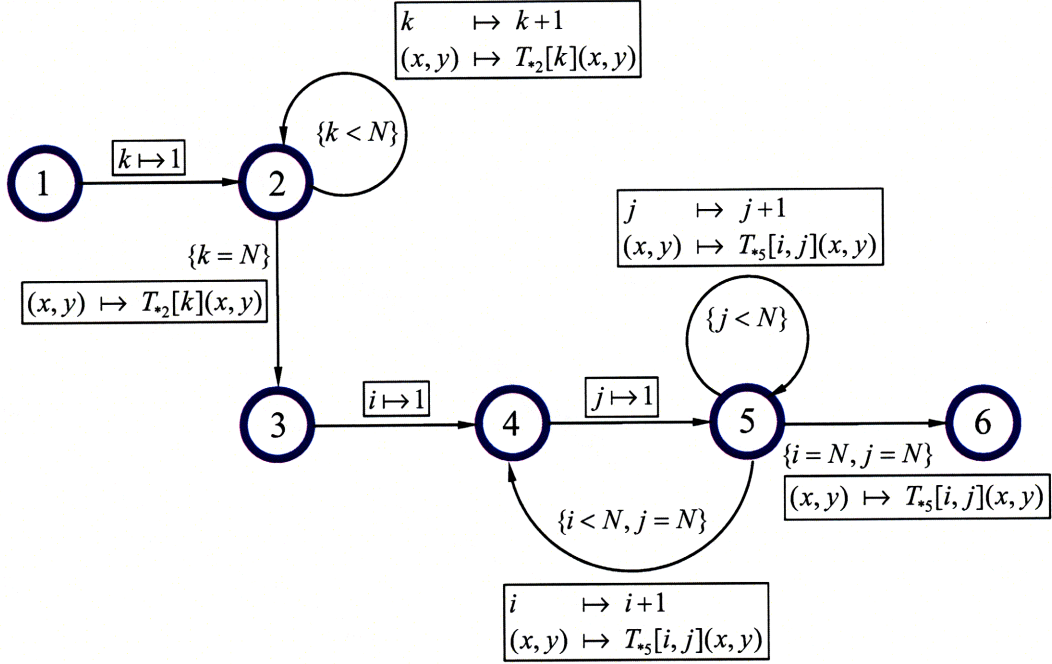


Figure 2-2: Graph model of a code fragment (Program 2-5) with time varying arc labels. The transition labels are shown in boxes and the passport labels are in brackets. For simplicity, only the non-identity transition labels are shown.

A graph model of this code fragment is shown in Figure 2-2, where:

$$y = [y[1] \dots y[N]]^T, \quad x = [x[1] \dots x[N]]^T$$

$$T_{*2}[k] : \begin{bmatrix} y \\ x \end{bmatrix} \rightarrow \begin{bmatrix} I - e_{kk} & e_{kk} \\ 0 & I - e_{kk} \end{bmatrix} \begin{bmatrix} y \\ x \end{bmatrix},$$

$$T_{*5}[i, j] : \begin{bmatrix} y \\ x \end{bmatrix} \rightarrow \begin{bmatrix} I & 0 \\ A_{ij}e_{ij} & I \end{bmatrix} \begin{bmatrix} y \\ x \end{bmatrix},$$

where e_{ij} is an $N \times N$ matrix which is zero everywhere except at the (i, j) -th entry which is 1, and A_{ij} denotes the (i, j) -th entry of A ($A_{ij} \equiv A[i][j]$) which is an $N \times N$ array in the code [31]. The remaining transition labels correspond to the counter variables i , j , k , and have been

specified on the graph in Figure 2-2. From node 1 to node 3, the net effect is:

$$\prod_{k=1}^N T_{*2}[k] = \begin{bmatrix} 0 & I \\ 0 & 0 \end{bmatrix},$$

and from node 3 to node 6 the net effect is:

$$\prod_{i=1}^N \prod_{j=1}^N T_{*5}[i, j] = \begin{bmatrix} I & 0 \\ A & 0 \end{bmatrix}.$$

In the sequel, in reference to the graph model of a computer program, we will use the concise notation $G(\mathcal{N}, \mathcal{E})$, with the convention that the nodes and arcs of G are appropriately labeled to define a valid model $\mathcal{S}(X, f, X_0, X_\infty)$. Note that the invariant sets X_i are not essential elements of the model. However, whenever they can be constructed, they can simplify program analysis and help us find stronger Lyapunov invariants. As we will see in Chapter 3 Lyapunov invariant candidates for graph models are in the form of $V(\tilde{x}) \equiv V(i, x) := \sigma_i(x)$, where for every $i \in \{\emptyset\} \cup \{1, \dots, m\} \cup \{\infty\}$, the function $\sigma_i : \mathbb{R}^n \rightarrow \mathbb{R}$ is a polynomial, quadratic or an affine functional.

The graph model that we have introduced in this section is quite generic. It is worthwhile to consider and study a subclass of the generic graph models: Linear Models with Conditional Switching.

Linear Models with Conditional Switching (LMwCS):

Linear Models with Conditional Switching are an important subclass of graph models. LMwCS are suitable for programs with simple linear flow. In particular, these include programs written with a combination of *while-loop*, *for-loop*, *goto*, and *if-then-else* commands with affine conditions and affine assignments. Examples of such programs can be found in embedded systems controlling real-time interactions between simple logic and gain scheduled linear systems. Similar to the generic graph models, in this model too, the state space of the system is the direct product of a discrete set and an n -dimensional Euclidean space:

$$X := \{\emptyset\} \cup \{1, 2, \dots, m\} \cup \{\infty\} \times \mathbb{R}^n$$

The set of initial and terminal states are defined by $X_0 := \{(\emptyset, X_\emptyset)\}$, and $X_\infty := \{(\infty, X_\infty)\}$, where X_\emptyset, X_∞ are selected polyhedral or second order subsets of \mathbb{R}^n . The set-valued state transition map $f : X \mapsto 2^X$ is defined by matrices $C_i, c_i, A_{ik}, B_{ik}, L_{ik}, k = 1, 2$, where $i \in \{\emptyset, 1, \dots, m\}$, as well as by functions $\theta_k : \{\emptyset, 1, \dots, m\} \mapsto \{\emptyset, 1, \dots, m\} \cup \{\infty\}, k = 1, 2$, according to the following rule:

$$f(i, x) = \{(\theta_1(i), A_{i1}x + B_{i1}w + L_{i1}) : w \in [-1, 1]^q\}$$

when $C_i x + c_i \leq 0$ and $i \neq \infty$,

$$f(i, x) = \{(\theta_2(i), A_{i2}x + B_{i2}w + L_{i2}) : w \in [-1, 1]^q\}$$

when $C_i x + c_i > 0$ and $i \neq \infty$. Finally, $f(i, x) = \{(\infty, x)\}$ when $i = \infty$.

Similar to generic graph models, Lyapunov invariant candidates for LMwCS are in the form of $V(\tilde{x}) \equiv V(i, x) := \sigma_i(x)$, where for every $i \in \{\emptyset\} \cup \{1, \dots, m\} \cup \{\infty\}$ the function $\sigma_i : \mathbb{R}^n \rightarrow \mathbb{R}$ is a quadratic or an affine functional.

LMwCS can be regarded as a special case of linear graph models, which are defined as a special subclass of graph models where all the transformations, sets, and conditions are affine. The elements of a linear graph model are:

1. A set of nodes $\mathcal{N} := \{\emptyset\} \cup \{1, \dots, m\} \cup \{\infty\}$.
2. A set of arcs $\mathcal{E} := \{(i, j, k) \mid i \in \mathcal{N}, j \in \mathcal{O}(i)\}$.
3. A set of program variables $x_q \in \mathbb{R}, q \in \mathbb{Z}(1, n)$. Given \mathcal{N}, n , the state space is defined as: $X := \mathcal{N} \times \mathbb{R}^n$.
4. A set of *transition* labels T_{ji}^k , assigned to every arc $(i, j, k) \in \mathcal{E}$, where $T_{ji}^k : \mathbb{R}^n \rightarrow \mathbb{R}^n$ is defined in the following way:

$$T_{ji}^k : x \mapsto A_{ji}^k x + B_{ji}^k w + L_{ji}^k, w \in [-1, 1]^q.$$

5. A set of *passport* labels Π_{ji}^k , assigned to every arc $(i, j, k) \in \mathcal{E}$:

$$\Pi_{ji}^k := \{x \mid C_{ji}^k x + c_{ji}^k \leq 0, D_{ji}^k x - d_{ji}^k = 0\}$$

State transition along the arc (i, j, k) is possible if and only if $x \in \Pi_{ji}^k$.

6. A set of polyhedral invariant sets $X_i \subset \mathbb{R}^n$, $i \in \mathcal{N}$, assigned to every node on the graph, such that $s_l(i, x) \in X_i$. The invariant sets are of the form:

$$X_i := \{x \mid x^T Q_i x \leq 1, S_i x - s_i \leq 0, H_i x - h_i = 0\}.$$

2.3 Specifications

In this section, we present mathematical definitions for the specification that we consider in this document. The specifications that can be verified via our framework are:

- **Safety:** The property that a certain subset of the state space will never be reached.
- **Finite-time termination:** The property that all of the trajectories will enter a certain subset of the state space in finite-time. This property is sometimes referred to as the liveness property.

It will be shown in Chapter 3 that different variations of Lyapunov invariants satisfying certain technical conditions can be formulated for verification of these specifications.

2.3.1 Safety

Definition 2.3 Consider a program \mathcal{P} and its dynamical system representation $\mathcal{S}(X, f, X_0, X_\infty)$. Program \mathcal{P} is said to satisfy the safety property with respect to a certain subset $X_- \subset X$, if for every trajectory $\mathcal{X} \equiv x(\cdot)$ of (2.1):

$$x(0) \in X_0 \subseteq X, \quad x(t+1) \in f(x(t)) \quad \forall t \in \mathbb{Z}_+,$$

and for every $t \in \mathbb{Z}_+$, $x(t)$ does not belong to X_- .

Several critical specifications associated with runtime errors can be defined as special cases of the safety specification.

Overflow

An overflow runtime error occurs when a variable exceeds its available dynamic range. For instance, in standard C programs running on 16-bit processors, the type *int* defines an unsigned integer that takes integer values between -2^{15} and $2^{15} - 1$. For simplicity, we often make this interval symmetric and assume that the overflow limit of an unsigned integer is given by $2^{15} - 1 = 32767$. Similarly, on a 16-bit machine, the symmetric overflow limits for variables of the type *double* and *float* are 1.7×10^{308} and 3.4×10^{38} . As it can be observed, the overflow limits for *double* and *float* variables are extremely large numbers. Our framework exploits convex optimization tools to find the Lyapunov invariants that certify the specifications. If these extremely large values are embedded in the optimization problem, they will lead to poorly conditioned matrices and numerical inaccuracies that render the process either infeasible or erroneous. As a result, much smaller real numbers (usually not larger than 10^8 in order of magnitude) must be used to specify the overflow limit. Although this may seem a very conservative remedy, it has little if any practical consequence, as the variables in a well-written program in a safety-critical application are typically not expected to be extremely large numbers.

The absence of overflow specification can be characterized as a special case of the unreachability specification by specifying X_- in Definition 2.3 in the following way:

$$X_- := \{x \in X \mid |x_i| \geq \alpha_i\},$$

where α_i is the overflow limit for variable x_i . Equivalently, we can write:

$$X_- := \{x \in X \mid \|\alpha^{-1}x\|_\infty \geq 1\}.$$

where $\alpha = \text{diag}\{\alpha_i\}$ is a diagonal positive definite matrix specifying the overflow limit.

Remark 2.2 *Overflow in modulo arithmetic does not lead to a runtime error, rather, there will be a rollover into the same set. In safety-critical applications a rollover is an equally*

dangerous scenario, as it may lead to an extreme distortion of performance or to an abrupt and unpredictable change in the behavior of the system. Therefore, we consider a variable overflow an unsafe event, whether it leads to an actual runtime error or to a rollover.

Out-of-Bounds Array Indexing An out-of-bounds array indexing error occurs when an integer variable exceeding the length of an array references an element of the array. The “absence of out-of-bounds array indexing” specification is very similar in nature to the absence of overflow specification. Assuming that x_k is an integer that is used for indexing an array of length l , we can define:

$$X_- := \{x \in X \mid |x_k| > l\}.$$

If it can be proven that X_- is unreachable, then the specification is provably satisfied. Consider now the safe situation in which x_k can exceed the array length l at other locations in the program, but never at the locations where it references to the array element. In such situations, the graph models are more suitable for verification of the specification that x_k does not exceed l at location i . Using graph models we can define:

$$X_- := \{(i, x) \in X \mid |x_k| > l\}$$

and prove that X_- is unreachable. This is similar to assertion checking which we will discuss next.

Program Assertions

An *assertion* is a True-or-False statement that a programmer may insert at certain locations in the computer code to indicate his or her expectation from the behavior of the program. More specifically, the type of program assertions that we consider are in the form of (semialgebraic) set membership ($x \in X_a$) or set exclusion ($x \notin X_a$), immediately before the execution of a specific line of code. Since assertions must hold only at certain locations (that is, the property $x \in X_a$ or $x \notin X_a$ is not necessarily invariant throughout the execution of the program) graph models are most suitable for assertion checking. Using graph models, assertion checking can be

characterized as a special case of the safety (unreachability) specification:

at location i : assert $x \in X_a \Rightarrow$ define $X_- := \{(i, x) \in X \mid x \in X \setminus X_a\}$

at location i : assert $x \notin X_a \Rightarrow$ define $X_- := \{(i, x) \in X \mid x \in X_a\}$

Division-by-Zero A division-by-zero runtime error occurs when the value of the *divisor* variable becomes zero at the exact time when the division instruction is being executed. This can cause the program to abort, or continue with an unknown value registered as the result of the division operation. A conservative approach to ruling out this scenario would be to prove that $x \neq 0$ is an invariant property of the program. However, this can be quite conservative as the value of x could safely be equal to zero at other lines of the code which do not include a divide-by- x instruction. This problem is therefore, better addressed as an assertion checking problem over graph models with

$$X_- := \{(i, x) \in X \mid x \in X_a\}$$

where $X_a = \{0\}$, and i is the line number containing the divide-by- x instruction.

Square Root or Logarithm of Negative Numbers A variable of questionable sign should never be passed to a real-valued function that computes the square root (or logarithm, 4th root, etc...) of nonnegative numbers. Ruling out these unsafe events is very similar to assertion checking for division-by-zero. For the square root we must define:

$$X_- := X_- := \{(i, x) \in X \mid x < 0\},$$

and for the logarithm we must define:

$$X_- := X_- := \{(i, x) \in X \mid x \leq 0\}.$$

While certain program assertions are defined by the programmer, the safety assertions for division-by-zero or taking the square root (or logarithm) of negative numbers are standard assertions that must be automatically specified and verified.

Program Invariants

A program invariant is a property that always holds during the program execution. The property is often described as a semialgebraic relation between certain variables of the program. Equivalently, the program variables must always belong to a certain semialgebraic subset X_I of the state space. In reference to Definition 2.3, this is equivalent to unreachability of $X \setminus X_I$. Program invariants can be viewed as assertion that hold at all locations rather than one or few specific locations. Essentially, any method that can be used for verifying unreachability of X_- can be applied for verifying invariance of X_I by defining $X_- := X \setminus X_I$. Similarly, any method that can be used for verifying invariance of X_I can be used for verifying unreachability of X_- by defining $X_I := X \setminus X_-$.

2.3.2 Program Termination in Finite Time

Finite-time termination is the property that all the trajectories will enter the subset X_∞ in finite time.

Definition 2.4 Consider a program \mathcal{P} and its dynamical system representation $\mathcal{S}(X, f, X_0, X_\infty)$. Program \mathcal{P} is said to terminate in finite time if every solution $\mathcal{X} \equiv x(\cdot)$ of (2.1):

$$x(0) \in X_0 \subseteq X, \quad x(t+1) \in f(x(t)) \quad \forall t \in \mathbb{Z}_+,$$

satisfies $x(t) \in X_\infty$ for some $t \in \mathbb{Z}_+$.

2.4 The Implications of Abstractions

So far in this chapter, we have introduced several dynamical system models for computer programs and we have given mathematical definition of the properties that we would like to prove. The dynamical system models that we have introduced include both abstract and concrete models. In practice, however, the verification task is often performed for the abstract models of computer programs. For mathematical correctness, in this section we prove that if an abstract model satisfies the safety and liveness specifications then the actual program satisfies those specification as well.

Proposition 2.2 Consider a program \mathcal{P} and its dynamical system model $\mathcal{S}(X, f, X_0, X_\infty)$. Let $\overline{\mathcal{S}}(\overline{X}, \overline{f}, \overline{X}_0, \overline{X}_\infty)$ be an abstraction of \mathcal{P} . Let $X_- \subset X$, and $\overline{X}_- \subset \overline{X}$, representing the unsafe regions of \mathcal{S} and $\overline{\mathcal{S}}$ respectively be such that $X_- \subseteq \overline{X}_-$. Assume that the safety property w.r.t. \overline{X}_- has been certified for the abstract model of \mathcal{P} . Then, \mathcal{P} satisfies the safety property w.r.t. X_- . In addition, if the finite-time termination property has been certified for the abstract model, then \mathcal{P} terminates in finite time.

Proof. First, consider the safety property. Assume the contrary, that is, \mathcal{P} does not satisfy safety w.r.t. X_- . Then, there exists a solution $\mathcal{X} \equiv x(\cdot)$ of $\mathcal{S}(X, f, X_0, X_\infty)$, and a positive integer $t_- \in \mathbb{Z}_+$ such that $x(0) \in X_0$, and $x(t_-) \in X_-$. It follows from $X_0 \subseteq \overline{X}_0$, and $f(x) \subseteq \overline{f}(x)$ that $\mathcal{X} \equiv x(\cdot)$ is also a solution of $\overline{\mathcal{S}}(\overline{X}, \overline{f}, \overline{X}_0, \overline{X}_\infty)$. Therefore, we have:

$$x(t_-) \in X_- , X_- \subseteq \overline{X}_- \Rightarrow x(t_-) \in \overline{X}_-$$

However, $x(t_-) \in \overline{X}_-$ contradicts the fact that $\overline{\mathcal{S}}(\overline{X}, \overline{f}, \overline{X}_0, \overline{X}_\infty)$ satisfies safety w.r.t. \overline{X}_- . Proof of the finite time termination property is similar: let $\mathcal{X} \equiv x(\cdot)$ be any solution of $\mathcal{S}(X, f, X_0, X_\infty)$. Since $\mathcal{X} \equiv x(\cdot)$ is also a solution of $\overline{\mathcal{S}}(\overline{X}, \overline{f}, \overline{X}_0, \overline{X}_\infty)$, it follows that there exists $t_T \in \mathbb{Z}_+$ such that $x(t_T) \in \overline{X}_\infty$. Since $x(t_T)$ is also an element of X , it follows that $x(t_T) \in \overline{X}_\infty \cap X$. Since by definition $\overline{X}_\infty \cap X \subset X_\infty$ holds, we must have $x(t_T) \in X_\infty$. This proves that \mathcal{P} terminates in finite time. ■

2.5 Summary

In this chapter we developed and presented the first element of the software analysis framework that is introduced in this dissertation. We discussed our interpretation of numerical computer programs as dynamical systems and introduced generic dynamical system representations that formalize this interpretation. We also introduced specific modeling languages as special cases of the generic representations. These include the MILM, the graph models, and the LMwCS. These models can represent a broad range of computer programs of interest to the control community. Furthermore, these models provide a convenient platform for analysis of software via systems and control theoretic tools in an automated or semi-automated framework. The

dynamical system models, whether generic or specific, can be concrete or abstract. We showed how to construct abstract models for computer programs involving common nonlinearities. These abstract models have the potential to simplify the analysis significantly. We further presented mathematical definitions for the liveness, safety, and other performance specifications that are considered in this document. Finally, we showed that safety and liveness properties of the abstract model can be carried over to the actual program. In the next chapter, we will introduce Lyapunov invariants as behavior certificates for the mathematical models of software that we introduced in this chapter.

Chapter 3

Lyapunov Invariants as Behavior Certificates

In this Chapter, we introduce Lyapunov invariants as certificates for the behavioral properties of numerical computer programs. A Lyapunov invariant is a real-valued function of the program variables that satisfies a *difference inequality* along the trajectories of a computer program. We demonstrate that different variations of Lyapunov invariants satisfying certain technical conditions can be formulated for verification of safety and liveness specifications that were defined in Section 2.3. We have chosen the terminology *Lyapunov invariant* instead of *Lyapunov function* to convey that the structure of these functions is different from the standard Lyapunov functions. In particular, the zero level set of a Lyapunov invariant defines an invariant set for the variables of the computer program. Also, a Lyapunov invariant is not required to be nonnegative (or bounded from below). In some cases, a Lyapunov invariant may not even be monotonically decreasing. Moreover, the level sets of a Lyapunov invariant that proves the safety properties of a dynamical system are not necessarily bounded closed curves. Numerical computation of the Lyapunov invariants via convex optimization methods is the topic of Chapter 4.

3.1 Preliminaries

A Lyapunov function in its standard form is a function which is non-increasing along the trajectories of a dynamical systems and is positive everywhere except at the equilibrium point

where it is zero. In mathematical terms, a function $V : X \rightarrow \mathbb{R}_+$ is a Lyapunov function for the dynamical system $\dot{x} = f(x)$ if V satisfies:

$$\begin{aligned} V(x) &> 0 : \forall x \in X \setminus \{x^*\}, V(x^*) = 0, \\ \frac{D}{Dt}V(x(t)) &< 0 : \forall x \in X \setminus \{x^*\}. \end{aligned}$$

It can be shown that under some technical conditions, if such function exists, then $x = x^*$ is an asymptotically stable equilibrium point of the dynamical system $\dot{x} = f(x)$. Many variations of the above definition exist (e.g. with non-strict inequalities, or with $X = \mathcal{B}(x^*, \varepsilon)$ versus $X = \mathbb{R}^n$), which can be used to prove different types of stability, e.g. local or global asymptotic stability, or stability in the sense of Lyapunov. Extensions of this notion to the input-output stability analysis of nonlinear systems, or to the analysis of systems with parameter and/or dynamical uncertainties have also been considered. Furthermore, most of the existing results on the stability analysis of nonlinear systems can be shown to have an interpretation in terms of Lyapunov functions, even if a Lyapunov function does not appear explicitly in the stability criteria. More detailed discussion of Lyapunov functions in stability analysis of nonlinear systems can be found for instance in [48].

Inspired by the concept of Lyapunov functions in analysis of dynamical systems, we introduce Lyapunov invariants for analysis of computer programs. Analogous ideas have been used in [77, 80] for safety verification of hybrid systems, and in [17], [47] for stability analysis of switched hybrid system.

Definition 3.1 *A rate θ Lyapunov invariant for the dynamical system model $\mathcal{S}(X, f, X_0, X_\infty)$ is defined to be a function $V : X \mapsto \mathbb{R}$ such that*

$$V(x_+) - \theta V(x) < 0 \quad \forall x \in X, x_+ \in f(x) : x \notin X_\infty. \quad (3.1)$$

where $\theta > 0$ is a constant. Thus, a rate θ Lyapunov invariant satisfies a difference inequality ($V(x_+) - \theta V(x) < 0$) along the trajectories of \mathcal{S} (as defined in (2.1)) until they reach a terminal state. Note that according to this definition, depending on the initial conditions and the constant θ , a Lyapunov invariant may or may not monotonically decrease along the trajectories of \mathcal{S} . For

instance, while adding the constraint $V(x) < 0, \forall x \in X_0$ to (3.1) implies that $V(x)$ is negative along the trajectories of \mathcal{S} , $V(x)$ may not be monotonic if θ is less than 1, and $V(x)$ will be monotonic along the trajectories of \mathcal{S} if θ is greater than or equal to 1.

The above definition is generic and applies to any software model $\mathcal{S}(X, f, X_0, X_\infty)$, in particular, it applies to MILMs and to graph models. It is worth exploring the interpretation of (3.1) for each of these models. Before we proceed, we present the following proposition which essentially states that if a function is a Lyapunov invariant for an abstract model of a program, it is also a Lyapunov invariant for the actual program.

Proposition 3.1 *Let $\mathcal{S}(X, f, X_0, X_\infty)$ be a concrete dynamical system representation of a computer program \mathcal{P} . Let $\mathcal{S}(\bar{X}, \bar{f}, \bar{X}_0, \bar{X}_\infty)$ be an abstract representation of \mathcal{P} , and let V be a rate θ Lyapunov invariant for $\mathcal{S}(\bar{X}, \bar{f}, \bar{X}_0, \bar{X}_\infty)$. Then V is a rate θ Lyapunov invariant for $\mathcal{S}(X, f, X_0, X_\infty)$.*

Proof. We are given that:

$$V(x_+) - \theta V(x) < 0 \quad \forall x \in \bar{X}, x_+ \in \bar{f}(x) : x \notin \bar{X}_\infty, \quad (3.2)$$

and we need to show that:

$$V(x_+) - \theta V(x) < 0 \quad \forall x \in X, x_+ \in f(x) : x \notin X_\infty.$$

First, we show that

$$X \setminus X_\infty \subseteq \bar{X} \setminus \bar{X}_\infty. \quad (3.3)$$

Assume for the sake of contradiction that (3.3) is not true. Then there exists $x \in X$ such that $x \notin X_\infty$, and $x \in \bar{X}_\infty$. Therefore, $x \in X \cap \bar{X}_\infty$ and $x \notin X_\infty$, which contradicts (2.2). Now, let $(x, x_+) \in (X \setminus X_\infty) \times f(x)$. It follows from (3.3) and $f(\cdot) \subseteq \bar{f}(\cdot)$ that $(x, x_+) \in (\bar{X} \setminus \bar{X}_\infty) \times \bar{f}(x)$. (3.2) then implies $V(x_+) - \theta V(x) < 0$. ■

If a Lyapunov invariant which proves liveness and/or safety properties for an abstract model can be found, then by Proposition 2.2 these specifications are valid for the actual code, and Proposition 3.1 may not be needed. Proposition 3.1, however, validates the Lyapunov invariants

(of the abstract models) on the actual computer code. The importance of this result is that due to the finiteness property of the state space of a concrete model, certain Lyapunov invariants may prove strong properties (such as finite-time termination) for the actual program but not for the abstract model. Furthermore, in light of Proposition 3.1, we can refer to Lyapunov invariants for a computer program and its abstract model indifferently, which is important for mathematical correctness, and convenient for presentation of the material in the remaining sections of this chapter.

3.1.1 Lyapunov Invariants for MILMs

The following proposition applies Definition 3.1 to MILMs. The proof is straightforward by inspection.

Proposition 3.2 *Consider a program \mathcal{P} and its MILM $\mathcal{S}(F, H, X_0, n, q, r)$. The function $V : [-1, 1]^n \rightarrow \mathbb{R}$ is a Lyapunov invariant for \mathcal{P} in the sense of Definition 3.1 if V satisfies:*

$$V(Fx_e) - \theta V(x) < 0 \quad \forall (x, x_e) \in [-1, 1]^n \times \Xi,$$

where

$$\Xi = \left\{ (x, w, v, 1) \mid H[x \ w \ v \ 1]^T = 0, (w, v) \in [-1, 1]^q \times \{-1, 1\}^r \right\}.$$

3.1.2 Lyapunov Invariants for Graph Models

Recall that the state space of a graph model is the direct product of a discrete set \mathcal{N} and an n -dimensional Euclidean space \mathbb{R}^n , and that the state of this model has two components: a discrete component which is an element of \mathcal{N} and a continuous component which is an element of \mathbb{R}^n . We define Lyapunov invariants for graph models in the following way:

$$V(\tilde{x}) \equiv V(i, x) := \sigma_i(x) \tag{3.4}$$

where, for every $i \in \mathcal{N}$ the function $\sigma_i : \mathbb{R}^n \rightarrow \mathbb{R}$ is a polynomial, quadratic, or an affine functional. In other words, a Lyapunov function is assigned to every node $i \in \mathcal{N}$ on the graph $G(\mathcal{N}, \mathcal{E})$. The proof of the following proposition is straightforward by inspection.

Proposition 3.3 Consider a computer program \mathcal{P} and its graph model $G(\mathcal{N}, \mathcal{E})$. The function $V : \mathcal{N} \times \mathbb{R}^n \rightarrow \mathbb{R}$, where $V(i, x) := \sigma_i(x)$, is a Lyapunov invariant for \mathcal{P} in the sense of Definition 3.1 if

$$\sigma_j(x_+) - \theta \sigma_i(x) < 0, \quad \forall (i, j, k) \in \mathcal{E}, \quad (x, x_+) \in \Pi_{ji}^k \times T_{ji}^k x. \quad (3.5)$$

Since all the regions of $X \setminus X_\infty$ may not be reachable, strictly speaking, it is not necessary to require (3.1) to hold on the entire $X \setminus X_\infty$. However, requiring (3.1) to hold only on the reachable regions of the state space leads to non-convex conditions on V , unless the reachable regions (or an over-approximation of them) are known and fixed a priori (this issue is discussed in further detail in Section 4.1). If the invariant sets corresponding to the nodes of the graph model are nontrivial subsets of the state space, this information can be included in (3.5), which would result in less restrictive criteria. Hence, there is a better chance that a Lyapunov invariant can be found this way. This approach is reflected in Proposition 3.4, which will be presented shortly.

Strictly speaking, the rate θ does not have to be a constant and it can be a positive definite function of x . However, since simultaneously searching for both $\theta(x)$ and $V(x)$ leads to non-convex conditions, it is generally hard to take advantage of the additional flexibility that a variable-rate Lyapunov invariant may provide compared to a constant-rate Lyapunov invariant; unless a good candidate function for $\theta(\cdot)$ is known a priori. A situation in which a non-constant rate $\theta(\cdot)$ can be logically chosen is in analysis of graph models. By assigning different θ 's to different arcs on the graph, we allow $\theta(\cdot)$ to be a specific function of the state. This function is an explicit function of the discrete component of the state, i.e. node number, and an implicit function of the continuous component of the state. While computing the optimal value of θ per arc is neither possible nor necessary, depending on the state transitions along the arcs, certain choices of θ may be more reasonable than others.

We present the following proposition, which summarizes the above two paragraphs.

Proposition 3.4 Consider a computer program \mathcal{P} and its graph model $G(\mathcal{N}, \mathcal{E})$. The function

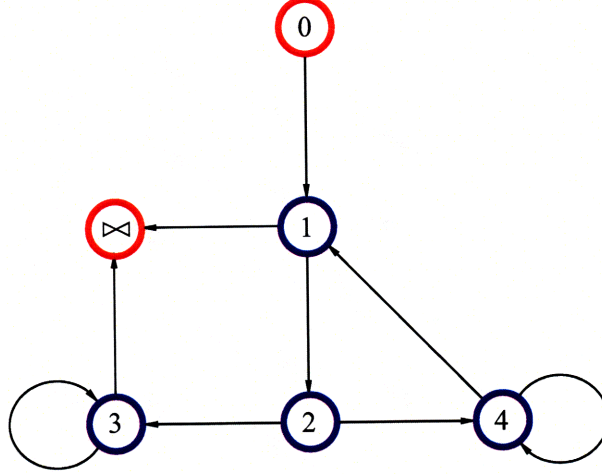


Figure 3-1: A graph model. There is an invariant set X_i assigned to each node. A transition label T_{ji} and a passport label Π_{ji} is assigned to each arc (i, j) from node i to node j .

$V : \mathcal{N} \times \mathbb{R}^n \rightarrow \mathbb{R}$ defined according to (3.4) is a Lyapunov invariant for \mathcal{P} in the sense that

$$V(\tilde{x}_+) < \theta(\tilde{x})V(\tilde{x}) \quad \forall \tilde{x} \in \mathcal{X}_r, \tilde{x}_+ \in f(\tilde{x}), \mathcal{X}_r := \bigcup_{i \in \mathcal{N} \setminus \{\infty\}} (i, X_i) \quad (3.6)$$

if

$$\sigma_j(x_+) - \theta_{ji}^k \sigma_i(x) < 0, \quad \forall (i, j, k) \in \mathcal{E}, (x, x_+) \in (X_i \cap \Pi_{ji}^k) \times T_{ji}^k x \quad (3.7)$$

Remark 3.1 A non-strict version of (3.1) can also be considered. In particular, if $\theta > 1$, replacing the strict inequality in (3.1) with the non-strict version has no theoretical or practical consequences on the results that will be derived later in this chapter. Some minor technicalities arise if $\theta = 1$ and (3.1) is replaced with the non-strict version. We will discuss how the criteria that we develop for safety and/or finite-time termination must be modified for the non-strict case as we present them. In particular, for the graph models, it is convenient to allow some of the strict inequalities in (3.7) or (3.5) be replaced with non-strict versions.

Let $a(G) := \sum_{(i,j,\cdot) \in \mathcal{E}} |\mathcal{A}_{ji}|$ denote the total number of arcs on $G(\mathcal{N}, \mathcal{E})$, excluding the arc $(\infty, \infty, 1)$. Then, according to Proposition 3.4, the Lyapunov condition (3.6) is equivalent to $a(G)$ constraints imposed on $|\mathcal{N}|$ functions $\sigma_i(\cdot)$ corresponding to the nodes of G .

Example 3.1 Consider the graph model $G(\mathcal{N}, \mathcal{E})$ in Figure (3-1), and suppose that for all $(i, j) \in \mathcal{E}$, $T_{ij} : \mathbb{R}^n \rightarrow \mathbb{R}^n$ is an affine function. Let $\sigma_i : \mathbb{R}^n \rightarrow \mathbb{R}$, $i \in \mathcal{N}$ be a collection of functions. Then (3.6) holds if:

- (1) $\sigma_1(x) < \sigma_0(x)$, $\forall x \in X_0$
- (2) $\sigma_2(T_{21}x) < \theta_{21}\sigma_1(x)$, $\forall x \in X_1 \cap \Pi_{21}$
- (3) $\sigma_3(T_{32}x) < \theta_{32}\sigma_2(x)$, $\forall x \in X_2 \cap \Pi_{32}$
- (4) $\sigma_4(T_{42}x) < \theta_{42}\sigma_2(x)$, $\forall x \in X_2 \cap \Pi_{42}$
- (5) $\sigma_{\kappa}(T_{\kappa 3}x) < \theta_{\kappa 3}\sigma_3(x)$, $\forall x \in X_3 \cap \Pi_{\kappa 3}$
- (6) $\sigma_3(T_{33}x) < \theta_{33}\sigma_3(x)$, $\forall x \in X_3 \cap \Pi_{33}$
- (7) $\sigma_4(T_{44}x) < \theta_{44}\sigma_4(x)$, $\forall x \in X_4 \cap \Pi_{44}$
- (8) $\sigma_1(T_{14}x) < \theta_{14}\sigma_4(x)$, $\forall x \in X_4 \cap \Pi_{14}$
- (9) $\sigma_{\kappa}(T_{\kappa 1}x) < \theta_{\kappa 1}\sigma_1(x)$, $\forall x \in X_1 \cap \Pi_{\kappa 1}$

3.2 Behavior Certificates

In this section, we show that different variations of Lyapunov invariants satisfying certain technical conditions can be formulated to prove liveness and safety specifications.

3.2.1 Liveness

Models with Finite State-Space

Proposition 3.5 Consider a program \mathcal{P} , and its dynamical system model $\mathcal{S}(X, f, X_0, X_\infty)$. Suppose that the function $V : X \mapsto \mathbb{R}$, is a rate 1 Lyapunov invariant for \mathcal{S} , that is:

$$V(x_+) - V(x) < 0 \quad \forall x \in X \setminus X_\infty, x_+ \in f(x). \quad (3.8)$$

If the state space X is a finite set, then \mathcal{P} terminates in finite time.

Proof. Since X is a finite set, the function V can take only finitely many values. In particular, V can take only finitely many values over the set $X \setminus X_\infty$. The rest of the proof proceeds by contradiction: assume that \mathcal{P} does not terminate in finite time. Then there exists a sequence $\mathcal{X} \equiv (x(0), x(1), \dots, x(t), \dots) \in \mathcal{S}$ with the following property:

$$\forall t \in \mathbb{Z}_+, \quad x(t) \in X \setminus X_\infty \quad (3.9)$$

Now, consider the sequence $\{V(x(t))\}$, $t = 0, 1, 2, \dots$. According to (3.9) and (3.8), $\{V(x(t))\}$ is a strictly monotonically decreasing sequence. Therefore, it must take infinitely many different values. This contradicts the fact that V can take values only within a finite set. ■

Remark 3.2 *The above proof relies on the fact that $V : X \mapsto \mathbb{R}$ can take only finitely many values. As long as this condition holds, Proposition 3.5 remains valid even if the entire state space is not finite. For instance, if a rate 1 Lyapunov invariant V happens to be only a function of a subset of the program variables that take values in a finite set, then V is a certificate for finite time termination, even though the whole state space may be infinite. This is useful for proving termination of loops where the iterations are controlled by integer counters, while other variables in the program (or the abstraction of them) may be real variables.*

The result of Proposition 3.5 holds whether the dynamical system model $\mathcal{S}(X, f, X_0, X_\infty)$ is concrete or abstract (see Proposition 3.7). Although the proof of the Proposition relies on the finiteness property of the state space, in light of Proposition 3.1, the result remains valid even if $\mathcal{S}(X, f, X_0, X_\infty)$ is an abstract model, perhaps with an infinite state space.

Example 3.2 *Consider the IntegerDivision program presented in Example 2.1. The Function $V : X \rightarrow \mathbb{R}$, defined according to:*

$$V : (dd, dr, q, r) \mapsto r$$

is a rate 1 Lyapunov invariant for IntegerDivision; at every step, V decreases by $dr > 0$. Since X is finite, the program IntegerDivision terminates in finite time. This, however, does not prove that the termination is safe. It only proves absence of infinite loops. The program could terminate with an overflow.

Models with Infinite State-Space

In the following proposition, we present a finite-termination criterion which is applicable to both finite and infinite state space cases.

Proposition 3.6 *Consider a program \mathcal{P} , and its dynamical system model $\mathcal{S}(X, f, X_0, X_\infty)$, and assume that $\theta > 1$. If there exists a rate θ Lyapunov invariant $V : X \mapsto \mathbb{R}$, uniformly*

bounded on X , satisfying

$$V(x) < 0 \quad \forall x \in X_0 \quad (3.10a)$$

$$V(x_+) - \theta V(x) < 0 \quad \forall x \in X \setminus X_\infty, x_+ \in f(x), \quad (3.10b)$$

then \mathcal{P} terminates in finite time.

Proof. Note that (3.10a) and (3.10b) imply that V is negative-definite along the trajectories of \mathcal{S} . Let \mathcal{X} be any solution of \mathcal{S} . Since V is uniformly bounded on X , we have:

$$\exists M \in \mathbb{R}_+, \text{ s.t. } -M \leq V(x(t)) < 0, \forall x(t) \in \mathcal{X}.$$

Now, assume that there exists a sequence $\mathcal{X} \equiv (x(0), x(1), \dots, x(t), \dots)$ of elements from X satisfying (3.10) that does not reach a terminal state in finite time. That is, $x(t) \notin X_\infty$, $\forall t \in \mathbb{Z}_+$. Then, if

$$t > \frac{\log M - \log |V(x(0))|}{\log \theta}, \quad (3.11)$$

there must hold $V(x(t)) < -M$, which contradicts bounded-ness of V . ■

Remark 3.3 *In Proposition 3.6, the inequalities in either (3.10a) or (3.10b) (but not both) can be replaced by the non-strict versions.*

While the state space of a computer program is always finite, we often search for Lyapunov invariants for real-valued abstractions of computer programs since working with exact models in discrete state spaces is more difficult. The following proposition is interesting as it states that a rate 1 Lyapunov invariant for an abstract model is a certificate for finite-time termination of the actual program, although it may not be a certificate for finite-time termination of the abstract model.

Proposition 3.7 *Let \mathcal{P} be a computer program and let $\mathcal{S}(X, f, X_0, X_\infty)$ and $\mathcal{S}(\overline{X}, \overline{f}, \overline{X}_0, \overline{X}_\infty)$ be an exact and an abstract representation of \mathcal{P} respectively. Suppose that X is a finite set and that the function $V : \overline{X} \rightarrow \mathbb{R}$ is a rate 1 Lyapunov invariant for $\mathcal{S}(\overline{X}, \overline{f}, \overline{X}_0, \overline{X}_\infty)$. Then V is a certificate for finite-time termination of \mathcal{P} .*

Proof. Proposition 3.1 implies that V is also a rate 1 Lyapunov invariant for the exact model $\mathcal{S}(X, f, X_0, X_\infty)$. Since the exact model has a finite state space, Proposition 3.5 implies that \mathcal{P} terminates in finite time. ■

Note that when \overline{X} is not a finite set, existence of $V : \overline{X} \rightarrow \mathbb{R}$ satisfying conditions of Proposition 3.7 does not prove finite-time termination property of the abstract model.

Liveness Analysis of Graph Models

In this section we study variations of Propositions 3.5 and 3.6 for liveness analysis with variable-rate Lyapunov invariants defined on graph models. Before we proceed, we present the following definition.

Definition 3.2 *A cycle \mathcal{C}_m on a graph $G(\mathcal{N}, \mathcal{E})$ is an ordered list of m triplets $(n_1, n_2, k_1), (n_2, n_3, k_2), \dots, (n_m, n_{m+1}, k_m)$, where $n_{m+1} = n_1$, and $\forall j \in \mathbb{Z}(1, m)$ we have $(n_j, n_{j+1}, k_j) \in \mathcal{E}$. A simple cycle is a cycle that does not visit any node more than once, except for the first and the last nodes. Thus, a simple cycle is a cycle with the following property:*

$$\begin{aligned} & \text{If } (n_i, n_{i+1}, k_i) \in \mathcal{C}_m \text{ and } (n_j, n_{j+1}, k_j) \in \mathcal{C}_m \text{ then} \\ & n_{j+1} = n_i \implies i = 1 \text{ and } j = m \end{aligned}$$

Proposition 3.8 *Consider a program \mathcal{P} and its graph model $G(\mathcal{N}, \mathcal{E})$. Let $V(i, x) := \sigma_i(x)$ be a variable-rate Lyapunov invariant defined on G , satisfying*

$$\sigma_\emptyset(x) < 0, \quad \forall x \in X_\emptyset \tag{3.12a}$$

$$\sigma_j(x_+) - \theta_{ji}^k \sigma_i(x) < 0, \quad \forall (i, j, k) \in \mathcal{E}, (x, x_+) \in (X_i \cap \Pi_{ji}^k) \times T_{ji}^k x \tag{3.12b}$$

In addition, assume that V is bounded from below. Then, (3.12) proves that \mathcal{P} terminates in finite time if and only if for every simple cycle $\mathcal{C} \in G$, we have:

$$\prod_{(i,j,k) \in \mathcal{C}} \theta_{ij}^k > 1, \quad \mathcal{C} \in G \tag{3.13}$$

Proof. Proof of sufficiency proceeds by contradiction. Assume that (3.12) and (3.13) hold, but \mathcal{P} does not terminate in finite time. Then, there exists a sequence $\mathcal{X} \equiv (\tilde{x}(0), \tilde{x}(1), \dots, \tilde{x}(t), \dots)$ of elements from X satisfying (2.1) that does not reach a terminal state in finite time. Let $S_l : X \rightarrow \mathcal{N}$ be a projection operator mapping every element \tilde{x} from X , to the discrete component of \tilde{x} . The sequence $S_l \mathcal{X} \equiv (\emptyset, 1, \dots)$ is then a sequence of infinite length that takes only finitely many different values. Therefore, there exists at least one element which repeats infinitely often in \mathcal{X} . Let $\omega \in \mathcal{N} \setminus \{\emptyset, \infty\}$ be an element that repeats infinitely often in $S_l \mathcal{X}$ and let $\mathcal{C}[\omega]$ denote the set of all cycles on $G(\mathcal{N}, \mathcal{E})$ that begin and end at ω . Define

$$\theta = \min_{\mathcal{C} \in \mathcal{C}[\omega]} \prod_{(i,j,k) \in \mathcal{C}} \theta_{ij}^k.$$

Note that (3.13) implies that $\theta > 1$. Let \mathcal{W} be a subsequence of \mathcal{X} consisting of all the elements from \mathcal{X} that satisfy $S_l \tilde{x} = \omega$, and rename the analog component of \tilde{x} at the k -th appearance of ω in $S_l \mathcal{X}$ by x_k to obtain the sequence $\mathcal{W} := ((\omega, x_1), (\omega, x_2), \dots, (\omega, x_t), \dots)$. Then we have $V_\omega(x_1) < 0$, and $V_\omega(x_{i+1}) < \theta V_\omega(x_i)$, and $\theta > 1$. The result then follows from Proposition 3.6. It is easy to construct a counterexample to prove necessity. For instance, consider a graph model defined via the following elements:

$$\begin{aligned} G(\mathcal{N} := \{\emptyset, 1, \infty\}, \mathcal{E} := \{(\emptyset, 1), (1, 1), (1, \infty)\}) \\ X := \{\emptyset, 1, \infty\} \times \mathbb{R}, X_0 := \{1\} \\ T_{11} := 0.9x + 1, \Pi_{11} := \mathbb{R}, \Pi_{1\infty} := \{0\}. \end{aligned}$$

Then $\sigma_i(x) := x^2 - 10$, $i \in \{\emptyset, 1, \infty\}$ is a rate 1 Lyapunov invariant. However, the process does not terminate in finite time. ■

Remark 3.4 *In Proposition 3.8, the inequalities in either (3.12a) or (3.12b) (but not both) can be replaced by the non-strict versions. Moreover, similar to Proposition 3.5, a variation of Proposition 3.8 can be formulated for finite state modes, where the expression in (3.13) is replaced by an equality or a non-strict inequality instead of the strict inequality:*

$$\prod_{(i,j,k) \in \mathcal{C}} \theta_{ij}^k \geq 1, \mathcal{C} \in G$$

In this case, existence of a Lyapunov invariant satisfying the conditions of Proposition 3.8 guarantees termination if the continuous component of the state belongs to a finite set.

3.2.2 Safety

As it was discussed in Section 2.3.1, safety in software systems is the property that an unsafe subset X_- of the state space X can never be reached. Consider a rate θ Lyapunov invariant V , defined according to (3.1), with rate $\theta = 1$. The level sets $\mathcal{L}_r(V)$ of V , are defined by: $\mathcal{L}_r(V) := \{x \in X : V(x) < r\}$. These level sets are invariant with respect to (2.1), in the sense that $x(t+1) \in \mathcal{L}_r(V)$ whenever $x(t) \in \mathcal{L}_r(V)$. We can use this fact, along with the monotonicity property of V , to establish a separating manifold between the reachable set and the unsafe region of the state space (cf. Theorem 3.1). Note that for $r = 0$, the level sets $\mathcal{L}_r(V)$ remain invariant with respect to (2.1) for any positive θ . Using this fact, we prove in the next theorem, that under some technical assumptions, $\theta = 1$ is not necessary for establishing the separation between the unsafe region and the reachable set.

Theorem 3.1 *Consider a program \mathcal{P} , and its dynamical system model $\mathcal{S}(X, f, X_0, X_\infty)$. Let \mathcal{V} denote the set of all rate θ Lyapunov invariants for \mathcal{S} . An unsafe subset X_- of the state space X can never be reached along the trajectories of \mathcal{P} , if there exists $V \in \mathcal{V}$ satisfying*

$$\sup_{x \in X_0} V(x) \leq \inf_{x \in X_-} V(x) \quad (3.14)$$

$$V(x_+) - \theta V(x) < 0 \quad \forall x \in X \setminus X_\infty, x_+ \in f(x). \quad (3.15)$$

and at least one of the following three conditions hold:

$$(I) \quad \theta = 1. \quad (3.16)$$

$$(II) \quad 0 \leq \theta < 1, \text{ and } \inf_{x \in X_-} V(x) \geq 0. \quad (3.17)$$

$$(III) \quad 0 \leq \theta, \quad \text{and } \sup_{x \in X_0} V(x) \leq 0. \quad (3.18)$$

Proof. First, consider case (I). Assume that \mathcal{S} has a solution $\mathcal{X} = (x(0), x(1), \dots, x(t_-), \dots)$, where $x(0) \in X_0$ and $x(t_-) \in X_-$. Since $V(x)$ is strictly monotonically decreasing along any

solution of \mathcal{S} , we must have:

$$\inf_{x \in X_-} V(x) \leq V(x(t_-)) < V(x(0)) \leq \sup_{x \in \bar{X}_0} V(x) \quad (3.19)$$

which contradicts (3.14). Next, consider case (II), for which V may not be monotonic along the trajectories. Partition X_0 into two subsets \bar{X}_0 and \underline{X}_0 (one of which could be empty) such that $X_0 = \bar{X}_0 \cup \underline{X}_0$ and

$$V(x) \leq 0 \quad \forall x \in \underline{X}_0, \text{ and } V(x) > 0 \quad \forall x \in \bar{X}_0$$

Now, assume that \mathcal{S} has a solution $\bar{\mathcal{X}} = (\bar{x}(0), \bar{x}(1), \dots, \bar{x}(t_-), \dots)$, where $\bar{x}(0) \in \bar{X}_0$ and $\bar{x}(t_-) \in X_-$. Note that (3.17) implies that $V(\bar{x}(t_-)) \geq 0$ and thus: $V(\bar{x}(t)) > 0, \quad \forall t < t_-$. Therefore, $V(\bar{x}(t))$ is strictly monotonically decreasing over the sequence $\bar{x}(0)$ to $\bar{x}(t_-)$. Therefore,

$$\inf_{x \in X_-} V(x) \leq V(\bar{x}(t_-)) < V(\bar{x}(0)) \leq \sup_{x \in \bar{X}_0} V(x)$$

which contradicts (3.14). Finally, assume that \mathcal{S} has a solution $\underline{\mathcal{X}} = (\underline{x}(0), \underline{x}(1), \dots, \underline{x}(t_-), \dots)$, where $\underline{x}(0) \in \underline{X}_0$ and $\underline{x}(t_-) \in X_-$. In this case, regardless of the value of θ , we must have $V(\underline{x}(t)) \leq 0, \forall t$. This implies that $V(\underline{x}(t_-)) < 0$, which contradicts (3.17). The proof for case (III) is similar to the proof for case (II) with $x(0) \in \underline{X}_0$. ■

Remark 3.5 *In Theorem 3.1, if (3.14) is replaced by the strict version of the inequality, then (3.15) can be replaced by the non-strict version and the results of the theorem remain valid.*

Among several properties of safety-critical software, absence of overflow and finite-time termination are expected in most applications. In the following corollary, we provide the criteria for establishing finite-time termination and absence of overflow simultaneously via one Lyapunov invariant. In addition, the criteria for the Lyapunov invariants as presented in Corollary 3.1, are more convenient for carrying out numerical computations (cf. Chapter 4) than the more generic criteria in Theorem 3.1.

Corollary 3.1 *Consider a program \mathcal{P} , and its dynamical system model $\mathcal{S}(X, f, X_0, X_\infty)$. Suppose that the overflow limit is specified by a diagonal positive definite matrix $\alpha > 0$, that is,*

$X_- := \{x \in X \mid \|\alpha^{-1}x\|_\infty \geq 1\}$. Let $q \in \mathbb{N} \cup \{\infty\}$, and let the function $V : X \mapsto \mathbb{R}$ be a rate θ Lyapunov invariant for \mathcal{S} , satisfying the following constraints:

$$V(x) \leq 0 \quad \forall x \in X_0. \quad (3.20)$$

$$V(x) \geq \|\alpha^{-1}x\|_q - 1 \quad \forall x \in X_-. \quad (3.21)$$

$$V(x_+) - \theta V(x) < 0 \quad \forall x \in X \setminus X_\infty, x_+ \in f(x). \quad (3.22)$$

Then, an overflow runtime error will not occur during any execution of \mathcal{P} . In addition, if $\theta > 1$ then, \mathcal{P} terminates in at most T steps where:

$$T = \frac{\log \sup_{x \in X \setminus \{X_- \cup X_\infty\}} |V(x)| - \log \inf_{x \in X_0} |V(x)|}{\log \theta}.$$

Proof. It follows from (3.21) and the definition of X_- that:

$$V(x) > \|\alpha^{-1}x\|_q - 1 \geq \|\alpha^{-1}x\|_\infty - 1 \geq 0, \quad \forall x \in X_-. \quad (3.23)$$

It then follows from (3.23) and (3.20) that:

$$\inf_{x \in X_-} V(x) \geq 0 \geq \sup_{x \in X_0} V(x)$$

Hence, the first statement of the Corollary follows from Theorem 3.1. Since $X \setminus X_-$ is a bounded set, $\sup_{x \in X \setminus \{X_- \cup X_\infty\}} |V(x)|$ is a finite real number, and is a lower bound for $V(x)$. Finite-time termination in at most T steps then follows from Proposition 3.6. ■

Remarks

1. If (3.21) is replaced by:

$$V(x) \geq \|\alpha^{-1}x\|_q - 1 \quad \forall x \in X, \quad (3.24)$$

or by

$$V(x) \geq \|\alpha^{-1}x\|_q - 1 \quad \forall x \in X \setminus X_\infty$$

then $\sup_{x \in X \setminus X_-} |V(x)| \leq 1$, and an upper bound for T is given by:

$$T = -\frac{\log \inf_{x \in X_0} |V(x)|}{\log \theta}$$

(assuming that $X_\infty \cap X_- = \emptyset$ the result remains valid and the proof does not change). Now, if X_0 is a small finite set, then T is easy to compute. The drawback is that satisfying the conditions of Corollary 3.1 is typically harder when (3.21) is replaced by (3.24). This is due to the fact that requiring $V(x) > \|\alpha^{-1}x\|_q - 1$ over the whole state space, as in (3.24), is more restrictive than the case where it has hold only over a subset of the state space, as in (3.21).

2. In Corollary 3.1, the results hold for an arbitrary choice of the q -norm: $\|\cdot\|_q$, $q > 0$. In practice, however, there is little incentive for choices other than $q = 2$ or $q = \infty$. When $V : X \mapsto \mathbb{R}$ is a quadratic function of its argument, the 2-norm is the viable choice; since with $q = 2$, the search for a function V satisfying the criteria of Corollary 3.1 can be formulated as a semidefinite optimization problem. When $V : X \mapsto \mathbb{R}$ is a linear or piecewise linear function of its argument, the ∞ -norm is the viable choice; since with $q = \infty$, the search for a function V satisfying the criteria of Corollary 3.1 can be formulated as a linear optimization problem (cf. Chapter 4).
3. Corollary 3.1 can be used for checking against “out-of-bounds array indexing” errors. If the k -th variable x_k is an array index for an array of size s , then with $\alpha_k = s$, existence of a function satisfying the criteria of the corollary guarantees that an out-of-bounds array indexing error will not occur.
4. It can be observed from the proof of Corollary 3.1 that one potential source of conservatism in the formulation of the safety criteria is the over-approximation of the unsafe set $X_- := \{x \in X \mid \|\alpha^{-1}x\|_\infty \geq 1\}$ with the set $X_- := \{x \in X \mid \|\alpha^{-1}x\|_q \geq 1\}$. If the actual values of the program variables do get very close to the overflow limit α (e.g. for array indices), this conservatism may lead to infeasibility. A potential remedy is to replace (3.21) by n constraints in the following way:

$$V(x) \geq |\alpha_k^{-1}x_k| - 1 \quad \forall x \in X_{k-}, \quad k \in \mathbb{Z}(1, n) \quad (3.25)$$

where α_k is the overflow limit for scalar variable x_k , and $X_{k-} := \{x \in X \mid \|\alpha^{-1}x_k\|_\infty \geq 1\}$. Since $\alpha_k^{-1}x_k$ is a scalar quantity, $\|\alpha_k^{-1}x_k\|_\infty = |\alpha_k^{-1}x_k|$ and (3.25) can be used in conjunction with quadratic Lyapunov invariants and the \mathcal{S} -Procedure to formulate a semidefinite optimization problem (cf. Chapter 4). However, (3.25) is obviously computationally more expensive than (3.21), as it increases the number of constraints. An even less conservative but computationally more demanding approach would be to construct a different Lyapunov invariant for each variable and rewrite (3.25), (3.21), (3.20) along with (3.1), for n different functions $V_k(\cdot)$. For instance:

$$\begin{aligned} V_k(x) &\leq 0, & \forall x \in X_0. \\ V_k(x) &\geq |\alpha_k^{-1}x_k| - 1 & \forall x_k \in X_{k-}. \end{aligned}$$

Safety Analysis of Graph Models

The main results of the previous section, namely Theorem 3.1 and Corollary 3.1 are readily applicable to MILMs. We will use these results in Chapter 4, to formulate a convex optimization problem, the solution of which provides the certificates for safety specifications of the MILMs. In this section we study variations of Theorem 3.1 and Corollary 3.1 for safety analysis of graph models. The reinterpretation of the results of the previous section in terms of graph models is theoretically interesting and practically necessary for formulating convex programming criteria (cf. Chapter 4).

Corollary 3.2 *Consider a program \mathcal{P} and its graph model $G(\mathcal{N}, \mathcal{E})$. Suppose that the unsafe region associated with node i is given by X_{i-} . Let $V(i, x) := \sigma_i(x)$ be a Lyapunov invariant for $G(\mathcal{N}, \mathcal{E})$, satisfying the following constraints:*

$$\sigma_\emptyset(x) \leq 0 \quad \forall x \in X_\emptyset \tag{3.26}$$

$$\sigma_i(x) \geq 0 \quad \forall x \in X_i \cap X_{i-}, \quad i \in \mathcal{N} \setminus \{\emptyset\} \tag{3.27}$$

$$\sigma_j(x_+) - \theta_{ji}^k \sigma_i(x) < 0 \quad \forall (i, j, k) \in \mathcal{E}, \quad (x, x_+) \in (X_i \cap \Pi_{ji}^k) \times T_{ji}^k x \tag{3.28}$$

Then, \mathcal{P} satisfies the safety property w.r.t. the collection of sets X_{i-} , $i \in \mathcal{N} \setminus \{\emptyset\}$. In addition,

if (3.13) holds, then \mathcal{P} terminates in at most T steps, where

$$T = \sum_{\mathcal{C} \in \mathcal{G}} \frac{\log \max_{(i, \dots) \in \mathcal{C}} \sup_{x \in X_i \setminus X_{i-}} |\sigma_i(x)| - \log \inf_{x \in X_\emptyset} |\sigma_\emptyset(x)|}{\log \theta(\mathcal{C})}, \quad \theta(\mathcal{C}) := \prod_{(i, j, k) \in \mathcal{C}} \theta_{ij}^k.$$

Proof. The safety property follows directly from Theorem 3.1. The finite-time termination property also follows from Proposition 3.6 and Proposition 3.8. The bound on the number of steps is the sum of the maximum number of iterations around every simple cycle. ■

Overflow

Corollary 3.3 Consider a program \mathcal{P} and its graph model $G(\mathcal{N}, \mathcal{E})$. Suppose that the overflow limit is specified by a diagonal positive definite matrix $\alpha > 0$. That is:

$$X_- := \{x \in \mathbb{R}^n \mid \|\alpha^{-1}x\|_\infty \geq 1\}.$$

Let $V(i, x) := \sigma_i(x)$ be a Lyapunov invariant for G satisfying the following conditions:

$$\sigma_\emptyset(x) \leq 0 \quad \forall x \in X_\emptyset,$$

$$\sigma_i(x) \geq \|\alpha^{-1}x\|_q - 1 \quad \forall x \in X_i \cap X_{i-}, \quad i \in \mathcal{N} \setminus \{\emptyset\},$$

$$\sigma_j(x_+) - \theta_{ji}^k \sigma_i(x) < 0 \quad \forall (i, j, k) \in \mathcal{E}, \quad (x, x_+) \in (X_i \cap \Pi_{ji}^k) \times T_{ji}^k x$$

Then, an overflow runtime error will not occur during any execution of \mathcal{P} . In addition, if (3.13) holds, then \mathcal{P} terminates in at most T steps, where

$$T = - \sum_{\mathcal{C} \in \mathcal{G}} \frac{\log \inf_{x \in X_\emptyset} |\sigma_\emptyset(x)|}{\log \theta(\mathcal{C})}, \quad \theta(\mathcal{C}) := \prod_{(i, j, k) \in \mathcal{C}} \theta_{ij}^k.$$

Proof. Follows from Corollary 3.2 as a special case. ■

Program Assertions Verification of user-specified assertions in computer code, or other standard safety specifications such as absence of division-by-zero can be performed using Corollary 3.2. The results are summarized in the following table:

Table 3.1: Application of Corollary 3.2 to the verification of various safety specifications.

	The command line:		apply Corollary 3.2 with:
loc i	involves: assert $x \in X_a$	\Rightarrow	$X_{i-} := \{x \in \mathbb{R}^n \mid x \in \mathbb{R}^n \setminus X_a\}$
loc i	involves: assert $x \notin X_a$	\Rightarrow	$X_{i-} := \{x \in \mathbb{R}^n \mid x \in X_a\}$
loc i	involves: (expr.)/ x_o	\Rightarrow	$X_{i-} := \{x \in \mathbb{R}^n \mid x_o = 0\}$
loc i	involves: $\sqrt[k]{x_o}$	\Rightarrow	$X_{i-} := \{x \in \mathbb{R}^n \mid x_o < 0\}$
loc i	involves: $\log(x_o)$	\Rightarrow	$X_{i-} := \{x \in \mathbb{R}^n \mid x_o \leq 0\}$

Example 3.3 Consider the following program

```

void ComputeTurnRate (void)
L0 : {double x = {0}; double y = {*PtrToY};
L1 : while (1)
L2 : {   y = *PtrToY;
L3 :     x = (5 * sin(y) + 1)/3;
L4 :     if x > -1 {
L5 :         x = x + 1.0472;
L6 :         TurnRate = y/x; }
L7 :     else {
L8 :         TurnRate = 100 * y/3.1416 }}

```

Program 3-1.

Proving that $x \neq 0$ throughout the program is not possible, as indeed x can be zero right after the assignment $x = (5\sin(y) + 1)/3$. However, at location L6, x cannot be zero and division-by-zero

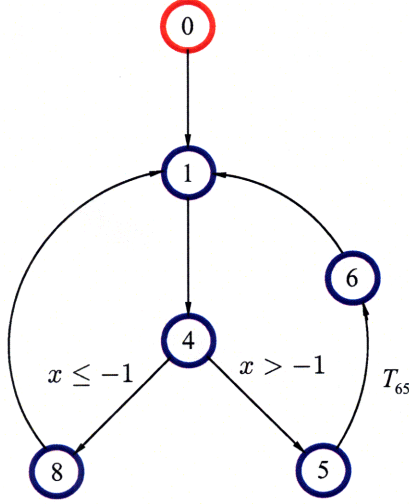


Figure 3-2: The graph model of an abstraction of Program 3-1.

will not occur. The graph model of an abstraction of Program 3-1 is shown in Figure 3-2 and is defined by the following elements: $T_{65} : x \rightarrow x + 1.0472$, and $T_{41} : x \rightarrow [-4/3, 2]$. The rest of the transition labels are identity. The only non-universal passport labels correspond to Π_{54} and Π_{84} . These are shown in Figure 3-2. Define:

$$\begin{aligned} \sigma_6(x) &= -x^2, \sigma_5(x) = -x^2 + 100x - 99, \sigma_4(x) = -x^2 - 1 \\ \sigma_1(x) &= -x^2 - 0.5, \sigma_8(x) = -x^2 - 2, \sigma_0(x) = -x^2 - 0.5 \end{aligned}$$

It can be verified that $V(x) = \sigma_i(x)$ is a Lyapunov invariant for Program 3-1 with variable rates: $\theta_{54} = 0.001$, $\theta_{41} = \theta_{18} = 0$, $\theta_{65} = \theta_{16} = \theta_{84} = 1$. Since

$$\inf_{x \in X_{-} := \{0\}} \sigma_6(x) = 0 \geq \sup_{x \in X_0} \sigma_0(x) = -0.5$$

the state $(6, x = 0)$ cannot be reached. Hence a division by zero will never occur. This example is chosen for its simplicity and is presented here to demonstrate applicability of the techniques. We will discuss in the next chapter how to find such functions in general.

Unreachability of Discrete Locations Assume that we want to verify that a discrete location $i \in \mathcal{N} \setminus \{\emptyset\}$ in a graph model $G(\mathcal{N}, \mathcal{E})$ is unreachable. Consider again Corollary 3.2, and define $X_{i-} = X_i$, where X_i is the invariant set of node i ($X_i = \mathbb{R}^n$ if no specific information is available about the variables at location i). The constraint (3.27) then becomes:

$$\sigma_i(x) \geq 0 \quad \forall x \in X_i, \quad i \in \mathcal{N} \setminus \{\emptyset\}.$$

If a Lyapunov invariant satisfying the criteria of Corollary 3.2 with $X_{i-} = X_i$ can be found, then location i can never be reached along any trajectory of $G(\mathcal{N}, \mathcal{E})$. This observation can be used for identifying dead-code.

3.3 Summary

In this chapter we introduced Lyapunov invariants as certificates for the behavior of mathematical models of computer programs. We formulated several theorems that establish criteria for verification of safety, liveness and other performance properties of software systems. We showed that different variations of Lyapunov invariants satisfying certain technical conditions can be formulated for proving absence of runtime errors and termination in finite time. The runtime errors that can be ruled out in this framework include overflow, out-of-bounds array indexing, division-by-zero, taking the square root or logarithm of a negative number, and various user-defined program assertions. Moreover, when finite-time termination can be guaranteed, the Lyapunov invariants provide an explicit upper bound on the maximum number of iterations. In the next chapter, we will present a computational procedure based on convex relaxations and numerical optimization for computation of the Lyapunov certificates.

Chapter 4

Computation of Lyapunov Invariants

In this Chapter, we present the details of a computational procedure based on convex optimization, for numerical computation of the Lyapunov invariants that prove the desired properties of computer programs. It is well known that the main difficulty in using Lyapunov functions in system analysis is finding them. However, the recent advances in the numerical optimization software technology, e.g. semi-definite programming software [33, 90, 86], or linear programming software [59, 34], along with the increased computational power that advances in hardware technology offer, provide a viable platform for computation of Lyapunov functions via numerical optimization tools. Hence, we propose exploiting convex optimization tools in analysis of software via Lyapunov invariants. The procedure is as follows: First, the search for V is restricted to a finite-dimensional linear subspace of the vector space of all real-valued functions $V : X \mapsto \mathbb{R}$. This subspace is specified by its basis which consists of a fixed finite set of appropriately selected (usually polynomial) functions. The basis is then used for constructing a finite-dimensional linear parameterization of a set of Lyapunov function candidates. The linear parameterization, along with various convex relaxation techniques make it possible to search for the Lyapunov invariants via convex optimization. If the convex optimization phase ends with a feasible solution, the result is a certificate for the properties of the computer program.

4.1 Preliminaries

4.1.1 Convex Parameterization of Lyapunov Invariants

As we mentioned earlier, the main difficulty in using Lyapunov functions in analysis of dynamical systems is finding them. Naturally, using Lyapunov invariants in software analysis inherits the same difficulties. The chances of finding a Lyapunov invariant successfully are increased when (3.1) is only required on a subset of $X \setminus X_\infty$. It is tempting to replace (3.1) with

$$V(x_+) - \theta V(x) < 0, \quad \forall x \in X \setminus X_\infty : V(x) < 1, \quad x_+ \in f(x), \quad (4.1)$$

while adding the constraint

$$V(x) < 1, \quad \forall x \in X_0.$$

In this formulation, V is not required to decrease monotonically for some of the states which cannot be reached from X_0 . Unfortunately, the set of all functions $V : X \rightarrow \mathbb{R}$ satisfying (4.1) is not a convex set. Therefore, in contrast with (3.1), formulation (4.1) has a significant disadvantage as finding a solution of (4.1) is typically much harder than finding a solution of (3.1). While several numerical optimization software that can handle non-convex problems currently exist (e.g. [50, 45]), for the most part, these software are not as reliable and well-behaved as the convex optimization software. They generally rely on a combination of heuristics and gradient methods to find a local extrema of the non-convex optimization problem. While they can be useful for solving specific non-convex optimization problems on a case-to-case basis, they cannot constitute the optimization engine of a software analysis framework as they can show unexpected behavior. In addition, these algorithms may easily get trapped in a local minima which is far from being globally optimal. Hence, there is no guarantee that a Lyapunov invariant satisfying (4.1) would be found even if a Lyapunov invariant satisfying the more restrictive condition (3.1) does exist. Therefore, we would like to avoid non-convex formulations like (4.1).

An alternative approach for improving the chances of finding Lyapunov invariants is to replace (3.1) by

$$V_i(x_+) - \theta V_i(x) < 0 \quad \forall x \in X, \quad x_+ \in f(x) : x \in X_{v_i}, \quad (4.2)$$

where X_{v_i} is a *fixed* subset of X which does not contain any terminal points. Since V does not enter into any conditioning of x here, the set of all functions $V : X \rightarrow \mathbb{R}$ satisfying (4.2) is convex. This technique, along with partitioning of the state space into appropriate subspaces of smaller size, and assigning different functions V_i to each subspace X_{v_i} leads to systematic improvement of analysis.

The first practical step in the search for a Lyapunov invariant satisfying (3.1) is selecting a finite-dimensional linear parameterization of a Lyapunov invariant candidate V :

$$V(x) = V_\tau(x) = \sum_{k=1}^n \tau_k V_k(x), \quad \tau = (\tau_k)_{k=1}^n, \quad \tau_k \in \mathbb{R}, \quad (4.3)$$

where $V_k : X \mapsto \mathbb{R}$ are fixed functions, used as a basis for a finite-dimensional linear subspace of the (infinite-dimensional) vector space of all real-valued functions $V : X \mapsto \mathbb{R}$. For instance, standard quadratic Lyapunov functions for linear time-invariant systems with n states are of the form:

$$V_\tau(x) = \sum_{i=1}^n \sum_{j=i}^n \tau_{ij} V_{ij}(x), \quad \tau = (\tau_k)_{k=1}^N, \quad \tau_k \in \mathbb{R}, \quad N = \frac{n(n+1)}{2},$$

where the functions $V_{ij} : x \mapsto x_i x_j$ form a basis for the N -dimensional ($N = n(n+1)/2$) vector space of all real-valued quadratic forms in n variables. Next, for every $\tau = (\tau_k)_{k=1}^N$ let

$$\phi(\tau) = \max_{x \in X \setminus X_\infty, x_+ \in f(x)} V_\tau(x_+) - \theta V_\tau(x),$$

(assuming for simplicity that the maximum does exist). Since $\phi(\cdot)$ is a maximum of a family of linear functions (for every fixed x , $V_\tau(x_+) - \theta V_\tau(x)$ is a linear function of τ), $\phi(\cdot)$ is a convex function of its argument (τ). Therefore, minimization of $\phi(\cdot)$ over the unit disk $\{\tau : \|\tau\| \leq 1\}$ is a well-defined convex optimization problem. If minimizing $\phi(\cdot)$ over the unit disk yields a negative minimum (if $\phi(\tau)$ can be made negative for some $\tau \in \mathbb{R}^N$, it can be made negative for some τ^* inside the unit disk by a simple scaling), the optimal τ^* defines a valid Lyapunov invariant $V_{\tau^*}(x)$ in the sense of (3.1). Otherwise, no linear combination (4.3) yields a valid Lyapunov invariant for (2.1).

The success and efficiency of the proposed convex optimization approach depend highly on

computability of $\phi(\cdot)$ and its subgradients. While $\phi(\cdot)$ is convex with respect to its argument, the same does not necessarily hold for $V_\tau(x_+) - \theta V_\tau(x)$. It is easy to see that even very simple computer programs lead to non-convex optimization in the problem of calculating the maximum of $V_\tau(x_+) - V_\tau(x)$. In fact, if $X \setminus X_\infty$ is non-convex, computation of $\phi(\cdot)$ becomes a non-convex optimization problem even if $V_\tau(x_+) - V_\tau(x)$ is a nice (e.g. linear or concave and smooth) function of x . Hence, in order to formulate the search for the parameters of the Lyapunov invariants as a convex optimization problem, a compromise has to be made. We propose using convex relaxation techniques which essentially lead to computing a convex upper bound for $\phi(\tau)$. We briefly review a few of these techniques in the next section.

4.1.2 Convex Relaxation Techniques

We refer to *convex relaxation* techniques as a broad class of techniques commonly used to construct finite-dimensional, convex counterparts for hard non-convex optimization problems. In some cases the relaxations can be exact, in the sense that the optimal solution of the convex counterpart equals that of the original non-convex problem. In general, however, convex relaxations are understood as approximate convexification techniques that result in upper or lower bounds for the original non-convex optimization problem. See for instance [63], where operator theoretic methods are used to provide error bounds for certain relaxation techniques commonly used in analysis of dynamical systems. However, quantifying the gap induced by a specific relaxation technique is often a mathematically challenging task and providing good error bounds may not always be possible.

Various convex relaxation techniques exist for both combinatorial and non combinatorial optimization problems. See for instance the results of Lovasz and Schrijver [57] for SDP relaxation of binary integer programs, Goemans and Williamson [39] or Laurent [55] for SDP relaxation of the max-cut problem, Megretski [63] and Nesterov [71], for SDP relaxations of quadratic programs, Yakubovic [92] for \mathcal{S} -Procedure losslessness in robustness analysis, and Parrilo [73, 74] for sum-of-squares relaxation in polynomial non-negativity verification. Here, we briefly review the \mathcal{S} -Procedure and the sum-of-squares relaxation techniques.

The \mathcal{S} -Procedure

The \mathcal{S} -Procedure is a convex relaxation method concerned with verification of positivity of a quadratic function subject to other quadratic or linear constraints. It was first introduced by Aizerman and Gantmakher [1] in the context of construction of Lyapunov functions for nonlinear systems [40], and has been ever since used frequently in analysis of dynamical systems. Let $\phi_i : X \rightarrow \mathbb{R}$, $i \in \mathbb{Z}(0, m)$, and $\psi_j : X \rightarrow \mathbb{R}$, $j \in \mathbb{Z}(1, n)$ be real-valued functions defined on a vector space X , and suppose that we want to evaluate the following assertions:

$$(I): \phi_0(x) > 0, \forall x \in \{x \in X \mid \phi_i(x) \geq 0, \psi_j(x) = 0, i \in \mathbb{Z}(1, m), j \in \mathbb{Z}(1, n)\} \quad (4.4)$$

$$(II): \exists \tau_i \in \mathbb{R}^+, \exists \mu_j \in \mathbb{R}, \text{ such that } \phi_0(x) > \sum_{i=1}^m \tau_i \phi_i(x) + \sum_{j=1}^n \mu_j \psi_j(x). \quad (4.5)$$

It is obvious that (I) is implied by (II). The process of replacing assertion (I) by its relaxed version (II) is called the \mathcal{S} -Procedure. Note that condition (II) is convex in decision variables τ_i and μ_j . Moreover, if ϕ_i and ψ_j are quadratic functionals then condition (II) is a semidefinite optimization problem in the decision variables which can be solved efficiently. The implication (I) \rightarrow (II) is not always true and hence the \mathcal{S} -Procedure in its general form provides only a sufficient condition for (I). The \mathcal{S} -Procedure is called lossless if (I) \rightarrow (II). For instance, a well-known case where the \mathcal{S} -Procedure is necessary and sufficient is when $m = 1$, $n = 0$, and ϕ_0, ϕ_1 are quadratic functionals. A comprehensive discussion of the available results on \mathcal{S} -Procedure losslessness can be found in [40]. Other variations of the \mathcal{S} -Procedure involving non-strict inequalities exist and are used frequently.

Sum-of-Squares Relaxation

The sum-of-squares (SOS) relaxation technique can be interpreted as the generalized version of the \mathcal{S} -Procedure. Suppose that we are concerned with the answer to the following question: Given the index sets $J = \mathbb{Z}(1, s)$, $K = \mathbb{Z}(1, t)$, $L = \mathbb{Z}(1, u)$, and polynomials f, g, h , when is it true that the following conditions:

$$f_j(x) \geq 0, \forall j \in J, \quad \text{and} \quad g_k(x) \neq 0, \forall k \in K, \quad \text{and} \quad h_l(x) = 0, \forall l \in L \quad (4.6)$$

imply that:

$$-f_0(x) \geq 0 ? \quad (4.7)$$

Note that (4.7) is implied by (4.6) if and only if the following semialgebraic set is empty:

$$\left\{ \begin{array}{l} x \in \mathbb{R}^n \mid \quad f_0(x) \geq 0, f_j(x) \geq 0, j \in J, \\ \quad \mid \quad f_0(x) \neq 0, g_k(x) \neq 0, k \in K, h_l(x) = 0, l \in L. \end{array} \right\} \quad (4.8)$$

Similar questions can be formulated concerning a polynomial vanishing or being nonzero on a semialgebraic set. That is, we can ask when does (4.6) imply that $g_0(x) = 0$, or that $h_0(x) \neq 0$? In a similar fashion, all these questions can be reformulated in terms of emptiness of semialgebraic sets. The sum-of-squares relaxation technique can in turn, be applied to formulate sufficient criteria for emptiness of semialgebraic sets. Before we proceed, for clarity of the exposition, we introduce some definitions.

Definition 4.1 Let $\mathbb{R}[x_1, \dots, x_n]$ denote the polynomial ring of n variables with real coefficients, and $\Sigma[x_1, \dots, x_n]$ denote the subset of sum of squares polynomials in $\mathbb{R}[x_1, \dots, x_n]$, that is, the set of polynomials that can be represented as $p = \sum_{i=1}^t p_i^2$, $p_i \in \mathbb{R}[x_1, \dots, x_n]$. Given $(g_k)_{k \in K} \in \mathbb{R}[x_1, \dots, x_n]$, the multiplicative monoid generated by g_k is the set:

$$M(g_k) := \left\{ \prod g_k^{a_k} \mid a_k \in \mathbb{N} \cup \{0\} \right\}.$$

Given $(h_l)_{l \in L} \in \mathbb{R}[x_1, \dots, x_n]$, the Ideal generated by h_l is the set:

$$I(h_l) := \left\{ \sum h_l \mu_l \mid \mu_l \in \mathbb{R}[x_1, \dots, x_n] \right\}.$$

Given $(f_j)_{j \in J} \in \mathbb{R}[x_1, \dots, x_n]$, the cone generated by f_j is the set:

$$P(f_j) := \left\{ \tau_0 + \sum \tau_i b_i \mid \tau_i \in \Sigma[x_1, \dots, x_n], b_i \in M(f_j) \right\}.$$

The Positivstellensatz Theorem [15] provides a necessary and sufficient criterion for emptiness of semialgebraic sets.

Theorem 4.1 ([15]) *The following assertions are equivalent:*

(I). *The set*

$$S := \left\{ \begin{array}{l} x \in \mathbb{R}^n \mid f_j(x) \geq 0, j \in J, g_k(x) \neq 0, k \in K, \\ \mid \\ h_l(x) = 0, l \in L \end{array} \right\}$$

is empty.

(II). *There exist $f \in P(f_j)$, $g \in M(g_k)$, $h \in I(h_l)$ such that $f + h + g^2 = 0$.*

Note that the implication (II) \rightarrow (I) is trivial. Although Theorem 4.1 provides a necessary and sufficient condition for emptiness of the set S , a systematic method for determining the minimal degrees of the polynomials f , g , and h is not known to this date, neither is known a systematic method for providing bounds on the maximum degrees of these polynomials in general settings. Hence, taking advantage of the implication (I) \rightarrow (II) is practically difficult. In practice, one must always resort to sufficient conditions that can be formulated by imposing degree bounds on f , g , h . A sufficient condition can be formulated as follows: The set S is empty if there exist polynomials $f \in P(f_j)$, $g \in M(g_k)$, $h \in I(h_l)$ of degrees less than or equal to d such that $f + h + g^2 = 0$. The following procedure or similar variations of it are often used for verifying emptiness of the set S while imposing degree bounds on f , g , h .

1. Fix a subset of $M(g_k)$, such as $\prod g_k$.
2. Fix a subset of $P(f_j)$, such as $\tau_0 + \sum_i \tau_i f_i + \sum_{i,j} \tau_{ij} f_i f_j$.
3. The set S is empty if there exist sum-of-squares polynomials, $\tau_0, \tau_1, \dots, \tau_s, \tau_{11}, \dots, \tau_{ss} \in \Sigma[x_1, \dots, x_n]$ and polynomials $\mu_l \in \mathbb{R}[x_1, \dots, x_n]$, such that $\tau_0 + \sum_i \tau_i f_i + \sum_{i,j} \tau_{ij} f_i f_j + \sum_l \mu_l h_l + (\prod g_k)^2 = 0$.

A good strategy is to choose the degrees of the SOS multipliers τ_i , τ_{ij} , and the polynomial multipliers p_l such that all the expressions in $f + h + g^2$ have the same degree. The problem of finding the multipliers is then an SOS optimization problem. The Matlab toolboxes SOSTOOLS [79], or YALMIP [56] automate the process of converting an SOS optimization problem to a semidefinite programming problem (SDP). The SDP is then subsequently solved by available software packages such as LMILAB [33], SDPT3 [90], or SeDumi [86]. Interested readers are referred to [73, 64, 74, 79] for more detailed information about the SOS optimization procedure.

4.2 Optimization of Lyapunov Invariants for Mixed-Integer Linear Models

We introduced the mixed-integer linear models for analysis of software in Chapter 2. In this section, we demonstrate in detail, the procedure for computation of Lyapunov invariants for these models. As we discussed in the previous section, the search for a Lyapunov invariant begins with a finite-dimensional linear parameterization of the search space. Natural Lyapunov invariant candidates for MILMs are quadratic functionals.

4.2.1 Quadratic Invariants

The linear parameterization of the space of quadratic functionals mapping \mathbb{R}^n to \mathbb{R} can be represented as:

$$\mathcal{V}_x^2 := \left\{ V : \mathbb{R}^n \rightarrow \mathbb{R} \mid V(x) = \begin{bmatrix} x \\ 1 \end{bmatrix}^T P \begin{bmatrix} x \\ 1 \end{bmatrix}, P \in \mathbb{S}^{(n+1) \times (n+1)} \right\}, \quad (4.9)$$

where, P is a symmetric matrix and the super-index 2 in \mathcal{V}_x^2 indicates the polynomial degree bound on the functional V . An attempt at finding a Lyapunov invariant in this linear subspace can be made by solving a convex optimization problem in which the elements of P appear as the decision variables (along with the multipliers from convex relaxations).

Before proceeding to the next lemma, recall from the \mathcal{S} -Procedure (cf. Section 4.1.2) that the assertion

$$\sigma(y) < 0, \forall y \in [-1, 1]^n$$

holds if there exists nonnegative constants $\tau_i \geq 0$, $i = 1, \dots, n$, such that

$$\sigma(y) < \sum \tau_i (y_i^2 - 1) = y^T \tau y - \text{Trace}(\tau),$$

where $\tau = \text{diag}(\tau_i)$ is a positive semidefinite diagonal matrix. Similarly, the assertion $\sigma(y) < 0, \forall y \in \{-1, 1\}^n$ holds if there exists a diagonal matrix μ (μ is not sign-definite) such that $\sigma(y) < \sum \mu_i (y_i^2 - 1) = y^T \mu y - \text{Trace}(\mu)$. These convex relaxations are exploited in the formulation of the following lemma.

Lemma 4.1 Consider a computer program \mathcal{P} and its Mixed-Integer Linear dynamical system model $\mathcal{S}(F, H, X_0, n, q, r)$, where $F \in \mathbb{R}^{n \times n_e}$, $H \in \mathbb{R}^{m \times n_e}$, n is the number of state variables, q , and r , represent the number of continuous and binary auxiliary variables respectively, and $n_e = n + q + r + 1$. There exists a rate θ Lyapunov invariant for \mathcal{P} in the class \mathcal{V}_x^2 , if there exists a matrix $Y \in \mathbb{R}^{n_e \times m}$, a diagonal matrix $D_v \in \mathbb{R}^{r \times r}$, a positive semidefinite diagonal matrix $D_{xw} \in \mathbb{R}^{(n+q) \times (n+q)}$, and a symmetric matrix $P \in \mathbb{S}^{(n+1) \times (n+1)}$, satisfying the following Linear Matrix Inequalities:

$$L_1^T P L_1 - \theta L_2^T P L_2 \prec \text{He}(YH) + L_3^T D_{xw} L_3 + L_4^T D_v L_4 - \lambda L_5^T L_5 \quad (4.10a)$$

$$0 \preceq D_{xw} \quad (4.10b)$$

$$\lambda = \text{Trace } D_{xw} + \text{Trace } D_v \quad (4.10c)$$

where

$$L_1 := \begin{bmatrix} F \\ L_5 \end{bmatrix}, \quad L_2 := \begin{bmatrix} I_n & 0_{n \times (n_e - n)} \\ 0_{1 \times (n_e - 1)} & 1 \end{bmatrix},$$

$$L_3 := \begin{bmatrix} I_{n+q} & 0_{(n+q) \times (r+1)} \end{bmatrix}, \quad L_4 := \begin{bmatrix} 0_{r \times (n+q)} & I_r & 0_{r \times 1} \end{bmatrix}, \quad L_5 := \begin{bmatrix} 0_{1 \times (n_e - 1)} & 1 \end{bmatrix}$$

Proof. Define $x_e = (x, w, v, 1)^T$, where $x \in [-1, 1]^n$, $w \in [-1, 1]^q$, $v \in \{-1, 1\}^r$. Recall that $(x, 1)^T = L_2 x_e$, and that for all x_e satisfying $Hx_e = 0$, there holds: $(x_+, 1) = (Fx_e, 1) = L_1 x_e$. It follows from Proposition 3.2 that the Lyapunov condition (3.1) holds if:

$$x_e^T L_1^T P L_1 x_e - \theta x_e^T L_2^T P L_2 x_e < 0, \text{ s.t. } Hx_e = 0, L_3 x_e \in [-1, 1]^{n+q}, L_4 x_e \in \{-1, 1\}^r. \quad (4.11)$$

Using the \mathcal{S} -Procedure to relax each of the constraints in (4.11), we obtain a sufficient condition for (4.11) to hold:

$$x_e^T L_1^T P L_1 x_e - \theta x_e^T L_2^T P L_2 x_e < x_e^T (YH + H^T Y^T) x_e + x_e^T L_3^T D_{xw} L_3 x_e$$

$$- \text{Trace } D_{xw} + x_e^T L_4^T D_v L_4 x_e - \text{Trace } D_v$$

$$0 \preceq D_{xw}$$

The first inequality can be rewritten as:

$$x_e^T [L_1^T P L_1 - \theta L_2^T P L_2] x_e \preceq x_e^T [(YH + Y^T H^T) + L_3^T D_{xw} L_3 + L_4^T D_v L_4 - \lambda L_5^T L_5] x_e$$

$$\lambda = \text{Trace } D_{xw} + \text{Trace } D_v$$

Together with $0 \preceq D_{xw}$, the above conditions are equivalent to the LMIs in (4.10). ■

The following theorem summarizes our results for verification of absence of overflow and/or finite-time termination for MILMs.

Theorem 4.2 *Consider a computer program \mathcal{P} and its Mixed-Integer Linear dynamical system model $\mathcal{S}(F, H, X_0, n, q, r)$. Suppose that the overflow limit is specified by a diagonal positive definite matrix $0 \prec \alpha \preceq I_n$. An overflow runtime error does not occur during any execution of \mathcal{P} if there exist matrices $Y_i \in \mathbb{R}^{n_e \times m}$, and diagonal matrices $D_{iv} \in \mathbb{R}^{r \times r}$, $i \in \{1, 2\}$, positive semidefinite diagonal matrices $D_{ixw} \in \mathbb{R}^{(n+q) \times (n+q)}$, $i \in \{1, 2\}$, and a symmetric matrix $P \in \mathbb{S}^{(n+1) \times (n+1)}$ satisfying the following conditions:*

$$[x_0 \ 1] P [x_0 \ 1]^T \leq 0, \quad \forall x_0 \in X_0 \quad (4.12a)$$

$$L_1^T P L_1 - \theta L_2^T P L_2 \prec \text{He}(Y_1 H) + L_3^T D_{1xw} L_3 + L_4^T D_{1v} L_4 - \lambda_1 L_5^T L_5 \quad (4.12b)$$

$$L_2^T \Lambda L_2 - L_2^T P L_2 \preceq \text{He}(Y_2 H) + L_3^T D_{2xw} L_3 + L_4^T D_{2v} L_4 - \lambda_2 L_5^T L_5 \quad (4.12c)$$

$$0 \preceq D_{ixw}, \quad i = 1, 2 \quad (4.12d)$$

$$\lambda_i = \text{Trace } D_{ixw} + \text{Trace } D_{iv}, \quad i = 1, 2. \quad (4.12e)$$

where $\Lambda := \text{diag}\{\alpha^{-2}, -1\}$. In addition, if $\theta > 1$, then \mathcal{P} terminates in a most T steps where

$$T = -\frac{\log \min_{x \in X_0} |[x_0 \ 1] P [x_0 \ 1]^T|}{\log \theta}.$$

Proof. The Theorem can be proven by applying a proof method similar to that of Lemma 4.1 to Corollary 3.1 with $q = 2$. ■

Remark 4.1 *The first condition in Theorem 4.2 guarantees that $V(x) \leq 0, \forall x \in X_0$, which conforms with Condition 3.20 of Corollary 3.1. If the set X_0 is a finite set of cardinality n_0 , then (4.12a) is equivalent to n_0 affine constraints on P , one for each x_0 in X_0 . However, if X_0 is not a finite set, or if n_0 is too large, we need to consider an over-approximation of X_0 by $\bar{X}_0 \supseteq X_0$. Convenient choices for \bar{X}_0 are sets of the form: $\bar{X}_0 := \{x_0 \in [-1, 1]^n \mid x_0^T Q x_0 \leq 1, Q \in \mathbb{S}^n\}$ or of the form $\bar{X}_0 := \{x_0 \in [-1, 1]^n \mid H_0[x_0 \ w \ v \ 1]^T = 0, (w, v) \in [-1, 1]^q \times \{-1, 1\}^r\}$. In either case, similar convex relaxation techniques can be applied to formulate the constraints on the initial conditions (4.12a) as an LMI.*

Example 4.1 *Consider Program 2-4. The MILM of the program was given in Chapter 2. Suppose that the overflow limit is specified as $\alpha = 1000$. By application of Theorem 4.2 with $\theta = 1.001$ to Program 2-4 with $a = 24$, and $b = 21$ (chosen arbitrarily) absence of overflow, and finite-time termination is certified. The function*

$$V(x_1, x_2) = \begin{bmatrix} x_1/\alpha \\ x_2/\alpha \\ 1 \end{bmatrix}^T \begin{bmatrix} 0.9350 & -0.0000 & 0.0069 \\ -0.0000 & -3.1599 & -4.6014 \\ 0.0069 & -4.6014 & -0.0001 \end{bmatrix} \begin{bmatrix} x_1/\alpha \\ x_2/\alpha \\ 1 \end{bmatrix}$$

is the certificate for finite-time termination and absence of overflow. The upper bound on the number of iterations provided by this certificate is $T = 1.8 \times 10^3$.

4.2.2 Linear Invariants

Linear invariants can be helpful in proving certain properties of computer programs. For instance, it is possible that the first attempt at proving strong properties (e.g. finite-time termination or absence of overflow) at one shot (e.g. via Theorem 4.2) would be a failure, while providing additional information about the behavior of the program in the form of linear invariants that constrain the evolution of certain variables would make the process a success. Once a linear invariant is found, it can be added to the set of constraints (e.g. the matrix H in a MILM model) that define the program's dynamics.

The search for linear invariants starts with a linear parameterization of the subspace of

linear functionals mapping \mathbb{R}^n to \mathbb{R} . This subspace can be represented as:

$$\mathcal{V}_x^1 := \left\{ V : \mathbb{R}^n \rightarrow \mathbb{R} \mid V(x) = K^T [x \ 1]^T, K \in \mathbb{R}^{n+1} \right\} \quad (4.13)$$

where the super-index 1 in \mathcal{V}_x^1 represents the degree bound on the polynomial function V . Here, K is a matrix whose elements define the linear parameterization of the search space. It is possible to search for the linear invariants via semidefinite programming.

Lemma 4.2 *Consider a computer program \mathcal{P} and its Mixed-Integer Linear dynamical system model $\mathcal{S}(F, H, X_0, n, q, r)$. There exists a (linear) rate θ Lyapunov invariant for \mathcal{P} in the class \mathcal{V}_x^1 , if there exists a matrix $Y \in \mathbb{R}^{n_e \times m}$, a diagonal matrix $D_v \in \mathbb{R}^{r \times r}$, a positive semidefinite diagonal matrix $D_{xw} \in \mathbb{R}^{(n+q) \times (n+q)}$, and a matrix $K \in \mathbb{R}^{n+1}$ satisfying the following Linear Matrix Inequalities:*

$$\begin{aligned} \text{He}(L_1^T K L_5 - \theta L_5^T K^T L_2) &< \text{He}(YH) + L_3^T D_{xw} L_3 + L_4^T D_v L_4 - \lambda L_5^T L_5 \\ 0 &\preceq D_{xw} \\ \lambda &= \text{Trace } D_{xw} + \text{Trace } D_v \end{aligned}$$

Proof. Proof is similar to the proof of Lemma 4.1. ■

Lemma 4.2 provides a criterion for computation of linear invariants via semidefinite programming. In the sequel, we present a method for computation of linear invariants via linear programming. The advantage of using Lemma 4.2 (semidefinite programming in general) for computation of linear invariants is that an efficient relaxation technique for treatment of the binary variables $v_i \in \{-1, 1\}$ exists. Compared with linear programming methods, Lemma 4.2 is at a disadvantage in terms of computational costs since solving semidefinite programs is generally more expensive than linear programs. However, linear programming relaxations of the binary constraints $v_i \in \{-1, 1\}$ are more involved than the corresponding semidefinite programming relaxations. Therefore, the same relaxation techniques that were used for treatment of the binary variables in the semidefinite programming formulation are not readily applicable to the linear programming version. See for instance the results of Sherali et. al. [84], [85] on construction of a hierarchy of linear relaxations of binary integer programs. Note that we

do not face the same difficulties in linear programming relaxation of the constraints for the continuous variables $w_i \in [-1, 1]$ (equivalently: $-1 \leq w_i$, and $w_i \leq 1$). Here we propose two possible remedies. The first is to relax the binary constraints and treat them as continuous variables $v_i \in [-1, 1]$ instead of $v_i \in \{-1, 1\}$, which is a conservative over-approximation approach. The second approach is to consider each of the 2^r different possibilities (one for each vertex of $\{-1, 1\}^r$) separately. This approach can be useful if r is small, and is otherwise impractical. More sophisticated schemes can be developed based on hierarchical linear programming relaxations of binary integer programs [84], [85].

Lemma 4.3 *Consider a computer program \mathcal{P} and its Mixed-Integer Linear dynamical system model $\mathcal{P}(F, H, X_0, n, q, r)$. There exists a (linear) rate θ Lyapunov invariant for \mathcal{P} in the class \mathcal{V}_x^1 , if there exists a scalar $D_1 \in \mathbb{R}$, a matrix $Y \in \mathbb{R}^{1 \times m}$, and nonnegative matrices $\underline{D}_v, \overline{D}_v \in \mathbb{R}^{1 \times r}$, $\underline{D}_{xw}, \overline{D}_{xw} \in \mathbb{R}^{1 \times (n+q)}$, and a matrix $K \in \mathbb{R}^{n+1}$ satisfying the following conditions:*

$$\begin{aligned} K^T L_1 - \theta K^T L_2 - YH - (\underline{D}_{xw} - \overline{D}_{xw})L_3 - (\underline{D}_v - \overline{D}_v)L_4 - D_1 L_5 &= 0 \\ D_1 + (\overline{D}_v + \underline{D}_v) \mathbf{1}_r + (\overline{D}_{xw} + \underline{D}_{xw}) \mathbf{1}_{n+q} &\geq 0 \\ \overline{D}_v, \underline{D}_v, \overline{D}_{xw}, \underline{D}_{xw} &\geq 0 \end{aligned}$$

In the following lemma we present sufficient conditions that are less conservative than the conditions of Lemma 4.3, that is, if a linear invariant can be found using Lemma 4.3, then it can be found using Lemma 4.4. The converse is not true. The trade off is that the number of constraints in Lemma 4.4 grows exponentially with respect to the number of binary variables. However, since linear programming software can typically handle very large optimization problems with thousands of constraints, it is feasible to use Lemma 4.4 for programs with about 20 binary variables.

Lemma 4.4 *Consider a computer program \mathcal{P} and its Mixed-Integer Linear dynamical system model $\mathcal{P}(F, H, X_0, n, q, r)$. There exists a (linear) rate θ Lyapunov invariant for \mathcal{P} in the class \mathcal{V}_x^1 , if there exists a scalar $D_1 \in \mathbb{R}$, a matrix $Y \in \mathbb{R}^{1 \times m}$, nonnegative matrices $\underline{D}_{xw}, \overline{D}_{xw} \in \mathbb{R}_+^{1 \times (n+q)}$ and 2^r matrices $D_{iv} \in \mathbb{R}^{1 \times r}$, $i \in \mathbb{Z}(1, 2^r)$, and a matrix $K \in \mathbb{R}^{n+1}$ satisfying the*

following conditions:

$$\begin{aligned}
K^T L_1 - \theta K^T L_2 - YH - (\underline{D}_{xw} - \overline{D}_{xw})L_3 - (D_{iv})L_4 - D_1 L_5 &= 0, \quad i \in \mathbb{Z}(1, 2^r) \\
D_1 + (D_{iv}) \mathbf{b}_{iv} + (\overline{D}_{xw} + \underline{D}_{xw}) \mathbf{1}_{n+q} &\geq 0, \quad i \in \mathbb{Z}(1, 2^r) \\
\overline{D}_{xw}, \underline{D}_{xw} &\geq 0
\end{aligned}$$

where \mathbf{b}_{iv} represents the i^{th} vertex of $\{-1, 1\}^r$.

Remark 4.2 It follows from Lemma 4.3 that a subset of all the linear invariants can be characterized as the set of all solutions of the following system of linear equations:

$$K^T L_1 - \theta K^T L_2 - YH - D_1 L_5 = 0, \quad D_1 \in \{0, 1, -1\}$$

which is obtained by selecting $\overline{D}_v = \underline{D}_v = (\mathbf{1}_r)^T$, and $\overline{D}_{xw} = \underline{D}_{xw} = (\mathbf{1}_{n+q})^T$ as the multiplier vectors. Similarly, using Lemma 4.2 we can characterize a subset of the linear invariants as the set of solutions to the following linear program:

$$\begin{aligned}
&\max \quad \gamma \\
&\text{s.t.} \quad \text{He}(L_1^T K L_5 - \theta L_5^T K^T L_2) = \text{He}(YH) + L_3^T D_{xw} L_3 + L_4^T D_v L_4 - \lambda L_5^T L_5 \\
&\quad \gamma \leq D_{xw} \\
&\quad \lambda = \text{Trace } D_{xw} + \text{Trace } D_v
\end{aligned}$$

If the optimal solution γ^* of the above linear program is nonnegative: $\gamma^* \geq 0$, then a linear invariant has been found. Similar reformulations for the criteria presented in Lemma 4.4 are possible.

Example 4.2 Consider Program 2-4. Application of Lemma 4.3 results in the following linear invariant:

$$x_2 \geq 0.$$

After adding this invariant to the matrix H of the MILM, we reapply Theorem 4.2. This improves the analysis as absence of overflow can now be certified w.r.t. the more restricted

overflow limit $\alpha = 750$. Moreover, the upper bound on the number of iterations improves: $T_{new} = 8.7 \times 10^2$.

4.3 Optimization of Lyapunov Invariants for Graph Models

In Chapter 2, we introduced the graph models for analysis of computer programs. Lyapunov invariants as behavior certificates for the graph models were introduced in Section 3.1.2. In this section, we describe in detail, the numerical procedure for computation of Lyapunov invariants for graph models. Following the same standard procedure that was applied to MILMs, the search for a Lyapunov invariant begins with a finite-dimensional linear parameterization of the search space. Recall that the state in this model is defined by $\tilde{x} := (i, x)$ where i is the discrete component representing a node on the graph, or a line number in the actual code. Further, recall from Section 3.1.2 that we defined Lyapunov invariants for graph models in the following way:

$$V(\tilde{x}) \equiv V(i, x) := \sigma_i(x)$$

where for every $i \in \mathcal{N}$, the function $\sigma_i : \mathbb{R}^n \rightarrow \mathbb{R}$ can be a polynomial, quadratic, or an affine functional. The computational procedure begins with a linear parameterization of the subspace of polynomial functionals of total degree less than or equal to d , mapping \mathbb{R}^n to \mathbb{R} :

$$\mathcal{V}_x^d := \left\{ V : \mathbb{R}^n \rightarrow \mathbb{R} \mid V(x) := K^T Z(x), K \in \mathbb{R}^N, N = \binom{n+d}{d} \right\} \quad (4.14)$$

where $Z(x)$ is the vector of all monomials of degree less than or equal to d in n variables x_1, \dots, x_n . The length of such vector is $N := \binom{n+d}{d}$. For instance, the linear parameterization of all polynomial functions in two variables with degree less than or equal to three is given by:

$$\mathcal{V}_{(x_1, x_2)}^3 := \left\{ \begin{array}{l} V : \mathbb{R}^2 \rightarrow \mathbb{R} \mid V(x) = K^T Z(x), K \in \mathbb{R}^{10} \\ Z(x) = \left[\begin{array}{cccccccccc} x_2^3 & x_2^2 x_1 & x_2 x_1^2 & x_1^3 & x_2^2 & x_1 x_2 & x_1^2 & x_2 & x_1 & 1 \end{array} \right] \end{array} \right\}.$$

Therefore, a linear parametrization of Lyapunov invariants for graph models is given by

$$V(\tilde{x}) \equiv V(i, x) := \sigma_i(x) \quad (4.15)$$

where for every $i \in \mathcal{N}$, the function $\sigma_i(\cdot) \in \mathcal{V}_x^{d(i)}$, and $d(i)$ is an appropriately chosen degree bound for the Lyapunov invariant at node i . Note that we do not require that the degree bounds $d(i)$ to be equal for all i . We will refer to Lyapunov invariants defined according to (4.15) as node-wise Lyapunov invariants. We will explain in the sequel, how convex relaxation methods can be used to formulate the search for a Lyapunov invariant for a graph model as a convex optimization problem. Depending on the dynamics of the model, the degree bounds $d(i)$, and the convex relaxation technique, the optimization problem will become a linear, semidefinite, or a sum-of-squares optimization problem.

4.3.1 Node-wise Polynomial Invariants

For graph models, we present the conditions for existence of polynomial Lyapunov invariants in their generic form in terms of emptiness of semialgebraic sets. The following theorem follows from Corollary 3.2.

Theorem 4.3 *Consider a program \mathcal{P} , and its graph model $G(\mathcal{N}, \mathcal{E})$. Let $V : \mathbb{R}^n \rightarrow \mathbb{R}$, be defined according to (4.15), where $\sigma_i(\cdot) \in \mathcal{V}_x^{d(i)}$. Then, the functions $\sigma_i(\cdot)$, $i \in \mathcal{N}$ define a Lyapunov invariant for \mathcal{P} , if and only if the following semialgebraic sets are empty:*

$$\{(x, x_+) \in \mathbb{R}^n \times \mathbb{R}^n \mid \sigma_j(x_+) - \theta_{ji}^k \sigma_i(x) \geq 0, x \in X_i \cap \Pi_{ji}^k, x_+ \in T_{ij}^k x\}, (i, j, k) \in \mathcal{E}. \quad (4.16)$$

In addition, if $T_{ij}^k x \equiv \{\overline{T}_{ij}^k(x, w) \mid (x, w) \in S_{ij}^k\}$ then, (4.16) can be alternatively described as:

$$\{(x, w) \in \mathbb{R}^n \times \mathbb{R}^q \mid \sigma_j(\overline{T}_{ij}^k(x, w)) - \theta_{ji}^k \sigma_i(x) \geq 0, x \in X_i \cap \Pi_{ji}^k, (x, w) \in S_{ij}^k\}, (i, j, k) \in \mathcal{E}. \quad (4.17)$$

Furthermore, \mathcal{P} satisfies the safety property w.r.t. the collection of sets X_{i-} , $i \in \mathcal{N} \setminus \{\emptyset\}$, if the following additional conditions are satisfied:

$$\{x \mid \sigma_\emptyset(x) \geq 0, x \in X_\emptyset\} = \emptyset \quad (4.18)$$

$$\{x \mid -\sigma_i(x) \geq 0, x \in X_i \cap X_{i-}\} = \emptyset, i \in \mathcal{N} \setminus \{\emptyset\} \quad (4.19)$$

As we discussed in Section 4.1.2, the sum-of-squares relaxation technique can be used for verification of conditions of Theorem 4.3. We do not present the sum-of-squares conditions involving the polynomial multipliers and the SOS multipliers as the notation can become cumbersome.

If $\sigma_i(\cdot)$, $i \in \mathcal{N}$ are quadratic functionals ($d(i) = 2$), the transition operators T_{ji}^k are affine, and the invariant sets X_i and the passport sets Π_{ji}^k have a second order description, then the conditions of Theorem 4.3 are equivalent to nonnegativity of several quadratic functionals subject to quadratic/linear inequalities and/or equalities. In this case, the standard \mathcal{S} -Procedure can be used as the convex relaxation method and the search for a Lyapunov invariant simplifies to a semidefinite optimization problem.

4.3.2 Node-wise Quadratic Invariants for Linear Graphs

Recall from Section 2.2.2 that linear graph models are graph models for which all the transition and passport labels are affine. The only nonlinear constraints are the invariant set constraints which are allowed to be quadratic. In this section, we present a theorem for verification of absence of overflow and finite-time termination for linear graph models via semidefinite optimization of quadratic Lyapunov invariants.

Assume for convenience in notation that for all $i \in \mathcal{N}$, and for all $(i, j, k) \in \mathcal{E}$, a compact description of the set $X_i \cap \Pi_{ji}^k$ is available:

$$X_i \cap \Pi_{ji}^k := \left\{ x \in \mathbb{R}^n \mid x^T Q_i x \leq 1, x^T R_i x = 1, G_{ji}^k x - g_{ji}^k \leq 0, H_{ji}^k x - h_{ji}^k = 0 \right\}.$$

Furthermore, assume that for all $(i, j, k) \in \mathcal{E}$, the row dimensions of the matrices H_{ji}^k are equal. We denote this dimension by n_H . The same assumption is made for the matrices G_{ji}^k , and the row dimension is denoted by n_G . Lastly, recall that the transition labels of the arcs are of the form $T_{ji}^k x = A_{ji}^k x + B_{ji}^k w + E_{ji}^k$, where $w \in [-1, 1]^q$. We now present the following theorem:

Theorem 4.4 *Consider a program \mathcal{P} and its linear graph model $G(\mathcal{N}, \mathcal{E})$. Suppose that the overflow limit is specified by a positive definite diagonal matrix $0 \prec \alpha$. An overflow runtime error does not occur during any execution of \mathcal{P} if there exist symmetric matrices $P_i \in \mathbb{S}^{(n+1) \times (n+1)}$,*

$i \in \mathcal{N}$, matrices $Y \in \mathbb{R}^{(n+q+1) \times n_H}$, $Z \in \mathbb{R}^{(n+q+1) \times n_G}$, a diagonal matrix $\tau \in \mathbb{R}^{q \times q}$, and scalars $\rho, \eta \in \mathbb{R}$, such that the following Linear Matrix Inequalities are satisfied:

$$\begin{bmatrix} x_0 & 1 \end{bmatrix} P_0 \begin{bmatrix} x_0 & 1 \end{bmatrix}^T \leq 0, \quad \forall x_0 \in X_0 \quad (4.20a)$$

$$\Lambda_\alpha - P_i \preceq 0, \quad \forall i \in \mathcal{N} \quad (4.20b)$$

$$-\tau \preceq 0 \quad (4.20c)$$

$$-Z \leq 0, \quad -\eta \leq 0, \quad (4.20d)$$

and for all $(\mathcal{F}, \mathcal{G}, \mathcal{H}, \theta) \in \{(\mathcal{F}_{ji}^k, \mathcal{G}_{ji}^k, \mathcal{H}_{ji}^k, \theta_{ji}^k) \mid (i, j, k) \in \mathcal{E}\}$:

$$\mathcal{F}^T P_j \mathcal{F} - \theta P_i \prec L_w^T \tau L_w - \text{Tr}(\tau) L_1^T L_1 + \text{He}((Y\mathcal{H} + Z\mathcal{G}) L_{x1}) + \lambda \quad (4.21a)$$

$$\lambda = \eta L_{x1}^T Q_i L_{x1} + \rho L_{x1}^T \mathcal{R}_i L_{x1}$$

where

$$\mathcal{F}_{ji}^k = \begin{bmatrix} A_{ji}^k & B_{ji}^k & E_{ji}^k \\ 0_{1 \times n} & 0_{1 \times q} & 1 \end{bmatrix}, \quad Q_i = \begin{bmatrix} Q_i & 0_{n \times 1} \\ 0_{1 \times n} & -1 \end{bmatrix}, \quad \mathcal{R}_i = \begin{bmatrix} R_i & 0_{n \times 1} \\ 0_{1 \times n} & -1 \end{bmatrix}$$

$$\mathcal{H}_{ji}^k := \begin{bmatrix} H_{ji}^k & -h_{ji}^k \end{bmatrix}, \quad \mathcal{G}_{ji}^k := \begin{bmatrix} G_{ji}^k & -g_{ji}^k \end{bmatrix}$$

$$L_w = \begin{bmatrix} 0_{q \times n} & I_q & 0_{q \times 1} \end{bmatrix}, \quad L_1 = \begin{bmatrix} 0_{1 \times n} & 0_{1 \times q} & 1 \end{bmatrix}, \quad L_x = \begin{bmatrix} I_n & 0_{n \times q} & 0_{n \times 1} \end{bmatrix}$$

$$L_{x1} = \begin{bmatrix} L_x \\ -L_1 \end{bmatrix}, \quad \Lambda_\alpha = \text{diag} \{ \alpha^{-2}, -1 \}$$

In addition, if (3.13) holds, then \mathcal{P} terminates in at most T steps, where

$$T = \sum_{\mathcal{C} \in \mathcal{G}} \frac{-\log \min_{x \in X_\emptyset} \left| \begin{bmatrix} x_0 & 1 \end{bmatrix} P_0 \begin{bmatrix} x_0 & 1 \end{bmatrix}^T \right|}{\log \theta(\mathcal{C})}, \quad \theta(\mathcal{C}) := \prod_{(i,j,k) \in \mathcal{C}} \theta_{ij}^k.$$

Proof. The theorem can be proven by direct application of the \mathcal{S} -Procedure relaxation

technique to Corollary 3.2, in a similar fashion to the proof of Lemma 4.10. ■

Remarks

1. In Theorem 4.4, the multipliers are fixed across all the arcs. That is, one set of multipliers is used for relaxations of analogous constraints corresponding to different arcs. Less conservative but more complicated versions can be formulated by allowing the multipliers to be functions of the arcs. In that case, a different set of multipliers must be defined for each arc $(i, j, k) \in \mathcal{E}$. This is equivalent to replacing the decision parameters τ, ρ, η, Z, Y in Theorem 4.4 by $\tau_{ji}^k, \rho_{ji}^k, \eta_{ji}^k, Z_{ji}^k, Y_{ji}^k$.
2. The constraints $\Lambda_\alpha - P_i \preceq 0, \forall i \in \mathcal{N}$ can be replaced by their relaxed versions. Assume for instance that

$$X_i := \{x \in \mathbb{R}^n \mid x^T Q_i x \leq 1, x^T R_i x = 1\}$$

Then, (4.20b) can be replaced by:

$$\Lambda_\alpha - P_i \preceq \gamma_i Q_i + \beta_i R_i, \quad \forall i \in \mathcal{N}$$

where $\gamma_i \in \mathbb{R}_+$, and $\beta_i \in \mathbb{R}$ are decision parameters, and Q_i and R_i are defined as before.

4.4 Case Study

Program 4-1 (see next page) has been adapted with some minor modifications from a similar program available at the ASTREE website. The only feature that we have added is the real-time input $w \in [-1, 1]$ at line L4. In Program 4-1, the function `saturate(.)` is a user-defined function inserted for safety reasons. The purpose of the function is to truncate the real-time input `*PtrToInput` so that $w \in [-1, 1]$ is guaranteed. We first build a dynamical system model of this program and then analyze its properties. The variables of the program are:

Z, Y, E[0], E[1], S[0], S[1], INIT.

INIT is Boolean which we model by $v \in \{-1, 1\}$. That is:

$$\text{INIT} = \text{True} \Leftrightarrow v = 1, \text{ and } \text{INIT} = \text{False} \Leftrightarrow v = -1.$$

```

/* filter.c */
typedef enum {FALSE = 0, TRUE = 1} BOOLEAN;
BOOLEAN INIT; float Y={0}, Z={0};

F0 : void filter () {
F1 : static float E[2], S[2];
F2 : if (INIT) {
F3 :     S[0] = Z;
F4 :     Y = Z;
F5 :     E[0] = Z;
F6 : } else {
F7 :     Y = (((((0.5*Z) - (E[0]*0.7)) + (E[1]*0.4)) + (S[0]*1.5)) - (S[1]*0.7));
F8 :     }
F9 : E[1] = E[0];
F10 : E[0] = Z;
F11 : S[1] = S[0];
F12 : S[0] = Y;
F13 : }

L0 : void main () {
L1 : Z = 0.2 * Z + 5;
L2 : INIT = TRUE;
L3 : while (1) {
        wait (0.001); w = saturate(*PtrToInput); /*updates real-time input*/
L4 :     Z = 0.9 * Z + 35+20*w;
L5 :     filter ();
L6 :     INIT = FALSE;
L7 :     }
L∞ : }

```

Program 4-1: Safety-critical software

Example adapted from Reference [94] with minor modifications.

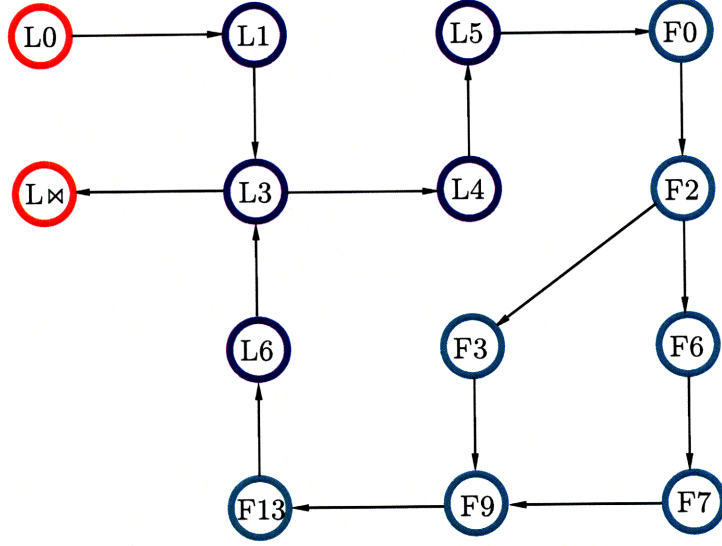


Figure 4-1: The graph of Program 4-4.

The remaining variables are floats. To simplify the exposition let us model them as reals and assume that the computations are exact. The graph model of this program is shown in Figure 4-1.

In order to construct a more compact model, several lines of code corresponding to consecutive linear assignments have been combined, and only the first line corresponding to a series of consecutive linear transformations is assigned a node on the graph. Specifically, the combined lines are: $\{L1, L2\}$, and $\{F3, F4, F5\}$, and $\{F9, F10, F11, F12\}$. The net effect of the eliminated lines will be captured in the transition labels by taking the composition of the functions that are coded at these lines.

Let us define $x = [Z, Y, E[0], E[1], S[0], S[1], \text{INIT}]^T$. The state space of this program is then $\mathcal{N} \times \mathbb{R}^7$, where \mathcal{N} is the set of nodes shown in Figure 4-1. The initial condition is fixed: $X_0 := \{0\} \times x_0$ where $x_0 = [0, 0, 0, 0, 0, 0, 1]^T$. Also, since $\text{INIT} \in \{-1, 1\}$, we add the invariant set $X_{\mathcal{N}} := \{x \in \mathbb{R}^7 \mid x_7^2 = 1\}$ to every node $i \in \mathcal{N}$.

The transitions associated with the arcs are all affine and are as follows ($w \in [-1, 1]$ is the real-time input):

$$T_{L3L1} : x \mapsto [0.2Z + 5, Y, E[0], E[1], S[0], S[1], 1]$$

$$T_{L5L4} : x \mapsto [0.9Z + 35 + 20w, Y, E[0], E[1], S[0], S[1], \text{INIT}]$$

$$T_{L3L6} : x \mapsto [Z, Y, E[0], E[1], S[0], S[1], -1]$$

$$T_{F9F3} : x \mapsto [Z, Z, Z, E[1], Z, S[1], \text{INIT}]$$

$$T_{F9F7} : x \mapsto [Z, 0.5Z - 0.7E[0] + 0.4E[1] + 1.5S[0] - 0.7S[1], E[0], E[1], S[0], S[1], \text{INIT}]$$

$$T_{F13F9} : x \mapsto [Z, Y, Z, E[0], Y, S[0], \text{INIT}]$$

The remaining transitions are identity. Furthermore, there are only two branches on the graph. The first one is at node L3. However, since $\Pi_{L4L3} := \mathbb{R}^7$, $\Pi_{L \times L3} := \emptyset$, the only possible transition is from L3 to L4. The second branch is at node F2, with $\Pi_{F6F2} := \{x \in \mathbb{R}^7 \mid x_7 = -1\}$, and $\Pi_{F3F2} := \{x \in \mathbb{R}^7 \mid x_7 = 1\}$. The remaining passport labels are universal ($\Pi = \mathbb{R}^7$). For computational purposes, it is convenient to represent all the transformations and sets with matrices. For instance, each transition label is of the form: $T_{ji} := A_{ji}x + B_{ji}w + E_{ji}$. The corresponding matrices are given in the Appendix section at the end of this chapter.

The graph model is linear and we can apply Theorem 4.4 to verify absence of overflow in the program via nodewise quadratic Lyapunov invariants. Since the program does not terminate in finite time, only absence of overflow can be verified. By applying Theorem 4.4 with $\theta_{ji} = 0.985$ for all i, j , and $\alpha = 1000$ as the overflow limit, absence of overflow is verified.

4.5 Summary

In this chapter, we presented a computational procedure based on convex relaxation techniques and convex, numerical optimization for computation of the Lyapunov invariants that we proposed in Chapter 3. We showed that sufficient criteria for verification of safety, liveness and critical performance properties of computer programs can be formulated as a semidefinite program, or a sum-of-squares program. We formulated these results for both the mixed-integer linear models and the graph models. The convex optimization phase is the final step in our software analysis framework. If the optimization problem is feasible, the result is a certificate

for safety and/or finite-time termination of the computer program, otherwise, the analysis is inconclusive. We also showed that the same techniques can be applied for numerical computation of linear invariants. While linear invariants are often insufficient for proving safety and liveness properties of computer programs of safety-critical systems, including the linear invariants in the model can improve the results of analysis via stronger (quadratic or polynomial) invariants.

4.6 Appendix

Each transition label is of the form: $T_{ji} := A_{ji}x + B_{ji}w + E_{ji}$. The matrices corresponding to the non-identity transition labels are:

$$A_{L3L1} = \begin{bmatrix} 0.2 & \mathbf{0}_{1 \times 5} & 0 \\ \mathbf{0}_{5 \times 1} & I_5 & \mathbf{0}_{5 \times 1} \\ 0 & \mathbf{0}_{1 \times 5} & 0 \end{bmatrix}, E_{L3L1} = \begin{bmatrix} 5 \\ \mathbf{0}_{5 \times 1} \\ 1 \end{bmatrix}, B_{L3L1} = \mathbf{0}_{7 \times 1}$$

$$A_{L5L4} = \begin{bmatrix} 0.9 & \mathbf{0}_{1 \times 6} \\ \mathbf{0}_{6 \times 1} & I_6 \end{bmatrix}, E_{L5L4} = \begin{bmatrix} 35 \\ \mathbf{0}_{6 \times 1} \end{bmatrix}, B_{L5L4} = \begin{bmatrix} 20 \\ \mathbf{0}_{6 \times 1} \end{bmatrix}$$

$$A_{F9F3} = \begin{bmatrix} a_{11} & \mathbf{0}_{3 \times 6} \\ a_{21} & a_{22} \end{bmatrix}, E_{F9F3} = \mathbf{0}_{7 \times 1}, B_{F9F3} = \mathbf{0}_{7 \times 1}$$

$$a_{11} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, a_{21} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, a_{22} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_{L3L6} = \begin{bmatrix} I_6 & \mathbf{0}_{6 \times 1} \\ \mathbf{0}_{1 \times 6} & 0 \end{bmatrix}, L_{L3L6} = \begin{bmatrix} \mathbf{0}_{6 \times 1} \\ -1 \end{bmatrix}, B_{L3L6} = \mathbf{0}_{7 \times 1}$$

$$A_{F13F9} = \begin{bmatrix} a_{11} & \mathbf{0}_{3 \times 5} \\ a_{21} & a_{22} \end{bmatrix}, E_{F13F9} = \mathbf{0}_{7 \times 1}, B_{F13F9} = \mathbf{0}_{7 \times 1}$$

$$a_{11} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}, a_{21} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, a_{22} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_{F9F7} = \begin{bmatrix} a_{11} & a_{12} \\ \mathbf{0}_{5 \times 2} & I_5 \end{bmatrix}, E_{F13F9} = \mathbf{0}_{7 \times 1}, B_{F13F9} = \mathbf{0}_{7 \times 1}$$

$$a_{11} = \begin{bmatrix} 1 & 0 \\ 0.5 & 0 \end{bmatrix}, a_{12} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ -0.7 & 0.4 & 1.5 & -0.7 & 0 \end{bmatrix}$$

The invariant set $X_{\mathcal{N}} := \{x \in \mathbb{R}^7 \mid x_7^2 = 1\}$ is represented in matrix form by: $X_{\mathcal{N}} := \{x \in \mathbb{R}^7 \mid x^T Q x = 1\}$, where

$$Q := \begin{bmatrix} \mathbf{0}_{6 \times 6} & \mathbf{0}_{6 \times 1} \\ \mathbf{0}_{1 \times 6} & 1 \end{bmatrix}$$

Finally, the non-universal passport labels are $\Pi_{F6F2} := \{x \in \mathbb{R}^7 \mid e_7^T x = -1\}$, and $\Pi_{F3F2} := \{x \in \mathbb{R}^7 \mid e_7^T x = 1\}$, where

$$e_7 = \begin{bmatrix} \mathbf{0}_{1 \times 6} & 1 \end{bmatrix}^T.$$

Chapter 5

Optimal Graph Models

In the previous chapters, we focused on developing a systematic framework for analysis of dynamical system models of software. In particular, we presented several results for analysis of graph models of software via Lyapunov invariants and convex optimization techniques. In this chapter, we introduce a notion of optimality for graph models. First, we motivate the discussion by comparing the results of application of the framework to two programs that are semantically identical but syntactically different. We observe that the results of node-wise Lyapunov analysis of graph models of these programs are not identical. This suggests that the success or failure of the method is contingent on the choice of the graph model. Based on these observations, we introduce the concepts of graph reduction, irreducible graphs, and minimal and maximal realizations of graph models. We present several new theorems that compare the original graph model of a computer program and its reduced offsprings in terms of existence and computability of the Lyapunov invariants. These results can be used for constructing efficient graph models that improve the analysis in a systematic way.

5.1 Motivation

We begin this section by presenting an example that motivates several discussions and concepts that will be introduced in the upcoming sections. The example is built based on the observation that a dynamical system or an algorithm can be implemented using a variety of different syntaxes that programming languages provide. However, the graph model interpretations of the

(syntactically) different programs that implement the same dynamical system are not identical. This leads to surprising results when analysis of the graph models via node-wise Lyapunov invariants is undertaken.

Consider the following two programs \mathcal{P}_1 , and \mathcal{P}_2 , which have identical functionality. They both implement a switched dynamical system in two variables x_1 and x_2 , with bounded initial conditions.

```

L0 : % pre:  $x_1, x_2 \in [-100, 100]$ 
L1 : while  $x_1^2 + x_2^2 \geq 0.001$ ,
L2 : if  $x_1^2 - x_2^2 \leq 0$ 
       $x_1 = 0.99x_1 + 0.01x_2$ ;
L3 :    $x_2 = -0.05x_1 + 0.99x_2$ ;
      else
L5 :    $x_1 = 0.99x_1 + 0.05x_2$ ;
       $x_2 = -0.01x_1 + 0.99x_2$ ;
      end
L∞: end

```

Program 5-1: \mathcal{P}_1

```

L0 : % pre:  $x_1, x_2 \in [-100, 100]$ 
L1 : while  $x_1^2 + x_2^2 \geq 0.001$ ,
L2 : while  $x_1^2 - x_2^2 \leq 0$ 
       $x_1 = 0.99x_1 + 0.01x_2$ ;
L3 :    $x_2 = -0.05x_1 + 0.99x_2$ ;
      end
L4 : while  $x_1^2 - x_2^2 > 0$ 
       $x_1 = 0.99x_1 + 0.05x_2$ ;
L5 :    $x_2 = -0.01x_1 + 0.99x_2$ ;
      end
L∞: end

```

Program 5-2: \mathcal{P}_2

Remark 5.1 *To avoid distracting details, we have eliminated the buffer variables from these programs. We assume that the assignments corresponding to the two consecutive lines at locations 3 and 5 execute simultaneously.*

These programs are semantically identical, but they are written in a slightly different way. In other words, if we define $x(t) := (x_1(t), x_2(t))$, $X := \mathbb{R}^2$, $X_0 := [-100, 100]^2 \subset \mathbb{R}^2$, and $X_\infty := \{x \in \mathbb{R}^2 \mid x_1^2 + x_2^2 < 10^{-3}\}$, then the set of all sequences $\mathcal{X}_{\mathcal{P}_1} := (x(0), x(1), \dots, x(t), \dots)$ corresponding to $\mathcal{S}_1(X, f, X_0, X_\infty)$ (the dynamical system model of program \mathcal{P}_1), and the set of all

sequences $\mathcal{X}_{\mathcal{P}_2} := (x(0), x(1), \dots, x(t), \dots)$ corresponding to $\mathcal{S}_2(X, f, X_0, X_\infty)$ (the dynamical system model of \mathcal{P}_2) are identical. Therefore, program \mathcal{P}_1 is correct if and only if program \mathcal{P}_2 is correct. Indeed, they are both correct in the sense of absence of overflow and termination in finite time. The graph models of \mathcal{P}_1 and \mathcal{P}_2 , which are of order 4 and 5 respectively¹, are shown in Figure 5-1. For the graph of \mathcal{P}_1 , the non-universal passport labels are:

$$\begin{aligned} \Pi_{12} &:= \{x \in \mathbb{R}^2 \mid x_1^2 + x_2^2 \geq 10^{-3}\}, \quad \Pi_{1\bowtie} := \{x \in \mathbb{R}^2 \mid x_1^2 + x_2^2 < 10^{-3}\} \\ \Pi_{23} &:= \{x \in \mathbb{R}^2 \mid x_1^2 - x_2^2 \leq 0\}, \quad \Pi_{25} := \{x \in \mathbb{R}^2 \mid x_1^2 - x_2^2 > 0\} \end{aligned}$$

and the non-identity transition labels are T_{3x} and T_{5x} where:

$$T_{3x} : \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \mapsto \begin{bmatrix} 0.99 & 0.01 \\ -0.05 & 0.99 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad \text{and} \quad T_{5x} : \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \mapsto \begin{bmatrix} 0.99 & 0.05 \\ -0.01 & 0.99 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}.$$

The remaining passport labels are universal, meaning that they represent the entire state space \mathbb{R}^2 . Also, the remaining transition labels represent the identity map. For clarity, only the non-identity transitions and non-universal passport labels are shown in Figure 5-1. For the graph of \mathcal{P}_2 , the passport labels Π_{12} , $\Pi_{1\bowtie}$, and Π_{23} are the same as the ones defined above for \mathcal{P}_1 . Furthermore,

$$\Pi_{24} = \Pi_{45} = \Pi_{25} := \{x \in \mathbb{R}^2 \mid x_1^2 - x_2^2 > 0\}, \quad \Pi_{41} = \Pi_{23} := \{x \in \mathbb{R}^2 \mid x_1^2 - x_2^2 \leq 0\}.$$

The remaining passport labels represent the entire state space \mathbb{R}^2 . The non-identity transition labels are T_{3x} and T_{5x} , which correspond to the arcs (3, 2) and (5, 4) respectively. These are defined as before. The remaining transition labels are identity.

We now consider the problem of finding a node-wise Lyapunov invariant for these models. For simplicity, let us consider the node-wise quadratic case with constant rate $\theta = 1$ across all the arcs. As discussed earlier, we assign a quadratic function $\sigma_i(x) := x^T P_i x$ to every node on the graph and impose the Lyapunov conditions according to Proposition 3.4. At this point, we are only concerned with finding a Lyapunov invariant which satisfies (3.6). We are not imposing

¹Recall that the order of a graph model is defined as the number of nodes on the graph not counting nodes \emptyset and \bowtie .

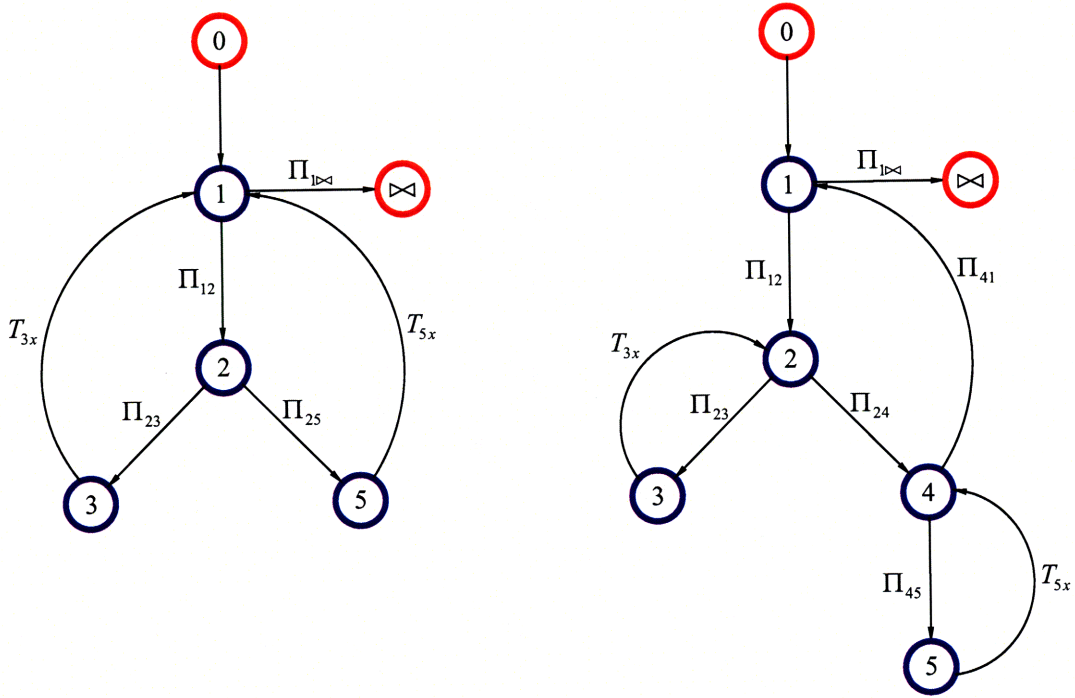


Figure 5-1: Graph Models of Programs \mathcal{P}_1 (left) and \mathcal{P}_2 (right).

the additional constraints for proving stronger properties. For these programs we have:

Lyapunov conditions for program \mathcal{P}_1

$$\begin{aligned} \sigma_1(x) &< \sigma_0(x) \text{ s.t. } x \in [-10^2, 10^2] \\ \sigma_2(x) &< \sigma_1(x) \text{ s.t. } x_1^2 + x_2^2 \geq 10^{-3} \\ \sigma_3(x) &< \sigma_2(x) \text{ s.t. } x_1^2 - x_2^2 \leq 0 \\ \sigma_5(x) &< \sigma_2(x) \text{ s.t. } x_1^2 - x_2^2 > 0 \\ \sigma_1(T_{x3}x) &< \sigma_3(x) \\ \sigma_1(T_{x5}x) &< \sigma_5(x) \\ \sigma_{\infty}(x) &< \sigma_1(x) \text{ s.t. } x_1^2 + x_2^2 < 10^{-3} \end{aligned}$$

Lyapunov conditions for program \mathcal{P}_2

$$\begin{aligned} \sigma_1(x) &< \sigma_0(x) \text{ s.t. } x \in [-10^2, 10^2] \\ \sigma_2(x) &< \sigma_1(x) \text{ s.t. } x_1^2 + x_2^2 \geq 10^{-3} \\ \sigma_3(x) &< \sigma_2(x) \text{ s.t. } x_1^2 - x_2^2 \leq 0 \\ \sigma_4(x) &< \sigma_2(x) \text{ s.t. } x_1^2 - x_2^2 > 0 \\ \sigma_2(T_{x3}x) &< \sigma_3(x) \\ \sigma_5(x) &< \sigma_4(x) \text{ s.t. } x_1^2 - x_2^2 > 0 \\ \sigma_4(T_{x5}x) &< \sigma_5(x) \\ \sigma_1(x) &< \sigma_4(x) \text{ s.t. } x_1^2 - x_2^2 \leq 0 \\ \sigma_{\infty}(x) &< \sigma_1(x) \text{ s.t. } x_1^2 + x_2^2 < 10^{-3} \end{aligned}$$

We then apply the \mathcal{S} -Procedure to formulate these conditions as convex constraints and solve for the parameters of P_i via semidefinite programming. The result of this experiment is somewhat surprising. Although the two programs define the exact same trajectories for the state variables x_1 and x_2 , the LMI optimization problem corresponding to the graph model of \mathcal{P}_2 is feasible, while the LMI optimization problem corresponding to \mathcal{P}_1 is infeasible. Interestingly, this phenomenon is not due to the fact that the graph model of \mathcal{P}_2 is of higher order and the corresponding Lyapunov invariant has more parameters. To understand this situation better, we introduce the notions of graph reduction, irreducible graph models, and minimal and maximal realizations of graphs.

5.2 Graph Reduction and Irreducible Graph Models

Definition 5.1 *A node $i \in \mathcal{N} \setminus \{\emptyset, \infty\}$ is called a focal node if there exists an arc from node i to itself, that is:*

$$\exists k, \text{ s.t. } (i, i, k) \in \mathcal{E}.$$

A node $i \in \mathcal{N} \setminus \{\emptyset, \infty\}$ is called a transient node if it is not a focal node. A graph model is called irreducible if every node $i \in \mathcal{N} \setminus \{\emptyset, \infty\}$ is a focal node.

In reference to the graph models of programs \mathcal{P}_1 and \mathcal{P}_2 (shown in Figure 5-1), every node $i \in \mathcal{N} \setminus \{\emptyset, \infty\}$ on those graphs is a transient node. Hence, the graphs are not irreducible. Now, we introduce the concept of reduction of graph models.

Algorithm 5.1 *Consider a graph $G(\mathcal{N}, \mathcal{E})$ and let $\alpha \in \mathcal{N} \setminus \{\emptyset, \infty\}$ be a transient node. A reduced graph $G_r(\mathcal{N}_r, \mathcal{E}_r)$ can be obtained from G according to the following procedure:*

1. Remove the transient node α , and all the pertinent incoming and outgoing arcs.
2. For every pair of arcs $\{(i, \alpha, r), (\alpha, j, s)\}$ where $i \in \mathcal{I}(\alpha)$ and $j \in \mathcal{O}(\alpha)$, add a new arc (i, j, k) with the transition label $T_{ji}^k := T_{j\alpha}^s T_{\alpha i}^r$ and passport label $\Pi_{ji}^k := \Pi_{j\alpha}^s (T_{\alpha i}^r) \cap \Pi_{\alpha i}^r$.

If $G_r(\mathcal{N}_r, \mathcal{E}_r)$ is a reduced graph model obtained from $G(\mathcal{N}, \mathcal{E})$, we call it an offspring of G and write $G_r \sqsubseteq G$. An irreducible offspring of a connected graph $G(\mathcal{N}, \mathcal{E})$ can be obtained by repeating the above process until every transient node is eliminated.

Remarks

1. If α and i are transient nodes such that $i \in \mathcal{I}(\alpha)$, and $i \in \mathcal{O}(\alpha)$ then by removing node α according to Algorithm 5.1, a loop will be added to node i . Hence, node i will be converted into a focal node after this reduction step.
2. A reduced offspring of a connected graph is connected.
3. Suppose that $G_r(\mathcal{N}_r, \mathcal{E}_r)$ is obtained from $G(\mathcal{N}, \mathcal{E})$ by removing node α in a reduction step. For $i \in \mathcal{N}$, let $\{\mathcal{I}(i)\}_G$ and $\{\mathcal{O}(i)\}_G$ denote the set of incoming and outgoing nodes corresponding to node i on graph G , that is:

$$\{\mathcal{I}(i)\}_G := \{j \mid (j, i, \cdot) \in \mathcal{E}\}, \quad \{\mathcal{O}(i)\}_G := \{j \mid (i, j, \cdot) \in \mathcal{E}\}.$$

Similarly, define:

$$\{\mathcal{I}(i)\}_{G_r} := \{j \mid (j, i, \cdot) \in \mathcal{E}_r\}, \quad \{\mathcal{O}(i)\}_{G_r} := \{j \mid (i, j, \cdot) \in \mathcal{E}_r\}.$$

Then, for $j \in \mathcal{N}_r$, we have

$$\begin{aligned} \{\mathcal{O}(j)\}_{G_r} &= \{\mathcal{O}(j)\}_G & \text{if } (j, \alpha, \cdot) \notin \mathcal{E} \\ \{\mathcal{I}(j)\}_{G_r} &= \{\mathcal{I}(j)\}_G & \text{if } (\alpha, j, \cdot) \notin \mathcal{E} \end{aligned}$$

4. The set of all reduced offsprings of G do not form an ordered set, in the sense that if $G_{r_1} \sqsubseteq G$ and $G_{r_2} \sqsubseteq G$, neither $G_{r_1} \sqsubseteq G_{r_2}$ nor $G_{r_2} \sqsubseteq G_{r_1}$ has to hold. However, \sqsubseteq defines a partial order on the set of all offsprings of G .

Note that the irreducible offspring of G is not unique, neither is its order (number of nodes). Among all the irreducible offsprings of G , we call the one(s) with minimal order a minimal realization of G . Similarly, among all the irreducible offsprings of G , we call the one(s) with maximal order a maximal realization of G . If G is reducible, the minimal or maximal realizations of G may not be unique either. The order of a minimal realization of G is called the minimal order of G and the order of a maximal realization of G is called the maximal order of G . The maximal order of G is equal to the order of G if and only if G is irreducible. The same is true

for the minimal order of G : the minimal order of G is equal to the order of G if and only if G is irreducible.

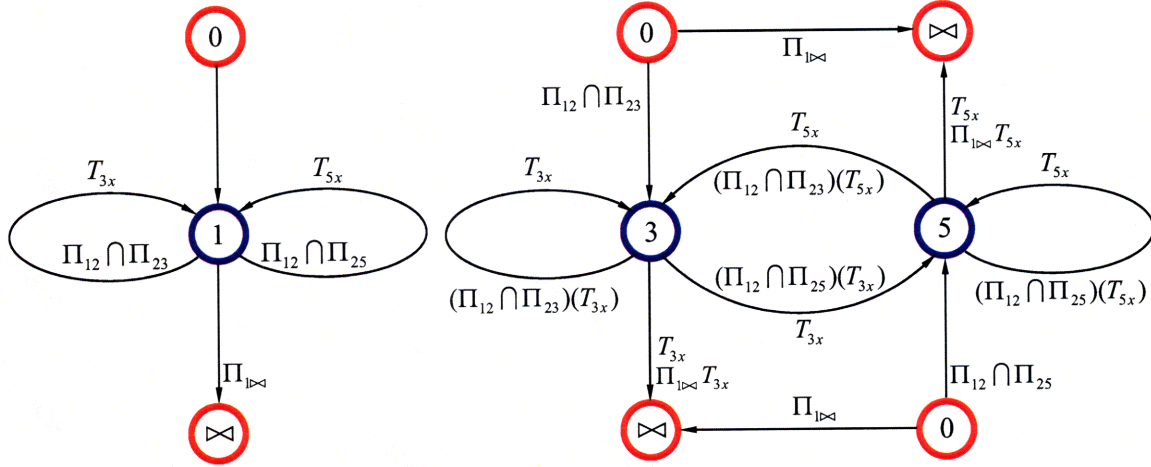


Figure 5-2: Minimal (left) and Maximal (right) realizations for program \mathcal{P}_1 .

The graph model of Program \mathcal{P}_1 is of minimal order 1 and maximal order 2. A particular minimal and a particular maximal realization are shown in Figure 5-2. An alternative minimal realization could have been obtained by eliminating nodes 3, 5, and 1, which would have left node 2 as the only remaining focal node. It happens that in this case, the maximal realization of \mathcal{P}_1 (Figure 5-2) is unique. It can be verified that the graph model of \mathcal{P}_2 is of minimal order 2, and maximal order 3. A minimal realization of \mathcal{P}_2 can be obtained through a reduction process that eliminates nodes 3, 5, and 1 (Figure 5-3). A maximal realization of \mathcal{P}_2 can be obtained via a reduction process that eliminates nodes 2, and 4 (Figure 5-4).

We learned in Section 5.1 that a node-wise quadratic Lyapunov invariant which is valid for the graph model of \mathcal{P}_2 does exist, while for the graph model of \mathcal{P}_1 such function could not be found. The following theorem states that existence of a node-wise Lyapunov invariant within a specific class of functions for a reduced offspring $G_r(\mathcal{N}_r, \mathcal{E}_r) \sqsubseteq G(\mathcal{N}, \mathcal{E})$ is a necessary but not sufficient condition for existence of a node-wise Lyapunov invariant within the same class of functions for the original graph $G(\mathcal{N}, \mathcal{E})$.

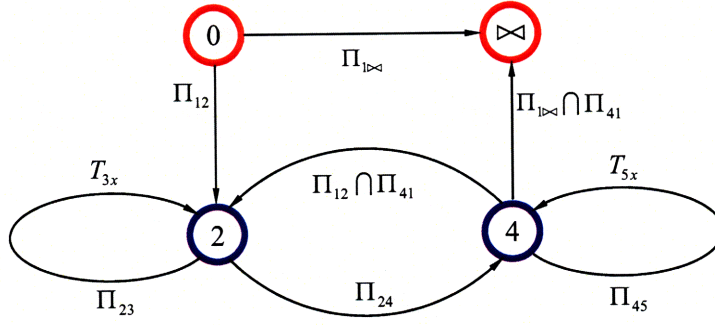


Figure 5-3: A minimal realization for program \mathcal{P}_2 .

Theorem 5.1 Consider a computer program \mathcal{P} and its graph model $G(\mathcal{N}, \mathcal{E})$. Let $G_r(\mathcal{N}_r, \mathcal{E}_r) \sqsubseteq G(\mathcal{N}, \mathcal{E})$ be a reduced offspring of G . If the function

$$V(i, x) := \sigma_i(x), \quad \sigma_i(x) \in \mathcal{V}_x^d, \quad i \in \mathcal{N}$$

is a node-wise polynomial Lyapunov invariant of maximum degree d for the graph $G(\mathcal{N}, \mathcal{E})$, then there exists a node-wise polynomial Lyapunov invariant $V_r(i, x)$ of maximum degree d , that is valid for $G_r(\mathcal{N}_r, \mathcal{E}_r)$. However, if

$$V_r(i, x) := \sigma_i(x), \quad \sigma_i(x) \in \mathcal{V}_x^d, \quad i \in \mathcal{N}_r$$

is a node-wise polynomial Lyapunov invariant of maximum degree d for the graph $G_r(\mathcal{N}_r, \mathcal{E}_r)$, a node-wise polynomial Lyapunov invariant of maximum degree d that is valid for $G(\mathcal{N}, \mathcal{E})$ does not necessarily exist.

Proof. If $G_r \sqsubseteq G$, then there exists a sequence of reduced graph models $G_i(\mathcal{N}_i, \mathcal{E}_i)$, $i = 1 \dots q$, where $G_1 = G$, $G_{i+1} \sqsubseteq G_i$, and $G_q = G_r$, with the property that $|\mathcal{N}_{i+1}| = |\mathcal{N}_i| - 1$. That is, G_{i+1} is obtained from G_i by removing one transient node. Furthermore, assume that V is a Lyapunov invariant for G_i and that G_{i+1} is derived from G_i by eliminating node n . Then:

$$\sigma_n(x_+) - \theta \sigma_m(x) < 0, \quad (x, x_+) \in \Pi_{nm}^r \times T_{nm}^r x, \quad m \in \mathcal{I}(n), \quad r \in \mathcal{A}_{nm}, \quad (5.1)$$

$$\sigma_l(x_+) - \theta \sigma_n(x) < 0, \quad (x, x_+) \in \Pi_{ln}^s \times T_{ln}^s x, \quad l \in \mathcal{O}(n), \quad s \in \mathcal{A}_{ln}. \quad (5.2)$$

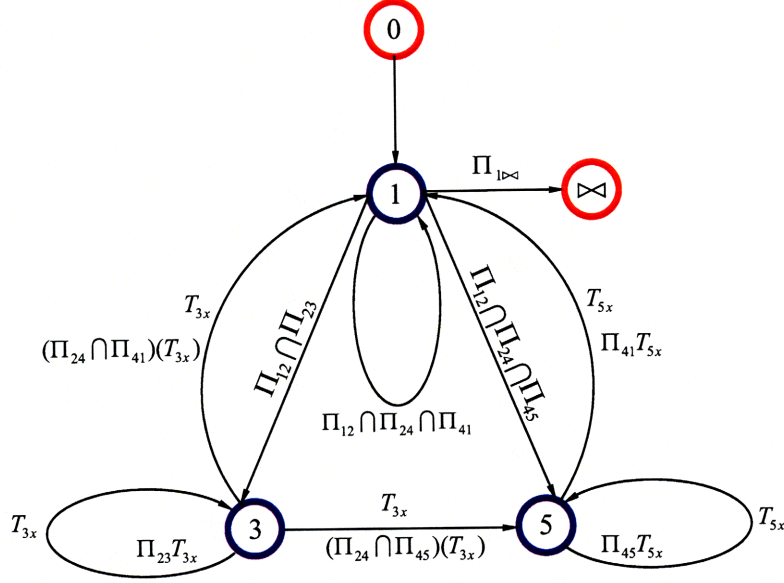


Figure 5-4: A Maximal realization for program \mathcal{P}_2 .

Let $y \in T_{nm}^r x$, and let $y_+ \in T_{ln}^s y$. Then if $y \in \Pi_{ln}^s$, the pair (y, y_+) must satisfy (5.2). Therefore, a necessary condition for (5.2) to hold is that:

$$\begin{aligned} \sigma_l(x_+) - \theta \sigma_n(x) < 0, \quad (x, x_+) \in \Pi_{ln}^s(T_{nm}^r) \times T_{ln}^s T_{nm}^r x, \\ l \in \mathcal{O}(n), \quad s \in \mathcal{A}_{ln}, \quad m \in \mathcal{I}(n), \quad r \in \mathcal{A}_{nm}. \end{aligned} \quad (5.3)$$

Now, for a fixed quadruplet (m, l, r, s) (representing a transition from m to n along arc r , followed by a transition from n to l along arc s), if $x \in \Pi_{nm}^r \cap \Pi_{ln}^s(T_{nm}^r)$ then the corresponding inequalities in both (5.3) and (5.1) hold. Thus, by multiplying the inequalities in (5.1) by θ and adding them to (5.3) we obtain:

$$\sigma_l(x_+) - \theta^2 \sigma_m(x) < 0, \quad (x, x_+) \in (\Pi_{nm}^r \cap \Pi_{ln}^s(T_{nm}^r)) \times (T_{ln}^s T_{nm}^r x), \quad r \in \mathcal{A}_{nm}, \quad s \in \mathcal{A}_{ln}.$$

By definition, this implies that σ_l and σ_m satisfy the Lyapunov conditions along all the arcs that were added between $\mathcal{I}(n)$ and $\mathcal{O}(n)$ in the reduction process (there are exactly $|\mathcal{A}_{nm}| \times |\mathcal{A}_{ln}|$ such arcs). Since σ_l and σ_m satisfy the Lyapunov conditions along any and all the existing

arcs (before reduction), we conclude that V satisfies the Lyapunov conditions for the reduced model. It follows by induction that the function

$$V_r(i, x) := \sigma_i(x), \quad i \in \mathcal{N}_r$$

is a valid Lyapunov invariant on G_r . This completes the proof of the first statement. To prove the second statement of the theorem, it is sufficient to present a counterexample. Indeed, the original graph of \mathcal{P}_1 (Figure 5-1) does not admit a nodewise quadratic Lyapunov invariant, while the maximal realization of \mathcal{P}_1 (Figure 5-2) admits a node-wise quadratic Lyapunov invariant (an explicit description of the invariant function is presented in the sequel). ■

In analysis of computer programs via Lyapunov invariants, an important issue is to determine whether a program admits a certain type of Lyapunov invariant, e.g. quartic polynomial, piece-wise quadratic, etc. For instance, consider program \mathcal{P}_1 , which does not admit a quadratic Lyapunov invariant as we are about to show. Recall that the graph model of this program is of order 4 (nodes \emptyset , and \bowtie are not counted), of minimal order 1, and maximal order 2. Lyapunov analysis of the minimal graph of \mathcal{P}_1 leads to the following LMI problem: Find a symmetric matrix $P_1 \in \mathbb{S}^{2 \times 2}$ such that

$$(T_{3x}x)^T P_1 (T_{3x}x) - x^T P_1 x \leq 0, \quad \forall x \in \Pi_{12} \cap \Pi_{23}, \quad (5.4a)$$

$$(T_{5x}x)^T P_1 (T_{5x}x) - x^T P_1 x \leq 0, \quad \forall x \in \Pi_{12} \cap \Pi_{25}. \quad (5.4b)$$

Since T_{3x} and T_{5x} are homogeneous, the constraint $x \in \Pi_{12} := \{x \in \mathbb{R}^2 \mid x_1^2 + x_2^2 \geq 10^{-3}\}$ becomes irrelevant. Therefore, a symmetric matrix satisfying (5.4) exists, if and only if there exists $P_1 \in \mathbb{S}^{2 \times 2}$ such that

$$(T_{3x}x)^T P_1 (T_{3x}x) - x^T P_1 x \leq 0, \quad \text{s.t. } x^T Q x \leq 0 \quad (5.5a)$$

$$(T_{5x}x)^T P_1 (T_{5x}x) - x^T P_1 x \leq 0, \quad \text{s.t. } x^T Q x > 0 \quad (5.5b)$$

where $Q = \text{diag}\{1, -1\}$. This is a case in constrained optimization where using the \mathcal{S} -Procedure for convexification is lossless. Therefore, (5.5) holds if and only if the following LMI problem is

feasible:

$$(T_{3x})^T P_1 (T_{3x}) - P_1 - \tau_1 Q \preceq 0 \quad (5.6a)$$

$$(T_{5x})^T P_1 (T_{5x}) - P_1 + \tau_2 Q \preceq 0 \quad (5.6b)$$

$$\tau_1, \tau_2 \geq 0 \quad (5.6c)$$

However, these LMIs turn out to be infeasible and a node-wise quadratic Lyapunov invariant for the minimal graph model of \mathcal{P}_1 cannot be found. Note that an important implication of the first statement of Theorem 5.1 is that as far as existence of Lyapunov invariants is concerned, assigning many Lyapunov functions to the original graph model of a program is only as good as assigning fewer functions to its minimal realization. Therefore, it follows from Theorem 5.1 and the infeasibility of (5.6), that a Lyapunov invariant cannot be found by assigning four different quadratic functions to the four nodes on the original graph of \mathcal{P}_1 .

The second statement of Theorem 5.1 is also very interesting since it states that performing analysis on the reduced graph models may be advantageous. This is indeed the case for program \mathcal{P}_1 : a Lyapunov invariant is found by assigning two Lyapunov functions to the two nodes (3 and 5) on its maximal realization. On the other hand, since the minimal graph of \mathcal{P}_2 is of order 2, the optimization problem arising from nodewise quadratic Lyapunov analysis of the original graph of \mathcal{P}_2 is likely to be feasible. This is indeed the case and a Lyapunov invariant is found for the original graph. The same is true for analysis of minimal and maximal realization of \mathcal{P}_2 . Nodewise quadratic Lyapunov invariants with fewer nodes can be found for the minimal and maximal realizations of the graph model of \mathcal{P}_2 .

Before we proceed to the next section, for convenience and clarity in presentation of the materials, we introduce the following definition.

Definition 5.2 *Given a graph model G and two offsprings $G_1 \sqsubseteq G$ and $G_2 \sqsubseteq G$, we say that G_2 outperforms G_1 and write $\mathcal{L}(G_1) \sqsubseteq \mathcal{L}(G_2)$ if existence of a nodewise Lyapunov invariant within a specific class of functions for G_2 is a necessary condition for existence of a nodewise Lyapunov invariant within the same class of functions for G_1 .*

Informally speaking, if G_2 outperforms G_1 then Lyapunov analysis of G_2 has a better chance of success than G_1 . It follows from Theorem 5.1, that a reduced offspring $G_1 \sqsubseteq G$

always outperforms the original graph G .

5.3 Comparison of Irreducible Graph Models

In the previous section we established that searching for Lyapunov invariants within a specific class of functions over an irreducible graph model G_i has a better chance of success than any reducible graph $G_r \sqsupseteq G_i$. We also observed that among the two irreducible offsprings of program \mathcal{P}_1 , Lyapunov analysis of the maximal realization resulted in a feasible LMI problem, while Lyapunov analysis of the minimal realization led to an infeasible set of LMIs. Here, several interesting questions arise concerning the comparison of different irreducible offsprings of a graph. For instance:

1. Is it always true that an irreducible offspring of a graph G outperforms all other irreducible graphs of lower order? That is:

$$\forall G_1, G_2 : G_1 \sqsubseteq G, G_2 \sqsubseteq G, \text{ord}(G_2) > \text{ord}(G_1) \stackrel{?}{\implies} \mathcal{L}(G_1) \sqsubseteq \mathcal{L}(G_2).$$

2. It is always true that a maximal realization of a graph outperforms a minimal realization?
3. How do we compute the minimal and maximal orders and the corresponding realizations?

For an arbitrary graph the answer to the first question is negative. A counterexample can be constructed in the following way: Consider the graph model $G(\{1, \dots, 15\} \cup \{\emptyset, \infty\}, \mathcal{E})$, shown in Figure 5-5. This graph can be viewed as the interconnection of two subgraphs $G_L(\{9, \dots, 15\}, \mathcal{E}_L)$ (which includes all the nodes to the left of node 1) and $G_R(\{1, \dots, 8\}, \mathcal{E}_R)$ (which includes all the nodes to the right of node 1). For the moment, let us focus on the subgraph G_R . An irreducible realization of G_R can be obtained by eliminating nodes 4, 5, 3, and 2, which leaves nodes 1, 6, 7, and 8 as the remaining focal nodes. This indeed produces a maximal realization of G_R (which is of order 4). Now, let us focus on G_L , the subgraph to the left of node 1. An irreducible realization of G_L can be obtained by eliminating nodes 15, 14, 13, 11, 12, and 10, which leaves node 9 as the only remaining focal node. This produces a minimal realization of G_L (which has order 1). The overall result is an irreducible graph $G_{\text{irr}} \sqsubseteq G$ with the following

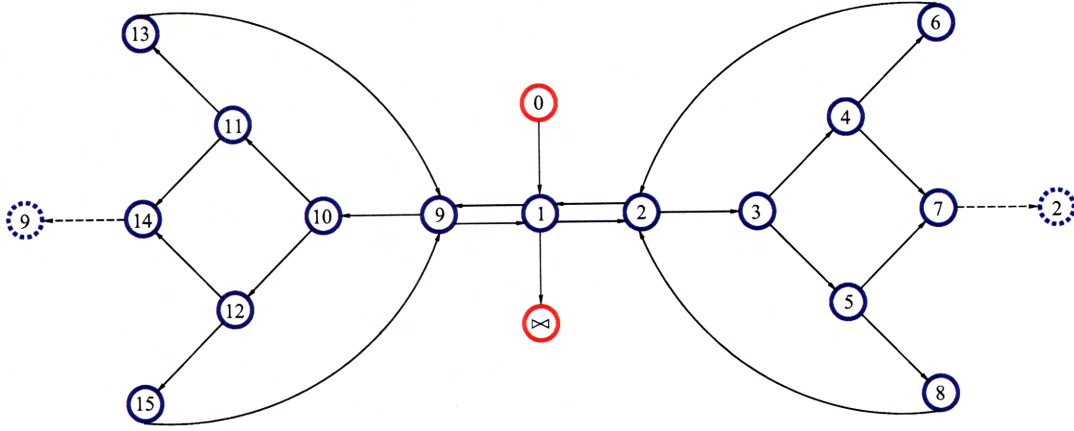


Figure 5-5: With proper labeling of this graph model, a counterexample can be constructed to prove that an irreducible realization of higher order does not always outperform an irreducible realization of lower order.

set of focal nodes: $\mathcal{N} = \{9, 1, 6, 7, 8\} \cup \{\emptyset, \infty\}$. The order of this irreducible model is 5. Now, if the transition labels are such that the dynamics defined over G_L is complicated enough, this realization (which is minimal for G_L) cannot support existence of a nodewise quadratic Lyapunov function. Therefore, the result of nodewise Lyapunov analysis over the G_{irr} would be infeasibility. On the other hand, if the transition labels are such that the dynamics defined over G_R is simple enough, even the minimal realization for G_R may be sufficient for existence of nodewise Lyapunov invariant with fewer many nodes. An alternative irreducible realization of G can be obtained by eliminating nodes 6, 7, 8, 4, 5, 3 from G_R , and nodes 15, 14, 13, 10, 9 from G_L . The resulting irreducible graph $\tilde{G}_{\text{irr}} \subseteq G$ would be of order 4, with $\tilde{\mathcal{N}} = \{1, 2, 11, 12\} \cup \{\emptyset, \infty\}$. As we already discussed, if the dynamics on G_L is complicated and the dynamics on G_R is simple, \tilde{G}_{irr} is a better model for nodewise Lyapunov analysis. In conclusion, we are able to make the following statement: It is not necessarily the case that an irreducible model of higher order outperforms an irreducible model of lower order.

The answer to the second question is trickier. Note that the argument that we made to answer the first question does not apply here as G_{irr} is not a maximal realization of G , nor is \tilde{G}_{irr} a minimal realization of G . For an arbitrary graph G the answer to the second question is

also negative.

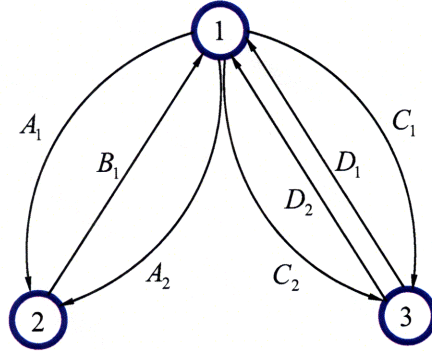


Figure 5-6: For this graph, it is possible to choose the state transition operators A_i , B_i , C_i , D_i such that the minimal realization outperforms the maximal realization.

For an arbitrary graph model, a maximal realization does not always outperform a minimal realization. For a counterexample, consider the graph model in Figure 5-6. The arc labels shown in the figure correspond to transition labels. For simplicity, there are no passport labels, which means that the discrete transitions are non-deterministic. Hence, a state transition along any outgoing arc is possible at any moment of time. The minimal realization of this graph is obtained by eliminating nodes 2 and 3, which leaves node 1 with 6 focal arcs: $\{B_1A_1, B_1A_2, D_1C_1, D_2C_1, D_1C_2, D_2C_2\}$. The maximal realization is obtained by eliminating node 1, which leaves nodes 2 and 3. In the maximal realization, there are two loops from node 2 to itself: $\{A_1B_1, B_1A_2\}$, and four loops from node 3 to itself: $\{C_1D_1, C_2D_1, C_1D_2, C_2D_2\}$. Finally, there are two arcs from node 2 to 3: $\{C_1B_1, C_2B_1\}$, and four arcs from node 3 to 2: $\{A_1D_1, A_2D_1, A_1D_2, A_2D_2\}$. If the linear operators A_i , B_i , C_i , D_i are chosen according to (5.7), then the minimal realization outperforms the maximal realization, in the sense that a single quadratic function $V_1(x) = x^T P_1 x$ which is valid for the minimal graph can be found, however, a pair of symmetric matrices P_i , $i = 2, 3$ such that the quadratic functionals $V_i(x) = x^T P_i x$, $i = 2, 3$, form a nodewise quadratic Lyapunov invariant for the maximal realization cannot be found. The following matrices were found through randomized search. Simpler examples (corresponding to simpler graph structures) may exist.

$$\begin{aligned}
A_1 &= \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0 \end{bmatrix}, \quad A_2 = \begin{bmatrix} 0.5 & 0.5 \\ 0.25 & 0.5 \end{bmatrix}, \quad B_1 = \begin{bmatrix} 0.63 & 0.05 \\ -0.12 & -0.19 \end{bmatrix}, \\
C_1 &= \begin{bmatrix} 0.4 & 0.5 \\ 0 & 0.5 \end{bmatrix}, \quad C_2 = \begin{bmatrix} 0 & 0.25 \\ 0.5 & 0.5 \end{bmatrix}, \quad D_1 = \begin{bmatrix} 1.01 & -1.71 \\ 0.35 & -0.34 \end{bmatrix}, \quad D_2 = \begin{bmatrix} -0.02 & 0.88 \\ 0.68 & 0.85 \end{bmatrix}.
\end{aligned} \tag{5.7}$$

A more detailed discussion concerning the answers to the second and third questions posed at the beginning of Section 5.3 is presented in the following subsections.

5.3.1 Comparison of Maximal and Minimal Realizations of K1 Graphs

Positive statements can be formulated regarding comparison of the performance of maximal and minimal realizations of certain graph models with a specific structure: the so-called Kn graphs. Before we proceed, we present the following proposition, which will be used in the proof of several theorems and lemmas in the remainder of this section.

Proposition 5.1 *The following statements are true for any connected graph $G(\mathcal{N}, \mathcal{E})$:*

1. *Given $i_1 \in \mathcal{N}$, it is possible to obtain an irreducible offspring $G_s(\mathcal{N}_s, \mathcal{E}_s)$ with $i_1 \in \mathcal{N}_s$, if and only if there exists a simple cycle $\mathcal{C} \in G$ that passes through i_1 and does not include any focal nodes.*
2. *Given $i_1, i_2 \in \mathcal{N}$, it is possible to obtain an irreducible offspring $G_s(\mathcal{N}_s, \mathcal{E}_s)$ with $i_1, i_2 \in \mathcal{N}_s$, if and only if there exists simple cycles $\mathcal{C}_1, \mathcal{C}_2 \in G$, such that \mathcal{C}_1 passes through i_1 and not through i_2 , and \mathcal{C}_2 passes through i_2 and not through i_1 and the subgraph² $\mathcal{C}_1 \setminus \{i_1\} \cup \mathcal{C}_2 \setminus \{i_2\}$ does not include any cycles.*
3. *Given $i_1, \dots, i_k \in \mathcal{N}$, it is possible to obtain an irreducible offspring $G_s(\mathcal{N}_s, \mathcal{E}_s)$ with $i_1, \dots, i_k \in \mathcal{N}_s$, if and only if there exists simple cycles $\mathcal{C}_1, \dots, \mathcal{C}_k \in G$, such that \mathcal{C}_j passes through i_j and not through i_q , $q \in \mathbb{Z}(1, k) \setminus \{j\}$ and the subgraph $\bigcup_{q \in \mathbb{Z}(1, k)} \mathcal{C}_q \setminus \{i_q\}$ does not include any cycles.*

What it takes to convert a transient node in a reducible graph to a focal node in an offspring is to eliminate (according to Algorithm 5.1) all the other nodes in at least one of the simple

²Recall that a subgraph of a graph $G(\mathcal{N}, \mathcal{E})$ is a graph $\underline{G}(\underline{\mathcal{N}}, \underline{\mathcal{E}})$ such that $\underline{\mathcal{N}} \subset \mathcal{N}$, and $\underline{\mathcal{E}}$ is the restriction of \mathcal{E} to $\underline{\mathcal{N}}$. Hence, to precisely specify a subgraph of a graph G , it is sufficient to specify the subset of nodes.

cycles that pass through that node. For instance, if there is only one cycle passing through node i , and that cycle passes through node j as well, it is impossible to obtain an irreducible realization that contains both i and j . The problems of computing the minimal and maximal orders as well as the corresponding realizations are hard combinatorial problems in the general case. We study certain graphs with specific structures that allow us to compute the minimal and maximal realizations and compare their performance.

Definition 5.3 *A graph $G(\mathcal{N}, \mathcal{E})$ is called a K1 Graph if it satisfies the following properties:*

I. The graph $G(\mathcal{N}, \mathcal{E})$ is connected, that is, for every pair of nodes $j_1, j_2 \in \mathcal{N} \setminus \{\emptyset, \infty\}$ there exists a path from j_1 to j_2 .

II. There exists a subset of nodes $\mathcal{N}^ \subset \mathcal{N}$ satisfying the following properties:*

IIa. Every simple cycle $\mathcal{C} \in G$ passes through every node $i^ \in \mathcal{N}^*$.*

IIb. For every $i^ \in \mathcal{N}^*$, and for every $j \in \mathcal{I}(i^*)$, we have $|\mathcal{O}(j)| = 1$. This implies in particular, that there is only one path from j to i^* .*

The graph model of program \mathcal{P}_1 (Figure 5-1) is a K1 graph. There are two simple cycles on the graph of $\mathcal{P}_1 : \{1 \rightarrow 2 \rightarrow 3 \rightarrow 1\}$ and $\{1 \rightarrow 2 \rightarrow 5 \rightarrow 1\}$. The set $\mathcal{N}^* := \{1, 2\}$ satisfies both properties IIa and IIb of Definition 5.3. The graph model of program \mathcal{P}_2 is not a K1 graph. There are three simple cycles on the graph of $\mathcal{P}_2 : \{1 \rightarrow 2 \rightarrow 4 \rightarrow 1\}$, $\{2 \rightarrow 3 \rightarrow 2\}$, and $\{4 \rightarrow 5 \rightarrow 4\}$, which do not share a common node. Note that it follows from Definition 5.3 that if $G(\mathcal{N}, \mathcal{E})$ is a K1 graph then every node $i \in \mathcal{N} \setminus \{\emptyset, \infty\}$ is a transient node, unless $|\mathcal{N} \setminus \{\emptyset, \infty\}| = 1$, in which case $\mathcal{N}^* = \{i^*\} = \mathcal{N} \setminus \{\emptyset, \infty\}$ and i^* is a focal node.

Theorem 5.2 *Let $G(\mathcal{N}, \mathcal{E})$ be a K1 graph. Then the minimal order of G is 1, and the maximal order of G is $d_{\max} \geq k$, where*

$$k = \max_{i^* \in \mathcal{N}^*} |\mathcal{I}(i^*)| \quad (5.8)$$

Furthermore, if $G(\mathcal{N}, \mathcal{E})$ is a linear graph model, then there exists an irreducible realization of order k that outperforms the minimal realization(s).

To make the presentation clearer, we first present Lemma 5.1 and Lemma 5.2, in which we prove the first and the second statement of Theorem 5.2 respectively.

Lemma 5.1 *The minimal order of a K1 graph is 1.*

Proof. Let $G(\mathcal{N}, \mathcal{E})$ be a K1 graph. If $|\mathcal{N} \setminus \{\emptyset, \infty\}| = 1$ the result is trivial. We prove the result for the case where $|\mathcal{N} \setminus \{\emptyset, \infty\}| > 1$. Since the graph is connected, it contains at least one cycle. Hence, the order of the minimal realization is at least 1 and it is sufficient to show that an irreducible realization with order 1 can be achieved. Let $i^* \in \mathcal{N}^*$ and assume for the moment that $\mathcal{O}(i^*) \setminus \mathcal{I}(i^*)$ is non-empty, and let $j \in \mathcal{O}(i^*) \setminus \mathcal{I}(i^*)$. Since the graph is K1, j is a transient node, otherwise, the simple cycle $j \rightarrow j$ does not pass through i^* which is a contradiction. Next, perform the following reduction step: remove node j and let $G_1(\mathcal{N}_1, \mathcal{E}_1)$ be the corresponding reduced graph. We claim that $G_1(\mathcal{N}_1, \mathcal{E}_1)$ is also a K1 graph. Connectivity is trivial. We prove that $\mathcal{N}^* \setminus \{j\}$ still satisfies properties IIa and IIb of Definition 5.3. Suppose that $i^* \in \mathcal{N}^* \setminus \{j\}$ and $z \in \mathcal{I}(i^*)$. Since $(z, j) \notin \mathcal{E}$ (which follows from IIb and $j \neq i^*$), we have³ $|\mathcal{O}(z)|_{G_1} = |\mathcal{O}(z)|_G = 1$. Hence, IIb holds for G_1 . To establish that IIa holds, it is sufficient to prove that removing j cannot add a cycle that does not go through i^* . Suppose on the contrary that G_1 includes a cycle $\mathcal{C}_1 := i_1 i_2 \dots i_m i_1$ that does not pass through i^* . Since such cycle cannot exist on G , at least one of the arcs on this cycle must have been added in the reduction step. Without loss of generality, assume that this arc is (i_1, i_2) . If $(i_1, i_2) \notin \mathcal{E}$ and $(i_1, i_2) \in \mathcal{E}_1$, then we must have $(i_1, j) \in \mathcal{E}$ and $(j, i_2) \in \mathcal{E}$. But this implies that $\mathcal{C}_1 := i_1 j i_2 \dots i_m i_1$ is a simple cycle in G that does not pass through i^* which contradicts IIa. Therefore, $G_1(\mathcal{N}_1, \mathcal{E}_1)$ is also a K1 graph. By induction, this process can be repeated until $\mathcal{O}(i^*) \setminus \mathcal{I}(i^*) = \emptyset$, in which case the only remaining nodes are i^* and $\{j_1, \dots, j_k\} \in \mathcal{O}(i^*) \cap \mathcal{I}(i^*)$. For this graph, each j_i can be removed without converting any of the other ones into a focal node, which leaves i^* as the only remaining node. Proof is complete. ■

Lemma 5.2 *The maximal order of a K1 graph is at least k , where:*

$$k = \max_{i^* \in \mathcal{N}^*} |\mathcal{I}(i^*)| \tag{5.9}$$

Proof. It is sufficient to show that an irreducible realization of order k exists. Let k be defined as in (5.9) and let i^* be such that $|\mathcal{I}(i^*)| = k$. Let $\{j_1, \dots, j_k\}$ be the set of nodes in

³See item 3 of the remarks after Algorithm 5.1 in Section 5.2.

$\mathcal{I}(i^*)$. Since the graph is connected, for every $j \in \mathcal{I}(i^*)$ there exists a simple cycle that passes through j . We first prove that there does not exist a simple cycle $\mathcal{C} \in G$ that includes a pair of nodes j_{k_1} and j_{k_2} in $\mathcal{I}(i^*)$. Assume for the sake of contradiction that there exists a simple cycle $\mathcal{C} \in G$ that starts at $j_{k_1} \in \mathcal{I}(i^*)$ and passes through $j_{k_2} \in \mathcal{I}(i^*)$. Since the graph is K1, property IIb implies that there is only one arc leaving j_{k_1} and that goes to i^* . Hence, cycle \mathcal{C} must include a path from i^* to j_{k_2} . For the cycle \mathcal{C} to close at j_{k_1} , there must be a path from j_{k_2} to j_{k_1} . But property IIb implies that there is only one arc leaving j_{k_2} and that goes to i^* . Therefore, \mathcal{C} visits i^* at least twice which contradicts the fact that \mathcal{C} is a simple cycle. Hence, for every $j \in \{j_1, \dots, j_k\}$ there exists a simple cycle \mathcal{C}_j that passes through j and not through $\mathcal{I}(i^*) \setminus \{j\}$. Furthermore, the subgraph $\bigcup_{i \in \mathbb{Z}(1,k)} \mathcal{C}_i \setminus \{j_i\}$ does not include any cycles. To see this, note that every simple cycle passes through i^* . Since the subgraph $\bigcup_{i \in \mathbb{Z}(1,k)} \mathcal{C}_i \setminus \{j_i\}$ excludes all the nodes in $\mathcal{I}(i^*)$ it cannot include any cycles. It then follows from Proposition 5.1 that an irreducible realization with $\{j_1, \dots, j_k\}$ as the set of focal nodes is achievable. This proves that the maximal order is at least k . ■

In Lemma 5.2 we proved that the maximal order of a K1 graph is at least k , where k is defined in (5.9). In the following algorithm, we present a procedure to obtain a realization with order k .

Algorithm 5.2 *To construct an irreducible realization of order k (k given in (5.9)) for a K1 graph, let $i^* \in \mathcal{N}^*$ be such that $|\mathcal{I}(i^*)| = k$ and let $\{j_1, \dots, j_k\}$ be the set of nodes in $\mathcal{I}(i^*)$. For $j_i \in \mathcal{I}(i^*)$ let $\{n_1, \dots, n_{v(j_i)}\}$ be the set of nodes in $\mathcal{I}(j_i)$. Then, perform the following set of reduction steps:*

0. Let $i = 1, l = 1, t = 1, G_t = G$.
1. Consider node $n_l \in \mathcal{I}(j_i)$. It follows from property IIb that $n_l \notin \{j_1, \dots, j_k\}$.
2. If $n_l = i^*$ and $l < v(j_i)$ then $l \rightarrow l + 1$ and go to 1.
3. If $n_l = i^*$ and $l = v(j_i)$ and $i < k$ then $i \rightarrow i + 1$ and $l \rightarrow 1$ and go to 1.
4. If $n_l = i^*$ and $l = v(j_i)$ and $i = k$ then goto 8.
5. Remove n_l from $G_t(\mathcal{N}_t, \mathcal{E}_t)$ and let $G_{t+1}(\mathcal{N}_{t+1}, \mathcal{E}_{t+1})$ be the corresponding reduced graph. Since the graph $G_t(\mathcal{N}_t, \mathcal{E}_t)$ is K1, every $n_l \in \mathcal{I}(j_i)$ is a transient node before this reduction step and can be removed. It can be shown using an argument similar to that in the proof of Lemma

- 5.1 that $G_{t+1}(\mathcal{N}_{t+1}, \mathcal{E}_{t+1})$ is also a K1 graph.
6. If $l < v(j_i)$ then $l \rightarrow l + 1$, and $t \rightarrow t + 1$, and go to 1.
7. If $l = v(j_i)$ and $i < k$ then $i \rightarrow i + 1$, and $t \rightarrow t + 1$, and $l \rightarrow 1$ and go to 1.
8. Remove node i^* .
- Step 8 will immediately convert every node in $\{j_1, \dots, j_k\}$ into a focal node.

We are now in a position to complete the proof of Theorem 5.2. To simplify the notation and avoid distracting details, we present the proof for the case where $\theta = 1$ across all the arcs and there are no passport labels, that is, the discrete transitions are non-deterministic. The proof at the presence of passport labels (or invariant sets X_i , $i \in \mathcal{N}$), and/or when the rate θ is variable across the arcs is similar.

Proof of Theorem 5.2. The first statement of the theorem was proven in Lemma 5.1. The existence of an irreducible realization of order k follows from Lemma 5.2. We prove that the irreducible realization which consists of $\mathcal{I}(i^*)$ (constructed according to Algorithm 5.2) always outperforms all the minimal realizations. Assume without loss of generality that $G_{\min}(\{1\}, \mathcal{E}_{\min})$ is the minimal realization in consideration and that $|\mathcal{I}(1)| = k$. Consider the set $\mathcal{I}(1)$ of the incoming nodes of node 1 on the original graph $G(\mathcal{N}, \mathcal{E})$. Let us denote the elements of this set by $\{j_1, \dots, j_k\}$. For $j \in \mathcal{I}(1)$ let $\pi(j)$ denote the set of all simple paths from node 1 to node j on $G(\mathcal{N}, \mathcal{E})$. Furthermore, for $j \in \mathcal{I}(1)$, let $\mathcal{T}(j)$ denote the set of transition labels obtained by taking the composition of the transition labels along all the simple paths from 1 to j , that is:

$$\mathcal{T}(j) := \{T_{j1} \mid T_{j1} = T_{j i_r} \dots T_{i_2 i_1} T_{i_1 1}, (j, i_r \dots i_2, i_1, 1) \in \pi(j)\}$$

Let $V(1, x) = \sigma_1(x)$ be a Lyapunov invariant for $G_{\min}(\{1\}, \mathcal{E}_{\min})$. Then we have:

$$\sigma_1(T_{1j}T_{j1}x) - \sigma_1(x) < 0, \forall j \in \{j_1, \dots, j_k\}, T_{j1} \in \mathcal{T}(j). \quad (5.10)$$

Recall (from Algorithm 5.2) that by construction, the set $\mathcal{I}(1) := \{j_1, \dots, j_k\}$ is exactly the set of focal nodes in the irreducible realization of order $k : \tilde{G}(\{j_1, \dots, j_k\}, \tilde{\mathcal{E}})$. Define:

$$\tilde{\sigma}_j(x) = \sigma_1(T_{1j}x), \quad \forall j \in \{j_1, \dots, j_k\}. \quad (5.11)$$

Since the graph is linear, the functions $\tilde{\sigma}_j(\cdot)$ belong to the same class of functions as $\sigma_i(\cdot)$. Our claim is that the functions $\tilde{\sigma}_j(\cdot)$ define a valid Lyapunov invariant for \tilde{G} . What needs to be proven is that:

$$\tilde{\sigma}_j(T_{ji}^r x) - \tilde{\sigma}_i(x) < 0, \quad \forall (i, j, r) \in \tilde{\mathcal{E}}, \text{ where } i, j \in \{j_1, \dots, j_k\}. \quad (5.12)$$

Note that by construction we have:

$$\forall (i, j, r) \in \tilde{\mathcal{E}}, \exists T \in \mathcal{T}(j), \text{ s.t. } T_{ji}^r = TT_{1i} = T_{j1}T_{1i}.$$

Therefore, (5.12) holds if and only if

$$\tilde{\sigma}_j(T_{j1}T_{1i}x) - \tilde{\sigma}_i(x) < 0, \quad \forall (i, j, \cdot) \in \tilde{\mathcal{E}}, T_{j1} \in \mathcal{T}(j).$$

Equivalently:

$$\sigma_1(T_{1j}T_{j1}T_{1i}x) - \sigma_1(T_{1i}x) < 0, \quad \forall (i, j, \cdot) \in \tilde{\mathcal{E}}, T_{j1} \in \mathcal{T}(j).$$

which follows immediately from 5.10 with x replaced by $T_{1i}x$. Proof is complete. ■

5.3.2 Comparison of Maximal and Minimal Realizations of Kn Graphs

In this section, we generalize the results of the previous section from K1 graphs to the so-called Kn graphs.

Definition 5.4 *A graph $G(\mathcal{N}, \mathcal{E})$ is called a Kn Graph if it satisfies the following properties:*

- I. The graph $G(\mathcal{N}, \mathcal{E})$ is connected, that is, for every pair of nodes $j_1, j_2 \in \mathcal{N} \setminus \{\emptyset, \infty\}$, there exists a path from j_1 to j_2 .*
- II. There exists a subset $\mathcal{N}^* \subset \mathcal{N}$, with cardinality $|\mathcal{N}^*| = n$, satisfying the following properties:*
 - IIa. Every simple cycle $\mathcal{C} \in G$ passes through at least one node $i^* \in \mathcal{N}^*$.*

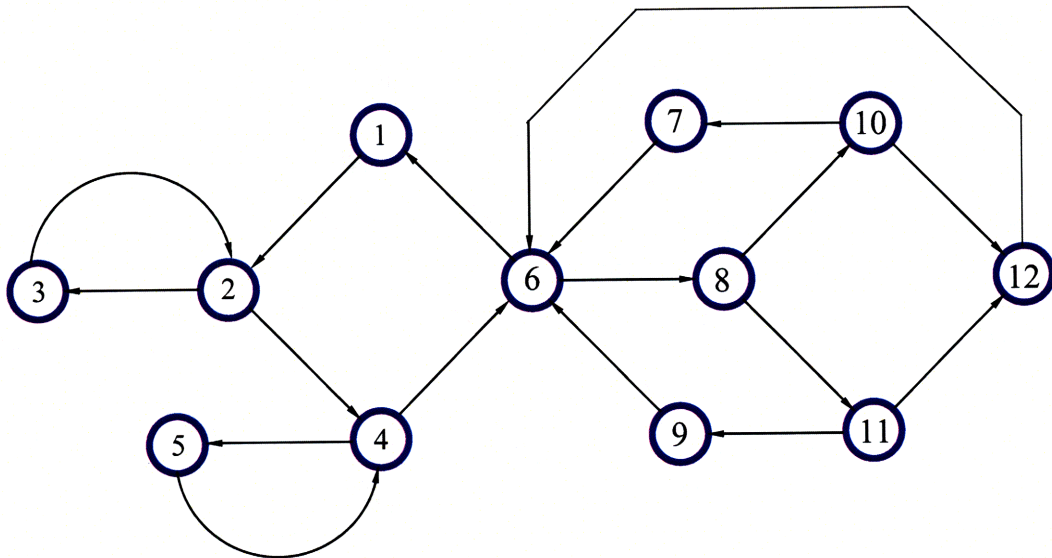


Figure 5-7: A K3 graph with $\mathcal{N}^* = \{2, 4, 6\}$. The minimal order is 3, and the maximal order is 6.

Iib. For every $i^ \in \mathcal{N}^*$, and for every $j \in \mathcal{I}(i^*) \setminus \mathcal{N}^*$, we have $|\mathcal{O}(j)| = 1$. This implies in particular, that there is only one path from j to i^* .*

Iic. There does not exist a simple cycle that passes through a pair of nodes $j_1, j_2 \in \mathcal{I}(\mathcal{N}^) \setminus \mathcal{N}^*$.*

III. The subgraph $G(\mathcal{N}^, \mathcal{E}^*)$ does not include any cycles.*

Remark 5.2 *Note that the set \mathcal{N}^* may not be unique, its cardinality, however, is fixed. In other words, a smaller set with cardinality strictly less than n , satisfying the same properties does not exist.*

For instance, the graph model of \mathcal{P}_2 (Figure 5-1) is a K2 graph with $\mathcal{N}^* = \{2, 4\}$. All simple cycles pass through either node 2 or node 4, and a smaller set satisfying this property cannot be found. It is easy to verify that properties Iib, Iic, and III also holds. Another example is shown in Figure 5-7. The graph in Figure 5-7 is a K3 graph.

Theorem 5.3 *Let $G(\mathcal{N}, \mathcal{E})$ be a K_n graph. Then the minimal order of G is n , and the maximal*

order of G is $d_{\max} \geq k$, where:

$$k = \max_{\mathcal{N}^*} \sum_{i^* \in \mathcal{N}^*} |\mathcal{I}(i^*) \setminus \mathcal{N}^*|,$$

Furthermore, if $G(\mathcal{N}, \mathcal{E})$ is a linear graph model, then there exists an irreducible realization of order k that always outperforms the minimal realization(s).

In a similar fashion to the proof of Theorem 5.2, we prove Theorem 5.3 in multiple steps to make the presentation clearer.

Lemma 5.3 *The minimal order of a Kn graph is n .*

Proof. Let $G(\mathcal{N}, \mathcal{E})$ be a Kn graph. Since a smaller set satisfying property IIa cannot exist, the order of the minimal realization is at least n . Hence, it is sufficient to show that an irreducible realization with order n can be achieved. Strictly speaking, only properties I, IIa, and IIb of Definition 5.4 are needed for the proof presented here. Let $\mathcal{N}^* \subset \mathcal{N}$ be a set of nodes satisfying these properties. Let $i^* \in \mathcal{N}^*$ and assume for the moment that $\{\mathcal{O}(i^*) \setminus \mathcal{I}(i^*)\} \setminus \mathcal{N}^*$ is non-empty, and let $j \in \{\mathcal{O}(i^*) \setminus \mathcal{I}(i^*)\} \setminus \mathcal{N}^*$. Note that j is a transient node, otherwise, the simple cycle $j \rightarrow j$ does not pass through any node in \mathcal{N}^* which is a contradiction. Next, perform the following reduction step: remove node j and let $G_1(\mathcal{N}_1, \mathcal{E}_1)$ be the corresponding reduced graph. It is true that $G_1(\mathcal{N}_1, \mathcal{E}_1)$ is also a Kn graph, however, we focus only on the properties that we use in the proof. Connectivity is trivial. We prove that $\mathcal{N}^* \subset \mathcal{N}_1$ still satisfies properties IIa and IIb w.r.t. G_1 . If $z \in \mathcal{I}(i^*)$, then $(z, j) \notin \mathcal{E}$ (which follows from IIb and $j \neq i^*$), and there holds $|\mathcal{O}(z)|_{G_1} = |\mathcal{O}(z)|_G = 1$. Hence, IIb holds for G_1 . To show that IIa holds, we prove that removing j cannot add a cycle that does not go through \mathcal{N}^* . Suppose on the contrary that G_1 includes a cycle $\mathcal{C}_1 := i_1 i_2 \dots i_m i_1$ that does not pass through \mathcal{N}^* . Since such cycle cannot exist on G , at least one of the arcs on this cycle must have been added in the reduction step. Without loss of generality assume that this arc is (i_1, i_2) . If $(i_1, i_2) \notin \mathcal{E}$ and $(i_1, i_2) \in \mathcal{E}_1$, then we must have $(i_1, j) \in \mathcal{E}$ and $(j, i_2) \in \mathcal{E}$. But this implies that $\mathcal{C} := i_1 j i_2 \dots i_m i_1$ is a simple cycle on G that does not pass through \mathcal{N}^* which contradicts IIa. Therefore, $G_1(\mathcal{N}_1, \mathcal{E}_1)$ also satisfies properties I, IIa, IIb. By induction, this process can

be repeated until:

$$\{\mathcal{O}(i^*) \setminus \mathcal{I}(i^*)\} \setminus \mathcal{N}^* = \emptyset, \forall i^* \in \mathcal{N}^*,$$

in which case the only remaining nodes are the ones in \mathcal{N}^* and sets of nodes $\{j_1, \dots, j_{k(i^*)}\} \in \mathcal{O}(i^*) \cap \mathcal{I}(i^*)$, $\forall i^* \in \mathcal{N}^*$. Since the elements of $\mathcal{O}(i^*) \cap \mathcal{I}(i^*)$ do not share a common simple cycle (which follows from IIb), removing each $j_i \in \mathcal{O}(i^*) \cap \mathcal{I}(i^*)$ converts i^* , and only i^* , into a focal node. Since this is true for every $i^* \in \mathcal{N}^*$, a minimal realization of order $n = |\mathcal{N}^*|$ is achieved. ■

Lemma 5.4 *The maximal order of a Kn graph is at least k, where*

$$k = \max_{\mathcal{N}^*} \sum_{i^* \in \mathcal{N}^*} |\mathcal{I}(i^*) \setminus \mathcal{N}^*| \quad (5.13)$$

Proof. We show that an irreducible realization of order k exists. Let k be defined as in (5.13) and let \mathcal{N}^* be such that

$$\sum_{i^* \in \mathcal{N}^*} |\mathcal{I}(i^*) \setminus \mathcal{N}^*| = k.$$

Recall that by definition:

$$\mathcal{I}(\mathcal{N}^*) = \bigcup_{i^* \in \mathcal{N}^*} \mathcal{I}(i^*).$$

It follows from property IIb that $\mathcal{I}(i_1^*) \cap \mathcal{I}(i_2^*) \subseteq \mathcal{N}^*$, $\forall i_1^*, i_2^* \in \mathcal{N}^*$. Therefore, the sets $\mathcal{I}(i^*) \setminus \mathcal{N}^*$, $i^* \in \mathcal{N}^*$ are disjoint and $|\mathcal{I}(\mathcal{N}^*) \setminus \mathcal{N}^*| = k$. Let $\{j_1, \dots, j_k\}$ be the set of nodes in $\mathcal{I}(\mathcal{N}^*) \setminus \mathcal{N}^*$. Since the graph is connected, for every $j \in \mathcal{I}(\mathcal{N}^*) \setminus \mathcal{N}^*$ there exists a simple cycle that passes through j . Property IIc then implies that there does not exist a simple cycle $\mathcal{C} \in G$ that includes a pair of nodes j_{k_1} and j_{k_2} in $\mathcal{I}(\mathcal{N}^*) \setminus \mathcal{N}^*$. Therefore, for every $j \in \mathcal{I}(\mathcal{N}^*) \setminus \mathcal{N}^*$, there exists a simple cycle \mathcal{C}_j that passes through j , and does not pass through any other node in $\{\mathcal{I}(\mathcal{N}^*) \setminus \mathcal{N}^*\} \setminus \{j\}$. Furthermore, the subgraph

$$G_{\mathcal{C}} := \bigcup_{i \in \mathbb{Z}(1, k)} \mathcal{C}_i \setminus \{j_i\}$$

does not include any cycles. To see this, note that every simple cycle passes through \mathcal{N}^* . Since the subgraph $G_{\mathcal{C}}$ excludes all the nodes in $\mathcal{I}(\mathcal{N}^*) \setminus \mathcal{N}^*$, the only cycles it can contain are the

ones that are a subgraph of \mathcal{N}^* . However, property III implies that such cycle cannot exist. It then follows from Proposition 5.1 that an irreducible realization with $\{j_1, \dots, j_k\}$ as the set of focal nodes is achievable. This proves that the maximal order is at least k . ■

We are now in a position to complete the proof of Theorem 5.3. In this case too, to simplify the notation and avoid distracting details, we present the proof for the case where $\theta = 1$ across all the arcs and there are no passport labels. The proof at the presence of passport labels and/or when the rate θ is variable across the arcs is similar.

Proof of Theorem 5.3. The first statement of the theorem was proven in Lemma 5.3. The existence of an irreducible realization of order k follows from Lemma 5.4. We prove that the irreducible realization consisting of $\mathcal{I}(\mathcal{N}^*) \setminus \mathcal{N}^*$ always outperforms the minimal realizations. The proof is similar to the proof of Theorem 5.2. Consider the set $\mathcal{I}(\mathcal{N}^*) \setminus \mathcal{N}^*$ of the incoming nodes of the set \mathcal{N}^* on the original graph $G(\mathcal{N}, \mathcal{E})$. For $j \in \mathcal{I}(\mathcal{N}^*) \setminus \mathcal{N}^*$ and $i^* \in \mathcal{N}^*$ let $\pi_{i^*}(j)$ denote the set of all simple paths from i^* to j on $G(\mathcal{N}, \mathcal{E})$. Furthermore, for $j \in \mathcal{I}(\mathcal{N}^*) \setminus \mathcal{N}^*$, let $\mathcal{T}_{i^*}(j)$ denote the set of transition labels obtained by taking the composition of the transition labels along all the simple paths from i^* to j , that is:

$$\mathcal{T}_{i^*}(j) := \{T_{ji^*} \mid T_{ji^*} = T_{ji_r} \dots T_{i_2 i_1} T_{i_1 i^*}, (j, i_r \dots i_2, i_1, i^*) \in \pi_{i^*}(j)\}$$

Assume that the functions $\sigma_{i^*}(x)$, $i^* \in \mathcal{N}^*$ define a Lyapunov invariant for $G_{\min}(\mathcal{N}^*, \mathcal{E}_{\min})$. Then we have:

$$\sigma_{i_2^*}(T_{i_2^* j} T_{j i_1^*} x) - \sigma_{i_1^*}(x) < 0, \forall i_1^*, i_2^* \in \mathcal{N}^*, j \in \mathcal{I}(i_2^*), T_{j i_1^*} \in \mathcal{T}_{i_1^*}(j). \quad (5.14)$$

Recall that by construction, the set $\mathcal{I}(\mathcal{N}^*) \setminus \mathcal{N}^* := \{j_1, \dots, j_k\}$ is exactly the set of focal nodes in the irreducible realization of order k : $\tilde{G}(\{j_1, \dots, j_k\}, \tilde{\mathcal{E}})$. Define:

$$\tilde{\sigma}_j(x) = \sigma_{i^*}(T_{i^* j} x), \forall i^* \in \mathcal{N}^*, j \in \mathcal{I}(i^*) \setminus \mathcal{N}^*. \quad (5.15)$$

(Since for every $i^* \in \mathcal{N}^*$ there is only one path from $j \in \mathcal{I}(i^*)$ to i^* , $\tilde{\sigma}_j(\cdot)$ is well-defined). Since the graph is linear, the functions $\tilde{\sigma}_j(\cdot)$, $j \in \mathcal{I}(\mathcal{N}^*) \setminus \mathcal{N}^*$ belong to the same class of functions

as $\sigma_{i^*}(\cdot)$, $i^* \in \mathcal{N}^*$. Our claim is that the functions $\tilde{\sigma}_j(\cdot)$ define a valid Lyapunov invariant for $\tilde{\mathcal{G}}$. What needs to be proven is that:

$$\tilde{\sigma}_j(T_{ji}^r x) - \tilde{\sigma}_i(x) < 0, \forall (i, j, r) \in \tilde{\mathcal{E}}, \text{ where } i, j \in \{j_1, \dots, j_k\}. \quad (5.16)$$

Note that by construction we have:

$$\forall (i, j, r) \in \tilde{\mathcal{E}}, \exists i_1^* \in \mathcal{N}^*, T \in \mathcal{T}_{i_1^*}(j), \text{ s.t. } T_{ji}^r = TT_{i_1^*i} = T_{ji_1^*}T_{i_1^*i}, i \in \mathcal{I}(i_1^*).$$

Therefore, (5.16) holds if and only if:

$$\tilde{\sigma}_j(T_{ji_1^*}T_{i_1^*i}x) - \tilde{\sigma}_i(x) < 0, \forall (i, j, \cdot) \in \tilde{\mathcal{E}}, T_{ji_1^*} \in \mathcal{T}_{i_1^*}(j). \quad (5.17)$$

Moreover, it follows from the definition of $\tilde{\sigma}(\cdot)$ that there exist i_2^* such that (5.17) is equivalent to:

$$\sigma_{i_2^*}(T_{i_2^*j}T_{ji_1^*}T_{i_1^*i}x) - \sigma_{i_1^*}(T_{i_1^*i}x) < 0, \forall (i, j, \cdot) \in \tilde{\mathcal{E}}, T_{ji_1^*} \in \mathcal{T}_{i_1^*}(j).$$

which follows immediately from 5.14 with x replaced by $T_{i_1^*i}x$. Proof is complete. ■

The Effects of Convex Relaxations

So far in this chapter, we have established that from the theoretical viewpoint of existence of node-wise Lyapunov invariants within a specific class of functions, it is advantageous to search for these functions over the reduced graph models rather than the original graph models of computer programs. The result of Theorem 5.1 can be interpreted in terms of comparing two generally non-convex optimization problems. The theorem states that if the non-convex optimization problem associated with the original graph model is feasible, then so is the non-convex optimization problem associated with any reduced graph model. An interesting question that arises here is about the computational procedure that will be used to compute the Lyapunov invariants for the two graph models. More specifically, the effects of convex relaxations on the computation of these functions for the original and the reduced models must be investigated. It is interesting that the statement of Theorem 5.1 can be extended to the case where convex relaxation techniques are exploited for computation of the Lyapunov invariants.

For instance, if application of the \mathcal{S} -Procedure renders the computation of $V(i, x)$, $i \in \mathcal{N}$, (for the original model) a feasible semidefinite optimization problem, then application of the \mathcal{S} -Procedure renders the computation of $V_r(i, x)$, $i \in \mathcal{N}_r$ (for the reduced model) a feasible semidefinite optimization problem. To make this concept clearer, let us consider a specific example. Consider a graph G , and assume that G_r is obtained from G by eliminating node 2, where, $\mathcal{I}(2) := \{1\}$, $\mathcal{O}\{2\} := \{3, 5\}$ (e.g. as in the graph of program \mathcal{P}_1 , Figure 5-1). Assume that each transition label T_{ji} represents a finite-dimensional linear operator, and that each passport label is defined by a single quadratic constraint: $\Pi_{ji} = \{x \mid x^T Q_{ji} x \leq 0\}$. The Lyapunov conditions for state transitions on the original graph are:

$$\sigma_2(T_{21}x) - \theta\sigma_1(x) < 0, \text{ s.t. } x^T Q_{21}x \leq 0, \quad (5.18a)$$

$$\sigma_3(T_{32}x) - \theta\sigma_2(x) < 0, \text{ s.t. } x^T Q_{32}x \leq 0, \quad (5.18b)$$

$$\sigma_5(T_{52}x) - \theta\sigma_2(x) < 0, \text{ s.t. } x^T Q_{52}x \leq 0. \quad (5.18c)$$

Eliminating node 2, we write the Lyapunov conditions for the reduced model G_r in the following way:

$$\sigma_3(T_{32}T_{21}x) - \theta\sigma_1(x) < 0, \text{ s.t. } x^T Q_{21}x \leq 0, x^T T_{21}^T Q_{32} T_{21}x \leq 0, \quad (5.19a)$$

$$\sigma_5(T_{52}T_{21}x) - \theta\sigma_1(x) < 0, \text{ s.t. } x^T Q_{21}x \leq 0, x^T T_{21}^T Q_{52} T_{21}x \leq 0. \quad (5.19b)$$

Now, let each $\sigma_i(\cdot) \in \mathcal{V}_x^2$ be a quadratic functional: $\sigma_i(x) := x^T P_i x$. Using the \mathcal{S} -Procedure, (5.18) can be converted to Linear Matrix Inequalities in the following way:

$$T_{21}^T P_2 T_{21} - \theta P_1 - \tau_{21} Q_{21} < 0, \tau_{21} > 0, \quad (5.20a)$$

$$T_{32}^T P_3 T_{32} - \theta P_2 - \tau_{32} Q_{32} < 0, \tau_{32} > 0, \quad (5.20b)$$

$$T_{52}^T P_5 T_{52} - \theta P_2 - \tau_{52} Q_{52} < 0, \tau_{52} > 0, \quad (5.20c)$$

while (5.19) becomes:

$$T_{21}^T T_{32}^T P_3 T_{32} T_{21} - \theta P_1 - \tau_{21} Q_{21} - \tau_{32} T_{21}^T Q_{32} T_{21} < 0, \quad \tau_{21} > 0, \quad \tau_{32} > 0, \quad (5.21a)$$

$$T_{21}^T T_{52}^T P_5 T_{52} T_{21} - \theta P_1 - \tau_{21} Q_{21} - \tau_{52} T_{21}^T Q_{52} T_{21} < 0, \quad \tau_{21} > 0, \quad \tau_{52} > 0. \quad (5.21b)$$

It is now easy to verify that if $P_1, P_2, P_3, P_5, \tau_{21}, \tau_{32}, \tau_{52}$ are a feasible solution to the set of LMIs in (5.20), then $P_1, \theta P_3, \theta P_5, \tau_{21}, \theta \tau_{32}, \theta \tau_{52}$ are a feasible solution to the set of LMIs in (5.21). To obtain (5.21a) from (5.20), multiply (5.20b) on both sides by $T_{21}^T/\sqrt{\theta}$ and $T_{21}\sqrt{\theta}$, and add it to (5.20a). Inequality (5.21b) can be obtained similarly.

So far we have shown for a special case, that numerical computation of the Lyapunov invariants for the reduced graph is not more complicated than the original graph. As we will show in the sequel, the result remains true in general as long as the convex relaxation technique that is applied is the standard \mathcal{S} -Procedure (cf. Section 4.1.2). The same cannot be said if the sum-of-squares relaxation technique in its most general form (cf. Section 4.1.2) is used for computation of $V(i, x)$, $i \in \mathcal{N}$ over the original graph. However, under some reasonable assumptions, we are able to extend the results to mildly restricted versions of the SOS relaxation technique. The only restriction that we impose in the SOS relaxation is that those expressions that consist of the multiplication of $\sigma_j(x_+) - \theta \sigma_i(x)$ and other inequality constraints do not appear in the P-Satz polynomial. We summarize the above discussion in Theorem 5.4. To make the presentation clearer, we first introduce a new definition.

Definition 5.5 *Let S be a semialgebraic set:*

$$S := \left\{ \begin{array}{l} x \in \mathbb{R}^n \mid \quad \quad \quad f_j(x) \geq 0, \quad j \in J, \\ \quad \quad \quad \mid \quad g_k(x) \neq 0, \quad k \in K, \quad h_l(x) = 0, \quad l \in L. \end{array} \right\}$$

The P-Satz polynomial $P_S := f + h + g^2$, $f \in P(f_j)$, $g \in M(g_k)$, $h \in I(h_l)$ is said to be strongly linear w.r.t. f_{j^} , $j^* \in J$, if the coefficient of f_{j^*} in P_S is 1. We say that the certificate for emptiness of S can be generated by solving a sum-of-squares optimization problem that is strongly linear w.r.t. the collection of functions $\{f_i(x) \mid i \in \underline{J} \subset J\}$ if $f \in P(f_j)$, $g \in M(g_k)$, $h \in I(h_l)$ can be found by solving a SOS optimization problem such that $P_S \triangleq f + h + g^2 = 0$*

and P_S is strongly linear w.r.t. each f_i , $i \in \underline{J}$.

Theorem 5.4 Consider a graph model $G(\mathcal{N}, \mathcal{E})$ and let $G_r(\mathcal{N}_r, \mathcal{E}_r)$ be a reduced offspring of G . Consider the following two statements:

I. The function

$$V(\tilde{x}) \equiv V(i, x) := \sigma_i(x), \quad \sigma_i(x) \in \mathcal{V}_x^d, \quad i \in \mathcal{N}$$

satisfying $V(\tilde{x}_+) - \theta V(\tilde{x}) < 0$ w.r.t. the original graph model G can be found by solving a sum-of-squares optimization problem that is strongly linear w.r.t. the collection of functions $\{\sigma_j(x_+) - \sigma_i(x) \mid (i, j, \cdot) \in \mathcal{E}\}$.

II. The function

$$V_r(\tilde{x}) \equiv V_r(i, x) := \sigma_i(x), \quad \sigma_i(x) \in \mathcal{V}_x^d, \quad i \in \mathcal{N}_r$$

satisfying $V_r(\tilde{x}_+) - \theta V_r(\tilde{x}) < 0$ w.r.t. the reduced graph model G_r can be found by solving a sum-of-squares optimization problem that is strongly linear w.r.t. the collection of functions $\{\sigma_j(x_+) - \sigma_i(x) \mid (i, j, \cdot) \in \mathcal{E}_r\}$.

Then (I) \rightarrow (II). The converse is not true. Moreover, if all the transition labels of $G(\mathcal{N}, \mathcal{E})$ are linear transformations, then the sum-of-squares optimization problem corresponding to (II) has the same complexity as the one corresponding to (I), in the sense that the polynomial multipliers in both problems have identical degrees.

Remark 5.3 Theorem 5.4 is presented for the case where the sum-of-squares relaxation technique is used for computation of V . The case where the standard \mathcal{S} -Procedure is used is a special case of Theorem 5.4, and the statements of the theorem remain valid.

Proof of Theorem 5.4. We present the proof for the constant rate case. The proof for the variable rate case is similar. If $G_r \sqsubseteq G$, then there exists a sequence of reduced graph models $G_i(\mathcal{N}_i, \mathcal{E}_i)$, $i = 1 \dots q$, where $G_1 = G$, $G_{i+1} \sqsubseteq G_i$, and $G_q = G_r$, with the property that $|\mathcal{N}_{i+1}| = |\mathcal{N}_i| - 1$. That is, G_{i+1} is obtained from G_i by removing one transient node. Furthermore, assume that the functions $\sigma_i(x) \in \mathcal{V}_x^d$, $i \in \mathcal{N}$ define a Lyapunov invariant for G_i and that G_{i+1} is derived from G_i by eliminating node n . For every arc $(m, n, r) \in \mathcal{E}$, let

$\Gamma_{nm}^r(\cdot)$ denote the vector of polynomial multipliers (and/or SOS multipliers) used in the SOS relaxation, and let Ψ denote the corresponding P-Satz polynomial. Then:

$$\Psi_1(\sigma_n(T_{nm}^r x) - \theta\sigma_m(x), \Pi_{nm}^r, \Gamma_{nm}^r(x)) = 0, \quad m \in \mathcal{I}(n), \quad r \in \mathcal{A}_{nm} \quad (5.22)$$

$$\Psi_2(\sigma_l(T_{ln}^s x) - \theta\sigma_n(x), \Pi_{ln}^s, \Gamma_{ln}^s(x)) = 0, \quad l \in \mathcal{O}(n), \quad s \in \mathcal{A}_{ln} \quad (5.23)$$

A necessary condition for (5.23) to hold is that:

$$\begin{aligned} \Psi_2(\sigma_l(T_{ln}^s T_{nm}^r x) - \theta\sigma_n(T_{nm}^r x), \Pi_{ln}^s(T_{nm}^r), \Gamma_{ln}^s(T_{nm}^r x)) &= 0, \quad m \in \mathcal{I}(n), \quad r \in \mathcal{A}_{nm} \\ l \in \mathcal{O}(n), \quad s \in \mathcal{A}_{ln} & \quad (5.24) \end{aligned}$$

Now, for a fixed quadruplet (m, l, r, s) (representing a transition from m to n along arc r , followed by a transition from n to l along arc s), if $x \in \Pi_{nm}^r \cap \Pi_{ln}^s(T_{nm}^r)$ then the corresponding equalities in both (5.24) and (5.22) hold. Since Ψ is strongly linear w.r.t. $\{\sigma_i(\cdot)\}$, by multiplying (5.22) by θ and adding it to (5.24) we obtain:

$$\begin{aligned} \tilde{\Psi}(\sigma_l(T_{ln}^s T_{nm}^r x) - \theta^2\sigma_m(x), [\Pi_{nm}^r, \Pi_{ln}^s(T_{nm}^r)], [\Gamma_{nm}^r(x), \Gamma_{ln}^s(T_{nm}^r x)]) &= 0, \\ r \in \mathcal{A}_{nm}, \quad s \in \mathcal{A}_{ln}. & \end{aligned}$$

In a similar fashion to the procedure described earlier in this section (equations (5.18)–(5.21)), a simple rescaling of the decision variables (coefficients of the polynomials) can be applied to convert the Lyapunov invariant rate (θ^2) to θ . By definition, this implies that σ_l and σ_m satisfy the Lyapunov conditions along all the $|\mathcal{A}_{nm}| \times |\mathcal{A}_{ln}|$ arcs that were added between $\mathcal{I}(n)$ and $\mathcal{O}(n)$ in the reduction process. Since σ_l and σ_m satisfy the Lyapunov conditions along any and all the existing arcs (before reduction) between m and l , we conclude that V satisfies the Lyapunov conditions for the reduced model. It follows by induction that

$$V_r(i, x) := \sigma_i(x), \quad i \in \mathcal{N}_r$$

can be computed for G_r by solving a strongly linear (w.r.t. $\{\sigma_i(\cdot)\}$) SOS problem. Finally, if the transition labels of $G(\mathcal{N}, \mathcal{E})$ are linear transformations, the new vector of multipliers

$([\Gamma_{nm}^r(x), \Gamma_{ln}^s(I_{nm}^r x)])$ has the same degree as the vector of multipliers for the original graph: $([\Gamma_{nm}^r(x), \Gamma_{ln}^s(x)])$. Proof is complete. ■

In light of Theorem 5.1, the conclusion of the above discussion is that analysis of the reduced models are always beneficial, regardless of the convex relaxations that are used at the numerical optimization phase.

5.4 Summary

In this chapter, we presented several results that complement the software analysis framework that we presented in the previous chapters of this dissertation. We showed via an example that the application of the framework to graph models of programs that are semantically identical but syntactically different does not produce identical results, which suggests that the success or failure of the method is contingent on the choice of the graph model. Based on this observation, we introduced the concepts of graph reduction, irreducible graphs, and minimal and maximal realizations of graph models. Several new theorems that compare the performance of the original graph model of a computer program and its reduced offsprings were presented. In particular the so-called K1 graphs and their extension, the Kn graphs, were studied in great detail. While it is not true in general that an irreducible realization of higher order outperforms an irreducible realization of lower order, we showed that for the Kn graphs, the minimal realizations are always outperformed by a specific realization of higher order. The importance of the study of Kn graphs is that an arbitrary (connected) graph can be converted to a Kn graph by removing and adding auxiliary nodes. The results of this chapter therefore, can be used for construction of efficient graph models that systematically improve analysis of computer code via the framework that has been presented in this dissertation.

Chapter 6

Optimal Implementation of Linear Time-Invariant Systems for Safety-Critical Applications

In the previous chapters, we presented a framework based on convex optimization of Lyapunov invariants for verification of safety and liveness properties of computer programs, particularly those of safety-critical control systems. In this chapter, we take a dual approach to providing guarantees of safety and good performance for these systems. The approach is applicable to a specific but very important problem in safety-critical system design and implementation. We show that for the problem of code-level implementation of discrete-time linear time-invariant systems in digital processors, quadratic Lyapunov functions and semidefinite programming can be used for finding an implementation that not only is safe by design, but also is optimized to minimize loss of performance due to quantization effects.

The particular implementation that is considered is a finite word-length implementation with quantization after multiplication. The objective is to minimize the effects of finite word-length constraints on deviation from ideal performance, while respecting the overflow limits. The problem is first formulated as a special case of the linear controller synthesis problem where the controller has a particular structure. This problem is known to be a hard non-convex problem in general. We show that this special case can be convexified exactly, and the optimal

implementation can be computed by solving a set of linear matrix inequalities. It is observed that the transfer function corresponding to the optimal finite word-length implementation of a discrete-time linear system is not necessarily identical to that of an ideal implementation with infinite-precision arithmetic.

6.1 Introduction

An important problem that arises in software-enabled control applications as well as in open loop estimation and filtering applications, is the problem of implementation of a discrete-time linear time-invariant system in a digital processor [6, 7]. The system to be implemented is specified by its real-rational transfer matrix $H(z)$ which could be, for instance, the outcome of a multivariable controller design process or a Kalman filter design process. The current practice for the implementation problem generally involves the following two steps:

1. First, a state space realization of $H(z)$ is computed. That is, matrices A , B , C , and D are calculated such that $H(z) = C(zI - A)^{-1}B + D$. This process is standard and several algorithms exist for computing a state space realization of a transfer function based on the numerator and denominator coefficients. For instance, the MATLAB Real-Time Workshop [97] uses a canonical state-space realization for the code-level implementation of linear systems specified by transfer matrices, which can be inefficient and unsafe for safety-critical systems.
2. Second, computer code is generated (either manually or automatically) to implement the state space equations:

$$x[k+1] = Ax[k] + Bw[k] \quad (6.1a)$$

$$y[k] = Cx[k] + Dw[k] \quad (6.1b)$$

For instance, a particular pseudocode implementation of the state space equations (6.1) is given in Program 6-1. The pseudocode in Program 6-1 is very close to an actual implementation of (6.1) with a high-level programming language such as C.

```

//A: array [n,n]; B&C: array [n,1]; D: scalar;    // state-space matrices data
while (true)  {
    wait for the clock (.)                        // wait for the next sampling time, t=k
    w = *PtrToInput;                             // read the input signal from the memory
    y = 0;                                       // reset the output. begin the update
    for ( i = 1 ; i == N ; i++)  {
        q[ i ] = x[ i ];                        // fill in the buffer variables q
        x[ i ] = 0;                             // reset the state variables
    }
    for ( i = 1 ; i == N ; i++)  {              // start updating the state variables
        for ( j = 1 ; j == N ; j++)  {
            x[ i ] = x[ i ] + q[ j ] * A[ i ][ j ];
        }
        x[ i ] = x[ i ] + B[ i ] * w;
        y = y + C[ i ] * q [ i ];
    }                                           // state update for time t=k completed
    y = y + D * w;                             // output update for time t=k completed
    *PtrToOutput = y; }                       // write the output signal to the memory

```

Program 6-1: Pseudocode for state space implementation of a SISO linear system with state space matrices (A, B, C, D)

As it was already mentioned, the state space realization (A, B, C, D) corresponding to a transfer matrix $H(z)$ is not unique. In particular, if (A, B, C, D) is a state space realization of $H(z)$, then for any non-singular matrix T of appropriate dimension, $(T^{-1}AT, T^{-1}B, CT, D)$ is also a state space realization of $H(z)$. Furthermore, the set of all state space realizations of $H(z)$ can be completely characterized by (A, B, C, D) , in the sense that if (controllable and observable) matrices (a, b, c, d) satisfy $H(z) = c(zI - a)^{-1}b + d$, then there exists a non-singular

matrix T such that $(a, b, c, d) = (T^{-1}AT, T^{-1}B, CT, D)$ (see [9] for more details). Therefore, in theory, there are infinitely many ways (which are in practice not equivalent) to implement (e.g. via Program 6-1) a given transfer matrix on a digital processor. For the time being, we would like to emphasize that the approach that we will present in this chapter is different than the current practice, in the sense that it does not assume that matrices A , B , C , and D necessarily satisfy $H(z) = C(zI - A)^{-1}B + D$. More details will be provided as we proceed.

If arithmetic operations could be performed with infinite precision, any two implementations corresponding to different minimal realizations of $H(z)$ would be equivalent, in the sense that they would produce identical output signals in response to any given input (assuming they start from identical initial conditions). However, since computers are finite-state machines, (6.1) can only be implemented with finite precision arithmetic. Hence, in practice, the following compromises must be made to satisfy the finite precision constraints: First, the coefficients of the matrices (A, B, C, D) must be quantized before implementation in the system to satisfy the word-length constraints. Second, the internal signals must be quantized in real time to the nearest available quantization level.

Program 6-2 illustrates the quantization of the internal signals at the implementation level. The program represents a particular finite-precision implementation on a fixed-point processor, where the so-called quantization after multiplication is used for implementation. Program 6-2 is essentially a translation of the Program 6-1 into Assembly language for the Texas Instruments Inc.'s TMS320C5x DSP processor. The processor has a 32-bit two's complement ALU, a 16-bit \times 16-bit multiplier, a 32-bit accumulator (ACC), a 32-bit product register (PREG), 16-bit auxiliary registers (AR0-7), and 16-bit data memory [99]. Quantization occurs when the updated value of the state, currently stored at the 32-bit accumulator, is saved to the 16-bit memory location of `X_state`. The instruction "SACH" saves the 16 MSBs of the ACC into the data memory address of `X_state`, and hence, the 16 LSBs are lost.

The consequence is that the behavioral properties of different implementations corresponding to different state-space realizations are not identical. Indeed, the system response to the same input could vary drastically within a class of different state-space implementations. The extent of the deviation from the ideal response depends on the particular realization used in (6.1), the finite precision format for computations, and the pole/zero structure of $H(z)$ [83].

```

RPTB   INFINITE_WHILE_LOOP
include WAIT_FOR_CLOCK.asm      ; wait for the next sampling time
SPLK   #N-1, BRCR                ; loop N times
RPTB   END_LOOP                  ; for j=0; j<=BRCR;j++
      MAR   *, AR1                ; modify auxiliary register (AR)
      LMMR  *0+, A_MTRX           ; load memory value A_MTRX into AR1
                                      ; AR1 = A_MTRX(j*INDX), INDX = N
      RPTZ  #N-1                  ; clear ACC and PREG; for ( i = 0 ; i <= N-1 ; i++)
          MAC   X_state, *+        ; (1) ACC = ACC + PREG
                                      ; (2) PREG = X_state(i)*AR1
      MAR   *, AR2                ; modify auxiliary register (AR)
      LMMAR *+, B_MTRX           ; AR2 = B_MTRX(j)
      MAC   W_input, *+          ; (1) ACC = ACC + PREG
                                      ; (2) PREG = W_state(j)*AR2
      APAC                                ; ACC = ACC + PREG; ACC contains the next X(j)
      SACH  X_state, *           ; save 16 MSBs of ACC into X_state; 16 LSBs lost.
                                      ; (Quantization After Multiplication.)

      END_LOOP
      OPL   #1, PMST              ; set BRAF to continue loop indefinitely
INFINITE_WHILE_LOOP

```

*Program 6-2: Fixed-point implementation with quantization after multiplication (QAM) for a SISO linear system with state space matrices A_MTRX , and B_MTRX . The processor is a TMS320C5x DSP processor by Texas Instruments Inc. It has a 16-bit \times 16-bit multiplier, a 32-bit two's complement ALU, a 32-bit accumulator (ACC), and a 32-bit product register (PREG). Note that the instructions for the computation of the output are not included in the code. Quantization of the internal variables occurs when the updated value of the state X_state is removed from the 32-bit accumulator and saved to the 16-bit memory location of X_state (instruction: SACH $X_state, *$).*

As a very simple example of the effects of different state space realizations on performance, let us compare two state space realizations $\mathcal{R}_1(A, B, C, D)$ and $\mathcal{R}_2(A, \alpha B, \alpha^{-1}C, D)$, where α is a scalar. Although these realizations define the same transfer matrix, the internal state variables (the $x[i]$ variables in Program 6-1, or X_state in Program 6-2) corresponding to the implementation of \mathcal{R}_2 could become very large (for the same input) if α is a large number. However, a finite-state implementation is inherently restricted with a finite dynamic range, that is, the variables cannot exceed in magnitude a certain limit that is specified by the word-length and the format. If a variable exceeds the dynamic range then an arithmetic overflow will occur, the result of which is either a rollover to the smallest number in the range or saturation (clipping) to the largest number in the range. Although the consequences of an overflow may be less dramatic when saturation arithmetic is used, if occurred, it can still cause significant performance distortion or even worse, instability.

The problem of finding a state space realization of $H(z)$ that minimizes the performance degradation due to fixed-point roundoff quantization of the internal variables subject to overflow constraints was studied to a great extent by M. Rotea and D. Williamson in [83]. Other important references on the subject include [6] and [7]. In [83], the authors start with an arbitrary state space realization of $H(z)$, namely (A, B, C, D) , and search for an optimal similarity transformation matrix T such that the realization defined by $T : (T^{-1}AT, T^{-1}B, CT, D)$ does not lead to an overflow, and is optimal in the sense that some measure of performance degradation is minimal compared to all other realizations of $H(z)$. While we have learned and greatly benefitted from the problem formulation of Rotea and Williamson, our problem formulation is different; we do not assume that the optimal state space implementation is necessarily within the class of state space realizations that are related to an arbitrary realization of $H(z)$ by a similarity transformation. In other words, we search for optimal implementation matrices (A_c, B_c, C_c, D_c) and do not assume that $C_c(zI - A)^{-1}B_c + D_c$, the transfer matrix of (A_c, B_c, C_c, D_c) , is necessarily identical to $H(z)$. This formulation is more generic and includes the formulation of [83] as a special case, and hence, is expected to lead to a better design. This intuition is confirmed by the numerical simulations presented in Section 6.4. The problem formulation is stated mathematically in the next section.

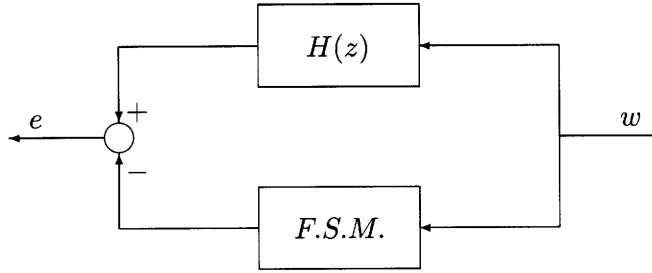


Figure 6-1: The error system

6.2 Problem Formulation

Given a strictly stable transfer matrix $H(z)$, we are interested in finding an optimal finite-state implementation of $H(z)$, such that the output of the error system (Figure 6-1), defined as the difference between $H(z)$ and the finite-state implementation, is small in some sense.

In this document, we focus on a particular F.S.M. (Finite-State Machine) implementation, which is a state-space implementation with quantization after multiplication¹:

$$x_c[k+1] = \Gamma(A_c x_c[k] + B_c w[k]), \quad w[k] \in [-1, 1] \quad (6.2a)$$

$$y_c[k] = C_c x_c[k] + D_c w[k] \quad (6.2b)$$

where, the quantization operator $\Gamma(\cdot)$ is a b -bit regular two's complement quantizer with saturation limits (Figure 6-2). We ignore the quantization error induced with the calculation of the output $y_c[k] = C_c x_c[k] + D_c w[k]$. The error system in Figure 6-1 is then defined by the following equations:

$$x_c[k+1] = \Gamma(A_c x_c[k] + B_c w[k]), \quad w[k] \in [-1, 1] \quad (6.3a)$$

$$x[k+1] = Ax[k] + Bw[k] \quad (6.3b)$$

$$y_c[k] = C_c x_c[k] + D_c w[k] \quad (6.3c)$$

$$y[k] = Cx[k] + Dw[k] \quad (6.3d)$$

$$e[k] = y[k] - y_c[k] \quad (6.3e)$$

¹The results of this chapter can be conveniently extended to at least two other finite state implementation schemes, namely, *quantization before multiplication* and *quantization with integer residue feedback* [83]. For clarity and convenience in notation, we only present the *quantization after multiplication* case.

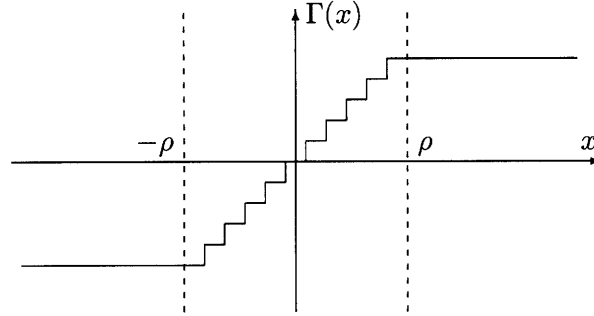


Figure 6-2: The Quantizer $\Gamma(\cdot)$: Two's complement rounding with saturation

In terms of equation (6.3), the objective is to find matrices (A_c, B_c, C_c, D_c) that minimize the measure of performance $\|e\|$, where $\|\cdot\|$ is an appropriately defined norm, and (A, B, C, D) is an arbitrary state space realization of $H(z)$. As it was mentioned in the introduction, Rotea *et. al.* [83] studied a very similar formulation of this problem with the additional assumption that $C_c(zI - A_c)^{-1}B_c + D_c = H(z)$. Hence, they start with an arbitrary state space realization (A, B, C, D) and search for an optimal similarity transformation such the realization $(T^{-1}AT, T^{-1}B, CT, D)$ meets the objectives. We search for the parameters of the optimal implementation without restricting ourselves to implementing the same transfer function. The formulation that we consider is more generic and covers that of Rotea *et. al.* as a special case, and is evidently much harder to solve. If the quantization levels are small relative to the dynamic range (i.e. high-bit quantizer), then, intuitively, we would expect that the coefficients of the transfer matrix of $H(z)$ and $C_c(zI - A_c)^{-1}B_c + D_c$ be very close. On the other hand, if the quantizer is coarse (i.e. low-bit quantizer), then we would expect that the difference between $H(z)$ and $C_c(zI - A_c)^{-1}B_c + D_c$ be larger. This intuition is confirmed by numerical simulations as we will see later in this chapter.

6.2.1 Linearization via signal + noise model

Since quantization is a nonlinear operation, the equations (6.2) and (6.3) define highly nonlinear systems. The problem of finding the optimal state-space matrices (A_c, B_c, C_c, D_c) for this highly nonlinear system is an intractable problem in many aspects. In order to simplify the

problem to some extent, we first apply a linearization technique that is commonly used in signal processing applications. Consider again the nonlinear quantization function $\Gamma : \mathbb{R} \rightarrow \mathbb{R}$, which corresponds to the b -bit two's complement uniform quantizer with saturation as shown in Figure 6-2. Let ρ represent the dynamic range and δ the height of the quantization levels. Then, $\delta := \rho (2^b - 1)^{-1}$, where b is the number of bits (equivalently the number of different levels in the quantizer). Define the nonlinear operator $N : \mathbb{R} \rightarrow \mathbb{R}$ according to $N(\eta) := \Gamma(\eta) - \eta$. The error system equation (6.3) can then be rewritten in the following way:

$$x[k+1] = Ax[k] + Bw[k], \quad w[k] \in [-1, 1] \quad (6.4a)$$

$$x_c[k+1] = \varepsilon[k] + \eta[k], \quad (6.4b)$$

$$\eta[k] = A_c x_c[k] + B_c w[k] \quad (6.4c)$$

$$\varepsilon[k] = N(\eta[k]), \quad (6.4d)$$

$$y[k] = Cx[k] + Dw[k], \quad (6.4e)$$

$$y_c[k] = C_c x_c[k] + D_c w[k], \quad (6.4f)$$

$$e[k] = y[k] - y_c[k] \quad (6.4g)$$

The block diagram interpretation of these equations is shown in Figure 6-3. It follows from the definitions of $\Gamma(\cdot)$ and $N(\cdot)$ that if $|\eta| \leq \rho$, then $|N(\eta)| \leq \delta$. This observation serves as the basis for construction of a linearized model. In many signal processing applications the standard procedure is to remove the nonlinear operator $N(\cdot)$, and assume that the quantization error ε is an uncorrelated exogenous input. This is the so-called signal+noise model of the quantization process. The main assumption behind this model is that the internal variables do not overflow and that the properties of the exogenous input ε are independent of the particular implementation, that is, matrices A_c , B_c , C_c , D_c . We assume that these assumptions hold and remove the nonlinear operator $N(\cdot)$ and represent its output by an external (uncorrelated) noise $\varepsilon[k]$, where, subject to the constraint $\|\eta[k]\|_\infty \leq \rho$, the signal $\varepsilon[k]$ satisfies: $\|\varepsilon[k]\|_\infty \leq \delta$. This simplification allows us to formulate the problem of finding the optimal realization as a constrained linear model matching problem subject to external noise. This formulation is presented next.

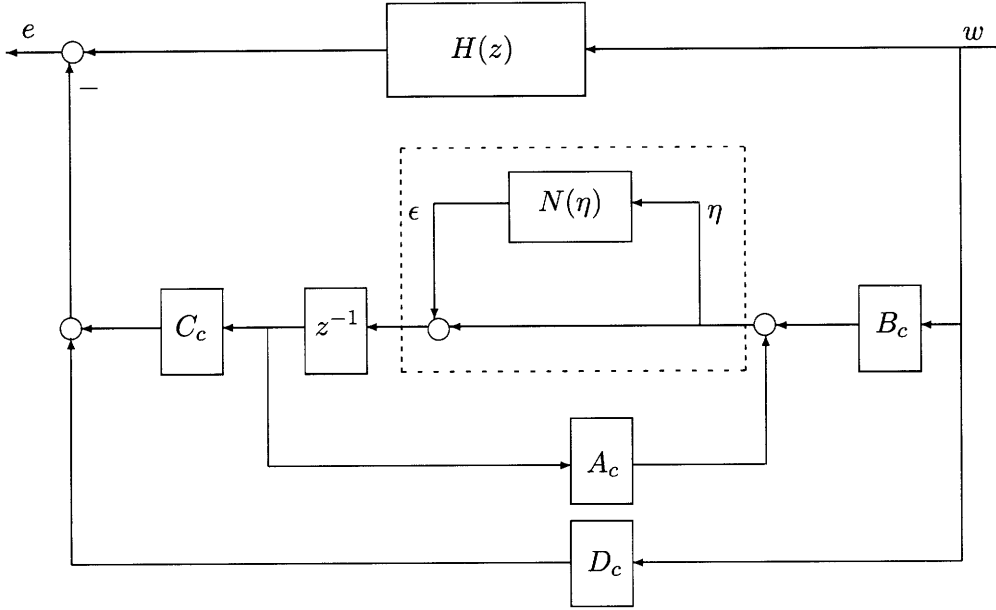


Figure 6-3: The error system corresponding to a particular finite-state implementation with quantization after multiplication $x_c[k+1] = \Gamma(A_c x_c[k] + B_c w[k])$. Inside the dashed box is the quantizer. Given $H(z)$, the objective is to find (A_c, B_c, C_c, D_c) such that the error $\|e\|$ is small for some appropriately defined norm.

6.2.2 Structured Linear Controller Synthesis Formulation

In the previous section, we constructed a linearized model of the error system (6.3). The linearized model was obtained by removing the nonlinear operator $N(\cdot)$ and representing its output $\epsilon[k]$ as an exogenous noise with bounded \mathcal{L}_∞ norm: $\|\epsilon[k]\|_\infty \leq \delta = \rho(2^b - 1)^{-1}$. We argued that subject to the overflow constraint $\|\eta[k]\|_\infty \leq \rho$, the assumption $\|\epsilon[k]\|_\infty \leq \delta$ is valid. In this section, we show that the problem of finding the optimal implementation can be formulated as a special case of the linear \mathcal{L}_1 controller synthesis problem where the controller has a specific structure.

Let $G_{w\eta}(z)$, $G_{\epsilon\eta}(z)$, $G_{wy_c}(z)$, and $G_{\epsilon y_c}(z)$, denote the transfer functions from w and ϵ to η and y_c respectively. Then, the linearized system can be described in the frequency domain by the following equations:

$$\begin{bmatrix} \tilde{e} \\ \tilde{\eta} \end{bmatrix} = G_{cl}(z) \begin{bmatrix} w \\ \tilde{\epsilon} \end{bmatrix}$$

$$G_{cl}(z) = \begin{bmatrix} \sigma(H(z) - G_{wy_c}(z)) & -\sigma G_{\varepsilon y_c}(z) \delta \\ \rho^{-1} G_{w\eta}(z) & \rho^{-1} G_{\varepsilon\eta}(z) \delta \end{bmatrix} \quad (6.5)$$

where ρ and δ are the parameters of the quantizer, $\tilde{\varepsilon} := \delta^{-1}\varepsilon$ is the normalized vector of ε , such that $\|(w, \tilde{\varepsilon})\|_\infty \leq 1$, and $\tilde{e} := \sigma e$ and $\tilde{\eta} := \rho^{-1}\eta$ are the scaled versions of e and η such that $\|(\tilde{e}, \tilde{\eta})\|_\infty \leq 1$. Under this formulation, the problem of finding the optimal implementation is equivalent to finding an implementation that would maximize σ subject to the following constraint:

$$\|G_{cl}(z)\|_1 \leq 1 \quad (6.6)$$

For convenience in notation define:

$$y_1 := \begin{bmatrix} \tilde{e} \\ \tilde{\eta} \end{bmatrix}, \quad y_2 := \begin{bmatrix} w \\ \tilde{\varepsilon} \end{bmatrix}, \quad u_1 := \begin{bmatrix} w \\ \tilde{\varepsilon} \end{bmatrix}, \quad u_2 := \begin{bmatrix} y_c \\ \tilde{\eta} \end{bmatrix}$$

Then, (6.6) can be viewed as a *non-standard* robust controller synthesis problem in the following way: Find $K(z)$, such that the closed loop system $\mathcal{F}(K(z), P(z))$ defined by:

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = P(z) \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

$$u_2 = K(z) y_2$$

satisfies $\|\mathcal{F}(K(z), P(z))\|_\infty \leq 1$, where:

$$P(z) = \begin{bmatrix} \sigma H(z) & 0 & -\sigma & 0 \\ 0 & 0 & 0 & \rho^{-1} I_n \\ 1 & 0 & 0 & 0 \\ 0 & \delta I_n & 0 & 0 \end{bmatrix}$$

and $K(z)$ has the specific structure:

$$K(z) := \left[\begin{array}{c|c} A_c & \begin{bmatrix} B_c & I \end{bmatrix} \\ \hline \begin{bmatrix} C_c \\ A_c \end{bmatrix} & \begin{bmatrix} D_c & 0 \\ B_c & 0 \end{bmatrix} \end{array} \right].$$

The problem that we have formulated is a particular instance of a more general problem which is unsolved in many aspects; a state space characterization which is convenient for synthesis under \mathcal{L}_1 norm constraints is not known to date [29], and furthermore, the problem of controller synthesis under structure constraints on the controller is also known to be a hard non-convex problem. In an attempt to simplify the problem we replace the constraint $\|G_{cl}\|_1 \leq 1$ by the constraint $\|G_{cl}\|_\infty \leq \gamma$, where γ is a positive constant to be determined (perhaps via iterative design) such that $\|G_{cl}\|_1 \leq 1$ holds. A suitable value of γ can be found by a line search. First, γ is fixed to a positive number and the maximal σ is found. It is then checked if $\|G_{cl}\|_1 \leq 1$. If $\|G_{cl}\|_1 > 1$, the process must be repeated with a smaller γ . If $\|G_{cl}\|_1 \ll 1$ the process can be repeated by a larger γ . Although, we have modified and in some sense relaxed the original problem, we do not consider this modification significant in terms of affecting the optimal solution. The rest of this chapter focuses on solving the following problem:

$$\max \sigma, \text{ subject to } \|G_{cl}\|_\infty \leq \gamma \quad (6.7)$$

Due to the specific structure imposed on the controller, this problem is still a hard problem for which the application of standard synthesis algorithms leads to non-convex criteria. In the next section, we first formulate an equivalent (non-convex) matrix inequality version of (6.7). We then show that despite the structure imposed on the controller, due to the sparsity of the plant, this problem can be convexified exactly. We then present an LMI criterion which solves problem (6.7).

6.3 Optimal State Space Implementation via \mathcal{H}^∞ Optimization

6.3.1 Nonconvex Matrix Inequality Criterion

We have so far established our interest in finding matrices A_c, B_c, C_c, D_c , such that the system G_{cl} defined in (6.5), satisfies $\|G_{cl}\|_\infty \leq \gamma$. In the following theorem, we present a necessary and sufficient condition for existence of such matrices. For convenience in notation, we have defined $\alpha := \gamma\rho^2 - \gamma^{-1}\delta^2$.

Theorem 6.1 *Given $\sigma > 0$, there exist matrices A_c, B_c, C_c, D_c s.t. $\|G_{cl}\|_\infty < \gamma$, if and only*

if there exist symmetric positive definite matrices $X = X^T \succ 0$, $Y = Y^T \succ 0$, $M = M^T \succ 0$, $V = V^T \succ 0$ and matrices $N \in \mathbb{R}^{n \times n}$, and $U \in \mathbb{R}^{n \times n}$, such that the following conditions hold:

$$\Pi_1 := \begin{bmatrix} X - AXA^T & B & U \\ B^T & \gamma & 0 \\ U^T & 0 & \alpha I + V \end{bmatrix} \succ 0 \quad (6.8)$$

$$\Pi_2 := \begin{bmatrix} Y - A^T Y A & \sigma C^T & -A^T N \\ \sigma C & \gamma & 0 \\ -N^T A & 0 & \gamma \delta^{-2} I - M \end{bmatrix} \succ 0 \quad (6.9)$$

$$P := \begin{bmatrix} Y & N \\ N^T & M \end{bmatrix} = \begin{bmatrix} X & U \\ U^T & V \end{bmatrix}^{-1} \succ 0 \quad (6.10)$$

In addition, if matrices Y , N , M and X , U , V satisfy the above conditions, then A_c , B_c , C_c , D_c can be obtained by solving the following LMI problem:

$$\begin{bmatrix} \Theta_{11} & \Theta_{12} \\ \Theta_{12}^T & \Theta_{22} \end{bmatrix} \succ 0 \quad (6.11)$$

where,

$$\Theta_{11} := \begin{bmatrix} X & U & A & 0 \\ U^T & V & 0 & A_c \\ A^T & 0 & Y & N \\ 0 & A_c^T & N^T & M \end{bmatrix} \quad \Theta_{12} := \begin{bmatrix} B & 0 & 0 & 0 \\ B_c & \delta I & 0 & 0 \\ 0 & 0 & \sigma C^T & 0 \\ 0 & 0 & -\sigma C_c^T & \rho^{-1} A_c^T \end{bmatrix}$$

$$\Theta_{22} := \begin{bmatrix} \gamma & 0 & \sigma(D - D_c)^T & \rho^{-1} B_c^T \\ 0 & \gamma I_n & 0 & 0 \\ \sigma(D - D_c) & 0 & \gamma & 0 \\ \rho^{-1} B_c & 0 & 0 & \gamma I_n \end{bmatrix}$$

Proof. The proof is presented in the Appendix. ■

Due to the inverse matrix condition (6.10), the constraints of Theorem 6.1 are non-convex with respect to the decision parameters X , U , V , and Y , N , M . We will show that these constraints can be convexified via a series of suitably defined change of variables and congruence transformations. The result is a convex criterion in the form of Linear Matrix Inequalities.

6.3.2 Convexification of the Matrix Inequality Criterion

The following theorem is the most important contribution of this chapter.

Theorem 6.2 *Assume that all the internal variables in the state space implementation (6.2) are allocated an equal number of bits, that is, the diagonal matrices ρ and δ are scalar multiples of the identity matrix. Then, given $\sigma > 0$, there exist matrices A_c , B_c , C_c , D_c s.t. $\|G_{cl}\|_\infty < \gamma$ if and only if there exist matrices $Z = Z^T \succ 0$, $T = T^T \succ 0$, and $W = W^T \succ 0$, such that:*

$$\Upsilon_1 := \begin{bmatrix} Z + T - A(Z + T)A^T & B & -T \\ & B^T & \gamma & 0 \\ & -T & 0 & T + \alpha W \end{bmatrix} \succ 0 \quad (6.12)$$

and

$$\Upsilon_2 := \begin{bmatrix} Z + T - AZA^T & T & -\sigma AZC^T \\ & T & T - \delta^2 \gamma^{-1} W & 0 \\ & -\sigma CZA^T & 0 & \gamma - \sigma^2 CZC^T \end{bmatrix} \succ 0 \quad (6.13)$$

Proof. It is sufficient to prove that the conditions (6.8), (6.9), and (6.10) of Theorem 6.1 hold if and only if (6.12) and (6.13) hold. First, recall the generic matrix inversion formulae:

$$\text{If } \begin{bmatrix} Y & N \\ N^T & M \end{bmatrix}^{-1} = \begin{bmatrix} X & U \\ U^T & V \end{bmatrix}, \text{ then:}$$

$$\begin{bmatrix} X & U \\ U^T & V \end{bmatrix} \equiv \begin{bmatrix} Y^{-1} + Y^{-1}NVN^TY^{-1} & -Y^{-1}NV \\ -VN^TY^{-1} & V \end{bmatrix} \quad (6.14)$$

Now, define:

$$Z = Y^{-1} \quad (6.15a)$$

$$T = Y^{-1}NVN^TY^{-1} \quad (6.15b)$$

$$W = Y^{-1}NN^TY^{-1} \quad (6.15c)$$

Note that with this change of variables we have $X = Z + T$. Using (6.14) and (6.15), substitute for X and U in Π_1 of (6.8) to obtain:

$$\Upsilon := \begin{bmatrix} (Z+T) - A(Z+T)A^T & B & -Y^{-1}NV \\ B^T & \gamma & 0 \\ -VN^TY^{-1} & 0 & \alpha I + V \end{bmatrix} \succ 0$$

Let $E_1 := \text{diag}\{I_{n+1}, N^TY^{-1}\}$, then the condition $\Upsilon \succ 0$ is equivalent to $\Upsilon_1 := E_1^T \Upsilon E_1 \succ 0$, where:

$$\Upsilon_1 := \begin{bmatrix} (Z+T) - A(Z+T)A^T & B & -T \\ B^T & \gamma & 0 \\ -T & 0 & \alpha W + T \end{bmatrix}$$

which is exactly (6.12). Now, we focus on Π_2 . It can be shown via the Schur complement and switching the rows and columns of Π_2 that the condition $\Pi_2 \succ 0$ is equivalent to

$$\bar{\Pi}_2 := \begin{bmatrix} X & U & A & 0 \\ U^T & V - \delta^2 \gamma^{-1} & 0 & 0 \\ A^T & 0 & Y & \sigma C^T \\ 0 & 0 & \sigma C & \gamma \end{bmatrix} \succ 0$$

Let $E_2 := \text{diag}\{I_n, Y^{-1}, Y^{-1}, I_n\}$, then, $\Pi_2 \succ 0$ is equivalent to $\bar{\Upsilon}_2 := E_2^T \bar{\Pi}_2 E_2 \succ 0$, where:

$$\bar{\Upsilon}_2 := \begin{bmatrix} Z+T & T & AZ & 0 \\ T & T - \delta^2 \gamma^{-1} W & 0 & 0 \\ ZA^T & 0 & Z & \sigma ZC^T \\ 0 & 0 & \sigma CZ & \gamma \end{bmatrix}$$

Again, switching rows and columns and using Schur complement yields $\Upsilon_2 \succ 0$ where

$$\Upsilon_2 := \begin{bmatrix} (Z + T) - AZA^T & T & -\sigma AZC^T \\ T & T - \delta^2 \gamma^{-1} W & 0 \\ -\sigma CZ A^T & 0 & \gamma - \sigma^2 CZC^T \end{bmatrix}$$

which is exactly (6.13). Finally, by the Schur Complement, (6.10) is equivalent to $X \succ 0$, and $X - UV^{-1}U^T \succ 0$. These are in turn equivalent to $Z + T \succ 0$, and $Z \succ 0$, which hold automatically. ■

Remarks

1. After solving the above system of linear matrix inequalities, X, U, V, N, Y, M can be recovered exactly in the following order:

- (1) $Y = Z^{-1}$
- (2) $N = \text{chol}(YWY)$
- (3) $V = N^{-1}YTYN^{T-1}$
- (4) $M = V + N^T Y^{-1} N$
- (5) $X = Y^{-1} + Y^{-1} N V N^T Y^{-1}$
- (6) $U = -Y^{-1} N V$

2. We use Theorem 6.2, to find the Lyapunov matrices Y, N, M that satisfy the constraints of Theorem 6.1. Once we have found these matrices, the optimal implementation matrices (A_c, B_c, C_c, D_c) are found by solving the LMI problem (6.11).
3. The quantity $\rho \delta^{-1}$ is only a function of the number of bits allocated to register each state variable:

$$\rho \delta^{-1} = 2^b - 1$$

Define $\widetilde{W} := \delta^2 W$. Then, the conditions (6.12) and (6.13) depend only on T, Z, \widetilde{W} , and the quantity $\alpha \delta^{-2}$ which is equal to $\gamma (2^b - 1)^2 - \gamma^{-1}$. Therefore, the LMIs in Theorem 6.2 are only a function of the number of bits. This observation conforms with our intuition

since the performance should not depend on ρ and δ independently, but only on the number of bits. In other words, the performance is independent of scaling the quantizer.

4. We assume that an equal amount of memory is allocated to register each state variable, which is the common case in standard processors. Under this assumption, the variables δ and ρ are scalar multiples of the identity matrix. An interesting case where a non-uniform number of bits is allocated to register different state variables may also be considered. In this case δ and ρ are positive definite diagonal matrices. Unfortunately, this case cannot be formulated as a convex optimization problem. It would be still interesting to investigate the sub-optimal distribution of bits between the states and its effect on improving performance. This may be accomplished by employing an iterative algorithm to solve the (non-convex) conditions of Theorem 6.1.

6.4 Numerical Simulations

Example 6.1 Let $\gamma = 1$, $\rho = 1$, $b = 5$, i.e. $\delta = 2^{-5}$, and let $H(z)$ be given by

$$H(z) = \frac{2z + 1}{z^2 + z + 5/6}$$

Applying our method we obtain $\sigma_{\max} = 0.2493$, with

$$\left[\begin{array}{c|c} A_c & B_c \\ \hline C_c & D_c \end{array} \right] = \left[\begin{array}{c|c} \left[\begin{array}{cc} -0.5545 & -0.6702 \\ 0.8323 & -0.4204 \end{array} \right] & \left[\begin{array}{c} 0.1075 \\ 0.0753 \end{array} \right] \\ \hline \left[\begin{array}{cc} -1.3489 & 16.7983 \end{array} \right] & [0.00136] \end{array} \right]$$

which implements the transfer function

$$H_c(z) := \frac{0.00136z^2 + 1.12z + 2.212}{z^2 + 0.9749z + 0.7909}$$

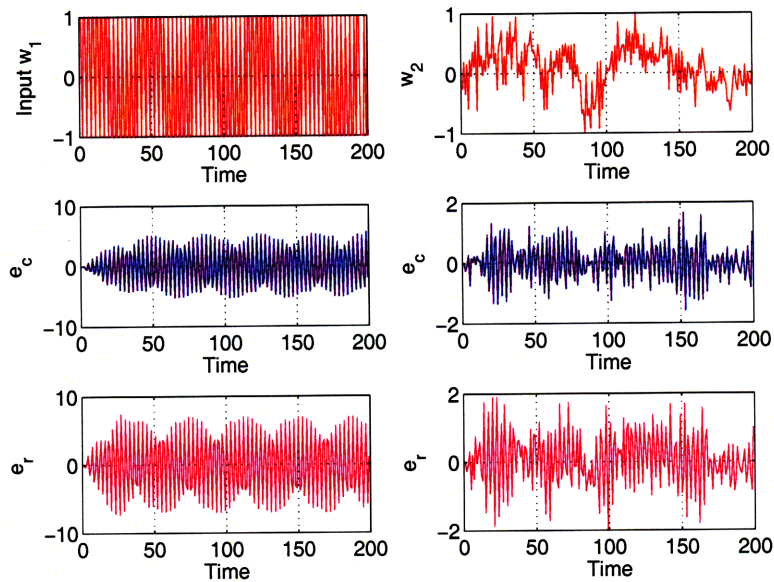


Figure 6-4: Numerical simulations: comparison of our results with [83].

We compare our design with that of Rotea et. al. [83]. Their result gives

$$\left[\begin{array}{c|c} A_{cr} & B_{cr} \\ \hline C_{cr} & D_{cr} \end{array} \right] = \left[\begin{array}{c|c} \left[\begin{array}{cc} -0.5584 & -0.7007 \\ 0.8358 & -0.4516 \end{array} \right] & \left[\begin{array}{c} 0.0659 \\ 0.0425 \end{array} \right] \\ \hline \left[\begin{array}{cc} -1.2704 & 25.5092 \end{array} \right] & [0] \end{array} \right]$$

which implements the same transfer function as $H(z)$.

Figure 6-4 shows the numerical simulation of the performance of the two designs in response to two different Inputs (ω_1 and ω_2). The comparison shows that our design is better in the sense that the error is smaller.

Example 6.2 Let $\gamma = 1.5$, $\rho = 1$, $b = 6$, i.e. $\delta = 2^{-6}$, and let $H(z)$ be given by

$$H(z) = \frac{1}{z^2 + 0.8z + 0.9}$$

Once again, by applying our method we obtain $\sigma_{\max} = 0.2984$, with

$$\left[\begin{array}{c|c} A_c & B_c \\ \hline C_c & D_c \end{array} \right] = \left[\begin{array}{cc|c} \begin{bmatrix} -0.4425 & -0.7952 \\ 0.8822 & -0.3343 \end{bmatrix} & \begin{bmatrix} 0.1012 \\ 0.0489 \end{bmatrix} \\ \hline \begin{bmatrix} -5.1684 & 10.8928 \end{bmatrix} & [-0.0487] \end{array} \right]$$

which implements the transfer function

$$H_c(z) := \frac{-0.0487z^2 + 0.009582z + 1.235}{z^2 + 0.7768z + 0.8494}$$

We compare our design with that of Rotea et al [83]. Their result gives

$$\left[\begin{array}{c|c} A_{cr} & B_{cr} \\ \hline C_{cr} & D_{cr} \end{array} \right] = \left[\begin{array}{cc|c} \begin{bmatrix} -0.4336 & -0.8307 \\ 0.8922 & -0.3664 \end{bmatrix} & \begin{bmatrix} 0.0498 \\ 0.0219 \end{bmatrix} \\ \hline \begin{bmatrix} -8.1807 & 18.5500 \end{bmatrix} & [0] \end{array} \right]$$

which implements the same transfer function as $H(z)$.

Figure 6-5 shows the numerical simulation of the performance of the two designs in response to two different Inputs (ω_1 and ω_2). The comparison shows that our design is better in the sense that the error is smaller.

6.5 Summary

We presented an algorithm based on optimization of quadratic Lyapunov functions for finding optimal finite-state implementations for discrete-time strictly stable linear time-invariant systems. The particular finite-state implementation that we considered consists of quantization of the state variables after multiplication, that is:

$$\begin{aligned} x[k+1] &= \Gamma(A_c x[k] + B_c w[k]) \\ y[k] &= C_c x[k] + D_c w[k] \end{aligned}$$

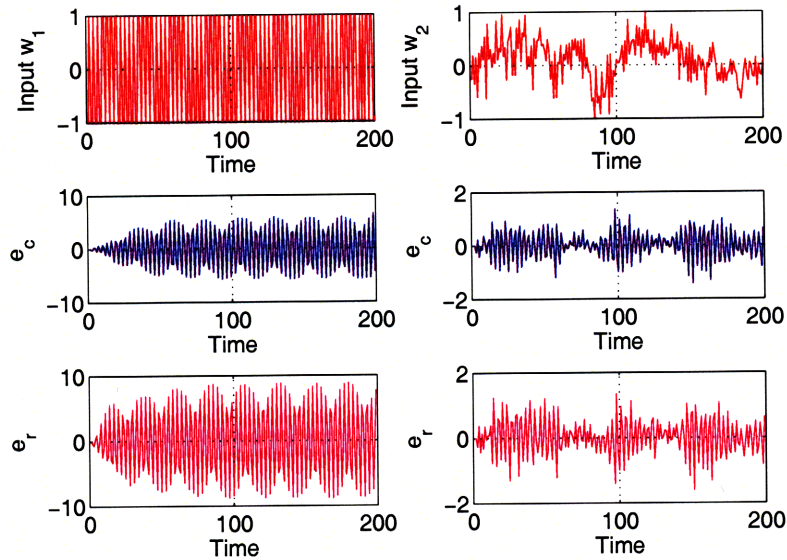


Figure 6-5: Numerical simulations: comparison of our results with [83].

where Γ is the quantizer. We defined the optimal implementation problem as the problem of finding matrices (A_c, B_c, C_c, D_c) such that the internal variables do not saturate the quantizer and the loss of performance due to quantization effects is minimal. We showed that after linearization of the quantization process via the so-called signal+noise model, the problem of finding the optimal implementation can be formulated as a special case of an \mathcal{L}_1 optimal synthesis problem where the controller has a specific structure. We relaxed the \mathcal{L}_1 synthesis problem and replaced it with an \mathcal{H}_∞ synthesis problem. The \mathcal{H}_∞ controller synthesis problem is still in non-standard form due to the special structure of the controller. We showed that this problem can be convexified via a series of congruent transformations and appropriate parameterizations. We presented a linear matrix inequality criterion, the solution of which provides the optimal state space implementation. We compared our results with the results of Rotea *et. al.* [83]. Since our formulation is less constrained and allows more freedom in the search parameters, intuitively, we expect better performance from our results. This intuition was confirmed by numerical simulations. The results of this chapter can be used for software implementation in safety-critical applications.

6.6 Appendix

Proof of Theorem 6.1. The proof of the theorem lies along the same lines as the standard proofs of \mathcal{H}_∞ controller synthesis via LMIs (see for instance [32]). The closed loop system is the feedback interconnection of a plant $P(z)$, and a structured controller $K(z)$, where

$$P(z) := \left[\begin{array}{c|cc} [A] & [B \ 0] & [0 \ 0] \\ \hline [\sigma C] & [\sigma D \ 0] & [-\sigma \ 0] \\ [0] & [0 \ 0] & [0 \ \rho^{-1}I_n] \\ [0] & [1 \ 0] & [0 \ 0] \\ [0] & [0 \ \delta I_n] & [0 \ 0] \end{array} \right]$$

and

$$K(z) := \left[\begin{array}{c|cc} A_c & B_c & I_n \\ \hline C_c & D_c & 0 \\ A_c & B_c & 0 \end{array} \right]$$

The state space equations of the system are given by:

$$\begin{aligned} \begin{bmatrix} x[k+1] \\ x_c[k+1] \end{bmatrix} &= \begin{bmatrix} A & 0 \\ 0 & A_c \end{bmatrix} \begin{bmatrix} x[k] \\ x_c[k] \end{bmatrix} + \begin{bmatrix} B & 0 \\ B_c & \delta I \end{bmatrix} \begin{bmatrix} w[k] \\ \tilde{\varepsilon}[k] \end{bmatrix} \\ \begin{bmatrix} \tilde{\varepsilon}[k] \\ \tilde{\eta}[k] \end{bmatrix} &= \begin{bmatrix} \sigma C & -\sigma C_c \\ 0 & \rho^{-1}A_c \end{bmatrix} \begin{bmatrix} x[k] \\ x_c[k] \end{bmatrix} + \begin{bmatrix} \sigma D - \sigma D_c & 0 \\ \rho^{-1}B_c & 0 \end{bmatrix} \begin{bmatrix} w[k] \\ \tilde{\varepsilon}[k] \end{bmatrix} \end{aligned}$$

Let us introduce the following notation:

$$\begin{aligned} A &:= \begin{bmatrix} A & 0_n \\ 0_n & 0_n \end{bmatrix}, & \mathcal{B}_1 &:= \begin{bmatrix} B & 0_n \\ 0_{n \times 1} & \delta I_n \end{bmatrix}, & \mathcal{B}_2 &:= \begin{bmatrix} 0_n & 0_{n \times 1} \\ I_n & 0_{n \times 1} \end{bmatrix}, \\ \mathcal{C}_1 &:= \begin{bmatrix} \sigma C & 0_{1 \times n} \\ 0_n & 0_n \end{bmatrix}, & \mathcal{C}_2 &:= \begin{bmatrix} 0_n & I_n \\ 0_{1 \times n} & 0_{1 \times n} \end{bmatrix}, & \mathcal{D}_{11} &:= \begin{bmatrix} \sigma D & 0 \\ 0 & 0 \end{bmatrix}, \\ \mathcal{D}_{21} &:= \begin{bmatrix} 0_{n \times 1} & 0_n \\ 1 & 0_{1 \times n} \end{bmatrix}, & \mathcal{D}_{12} &:= \begin{bmatrix} 0_{1 \times n} & -\sigma \\ \rho^{-1}I_n & 0_{n \times 1} \end{bmatrix}. \end{aligned}$$

Now, define \mathcal{K} to be

$$\mathcal{K} := \begin{bmatrix} A_c & B_c \\ C_c & D_c \end{bmatrix}$$

It can then be verified that the closed loop matrices are given by:

$$\begin{aligned} \mathcal{A}_{cl} &\triangleq A + \mathcal{B}_2 \mathcal{K} \mathcal{C}_2 \\ \mathcal{B}_{cl} &\triangleq \mathcal{B}_1 + \mathcal{B}_2 \mathcal{K} \mathcal{D}_{21} \\ \mathcal{C}_{cl} &\triangleq \mathcal{C}_1 + \mathcal{D}_{12} \mathcal{K} \mathcal{C}_2 \\ \mathcal{D}_{cl} &\triangleq \mathcal{D}_{11} + \mathcal{D}_{12} \mathcal{K} \mathcal{D}_{21} \end{aligned}$$

The original structure of the controller has been captured in the (non-standard) structure of \mathcal{B}_1 and \mathcal{D}_{12} . In a standard design situation, the matrices \mathcal{B}_1 and \mathcal{D}_{12} are of the form:

$$\mathcal{B}_1 := \begin{bmatrix} B_1 \\ 0 \end{bmatrix} \text{ and } \mathcal{D}_{12} := \begin{bmatrix} 0 & D_{12} \end{bmatrix}$$

whereas here, the second row block of \mathcal{B}_1 or the first column block of \mathcal{D}_{12} are defined by nonzero matrices. Next, recall the discrete-time version of the bounded real lemma: The system

$$G_{cl}(z) := \left[\begin{array}{c|c} \mathcal{A}_{cl} & \mathcal{B}_{cl} \\ \hline \mathcal{C}_{cl} & \mathcal{D}_{cl} \end{array} \right]$$

satisfies

$$\|G_{cl}(z)\|_{\infty} < \gamma$$

if and only if $\exists P = P^T \succ 0$ (of compatible dimensions), such that:

$$\Pi := \begin{bmatrix} P^{-1} & \mathcal{A}_{cl} & \mathcal{B}_{cl} & 0 \\ \mathcal{A}_{cl}^T & P & 0 & \mathcal{C}_{cl}^T \\ \mathcal{B}_{cl}^T & 0 & \gamma I & \mathcal{D}_{cl}^T \\ 0 & \mathcal{C}_{cl} & \mathcal{D}_{cl} & \gamma I \end{bmatrix} \succ 0.$$

Using the projection lemma ([32]), it can be shown that $\Pi \succ 0$ holds if and only if:

$$\begin{aligned}\Gamma^\perp \Theta \Gamma^{\perp T} &\succ 0 \\ \Lambda^\perp \Theta \Lambda^{\perp T} &\succ 0\end{aligned}$$

where Γ , Λ , Θ are defined by:

$$\Gamma := \begin{bmatrix} \mathcal{B}_2 \\ 0_{2n \times (n+1)} \\ 0_{n+1} \\ \mathcal{D}_{12} \end{bmatrix}, \quad \Lambda^T := \begin{bmatrix} 0_{(n+1) \times 2n} & \mathcal{C}_2 & \mathcal{D}_{21} & 0_{n+1} \end{bmatrix}$$

and

$$\Theta := \begin{bmatrix} P^{-1} & A & B_1 & 0 \\ A^T & P & 0 & C_1^T \\ B_1^T & 0 & \gamma I & D_{11}^T \\ 0 & C_1 & D_{11} & \gamma I \end{bmatrix}.$$

(Γ^\perp and Λ^\perp represent the left annihilator of Γ and Λ respectively). Now, define

$$P := \begin{bmatrix} Y & N \\ N^T & M \end{bmatrix}, \quad \text{and } P^{-1} \equiv Q := \begin{bmatrix} X & U \\ U^T & V \end{bmatrix}$$

and expand the matrix Θ :

$$\Theta := \begin{bmatrix} X & U & A & 0_n & B & 0_n & 0_{n \times 1} & 0 \\ U^T & V & 0_n & 0_n & 0_{n \times 1} & \delta I_n & 0_{n \times 1} & 0 \\ A^T & 0_n & Y & N & 0 & 0 & \sigma C^T & 0_n \\ 0_n & 0_n & N^T & M & 0 & 0 & 0_{n \times 1} & 0 \\ B^T & 0_{1 \times n} & 0 & 0 & \gamma & 0 & \sigma D^T & 0 \\ 0_n & \delta I_n & 0 & 0 & 0 & \gamma I_n & 0 & 0 \\ 0_{1 \times n} & 0_n & \sigma C & 0_{1 \times n} & \sigma D & 0 & \gamma & 0 \\ 0_n & 0_n & 0_n & 0 & 0 & 0 & 0 & \gamma I_n \end{bmatrix}$$

The matrix Λ (defined in (6.16)) has rank $n + 1$ and its left annihilator Λ^\perp can be defined by:

$$\Lambda := \begin{bmatrix} 0_n & 0_{n \times 1} \\ 0_n & 0_{n \times 1} \\ 0_n & 0_{n \times 1} \\ I_n & 0_{n \times 1} \\ 0 & 1 \\ 0_n & 0_{n \times 1} \\ 0_{(n+1) \times n} & 0_{(n+1) \times 1} \end{bmatrix}, \quad \Lambda^\perp := \begin{bmatrix} I_n & 0_n & 0 & 0_n & 0_{n \times 1} & 0 & 0 & 0 \\ 0 & I_n & 0 & 0 & 0_{n \times 1} & 0 & 0 & 0 \\ 0 & 0 & I_n & 0 & 0_{n \times 1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & I_n & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & I_n \end{bmatrix} \quad (6.16)$$

Thus, $\Lambda^\perp \Theta \Lambda^{\perp T} \succ 0$ is equivalent to

$$\begin{bmatrix} X & U & A & 0_n & 0_{n \times 1} & 0 \\ U^T & V & 0_n & \delta I_n & 0_{n \times 1} & 0 \\ A^T & 0_n & Y & 0 & \sigma C^T & 0_n \\ 0_n & \delta I_n & 0 & \gamma I_n & 0 & 0 \\ 0_{1 \times n} & 0_{1 \times n} & \sigma C & 0 & \gamma & 0 \\ 0_n & 0_n & 0_n & 0 & 0 & \gamma I_n \end{bmatrix} \succ 0.$$

Apply the Schur complement:

$$\begin{bmatrix} Y - A^T Y A & -A^T N \delta & \sigma C^T & 0 \\ -\delta N^T A & \gamma I_n - \delta^2 M & 0 & 0 \\ \sigma C & 0 & \gamma & 0 \\ 0 & 0 & 0 & \gamma I_n \end{bmatrix} \succ 0.$$

Finally, eliminate the last row and column and then switch the second rows and columns to obtain (6.8):

$$\begin{bmatrix} Y - A^T Y A & \sigma C^T & -A^T N \delta \\ \sigma C & \gamma & 0 \\ -\delta N^T A & 0 & \gamma I_n - \delta M \delta \end{bmatrix} \succ 0.$$

Now, we focus on the second condition, i.e. $\Gamma^\perp \Theta \Gamma^{\perp T} \succ 0$. The matrix Γ and its left annihilator are defined as:

$$\Gamma := \begin{bmatrix} 0_n & 0_{n \times 1} \\ I_n & 0_{n \times 1} \\ 0_{3n \times n} & 0_{n \times 1} \\ 0_{1 \times n} & 0 \\ 0_{1 \times n} & -\sigma \\ \rho^{-1} I_n & 0_{n \times 1} \end{bmatrix}, \quad \Gamma^\perp := \begin{bmatrix} I_n & 0_n & 0 & 0_n & 0 & 0 & 0 & 0 \\ 0 & 0 & I_n & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & I_n & 0_n & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & I_n & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & I_n & 0 & 0 & 0 & 0 & 0 & -I_n \rho \end{bmatrix} \quad (6.17)$$

Thus, $\Gamma^\perp \Theta \Gamma^{\perp T} \succ 0$ is equivalent to:

$$\begin{bmatrix} X & A & 0 & B & 0 & U \\ A^T & Y & N & 0 & 0 & 0 \\ 0 & N^T & M & 0 & 0 & 0 \\ B^T & 0 & 0 & \gamma I_n & 0 & 0 \\ 0 & 0 & 0 & 0 & \gamma I_n & \delta I_n \\ U^T & 0 & 0 & 0 & \delta I_n & V + \gamma \rho^2 I_n \end{bmatrix} \succ 0.$$

Now, switch rows and columns and apply the Schur complement:

$$\begin{bmatrix} X - AXA^T & B & 0 & U \\ B^T & \gamma & 0 & 0 \\ 0 & 0 & \gamma I & \delta I_n \\ U^T & 0 & \delta I_n & V + \rho^2 \gamma I_n \end{bmatrix} \succ 0$$

Again, switch rows and columns and apply the Schur complement one more time:

$$\begin{bmatrix} X - AXA^T & B & U \\ B^T & \gamma & 0 \\ U^T & 0 & V + (\rho^2 \gamma - \delta^2 \gamma^{-1}) I_n \end{bmatrix} \succ 0.$$

which is exactly (6.9). Proof is complete. ■

Chapter 7

Conclusions and Future work

7.1 Summary

In this dissertation, we presented a systematic framework based on convex optimization of Lyapunov invariants for verification of various critical properties of software systems. The principal elements of this software analysis framework are:

1. Dynamical system interpretation and modeling of computer programs.
2. Lyapunov invariants as certificates for the behavior of computer programs.
3. A computational procedure for finding the Lyapunov invariants.

In Chapter 2 we introduced generic dynamical system representations that formalize our interpretation of numerical computer programs as dynamical systems. We also introduced the Mixed-Integer Linear Models (MILM), the graph models, and the Linear Models with Conditional Switching (LMwCS) as special cases of the generic representations. These modeling languages can represent a broad range of computer programs of interest to the control community and provide a convenient platform for analysis of software via systems and control theoretic tools in an automated or semi-automated framework.

In Chapter 3 we presented several theorems that establish criteria for verification of safety, liveness, and other performance properties of software systems via Lyapunov invariants. The

safety specifications that can be verified in this framework include but are not limited to overflow, out-of-bounds array indexing, division-by-zero, taking the square root or logarithm of a negative number, and various user-defined program assertions. Moreover, when finite-time termination can be guaranteed, the Lyapunov invariants provide an explicit upper bound on the maximum number of iterations.

In Chapter 4 we presented a computational procedure based on linear parametrization of the search space followed by Lagrangian relaxation techniques and convex optimization for computation of the Lyapunov invariants. We showed that sufficient criteria for verification of safety, liveness and certain critical performance properties of computer programs can be formulated as a semidefinite program, a sum-of-squares program, or a linear program. The convex optimization phase is the final step in our software analysis framework. If the optimization problem is feasible, the result is a certificate for safety and/or finite-time termination of the computer program, otherwise, the analysis is inconclusive.

In Chapter 5 we presented several results that complement our software analysis framework. We introduced the concepts of graph reduction, irreducible graphs, and minimal and maximal realizations of graph models. Several new theorems that compare the performance of the original graph model of a computer program and its reduced offsprings were presented. In particular the so-called K1 graphs and their extension, the Kn graphs, were studied in great detail. These results can be used for construction of efficient graph models that systematically improve analysis of computer programs via the framework that has been presented in this dissertation.

In Chapter 6 the framework was further extended to the implementation problem. We introduced an algorithm based on optimization of quadratic Lyapunov functions and semidefinite programming for computation of optimal state space implementations of linear time-invariant systems in digital processors. While respecting the overflow limits, the algorithm minimizes the effects of finite word-length constraints on performance deviation. It was shown that the optimal implementation can be computed by solving a semidefinite optimization problem. It is observed that the optimal state space implementation of a digital filter on a machine with finite memory does not necessarily define the same transfer function as that of an ideal implementation.

7.2 Future Work

The work that we have developed in this dissertation can be extended in several important directions. Herein, we present some ideas that we feel are most interesting for future research.

- **Modular analysis:** Modular analysis is an approach for reducing the computational costs and improving the scalability of the proposed framework as analysis of large size computer programs is undertaken. The idea is to model large size software as the interconnection of smaller size dynamical systems which are referred to as *modules*. These modules interact via a subset of the program variables, namely, the global variables. Modular analysis starts with abstraction of the dynamics of the building blocks of the computer code, that is, the modules, with Input/Output behavioral models. These models typically constitute equalities and/or inequalities relating the input and the output variables. The correctness of each module must be established separately. Correctness of the entire program will be established by verifying safety w.r.t. the global variables, as well as verifying that a terminal global state will be reached in finite-time. This way, the variables that are local to each module are eliminated from the global model, which has the potential to simplify the analysis significantly. Some preliminary results are reported in [81].
- **The complexity tradeoff in graph reduction (symbolic calculations) versus numerical optimization:** The concepts of reduction of graph models and irreducible graphs were introduced and discussed in detail in chapter 5. It was shown that reducing a graph model to an irreducible model is advantageous in the sense that the optimization problem arising from analysis of the reduced graph has fewer decision variables and yet, Lyapunov analysis of the reduced graph is less conservative than the original graph. However, there is a cost associated with the symbolic computations that are performed in the process of graph reduction and building irreducible graph models. A very interesting research direction would be to compare and contrast the added computational costs of the symbolic operations in building irreducible models and the reduced computational costs (due to fewer decision variables) in the convex optimization phase.
- **Perturbation analysis of the Lyapunov Certificates:** The approach presented in this dissertation for taking into account the effects of floating point operations is to include the

roundoff errors in the model as additional noise. The inclusion of roundoff errors as additional noise in the model is similar in nature to the approach taken by many of the existing methods, e.g. abstract interpretation. An interesting alternative to explore would be to assume that the variables are *real* and computations are ideal, and find Lyapunov invariants for the ideal system. A perturbation analysis of the Lyapunov certificates can then determine how much noise the system can tolerate without invalidating the Lyapunov certificates. If the roundoff errors are within acceptable ranges, then the Lyapunov invariant for the nominal system would provide a certificate for the properties of the perturbed system (with floating point computations) as well.

- Extension to systems with software in closed loop with hardware: In this dissertation we focused on verification of computer programs as stand alone dynamical systems. It would be interesting to study extensions of the framework to verification of closed loop systems consisting of the feedback interconnection of analog, continuous-time plants and digital, discrete-time computer programs. Since our framework is built on systems and control theoretic tools, it appears that the framework is readily extendable to such systems; however, this extension does not seem to be as straightforward for some of the existing methods such as abstract interpretation.
- Extension of the implementation results to nonlinear systems: As presented, the approach developed in chapter 6 for optimal software implementation in digital processors is applicable to LTI systems. An interesting and important direction for future research would be to investigate extension of the results to systems involving common nonlinearities such as saturation, time-varying uncertainties, or monotonic odd nonlinearities. It appears that integral quadratic constraints can be exploited for partial or full extension of the results of chapter 6 to such systems.
- Adaptation of the framework to specific classes of software: The framework that is proposed in this dissertation is generic. An interesting direction for future research would be to adapt the framework to specific classes of software, e.g. those of adaptive control systems, or gain-scheduled systems. Adaptation to specific classes of software can improve the efficiency and applicability of the method and further reduce the computational costs.

Bibliography

- [1] M. A. Aizerman, and F. R. Gantmakher. *Absolyutnaya ustoichivost' reguliruemyykh sistem* (Absolute Stability of the Control Systems), Moscow: Akad. Nauk SSSR, 1963.
- [2] R. Alur. *Techniques for Automatic Verification of Real-time Systems*. PhD Thesis, Stanford University, 1991.
- [3] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho X. Nicollin, A. Oliviero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems, *Theoretical Computer Science*, vol. 138, pp. 3–34, 1995.
- [4] R. Alur, T. Dang, and F. Ivancic. Reachability analysis of hybrid systems via predicate abstraction. In *Hybrid Systems: Computation and Control*. Lecture Notes in Computer Science v. 2289, pp. 35–48. Springer Verlag, 2002.
- [5] R. Alur, and G. J. Pappas (Eds.). In *Hybrid Systems: Computation and Control*. Lecture Notes in Computer Science, v. 2993, Springer-Verlag, 2004.
- [6] K. J. Astrom, and B. Wittenmark. *Computer Controlled Systems: Theory and Design*. Prentice Hall Information and System Sciences Series, 1996.
- [7] D. Auslander, J. R. Ridgely, and J. D. Ringgenberg. *Control Software for Mechanical Systems, Object-Oriented Design in a Real-Time World*. Prentice-Hall, 2002.
- [8] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Model-checking algorithms for continuous-time Markov chains. *IEEE Transactions on Software Engineering*, 29(6):524–541, 2003.

- [9] J. S. Bay. *Fundamentals of Linear State Space Systems*. McGraw-Hill, 1999.
- [10] A. Bemporad, D. Mignone, and M. Morari. Moving horizon estimation for hybrid systems and fault detection. In *Proc. American Control Conference*, Pages 2471–2475, 1999.
- [11] A. Bemporad, and M. Morari. Control of systems integrating logic, dynamics, and constraints. *Automatica*, 35(3):407–427, 1999.
- [12] A. Bemporad, F. D. Torrisi, and M. Morari. Optimization-based verification and stability characterization of piecewise affine and hybrid systems. In *Hybrid Systems: Computation and Control*, Lecture Notes in Computer Science v. 1790, pp. 45–58. Springer-Verlag, 2000.
- [13] D. Bertsimas, and J. Tsitsikilis. *Introduction to Linear Optimization*. Athena Scientific, 1997.
- [14] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. Design and implementation of a special-purpose static program analyzer for safety-critical real-time embedded software. In *The Essence of Computation: Complexity, Analysis, Transformation*. Lecture Notes in Computer Science v. 2566, pp. 85–108, Springer-Verlag, 2002.
- [15] J. Bochnak, M. Coste, and M. F. Roy. *Real Algebraic Geometry*. Springer, 1998.
- [16] S. Boyd, L.E. Ghaoui, E. Feron, and V. Balakrishnan. *Linear Matrix Inequalities in Systems and Control Theory*. *Studies in Applied Mathematics*, v. 15, SIAM, 1994.
- [17] M. S. Branicky. Multiple Lyapunov functions and other analysis tools for switched and hybrid systems. *IEEE Transactions on Automatic Control*, 43(4):475–482, 1998.
- [18] M. S. Branicky, V. S. Borkar, and S. K. Mitter. A unified framework for hybrid control: model and optimal control theory. *IEEE Transactions on Automatic Control*, 43(1):31–45, 1998.
- [19] R. W. Brockett. Hybrid models for motion control systems. In *Essays in Control: Perspectives in the Theory and its Applications*, pp 20–53, Birkhauser, 1994.

- [20] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.
- [21] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.
- [22] E. M. Clarke, O. Grumberg, H. Hiraishi, S. Jha, D.E. Long, K.L. McMillan, and L.A. Ness. Verification of the Future-bus+cache coherence protocol. In *Formal Methods in System Design*, 6(2):217–232, 1995.
- [23] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.
- [24] C. Baier, E. M. Clarke, V. Hartonas-Garmhausen, M. Kwiatkowska, and M. Ryan. Symbolic model checking for probabilistic processes. In *Automata, Languages and Programming*. Lecture Notes in Computer Science v. 1256, pp 430–437, Springer-Verlag, 1997.
- [25] M. A. Colon, S. Sankaranarayanan, and H. B. Sipma. Linear invariant generation using non-linear constraint solving. In *Computer Aided Verification*. Lecture Notes in Computer Science, v. 2725, pp. 420-433, Springer-Verlag, 2003.
- [26] P. Cousot, and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the 4th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238–252, 1977.
- [27] P. Cousot, and R. Cousot. Systematic design of program analysis frameworks. In *Conference Record of the 6th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 269–282, 1979.
- [28] P. Cousot. Abstract interpretation based formal methods and future challenges. In *Informatics: 10 Years Back, 10 Years Ahead*. Lecture Notes in Computer Science, v. 2000, pp. 138 – 143, Springer-Verlag, 2001.
- [29] M. A. Dahleh, and I. J. Diaz-Bobillo. *Control of Uncertain Systems: a Linear Programming Approach*. Prentice-Hall, 1995.

- [30] D. Dams. Abstract Interpretation and Partition Refinement for Model Checking. Ph.D. Thesis, Eindhoven University of Technology, 1996.
- [31] E. Feron, and M. Roozbehani. Certifying controls and systems software. In *Proc. of the AIAA Guidance, Navigation and Control Conference*, Hilton Head, South Carolina, August 2007.
- [32] P. Gahinet, and P. Apkarian. A linear matrix inequality approach to \mathcal{H}_∞ control. *International Journal of Robust and Nonlinear Control*, 4(4):421–448, 1994.
- [33] P. Gahinet, A. Nemirovskii, and A. Laub. LMILAB: A Package for Manipulating and Solving LMIs. South Natick, MA: The Mathworks, 1994.
- [34] ILOG Inc. ILOG CPLEX 9.0 User’s guide. Mountain View, CA, 2003.
- [35] A. Girard, and G. J. Pappas. Approximate bisimulation relations for constrained linear systems. *Automatica*, 43(8):1307–1317, 2007.
- [36] A. Girard, A. A. Julius, and G. J. Pappas. Approximate simulation relations for hybrid systems. In *Proc. of the 2nd IFAC Conference on Analysis and Design of Hybrid Systems*, pages 106–111, 2006.
- [37] A. Girard, and G. J. Pappas. Verification using simulation. *Hybrid Systems : Computation and Control*. Lecture Notes in Computer Science, v. 3927, pp. 272–286 , Springer-Verlag, 2006.
- [38] S. Graf, and H. Saidi. Construction of abstract state graphs with PVS. In *Proc. 9th International Conference on Computer Aided Verification*, Lecture Notes in Computer Science v. 1254, pages 72–83. Springer Verlag, 1997.
- [39] M. X. Goemans, and D. P. Williamson, Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the Association for Computing Machinery (ACM)*, 42(6):1115–1145, 1995.
- [40] S. V. Gusev, and A. L. Likhtarnikov. Kalman–Popov–Yakubovich Lemma and the \mathcal{S} -procedure: A historical essay. *Journal of Automation and Remote Control*, 67(11):1768–1810, 2006.

- [41] J. Harper, and A. Megretski. Personal communication, 2000.
- [42] B. S. Heck, L. M. Wills, and G. J. Vachtsevanos. Software technology for implementing reusable, distributed control systems. *IEEE Control Systems Magazine*, 23(1):21–35, 2003.
- [43] M. S. Hecht. *Flow Analysis of Computer Programs*. Elsevier Science, 1977.
- [44] S. Hedlund, and A. Rantzer. Optimal control of hybrid systems. In *Proc. 38th IEEE Conference on Decision and Control*, pages 3972–3977, 1999.
- [45] K. Holmström. The TOMLAB Optimization Environment in Matlab. *Advanced Modeling and Optimization*, 1(1):47–69, 1999.
- [46] M. Johansson, and A. Rantzer. On the computation of piecewise quadratic Lyapunov functions. In *Proc. 36th IEEE Conference on Decision and Control*, pages 3515–3520, 1997.
- [47] M. Johansson, and A. Rantzer. Computation of piecewise quadratic Lyapunov functions for hybrid systems. *IEEE Transactions on Automatic Control*. 43(4):555–559, 1998.
- [48] H. K. Khalil. *Nonlinear Systems*. Prentice Hall, 2002.
- [49] H. Kopetz. *Real-Time Systems Design Principles for Distributed Embedded Applications*. Kluwer, 2001.
- [50] M. Kocvara, M. Stingl. PENBMI User’s Guide, Version 2.1, 2006. A free developer version available at www.penopt.com.
- [51] A. B. Kurzhanski, and I. Valyi. *Ellipsoidal Calculus for Estimation and Control*. Birkhauser, 1996.
- [52] G. Lafferriere, G. J. Pappas, and S. Sastry. Hybrid systems with finite bisimulations. In *Hybrid Systems V. Lecture Notes in Computer Science*, v. 1567, pp. 186–203, Springer-Verlag, 1999.
- [53] G. Lafferriere, G. J. Pappas, and S. Sastry. Reachability analysis of hybrid systems using bisimulations. In *Proc. 37th IEEE Conference on Decision and Control*, pages 1623–1628, 1998.

- [54] G. Lafferriere, G. J. Pappas, and S. Yovine. Symbolic reachability computations for families of linear vector fields. *Journal of Symbolic Computation*, 32(3):231–253, 2001.
- [55] M. Laurent. Tighter linear and semidefinite relaxations for max-cut based on the Lovász–Schrijver Lift-and-Project procedure. *SIAM Journal on Optimization*, 12(2):345–375, 2002.
- [56] J. Löfberg. YALMIP : A Toolbox for Modeling and Optimization in MATLAB. In Proc. of the CACSD Conference, 2004. URL: <http://control.ee.ethz.ch/~joloef/yalmip.php>
- [57] L. Lovasz, and A. Schrijver. Cones of matrices and set-functions and 0-1 optimization. *SIAM Journal on Optimization*, 1(2):166–190, 1991.
- [58] J. Lygeros, C. Tomlin, and S. Sastry. Controllers for reachability specifications for hybrid systems. *Automatica*, 35(3):349–370, 1999.
- [59] A. Makhorin. GLPK (GNU Linear Programming Kit). Available at <http://www.gnu.org/software/glpk/glpk.html>.
- [60] Z. Manna, and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, 1992.
- [61] Z. Manna, and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer-Verlag, 1995.
- [62] W. Marrero, E. Clarke, and S. Jha. Model checking for security protocols. In *Proc. DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997. Preliminary version appears as Technical Report TR–CMU–CS–97–139, Carnegie Mellon University, May 1997.
- [63] A. Megretski. Relaxations of quadratic programs in operator theory and system analysis. In *Systems, Approximation, Singular Integral Operators, and Related Topics* (Bordeaux, 2000). Operator Theory: Advances and Applications, v. 129, pp. 365–392. Birkhauser-Verlag, 2001.
- [64] A. Megretski. Positivity of trigonometric polynomials. In *Proc. 42nd IEEE Conference on Decision and Control*, pages 3814–3817, 2003.

- [65] R. Milner. A theory of type polymorphism in programming. *Journal of Computer and System Sciences*, 17(3):48–375, 1978.
- [66] A. Mine. *Weakly Relational Numerical Abstract Domains*. Ph.D. Thesis. 'Ecole Normale Sup'erieure, December 2004.
- [67] I. Mitchell, A. Bayen, and C. Tomlin. Validating a Hamilton-Jacobi approximation to hybrid system reachable sets. In *Hybrid Systems: Computation and Control*. Lecture Notes in Computer Science v. 2034, pp. 418-432, Springer-Verlag, 2001.
- [68] S. Mitra. *A Verification Framework for Hybrid Systems*. Ph.D. Thesis. Massachusetts Institute of Technology, September 2007.
- [69] C. S. R. Murthy, and G. Manimaran. *Resource Management in Real-Time Systems and Networks*. MIT Press, 2001.
- [70] G. Naumovich, L. A. Clarke, and L. J. Osterweil. Verification of communication protocols using data flow analysis. In *Proc. 4-th ACM SIGSOFT Symposium on the Foundation of Software Engineering*, pages 93–105, 1996.
- [71] Y.E. Nesterov, H. Wolkowicz, and Y. Ye. Semidefinite programming relaxations of non-convex quadratic optimization. In *Handbook of Semidefinite Programming: Theory, Algorithms, and Applications*. Dordrecht, Kluwer Academic Press, pp. 361–419, 2000.
- [72] F. Nielson, H. Nielson, and C. Hank. *Principles of Program Analysis*. Springer, 2004.
- [73] P. A. Parrilo. Minimizing polynomial functions. In *Algorithmic and Quantitative Real Algebraic Geometry*. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, v. 60, pp. 83-100, American Mathematical Society, 2003.
- [74] P. A. Parrilo. *Structured Semidefinite Programs and Semialgebraic Geometry Methods in Robustness and Optimization*. Ph.D. Thesis, California Institute of Technology, 2000.
- [75] D. A. Peled. *Software Reliability Methods*. Springer-Verlag, 2001.
- [76] B. C. Pierce. *Types and Programming Languages*. MIT Press, 2002.

- [77] S. Prajna. *Optimization-Based Methods for Nonlinear and Hybrid Systems Verification*. Ph.D. Thesis, California Institute of Technology, 2005.
- [78] S. Prajna, and A. Jadbabaie. Safety verification of hybrid systems using barrier certificates. In *Hybrid Systems: Computation and Control*. Lecture Notes in Computer Science, v. 2993, pp. 477–492, Springer-Verlag, 2004.
- [79] S. Prajna, A. Papachristodoulou, P. Seiler, and P. A. Parrilo, SOSTOOLS: Sum of squares optimization toolbox for MATLAB, 2004. <http://www.mit.edu/~parrilo/sostools>.
- [80] S. Prajna, and A. Rantzer, Convex programs for temporal verification of nonlinear dynamical systems, *SIAM Journal on Control and Optimization*, 46(3):999–1021, 2007.
- [81] M. Roozbehani, É. Feron, and A. Megretski. Modeling, optimization and computation for software verification. In *Hybrid Systems: Computation and Control*. Lecture Notes in Computer Science, v. 3414, pp. 606–622, Springer-Verlag 2005.
- [82] M. Roozbehani, A. Megretski, E. Feron. Safety verification of iterative algorithms over polynomial vector fields. In *Proc. 45th IEEE Conference on Decision and Control*, pages 6061–6067, 2006.
- [83] M. A. Rotea, and D. Williamson. Optimal realizations of finite wordlength digital filters and controllers. *IEEE Transactions on Circuits and Systems*, 42(2):61–72, 1995.
- [84] H. D. Sherali, and W. P. Adams. A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems. *SIAM Journal on Discrete Mathematics*, 3(3):411–430, 1990.
- [85] H. D. Sherali, and W. P. Adams. A hierarchy of relaxations and convex hull characterizations for mixed-integer zero-one programming problems. *Discrete Applied Mathematics*, 52(1):83–106, 1994.
- [86] J. F. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11–12:625–653, 1999. URL: <http://sedumi.mcmaster.ca>

- [87] A. M. van Tilborg, and G. M. Koob. *Foundations of Real-Time Computing: Scheduling and Resource Management*. Kluwer, 1991.
- [88] A. Tiwari. Approximate reachability for linear systems. In *Hybrid Systems: Computation and Control*, Lecture Notes in Computer Science, v. 2623, pp. 514–525. Springer Verlag, 2003.
- [89] A. Tiwari, and G. Khanna. Series of abstractions for hybrid automata. In *Hybrid Systems: Computation and Control*, Lecture Notes in Computer Science, v. 2289, pp. 465–478. Springer Verlag, 2002.
- [90] K. C. Toh, R. H. Tutuncu, and M. J. Todd. SDPT3 - a MATLAB software package for semidefinite-quadratic-linear programming. URL: <http://www.math.nus.edu.sg/~mattokc/sdpt3.html>.
- [91] L. Vandenberghe, and S. Boyd. Semidefinite programming. *SIAM Review*, 38(1):49–95, 1996.
- [92] V. A. Yakubovic. S-procedure in nonlinear control theory. *Vestnik Leningrad University*, 4(1):73–93, 1977. English translation; original Russian publication in *Vestnik Leningradskogo Universiteta, Seriya Matematika*, pp. 62–77, Leningrad, Russia, 1971.
- [93] H. Yazarel, and G. Pappas. Geometric programming relaxations for linear systems reachability. In *Proc. American Control Conference*, pages 553–559, 2004.
- [94] ASTRÉE: Static Software Analyzer. <http://www.astree.ens.fr/>
- [95] BLAST: Berkeley Lazy Abstraction Software Verification Tool. <http://mtc.epfl.ch/software-tools/blast/>
- [96] NuSMV: Symbolic Model Checker. <http://nusmv.irst.itc.it/>
- [97] Real-Time Workshop, MATLAB Toolbox, The Mathworks Inc., Natick, MA.
- [98] SPIN: Model Checking Software Tool. <http://spinroot.com/spin/whatispin.html>
- [99] Texas Instruments TMS320C5x User's Guide (Rev. D). Available at <http://focus.ti.com/general/docs/techdocsabstract.tsp?abstractName=spru056d>