Contents

I Need and Tools to Verify Critical Cyber-Physical Systems 1

1 Critical Embedded Software: Control Software Development and V&V 3

2 Formal Methods: Different Approaches for Verification 7
   2.1 Semantics and Properties 7
   2.2 A Formal Verification Methods Overview 11
   2.3 Deductive Methods 19
   2.4 SMT-based Model-checking 21
   2.5 Abstract Interpretation (of Collecting Semantics) 23
   2.6 Need for Inductive Invariants 29

3 Control Systems 31
   3.1 Controllers’ Development Process 31
   3.2 A Simple Linear System: Spring-mass Damper 35

II Invariant Synthesis: Convex-optimization Based Abstract Interpretation 41

4 Definitions–Background 43
   4.1 Discrete Dynamical Systems 43
   4.2 Elements of (Applied) Convex Optimization 54

5 Invariants Synthesis via Convex Optimization: Postfixpoint Computation as Semialgebraic Constraints 64
   5.1 Invariants, Lyapunov Functions, and Convex Optimization 64
   5.2 Quadratic Invariants 68
   5.3 Piecewise Quadratic Invariants 76
   5.4 \( k \)-inductive Quadratic Invariants 87

For general queries, contact webmaster@press.princeton.edu
CONTENTS

5.5 Polynomial Invariants 95
5.6 Image Measure Method 103
5.7 Related Works 108

6 Template-based Analyses and Min-policy Iteration 111
6.1 Template-based Abstract Domains 111
6.2 Template Abstraction Fixpoint as an Optimization Problem 112
6.3 SOS-relaxed Semantics 114
6.4 Example 122
6.5 Related Works 124

III System-level Analysis at Model and Code Level 127

7 System-level Properties as Numerical Invariants 129
7.1 Open-loop and Closed-loop Stability 130
7.2 Robustness with Vector Margin 139
7.3 Related Work 145

8 Validation of System-level Properties at Code Level 147
8.1 Axiomatic Semantics of Control Properties through Synchronous Observers and Hoare Triples 147
8.2 Generating Annotations: A Strongest Postcondition Propagation Algorithm 155
8.3 Discharging Proof Objectives using PVS 159

IV Numerical Issues 165

9 Floating-point Semantics of Analyzed Programs 167
9.1 Floating-point Semantics 167
9.2 Revisiting Inductiveness Constraints 170
9.3 Bound Floating-point Errors: Taylor-based Abstractions aka Zonotopic Abstract Domains 173
9.4 Related Works 190

10 Convex Optimization and Numerical Issues 191
10.1 Convex Optimization Algorithms 191
10.2 Guaranteed Feasible Solutions with Floats 196

Bibliography 201
Index 217
Acknowledgments 220
Chapter One

Critical Embedded Software

Control Software Development and V&V

Cyber-physical systems (CPS) is a kind of buzzword capturing the set of physical devices controlled by an onboard computer, an embedded system. Critical embedded systems are a subset of these for which failure is not acceptable. Typically this covers transportation systems such as cars, aircraft, railway systems, space systems, or even medical devices, all of them either for the expected harmfulness for people, or for the huge cost associated with their failure.

A large part of these systems are controllers. They are built as a large running loop which reads sensor values, computes a feedback, and applies it to the controlled system through actuators. For most systems, at least in the aerospace industry, the time schedule for controllers is so tight that these systems have to be “real time.” The way these systems have been designed requires the execution of the loop body to be performed within some time to maintain the system in a reasonable state. In the civil aircraft industry, the controller itself is rather complex, but is built as a composition of simpler controllers. Furthermore, the global system accounts for potential failures of components: sensors, networks, computers, actuators, etc., and adapts the control to these discrepancies.

The increase of computer use in those systems has led to huge benefits but also an exponential growth in complexity. Computer based systems compared to analog circuits enable more efficient behaviors, as well as size and weight reductions. For example, aircraft manufacturers are building control laws for their aircraft that maintain them at the limit of instability, allowing more fuel efficient behavior;\(^1\) Rockwell Collins implemented a controller for a fighter aircraft able to recover controllability when the aircraft loses, in flight, from 60 to 80% of one of its wings;\(^2\) United Technology has been able to replace huge and heavy

---

\(^1\) In an A380, fuel is transferred between tanks to move the center of gravity to the aft (backward). This degrades natural stability but reduces the need for lift surfaces and therefore improves fuel efficiency by minimizing total weight and drag. See the book *Airbus A380: Superjumbo of the 21st Century* by Noris and Wagner [58].

\(^2\) Search for Damage Tolerance Flight Test video, e.g., at https://www.youtube.com/watch?v=PTMpq_8SSCI
power electric systems with their electronic counterpart, with a huge reduction in size and weight.\(^3\)

The drawback of this massive introduction of computers to control systems is the lack of predictability for both computer and software. While the industry has been accustomed to having access to the precise characteristic of its components, e.g., a failure rate for a physical device running in some specific conditions, these figures are hardly computable for software, because of the intrinsic complexity of computer programs.

Still, all of us are now used to accepting software licenses where the software vendor assumes nothing related to the use of the software and its possible impact. These kinds of licenses would be unacceptable for any other industry.

To conclude with this brief motivation, the aerospace industry, and more generally critical embedded systems industries, is are now facing a huge increase in the software size in their systems. This is motivated first by system complexity increases because of safety or performance objectives, but also by the need to integrate even more advanced algorithms to sustain autonomy and energy efficiency.

Guaranteeing the good behavior of those systems is essential to enable their use.

Until now, classical means to guarantee good behavior were mainly relying on tests. In the aerospace industry the development process is strictly constrained by norms such as the DO-178C [104] specifying how to design software and perform its verification and validation (V&V). This document shapes the V&V activities and requires the verification to be specification-driven. For each requirement expressed in the design phases, a set of tests has to be produced to argue that the requirement is satisfied. However, because of the increase in complexity of the current and future systems, these test-based verifications are reaching their limit. As a result the cost of V&V for systems has exploded and the later a bug is found, the more expensive it is to solve.\(^4\)

Last, these certification documents such as DO-178C have been recently updated, accounting for the recent applicability of formal methods to argue about the verification of a requirement. Despite their possible lack of results in a general setting, these techniques, in cases of success, provide an exhaustive result, i.e., they guarantee that the property considered is valid for all uses, including systems admitting infinite behaviors.

All the works presented in this book are motivated by this context. We present formal methods sustaining the verification of controller properties at multiple stages of their development. The goal is to define new means of verification, specific to controller analysis.

\(^3\)E.g., Active EMI filtering for inverters used at Pratt and Whitney, Patent US20140043871.

\(^4\)USA NIST released in 2002 an interesting survey, “The Economic Impacts of Inadequate Infrastructure for Software Testing,” detailing the various costs of verification and bugs. Chapter 6 is focused on the transportation industry.
Current limits & Objectives

The objectives of the presented works are restricted to the definition of formal methods-based analyses to support the verification of controller programs. 

More specifically we can identify the following limits in the current state of the art:

Need to compute invariants of dynamical systems  New advances in formal methods are often not specialized for a particular kind of program. They rather try to handle a large set of programming language constructs and deal with scalability issues. In specific cases, such as the application of static analysis to Airbus programs [54], dedicated analyses, like the second-order filter abstraction [47], have been defined. But the definition of these domains is tailored to the program for which they are defined.

Lack of means to compute nonlinear invariants  As we will see in this book, the simplest properties of controllers are often based on at least quadratic properties. Again, because of efficiency and scalability, most analyses are bound to linear properties. We claim that more expressive yet more costly analyses are required in specific settings such as the analysis of control software. The scalability issues have to be addressed by carefully identifying the local part of the program on which to apply these more costly analyses.

Expressivity of static analysis properties  Formal methods applied at model or code level are hardly used to express or analyze system-level properties. In practice, static analysis is mainly bound to numerical invariants while deductive methods or model-checking can manipulate more expressive first-order logic formulas. However, computer scientists are usually not aware of the system-level properties satisfied or to be satisfied by the control program they are analyzing. An important research topic is therefore the use of these formalisms (first-order logic and numerical invariants) to express and analyze system-level properties.

Scope of current analyses  In the current state of the practice, concerns are split and analyzed locally. For example the control-level properties such as stability are usually analyzed by linearizing the plant and the controller description. At the code level this can be compared to the analysis of a simplified program without if-then-else or nonlinear computations. Similarly, the complete fault-tolerant architecture, which is part of the implemented embedded program, is abstracted away when analyzing system-level properties. A last example of such—potentially unsound—simplifications, is the assumption of a real semantics when performing analyses, while the actual implementation will be executed with floating-point semantics and the associated errors. The vision supported by the book is that more integrated analyses should address the study of the global system.
CHAPTER 1

The proposal is mainly developed in two complementary directions:

- nonlinear invariant synthesis mainly based on the use of convex optimization techniques;
- consideration of system-level properties on discrete representation, at code level, with floating-point semantics.

This book is structured in four parts:

Part I introduces formal methods and controller design. It is intended to be readable both by a control scientist unaware of formal methods, and by a computer scientist unaware of controller design. References are provided for more scholastic presentations.

Part II focuses on invariant synthesis for discrete dynamical systems, assuming a real semantics. All techniques are based on the computation of an inductive invariant as the resolution of a convex optimization problem.

Part III revisits basic control-level properties as numerical invariants. These properties are typically expressed on the so-called closed-loop representation. In these chapters we assume that the system description is provided as a discrete dynamical system, without considering its continuous representation with ordinary differential equations (ODEs).

Part IV extends the previous contributions by considering floating-point computations. A first part considers that the program analyzed is executed with floating-point semantics and searches for an inductive invariant considering the numerical errors produced. A second part ensures that the use of convex optimization, a numerical technique, does not suffer from similar floating-point errors.
Index

abstract
domain, 25
interpretation, 11, 16, 23
template abstraction, 67
affine
arithmetics, 179
constraint, 48
alarm, 25
boundedness, 93, 96, 98
completeness, 11
incompleteness, 25
condition number, 69
conditional, 14
conservative techniques, 12
constraint
linear, 48, 55, 80, 199
quadratic, 76
contract, 10
control flow graph, 8
convexification, 136
coupling conditions, 82
decay rate, 70, 138
deductive method, 11, 13
Dijkstra, 20, 29
predicate transformation, 20
duality, 61, 62
dynamics
continuous, 35
discrete, 36, 37, 65
fixpoint, 25, 27, 28, 30, 95
classification, 65
floating-point
error, 173
guard, 175
flowchart, 9
Floyd, 9
Hoare, 9
model, 20
triple, 1, 2, 10, 13, 14
homogenization, 48, 77
IEEE 754, 171
image measure, 103
induction
k-induction, 88
k-inductive invariant, 87
inductiveness, 95
interior point method, 198
invariant
inductive invariant, 29, 30
local invariant, 82
loop invariant, 6, 29
difference invariant, 96
sublevel invariant, 96
Kleene, 28, 43, 68, 109, 124, 126, 177, 180
Lagrangian relaxation, 58, 62
SOS relaxation, 60, 114
Lasserre’s hierarchy, 106
linear matrix inequality (LMI), 138
condition number, 69
convexification, 136
encoding, 82
floating-point issues, 204
Lyapunov function, 68
positive semidefinite cone, 56
vector margins, 143

For general queries, contact webmaster@press.princeton.edu
Liouville’s equation, 105
floating-point
assignment, 175
local positivity, 79
loop, 19
Lyapunov function, 66, 131, 132
  k-inductive function, 88
  piecewise quadratic function, 82
  polynomial function, 95
  quadratic function, 68
  semialgebraic function, 67
SOS, 57
margins
  phase and gain, 141
  vector, 139, 142
memory model, 20
model-checking, 11, 16, 21, 29
Motzkin’s transposition, 80
narrowing, 118, 124, 146
optimization, 54
  convex, 54
  convex conic, 54
  duality, 62
  guaranteed feasible solution, 202
over-approximation, 12, 16
policy iteration, 117, 175
  max-policy iteration, 125
  strategy iteration, 124
  Sum-of-square extension, 121
polyhedral partitioning, 47
polytope, 49, 56, 180, 197, 199
positive semidefinite
  cone, 56
postcondition, 10
  strongest postcondition, 9
precondition, 10
  weakest precondition, 13
predicate
  ACSL, 6
  encoding, 21
  modulo theory, 15, 16
PVS, 3
  transformation, 20
  transformer, 13, 29
property, 101
  selection property, 118
  system-level property, 129
propositional encoding, 16
quadratization, 77, 83
reachable states, 9, 16
recursion, 19
region
  bad, 96
  Rice’s theorem, 11
S-procedure, 13, 58, 73, 82, 136
safety, 96
  satisfiability, 16
  saturation, 40, 133
semantics
  axiomatic, 1, 9
  closed-loop, 8
  collecting, 9, 16, 23, 65
  denotational, 8
  operational, 8
  trace semantics, 9
semialgebraic
  partitioning, 52
  set, 51
soundness, 11, 25, 26
stability, 4, 129, 130
  closed-loop, 131, 133
  robustness, 139
strongest postcondition, 13
sublevel set, 82
  property, 95
sum-of-square (SOS)
  polynomial, 57
  relaxation, 97
switching condition, 79
synchronous observer, 1
system
  closed-loop, 37
  linear, 45

For general queries, contact webmaster@press.princeton.edu
<table>
<thead>
<tr>
<th>Term</th>
<th>Page(s)</th>
<th>Term</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>piecewise affine</td>
<td>47, 76</td>
<td>relation</td>
<td>29</td>
</tr>
<tr>
<td>piecewise polynomial</td>
<td>51</td>
<td>system</td>
<td>9</td>
</tr>
<tr>
<td>switched linear</td>
<td>47</td>
<td>undecidability</td>
<td>11</td>
</tr>
<tr>
<td>template</td>
<td>67, 132</td>
<td>volume</td>
<td>103</td>
</tr>
<tr>
<td>abstraction</td>
<td>111</td>
<td>widening</td>
<td>28, 124, 125</td>
</tr>
<tr>
<td>shape</td>
<td>70</td>
<td>zonotope</td>
<td>177</td>
</tr>
<tr>
<td>termination</td>
<td>9, 11, 12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>theory interpretation</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>transition</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>