Convex Optimization-based Static Analysis for Control Systems

Pierre-Loïc Garoche

Habilitation à diriger des recherches

de l'INSTITUT NATIONAL POLYTECHNIQUE DE TOULOUSE

Pierre-Loïc Garoche

September 19th 2016

Convex Optimization-based Static Analysis for Control Systems

---

Rapporteurs :

Éric Goubault
Professeur au LIX à l'École Polytechnique

Ilya Kolmanovsky
Professor of Aerospace Engineering at Michigan University, USA

David Monniaux
Directeur de recherches au CNRS, Laboratoire Vérimag

Examinateurs :

Behçet Açikmeşe
Associate Professor of Aeronautics & Astronautics at University of Washington, USA

Éric Féron
Professor of Aerospace Engineering at Georgia Tech, USA

John Hauser
Associate Professor, Dept. of Electrical and Computer Engineering at University of Colorado Boulder, USA

Didier Henrion
Directeur de recherches au LAAS-CNRS, Université de Toulouse

Matthieu Martel
Professeur à l'Université de Perpignan Via Domitia, Laboratoire de Mathématiques et de Physique (LAMPS)

Philippe Queinnec
Professeur à l'INPT/IRIT, Université de Toulouse
correspondant INPT

PUBLICATIONS & PROTOTYPES

Some ideas and figures have appeared previously in the following publications:

[RDG10] Pierre Roux, Rémi Delmas, and Pierre-Loïc Garoche. "SMT-AI: an Abstract Interpreter for a Synchronous Extension of SMT-lib." In: *1st International Workshop on Tools for Automatic Program AnalysiS (TAPAS 2010), SAS'10 satellite event, Perpignan, France*. Ed. by David Delmas and Xavier Rival. Vol. 267. 2. Elsevier Electr. Notes Theor. Comput. Sci., Sept. 2010, pp. 55–68. DOI: 10.1016/j.entcs.2010.09.018.

[Cha+11] Adrien Champion, Rémi Delmas, Pierre-Loïc Garoche, and Pierre Roux. "Towards Cooperation of Formal Methods for the Analysis of Critical Control Systems." In: *SAE International Journal of Aerospace* 4.2 (Nov. 2011). **Arch T. Colwell Merit Award**, pp. 850–858. DOI: 10.4271/2011-01-2558.

[Her+12] Heber Herencia-Zapana, Romain Jobredeaux, Sam Owre, Pierre-Loïc Garoche, Éric Féron, Gilberto Perez, and Pablo Ascariz. "PVS Linear Algebra Libraries for Verification of Control Software Algorithms in C/ACSL." In: *NASA Formal Methods - Forth International Symposium, NFM 2012, Norfolk, VA USA, April 3-5, 2012. Proceedings*. Ed. by Alwyn Goodloe and Suzette Person. Vol. 7226. Lecture Notes in Computer Science. Springer, 2012, pp. 147–161. DOI: 10.1007/978-3-642-28891-3_15.

[Kah+12] Temesghen Kahsai, Pierre-Loïc Garoche, Cesare Tinelli, and Mike Whalen. "Incremental verification with mode variable invariants in state machines." In: *NASA Formal Methods - Forth International Symposium, NFM 2012, Norfolk, VA USA, April 3-5, 2012. Proceedings*. Ed. by Alwyn Goodloe and Suzette Person. Vol. 7226. Lecture Notes in Computer Science. Springer, 2012, pp. 388–402. DOI: 10.1007/978-3-642-28891-3_35.

[Rou+12] Pierre Roux, Romain Jobredeaux, Pierre-Loïc Garoche, and Éric Féron. "A Generic Ellipsoid Abstract Domain for Linear Time Invariant Systems." In: *Proceedings of the 15th ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2012, Beijing, China, April 17-19, 2012*, ed. by Thao Dang and Ian Mitchell. ACM, 2012, pp. 105–114. ISBN: 978-1-4503-1220-2. DOI: 10.1145/2185632.2185651. URL: doi.acm.org/10.1145/2185632.2185651.

[Wie+12] Virginie Wiels, Rémi Delmas, David Doose, Pierre-Loïc Garoche, Jacques Cazin, and Guy Durrieu. "Formal Verification of Critical Aerospace Software." In: *Aerospace Lab Journal* 4 (May 2012). URL: www.aerospacelab-journal.org/al4/.

[Cha+13a] Adrien Champion, Rémi Delmas, Michael Dierkes, Pierre-Loïc Garoche, Romain Jobredeaux, and Pierre Roux. "Formal Methods for the Analysis of Critical Control Systems Models: Combining Non-Linear and Linear Analyses." In: *Formal Methods for Industrial Critical Systems (FMICS'13)*. Ed. by Charles Pecheur and Michael Dierkes. Vol. 8187. Lecture Notes in Computer Science. **Best paper award**. Springer, 2013, pp. 1–16. ISBN: 978-3-642-41009-3. DOI: 10.1007/978-3-642-41010-9_1.

[Cha+13b] Adrien Champion, Rémi Delmas, Michael Dierkes, Pierre-Loïc Garoche, Romain Jobredeaux, and Pierre Roux. "Formal Methods for the Analysis of Critical Control Systems Models: Combining Non-Linear and Linear Analyses." In: *SAE International Journal of Aerospace* 6.1 (2013), pp. 150–160. DOI: 10.4271/2013-01-2109.

[GKT13] Pierre-Loïc Garoche, Temesghen Kahsai, and Cesare Tinelli. "Incremental Invariant Generation Using Logic-Based Automatic Abstract Transformers." In: *NASA Formal Methods - Fifth International Symposium, NFM 2013, Moffett Field, CA USA, May 14-16, 2013. Proceedings*. Ed. by Guillaume Brat, Neha Rungta, and Arnaud Venet. Vol. 7871. Lecture Notes in Computer Science. Springer, 2013, pp. 139–154. ISBN: 978-3-642-38087-7. DOI: 10.1007/978-3-642-38088-4_10.

[RG13a] Pierre Roux and Pierre-Loïc Garoche. "A Polynomial Template Abstract Domain based on Bernstein Polynomials." In: *Numerical Software Verification*. 2013.

[RG13b] Pierre Roux and Pierre-Loïc Garoche. "Integrating Policy Iterations in Abstract Interpreters." In: *Automated Technology for Verification and Analysis - 11th International Symposium, ATVA 2013, Hanoi, Vietnam, October 15-18, 2013. Proceedings*. Ed. by Dang Van Hung and Mizuhito Ogawa. Vol. 8172. Lecture Notes in Computer Science. Springer, 2013, pp. 240–254. ISBN: 978-3-319-02443-1. DOI: 10.1007/978-3-319-02444-8_18.

[AGW14] Assalé Adjé, Pierre-Loïc Garoche, and Alexis Werey. *Quadratic Zonotopes: An extension of Zonotopes to Quadratic Arithmetics*. 2014. URL: arxiv.org/abs/1411.5847.

[GGK14] Pierre-Loïc Garoche, Arie Gurfinkel, and Temesghen Kahsai. "Synthesizing Modular Invariants for Synchronous Code." In: *Proceedings of the First Workshop on Horn Clauses for Verification and Synthesis, HCVS 2014, Vienna, Austria, 17 July 2014*. Ed. by Nikolaj Bjørner, Fabio Fioravanti, Andrey Rybalchenko, and Valerio Senni. Vol. 169. EPTCS. 2014, pp. 19–30. DOI: 10.4204/EPTCS.169.4.

[Gar+14] Pierre-Loïc Garoche, Falk Howar, Temesghen Kahsai, and Xavier Thirioux. "Testing-Based Compiler Validation for Synchronous Languages." In: *NASA Formal Methods - 6th International Symposium, NFM 2014, Houston, TX, USA, April 29 - May 1, 2014. Proceedings*. Ed. by Julia M. Badger and Kristin Yvonne Rozier. Vol. 8430. Lecture Notes in Computer Science. Short paper. Springer, 2014, pp. 246–251. ISBN: 978-3-319-06199-3. DOI: 10.1007/978-3-319-06200-6_19.

[RG14] Pierre Roux and Pierre-Loïc Garoche. "Computing Quadratic Invariants with Min- and Max-Policy Iterations: A Practical Comparison." In: *FM 2014: Formal Methods - 19th International Symposium, Singapore, May 12-16, 2014. Proceedings*. Ed. by Cliff B. Jones, Pekka Pihlajasaari, and Jun Sun. Vol. 8442. Lecture Notes in Computer Science. Springer, 2014, pp. 563–578. ISBN: 978-3-319-06409-3. DOI: 10.1007/978-3-319-06410-9_38.

[AG15a] Assalé Adjé and Pierre-Loïc Garoche. "Automatic synthesis of k-inductive piecewise quadratic invariants for switched affine control programs." In: *Computer Languages, Systems & Structures (COMLAN)* (2015). ISSN: 1477-8424. DOI: 10.1016/j.cl.2015.12.002. URL: www.sciencedirect.com/science/article/pii/S1477842415000937.

[AG15b] Assalé Adjé and Pierre-Loïc Garoche. "Automatic Synthesis of Piecewise Linear Quadratic Invariants for Programs." In: *Verification, Model Checking, and Abstract Interpretation - 16th International Conference, VMCAI 2015, Mumbai, India, January 12-14, 2015. Proceedings*. 2015, pp. 99–116. DOI: 10.1007/978-3-662-46081-8_6.

[AGM15a] Assalé Adjé, Pierre-Loïc Garoche, and Victor Magron. *A Sums-of-Squares Extension of Policy Iterations*. 2015. URL: arxiv.org/abs/1503.08090.

[AGM15b] Assalé Adjé, Pierre-Loïc Garoche, and Victor Magron. "Property-based Polynomial Invariant Generation using Sums-of-Squares Optimization." In: *Static Analysis - 22nd International Symposium, SAS 2015, St Malo, France, 2015. Proceedings*. Ed. by Sandrine Blazy and Thomas Jensen. Vol. 9291. Lecture Notes in Computer Science. Springer, 2015, pp. 235–251. ISBN: 978-3-662-48287-2. DOI: 10.1007/978-3-662-48288-9_14.

[AGW15] Assalé Adjé, Pierre-Loïc Garoche, and Alexis Werey. "Quadratic Zonotopes - An Extension of Zonotopes to Quadratic Arithmetics." In: *Programming Languages and Systems - 13th Asian Symposium, APLAS 2015, Pohang, South Korea, November 30 - December 2, 2015, Proceedings*. 2015, pp. 127–145. DOI: 10.1007/978-3-319-26529-2_8.

[Die+15] Arnaud Dieumegard, Pierre-Loïc Garoche, Temesghen Kahsai, Alice Tailliar, and Xavier Thirioux. "Compilation Of Synchronous Observers As Code Contracts." In: *30th ACM/SIGAPP Symposium on Applied Computing, SAC 2015, Salamanca, Spain - April 13 - 17, 2015*. Ed. by Roger L. Wainwright, Juan Manuel Corchado, Alessio Bechini, and Jiman Hong. Short paper. ACM, 2015, pp. 1933–1939. ISBN: 978-1-4503-3196-8. DOI: 10.1145/2695664.2695819. URL: doi.acm.org/10.1145/2695664.2695819.

[RG15] Pierre Roux and Pierre-Loïc Garoche. "Practical Policy Iterations A practical use of policy iterations for static analysis - The quadratic case." In: *Formal Methods in System Design* 46.2 (2015), pp. 163–196. DOI: 10.1007/s10703-015-0230-7.

[RJG15] Pierre Roux, Romain Jobredeaux, and Pierre-Loïc Garoche. "Closed Loop Analysis of Control Command Software." In: *18th International Conference on Hybrid Systems: Computation and Control (part of CPS Week), HSCC'15, Seattle, Washington, USA, April 14-16, 2015*, ed. by Antoine Girard and Sriram Sankaranarayanan. 2015, pp. 108–117. ISBN: 978-1-4503-3433-4. DOI: 10.1145/2728606.2728623. URL: doi.acm.org/10.1145/2728606.2728623.

[KTG16] Temesghen Kahsai, Xavier Thirioux, and Pierre-Loïc Garoche. "Hierarchical state machines as modular Horn clauses." In: *Proceedings of the Second Workshop on Horn Clauses for Verification and Synthesis, HCVS 2016, Eindhoven, The Netherlands, April 3rd 2016*. 2016.

[Wan+16a] Timothy Wang, Pierre-Loïc Garoche, Pierre Roux, Romain Jobredeaux, and Éric Féron. "Formal Analysis of Robustness at Model and Code Level." In: *19th International Conference on Hybrid Systems: Computation and Control (part of CPS Week), HSCC'16, Vienna, Austria, April 12-14, 2015, to appear*. 2016.

[Wan+16b] Timothy Wang, Romain Jobredeaux, Heber Herencia-Zapana, Pierre-Loïc Garoche, Arnaud Dieumegard, Éric Féron, and Marc Pantel. "From Design to Implementation: An Automated, Credible Autocoding Chain for Control Systems." In: *Advances in Control System Technology for Aerospace Applications*. Ed. by Éric Féron. Vol. 460. Lecture Notes in Control and Information Sciences. Springer Berlin Heidelberg, 2016, pp. 137–180. ISBN: 978-3-662-47693-2. DOI: 10.1007/978-3-662-47694-9_5.

[Wan+16c] Timothy Wang, Romain Jobredeaux, Marc Pantel, Pierre-Loïc Garoche, Éric Féron, and Didier Henrion. "Credible Autocoding of Convex Optimization Algorithms." In: *Optimization and Engineering* (2016). Ed. by Springer. to appear. URL: arxiv.org/abs/1403.1861.

In the following prototypes:

[GT11] Pierre-Loïc Garoche and Xavier Thirioux. *YASA: Yet Another Static Analyzer*. 2006–2011.

[GR11] Pierre-Loïc Garoche and Pierre Roux. *SMT-AI, abstract interpreter for a temporal extension of SMT-lib*. 2011–. URL: https://cavale.enseeiht.fr/smt-ai/.

[GKT12a] Pierre-Loïc Garoche, Temesghen Kasai, and Cesare Tinelli. *Kind-AI, automatic abstract interpreter module for Lustre model-checker Kind*. 2012. URL: clc.cs.uiowa.edu/Kind/NFM13/.

[GKT12b] Pierre-Loïc Garoche, Temesghen Kasai, and Cesare Tinelli. *Kind, a k-induction based model-checker*. 2012–. URL: clc.cs.uiowa.edu/Kind/.

[GTK12] Pierre-Loïc Garoche, Xavier Thirioux, and Temesghen Kahsai. *LustreC: a modular Lustre compiler*. 2012–. URL: https://github.com/coco-team/lustrec.

[RGW12] Pierre Roux, Pierre-Loïc Garoche, and Alexis Werey. *TINY: Simple Static Analyzer for Imperative Code and Numerical precision analysis*. 2012–. URL: https://cavale.enseeiht.fr/QuadZonotopes/.

[RG14] Pierre Roux and Pierre-Loïc Garoche. *Osdp, an Ocaml library for Semi-Definite Programming (SDP, SOS)*. 2014–. URL: https://cavale.enseeiht.fr/osdp/.

And the following presentations:

[Gar12] Pierre-Loïc Garoche. "Verification of aircraft controller: from process-based certification to product-based certification." In: *Air Force Safe & Secure Systems & Software Symposium, S5 Conference, Dayton, OH, USA*. (abstract only). June 2012.

[AGM15] Assalé Adjé, Pierre-Loïc Garoche, and Victor Magron. "Property-based polynomial invariant generation using sums-of-squares optimization." In: *17th British-French-German Conference on Optimization 15-17 June 2015 London, United Kingdom*. (abstract only). 2015.

[Gar15] Pierre-Loïc Garoche. "Certificate-carrying modular compilation." In: *10èmes Journées compilation du GDR GPL*. (abstract only). 2015.

[Wan+15] Timothy Wang, Éric Féron, Romain Jobredeaux, Marc Pantel, Pierre-Loïc Garoche, and Didier Henrion. "Credible autocoding of convex optimization algorithms." In: *17th British-French-German Conference on Optimization 15-17 June 2015 London, United Kingdom*. (abstract only). 2015.

[Gar+16] Pierre-Loïc Garoche, Didier Henrion, Victor Magron, and Xavier Thirioux. "Semidefinite Approximations of Reachability Sets for Discrete-time Polynomial Systems." In: *Journées MODE Mathématiques de l'Optimisation et de la DEcision de la SMAI Société de Mathématiques Appliquées et Industrielles*. 2016.

CONTENTS

Part I

MOTIVATION

1

CRITICAL EMBEDDED SOFTWARE: CONTROL SOFTWARE DEVELOPMENT AND V&V

Cyber physical systems (CPS) is a kind of buzz word 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 harmnessless for people, or for the huge cost associated to 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, network, computers, actuators, etc, and adapts the control to these discrepancies.

The increase of computer use in those systems has lead to huge benefits but also an exponential growth in complexity. Computer based systems compared to analog circuits enable more efficient behaviors, 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 looses, in flight, from 60 to 80% of one of its wings[2]; United Technology has been able to replace huge and heavy power electric systems by 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 computer and software. While the industry has been used for ages to have access to the precise characteristic of its components, eg. 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 nowadays used to accept 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 however unacceptable for any other industry.

To conclude with this brief motivation, the aerospace industry, and more generally critical embedded systems industries, 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 the need to integrate even more advanced algorithms to sustain autonomy and energy efficiency.

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

Until now, classical means to guaranty good behavior were mainly relying on tests. In the aerospace industry the development process is strictly constrained by norms such as the DO-178C [RTC11] specifying how to design a 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 be solved[4].

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 [NW05].

2 Search for Damage Tolerance Flight Test video, e.g. at https://www.youtube.com/watch?v=PTMpq_8SSCI

3 Eg. Active EMI filtering for inverters used at Pratt and Witney, 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 transportation industry.

Last, these certification documents such as DO178C 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 case of success, provide an exhaustive result, ie. they guarantee that the property considered is valid for all uses, including systems admitting infinite behaviors.

All the works presented here are motivated by this context. We aim at developing formal methods sustaining the verification of controller properties at multiple stages of their development. Our goal is to provide new means of verification, specific to controller analysis.

CURRENT LIMITS & OBJECTIVES The objectives of the presented works are restricted to the definition of formal methods based analyzes 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 programs. 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 [Cou+05], dedicated analyses, like the second-order filter abstraction [Fer04], have been defined. But these domains definition is tailored to the program for which they are defined.

Lack of means to compute non linear invariants As we will see in this document, the simplest properties of controllers are often based on at least quadratic properties. Again, because of efficiency and scalability, most analyzes are bound to linear properties. We claim that more expressive yet more costly analyzes 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 most of the time 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 non linear 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 a floating-point semantics and the associated errors. We think that more integrated analyzes should address the study of the global system.

Our proposal is mainly developed in two complementary directions:

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

This document is structured in five parts:

Part I introduces formal methods and controller design. It intends 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 considering floating-point computations. A first part consider that the program analyzed is executed with floating-point semantics and search for an inductive invariant considering the numerical errors produced. A second part ensures that the use of con-

vex optimization, a numerical technique, does not suffer from similar floating point errors.

Part V outlines possible research directions. They range from the definition of new analyses, the integration of the analyses in a realistic development process, to the extension of the presented approaches to more systems and more properties. A last perspective is the study of optimization algorithm *per se* in order to enable their use in critical applications.

2

FORMAL METHODS: DIFFERENT APPROACHES FOR VERIFICATION

While testing is a common practice for a lot of engineers as a way to evaluate whether the program they developed fulfill its needs, formal methods are less known and may require a little introduction to the non-expert. This chapter can be easily skipped by the formal verification reader but should be a reasonable introduction to the control expert engineer.

In this chapter we will try to give a brief overview of some of these formal methods, and their use in the context of critical embedded systems development. We will first define the semantics of programs: their basic properties and their meaning. Then, we will outline different formal verifications and explain how they reason on the program artifact. A last part will address the soundness of the analyses with respect to the actual semantics.

2.1 SEMANTICS AND PROPERTIES

Let us first consider a simple imperative program as we could write in C code and use it to introduce basic notions:

C

```
1  int f (x) {
2    int y = 42;
3    while (x > 0) {
4      x = x - 2;
5      y = y + 4;
6    }
7    return y;
8  }
```

For a given input x, this program is deterministic: it admits a single execution. Let us assume it is called with $x = 3$. In that case the execution is finite and will stop once x becomes non positive, here $x = -1$. This happens after two executions of the loop body. Therefore, $y = 42 + (2 * 4) = 50$ when the program stops.

The semantics, ie. the meaning of this program can be characterized in different ways. One approach is to see it as a function that takes inputs – here x – and returns the output value y. We speak of a *denotational semantics*: $[\![f]\!]_{den}(3) = 50$. We could characterize the output of the program `f` as a mathematical function $f : \mathbb{Z} \to \mathbb{Z}$ of x:

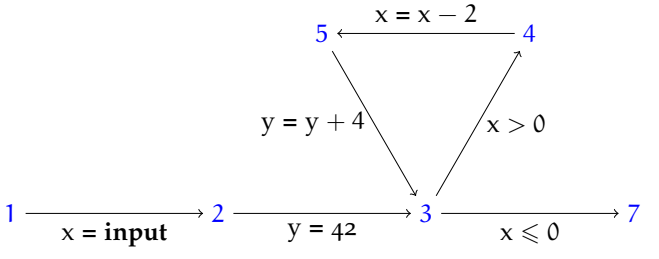
$$f(x) = 42 + 4 * \left\lceil \frac{x}{2} \right\rceil$$

Another approach details the steps of the computation and does not only focus on the result. This is the *operational semantics*. In operational semantics, one describes the behavior of the program as a sequence of transitions between states. A state denotes a current view of the program. In this simple case, a state can be characterized by a triple program point (pp), x, and y. In the following, we denote by Σ such set of states. Let us look at the simple execution of `f` with input 5:

state	0	1	2	3	4	5	6	7	8
pp	2	3	4	5	3	4	5	3	7
x	5	5	3	3	3	1	1	1	-1
y	42	42	42	46	46	46	50	50	50

The run of the program is here described by a sequence of states, a trace: $s_0 \to s_1 \to \ldots \to s_8$. In this case of a deterministic function, each trace is only characterized by its initial element s_0. Initial elements are a subset of states: let $Init \subseteq \Sigma$ be such set. The set of rules describing possible transitions from one state to the other characterizes the operational semantics of the program. Let us denote it by $[\![f]\!]_{op} \in \Sigma \times \Sigma$, the set of transitions from state to state. One can also represent it as a kind of automaton: the control flow graph.

Figure 2.1 Control flow graph



By interpreting a program as a set of states Σ, an initial set of states $Init \subseteq \Sigma$ and a transition relation $[\![\cdot]\!]_{op} \subseteq \Sigma \times \Sigma$, we defined a transition system $(\Sigma, Init, [\![\cdot]\!]_{op})$.

In practice, one is not necessarily interested directly in the program semantics in a denotational or operational form but rather by the properties of the program when executed.

The most precise definition of a program behavior is to characterize exactly its set of traces, its *trace semantics*:

$$[\![f]\!]_{\text{trace}} = \left\{ s_0 \to \ldots \to s_n \;\middle|\; \begin{array}{l} \forall i \in [0, n-1], (s_i, s_{i+1}) \in [\![f]\!]_{\text{op}} \\ s_0 \in \text{Init} \end{array} \right\}$$

In case of non terminating programs, traces could be infinite. While non terminating programs are usually seen as bad programs in computer science, controllers are supposed to be executed without time limit, in a while true loop. We can extend the definition of trace semantics for infinite traces:

$$[\![f]\!]_{\text{trace}} = \left\{ s_0 \to \ldots \to s_i \to \ldots \;\middle|\; \begin{array}{l} \forall i \geqslant 0, (s_i, s_{i+1}) \in [\![f]\!]_{\text{op}} \\ s_0 \in \text{Init} \end{array} \right\}$$

To summarize, the trace semantics captures the possibly infinite set of possibly infinite traces. If provided with such set, one can observe any properties related to intermediate computed values, occurrence of states within traces, infinite behavior, finite behavior such as deadlocks,... These properties are usually defined as temporal properties.

Another semantics of interest, with respect to the program semantics, is the *collecting semantics*. This semantics focuses only on reachable states in traces but not on their specific sequences.

One can define it as follows:

$$[\![f]\!]_{\text{coll}} = \left\{ s_n \;\middle|\; \begin{array}{l} \exists\, s_0 \to \ldots \to s_n \in [\![f]\!]_{\text{trace}} \text{ i.e. such that} \\ \exists s_0, \ldots, s_n, \ldots \in \Sigma \\ \forall i \in [0, n-1], (s_i, s_{i+1}) \in [\![f]\!]_{\text{op}} \\ s_0 \in \text{Init} \end{array} \right\}$$

As such, collecting semantics is an abstraction of trace semantics: it characterizes a set of reachable states but loses information on their relationship. This semantics is however extremely useful: it can capture all reachable states and therefore guarantee that all such states verify a given invariant, or avoid a given bad region.

A last way to express the behavior of a program is the axiomatic semantics. First ideas were proposed by TURING [Tur49], then this notion of *axiomatic semantics* was introduced by HOARE in 1969 [Hoa69]. In 1967 FLOYD [Flo67] proposed to annotate a flowchart by its local invariants. The following figure is extracted from that paper.

Figure 2.2 Assigning meanings to programs by FLOYD

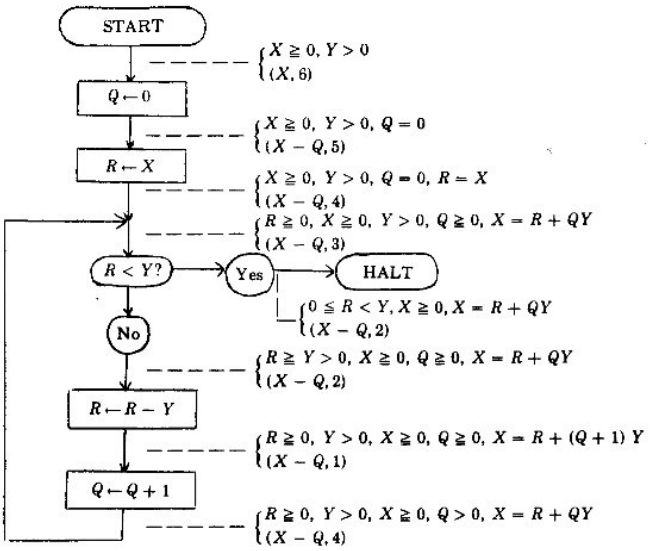


FIGURE 5. Algorithm to compute quotient Q and remainder R of $X \div Y$, for integers $X \geqq 0$, $Y > 0$

In [Hoa69], "An Axiomatic Basis for Computer Programming", HOARE defines a deductive reasoning to validate code level annotations. This paper introduces the concept of HOARE triple {Pre}code{Post} as a way to express the semantics of a piece of code by specifying the postconditions (Post) that are guaranteed after the execution of the code, assuming that a set of preconditions (Pre) was satisfied. HOARE supports a vision in which this axiomatic semantics is used as the "ultimately definitive specification of the meaning of the language [...], leaving certain aspects undefined. [...] Axioms enable the language designer to express its general *intentions* quite simply and directly, without the mass of detail which usually accompanies algorithmic descriptions."

Assuming the Euclidian division algorithm presented in Fig. 2.2 is implemented in a C function `div(x,y,*q,*r)`, one can specify the contract as follows:

```c
void div(x,y,*q,*r) {
  // { x⩽0 ∧ y>0 }
  *q = 0;
  *r = x;
  while (*r < y) { ... };
  // { 0 ⩽*r<y ∧ x⩾0 ∧ x=*r+*q×y }
}
```

As envisioned by HOARE, this approach has been largely developed and is used to specify formally the intended behavior of a program as a set of HOARE triples. Theoretically speaking, axiomatic semantics is a

further abstraction of operational or denotational semantics since it only constrains valid implementations.

2.2 A FORMAL VERIFICATION METHODS OVERVIEW

We will now illustrate the basic principles behind main verification methods: deductive methods (DM), SMT-based model-checking (MC) and abstract interpretation (AI). First, we sketch here how these techniques work on simple loopless examples. Then, we elaborate more on some details of their implementation or their use on more realistic examples. The exhaustive presentation of these techniques, developed since thirty to forty years, cannot be done in a few pages. The presentation reflects the author's view and understanding of these approaches.

First let us make a disappointing statement:

Theorem 2.1 (Rice's theorem) *It is undecidable to determine whether the language recognized by an arbitrary* Turing *machine* T *lies in a non trivial set of languages* S*.*

$$\mathcal{L}(T) \subseteq S \text{ is undecidable}$$

where $\mathcal{L}(T)$ *denotes the language recognized by the Turing machine* T*.*

Here the *non trivial set of languages* S denotes a valid output of the program, ie. a property of its trace semantics. This theorem, which may not be easily readable for the theoretical computer science agnostic, states that any property of interest is hardly analyzable on a program. In other words: "it is undecidable to determine whether an arbitrary program satisfies a non trivial property".

Because of undecidability it is worthless to design sound, complete and terminating techniques for arbitrary programs and properties. Let us denote by $\mathtt{Prog} \models P$ the validity of property P for program $\mathtt{Prog}$ and by $\mathtt{Prog} \vdash_A P$ the fact that the analysis A stated that P was valid for program $\mathtt{Prog}$. We can define, for all program $\mathtt{Prog}$ and property P:

$$\begin{aligned} \mathtt{Prog} \vdash_A P \;&\Rightarrow\; \mathtt{Prog} \models P && \text{(soundness)} \\ \mathtt{Prog} \models P \;&\Rightarrow\; \mathtt{Prog} \vdash_A P && \text{(completeness)} \\ \mathtt{Prog} &\vdash_A P \text{ terminates} \end{aligned}$$

Formal verification techniques usually address this issue by focusing on sound and terminating methods, that is without the completeness property. This amount to compute an intermediate stronger property P' such that

$$\begin{aligned} \big((\mathtt{Prog} \vdash_A P') \wedge (P' \Rightarrow P)\big) \;&\Rightarrow\; \mathtt{Prog} \models P \\ \mathtt{Prog} &\vdash_A P' \text{ terminates} \end{aligned}$$

This is often referred to as over-approximation techniques, or conservative techniques: showing the validity of P amounts to compute a less precise property P' which may imply P. Even if the property P was actually valid on the program, the lack of precision of P' may not permit to prove $P' \implies P$ leading to a lack of conclusion: P has a unknown status for program $\mathtt{Prog}$, the analysis has been unable to conclude with respect to P.

Remark 1 (Termination of analysis vs program) *Note that termination of analysis is unrelated to the existence of infinite traces in the analyzed program. A non terminating formal verification technique may fail to return a result on a finite transition system admitting only finite traces, while a terminating analysis will conclude even for systems admitting infinite behaviors.*

2.2.1 *Basic principles illustrated on a loopless example*

Let us first focus on a simple loopless example, for example the infinity norm in $\mathbb{R}^2$:

C
```
1  real norminf(real x, y) {
2   real xm, ym, r;
3   if (x >= 0) // compute abs(x)
4    {xm = x;}
5   else
6    {xm = -x;};
7   if (y >= 0) // compute abs(y)
8    {ym = y;}
9   else
10   {ym = -y;};
11  if (xm >= ym) // compute max(xm, ym)
12   {r = xm;}
13  else
14   {r = ym;};
15  return r;
16 }
```

We are interested in the following properties:

- null on zero: $\mathtt{norminf}(0,0) = 0$;
- positivity: $\forall(x,y), \mathtt{norminf}(x,y) \geqslant 0$.

Note that, in that case, the formalization of the specification, that is the properties of interest, as formal artifacts was rather straightforward. It may be more difficult when considering natural language description with ambiguous statements. This is another added value of formal methods: disambiguation of specification by imposing the need of strict formalization.

Of course, a first classical approach could rely on tests to evaluate the validity of these properties. We will see how various formal method reason on that program, trying to prove the desired properties:

- DM: use of predicate transformation, either forward or backward reasoning;

- MC: propositional encoding and SMT-based reasoning;
- AI: interpretation of each computation in an abstract domain.

Deductive methods: predicate transformers

Deductive methods are the evolution of the ideas proposed by HOARE [Hoa69]. Predicate transformation allows to apply the semantics of the considered program on the formal representation of the property. These manipulations can be either performed in a forward manner, transforming the precondition through the code – we speak about *strongest postcondition* –, or, in the opposite direction, propagating back the postcondition through the code – we speak about *weakest precondition*. While both techniques should be equally sound, most implementations used in C code analysis [App11; Bau+02; Cuo+12; FM07] rely on the weakest precondition algorithm.

This method computes $wp(\mathtt{code}, \mathtt{Post})$, the weakest precondition such that, when executing the code, Post is guaranteed. The rules are defined on the structure on the imperative code, per statement kind and applied iteratively. On naive imperative languages statements can be either assignements or control structures such as sequencing of statements, conditionals (if-then-else) or loops:

The assignment rule amounts to substitute in the postcondition B any occurrence of x by its definition e:

$$wp(x := e, B) \triangleq [e/x]B$$

Example 1 *Let us illustrate this mechanism on the simplest example. Assuming the postcondition requires* `y ⩽0`*. The weakest precondition of the instruction* `y = x+1;` *imposes* `x ⩽-1`*.*

C
```
// { x+1 ⩽ 0 ≡ x ⩽ -1 }
y = x + 1;
// { y ⩽ 0 }
```

Weakest precondition composes well: once the computation of the impact of c_2 to B has been computed, it can be used to propagate the impact of statement c_1:

$$wp(c_1; c_2, B) \triangleq wp(c_1, wp(c_2, B))$$

Conditional statements (if-then-else) are encoded as a disjunction: one obtains B after executing the statement, either because b holds and c_1 gives B, or because $\neg b$ holds and c_2 gives B:

$$wp(\text{if } b \text{ then } c_1 \text{ else } c_2, B) \triangleq \wedge \begin{matrix} b \Rightarrow wp(c_1, B) \\ \neg b \Rightarrow wp(c_2, B) \end{matrix}$$

In our example, we have two properties expressed as the following HOARE triples:

$$\{(x, y) = (0, 0)\}\ \mathtt{norminf}\ \{\backslash result = 0\} \quad (1)$$

$$\{True\}\ \mathtt{norminf}\ \{\backslash result \geqslant 0\} \quad (2)$$

The first HOARE triple states that when $(x, y) = (0, 0)$ the result is 0, while the second one makes no assumption on the input: it should be valid in any context.

Let us look, manually, at this computation on the first property:

$$\backslash result = 0$$

is transformed through the last statement, a conditional statement (ite) on line 11. We obtain the weakest precondition of the statement line 11 guaranteeing $\backslash result = 0$. Each then and else block is analyzed with the wp algorithm, producing the required predicate $xm = 0$ or $ym = 0$. Then the weakest precondition of the conditional statement is produced:

$$(xm \geqslant ym \Rightarrow xm = 0) \wedge (xm < ym \Rightarrow ym = 0)$$

This predicate is further transformed in the leaves of the previous statement, at line 7. Then, block at line 8 is associated to the weakest precondition:

$$(xm \geqslant y \Rightarrow xm = 0) \wedge (xm < y \Rightarrow y = 0)$$

while the else-block at line 10 gives

$$(xm \geqslant -y \Rightarrow xm = 0) \wedge (xm < -y \Rightarrow -y = 0)$$

Combined with the conditional rule, this gives:

$$\left(y \geqslant 0 \Rightarrow \begin{pmatrix} (xm \geqslant y \Rightarrow xm = 0) \\ \wedge (xm < y \Rightarrow y = 0) \end{pmatrix}\right) \wedge \left(y < 0 \Rightarrow \begin{pmatrix} (xm \geqslant -y \Rightarrow xm = 0) \\ \wedge (xm < -y \Rightarrow -y = 0) \end{pmatrix}\right)$$

Let us, again propagate this weakest precondition to the previous statement at line 3. We obtain, for its then-block the predicate

$$\left(y \geqslant 0 \Rightarrow \begin{pmatrix} (x \geqslant y \Rightarrow x = 0) \\ \wedge (x < y \Rightarrow y = 0) \end{pmatrix}\right) \wedge \left(y < 0 \Rightarrow \begin{pmatrix} (x \geqslant -y \Rightarrow x = 0) \\ \wedge (x < -y \Rightarrow -y = 0) \end{pmatrix}\right)$$

and for its else-block:

$$\left(y \geqslant 0 \Rightarrow \begin{pmatrix} (-x \geqslant y \Rightarrow -x = 0) \\ \wedge (-x < y \Rightarrow y = 0) \end{pmatrix}\right) \wedge \left(y < 0 \Rightarrow \begin{pmatrix} (-x \geqslant -y \Rightarrow -x = 0) \\ \wedge (-x < -y \Rightarrow -y = 0) \end{pmatrix}\right)$$

Last the conditional rule is applied:

$$\left(\begin{array}{r} x \geqslant 0 \Rightarrow \\ \left(\begin{array}{r} \left(y \geqslant 0 \Rightarrow \left(\begin{array}{r} (x \geqslant y \Rightarrow x = 0) \\ \wedge (x < y \Rightarrow y = 0) \end{array}\right)\right) \\ \wedge \left(y < 0 \Rightarrow \left(\begin{array}{r} (x \geqslant -y \Rightarrow x = 0) \\ \wedge (x < -y \Rightarrow -y = 0) \end{array}\right)\right) \end{array}\right) \end{array}\right) \wedge$$
$$\left(\begin{array}{r} x < 0 \Rightarrow \\ \left(\begin{array}{r} \left(y \geqslant 0 \Rightarrow \left(\begin{array}{r} (-x \geqslant y \Rightarrow -x = 0) \\ \wedge (-x < y \Rightarrow y = 0) \end{array}\right)\right) \\ \wedge \\ \left(y < 0 \Rightarrow \left(\begin{array}{r} (-x \geqslant -y \Rightarrow -x = 0) \\ \wedge (-x < -y \Rightarrow -y = 0) \end{array}\right)\right) \end{array}\right) \end{array}\right) \quad (3)$$

This large predicate represents the weakest precondition, that, when satisfied, guarantee to obtain $\backslash result = 0$ after executing the code. In this first property, the precondition was $(x, y) = (0, 0)$. Therefore, we have to prove

$$(x, y) = (0, 0) \Rightarrow (3)$$

This proof is sent to a satisfiabiliy modulo theory solver (SMT) such as Alt-Ergo [Con+08], Z3 [MB08], CVC4 [BT07; Det+14] or Yices [Dut14; DM06]. These solvers extend a SAT[1] core to predicates whose atoms are expressed in other (numerical) theories.

In this specific case, the formula is easily analyzed – it can even be done by hand – and reduces to the predicate

$$True$$

The second property can be similarly analyzed and will generate the following proof objective

$$True \Rightarrow \{(3) \text{ in which } v = 0 \text{ becomes } v \geqslant 0\}$$

SMT-based model-checking: propositional encoding and satisfiability

SMT-based model checking will perform similarly on this specific example. The idea is to map all constructs as predicates. One can, for example, rename variables to avoid multiple assignments to the same variable. This amounts to embed the imperative program as functional dependencies between input and output. Let $[\![norminf]\!]_{MC}(x, y, r)$ be such function.

The proof objectives become:

$$(x = 0 \wedge y = 0) \wedge [\![norminf]\!]_{MC}(x, y, r) \wedge (r = 0) \quad (4)$$

$$[\![norminf]\!]_{MC}(x, y, r) \wedge (r \geqslant 0) \quad (5)$$

The difference with deductive methods is not really visible in this oversimple example. The main one is that no order is specified on the model-checking approach, while weakest precondition rules do transform the predicate statement after statement. One can also notice that the expression of the functional representation of $[\![norminf]\!]_{MC}(x, y, r)$ is identical in both properties (4) and (5). In deductive methods, the form of the predicate representation of the code widely depends on the property analyzed.

In both cases, the final validity of the propositional encoding of the property is delegated to external solvers such as SMT-solvers.

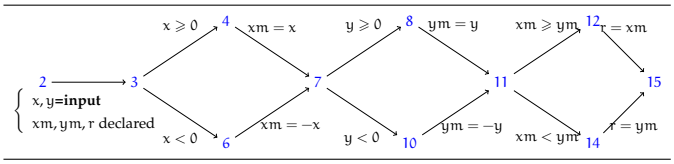
Abstract interpretation (of collecting semantics): over-approximating reachable states

Abstract interpretation relies on different algorithms. We will develop it in its general setting in the next section. In contrast to previous methods which are able to represent complex properties through logical predicates but rely on external solvers to determine the satisfiability of these formulas, the abstract interpretation paradigm intends to restrict *a priori* the form of the properties manipulated, providing constructive means to analyze them.

These constrained properties are called abstract domains and, since we are focused on the abstraction of the collecting semantics, they represent set of states. One can see an abstract domain D as a subset of set of states: $D \subseteq \wp(\Sigma)$. A classical example – and a widely used one – is the abstract domain of intervals $\mathcal{M}$ to represent subsets of $\mathbb{R}$ and the use of interval arithmetic to manipulate these abstract values. An abstract environment is used to represent a set of states. Let us informally show the computation of the abstract environment in our example before providing more theoretical background.

The computations are performed on the control flow graph. The following picture characterizes it for our example:

Figure 2.3 Control flow graph for infinity norm.



One can associate to each program point, its abstract collecting semantics equations. These equations define the local abstract environment, depending on the predecessor values:

1 A SAT(isfiability) solver aims at proving that a propositional formula (a formula composed of boolean variables, and logical operators $\wedge$, $\vee$, $\neg$) either admits a satisfiable assignment of the free variables that makes the formula true, or show that no such assignement exists.

$$
\begin{array}{lcl}
S_2 & = & \{\text{any value}\} \\
S_3 & = & S_2 \left[\begin{array}{lcl} x & \mapsto &]-\infty,+\infty[\\ y & \mapsto &]-\infty,+\infty[\\ xm & \mapsto &]-\infty,+\infty[\\ ym & \mapsto &]-\infty,+\infty[\\ r & \mapsto &]-\infty,+\infty[\end{array}\right] \\
S_4 & = & S_3[x \geqslant 0] \\
S_6 & = & S_3[x < 0] \\
S_7 & = & S_4 \sqcup S_6 \\
S_8 & = & S_7[y \geqslant 0] \\
S_{10} & = & S_7[y < 0] \\
S_{11} & = & S_8 \sqcup S_{10} \\
S_{12} & = & S_{11}[xm \geqslant ym] \\
S_{14} & = & S_{11}[xm < ym] \\
S_{15} & = & S_{12} \sqcup S_{14}
\end{array}
$$

where $S[e > 0]$ denotes the environment S in which the abstract evaluation of e is constrained to be positive; $S[x \mapsto e]$ denotes the environment S in which variable x is updated to the abstract value e; and $S_1 \sqcup S_2$ denotes the lift of interval join to maps: $[x \mapsto S_1(x) \cup_{\mathcal{M}} S_2(x)]$.

When entering in the function body, nothing is assumed on x and y. The abstract environment is then the map

$$x \mapsto]-\infty,+\infty[\qquad y \mapsto]-\infty,+\infty[$$

Depending on the language semantics, the declaration of local variables at line 2 can either assign them to a default value, or, like in C, give an value. We have the following updated abstract environment at line 3:

$$
\begin{array}{llll}
x & \mapsto]-\infty,+\infty[& y & \mapsto]-\infty,+\infty[\\
xm & \mapsto]-\infty,+\infty[& ym & \mapsto]-\infty,+\infty[\\
r & \mapsto]-\infty,+\infty[& &
\end{array}
$$

The evaluation of the first statement constrains the values of x depending on the active branch:

At line 4, we have

$$
\begin{array}{llll}
x & \mapsto [0,+\infty[& y & \mapsto]-\infty,+\infty[\\
xm & \mapsto]-\infty,+\infty[& ym & \mapsto]-\infty,+\infty[\\
r & \mapsto]-\infty,+\infty[& &
\end{array}
$$

while at line 6 we have:

$$
\begin{array}{llll}
x & \mapsto]-\infty,0[& y & \mapsto]-\infty,+\infty[\\
xm & \mapsto]-\infty,+\infty[& ym & \mapsto]-\infty,+\infty[\\
r & \mapsto]-\infty,+\infty[& &
\end{array}
$$

After the assignment of line 4, we obtain for variable xm the interval $[0,+\infty[$. Similarly, after the assignment of line 6, we obtain for variable xm the interval $-]-\infty,0[=]0,+\infty[$. The computation of the join in the definition of the abstract collecting semantics at program point 7 returns the interval $]0,+\infty[\cup_{\mathcal{M}}[0,+\infty[=[0,+\infty[$ for xm. However, the join of $]-\infty,0[$ and $[0,+\infty[$ returns the interval $]-\infty,+\infty[$ for variable x.

The abstract evaluation of program points 8 to 15 follows comparable patterns. Note that the conditions $xm \geqslant ym$ does not provide any meaningful information for this interval-based analysis. We eventually obtain the following abstract environment:

$$
\begin{array}{llll}
x & \mapsto]-\infty,+\infty[& y & \mapsto]-\infty,+\infty[\\
xm & \mapsto [0,+\infty[& ym & \mapsto [0,+\infty[\\
r & \mapsto [0,+\infty[& &
\end{array}
$$

This analysis has been able to obtain the positivity of r without any assumption on the input values. The same analysis can be done by assuming that the initial abstract environment is:

$$x \mapsto [0,0] \qquad y \mapsto [0,0]$$

In that case the final abstract environment obtained is:

$$
\begin{array}{llll}
x & \mapsto [0,0] & y & \mapsto [0,0] \\
xm & \mapsto [0,0] & ym & \mapsto [0,0] \\
r & \mapsto [0,0] & &
\end{array}
$$

Note that abstract environments associated to program points 6, 10 and 14 are associated to the empty environment denoting unreachable program points.

2.3 DEDUCTIVE METHODS

Weakest precondition methods are typically designed for imperative languages. A realistic application will reason on the program as outlined in Section 2.2.1 but shall also address the following items:

2.3.1 *Loops and recursion in programs*

While predicate transformation may seem natural in the previous example, it is less obvious in presence of loops in the control flow graph. A sufficient rule to validate annotations, as defined by FLOYD or HOARE could be:

$$\frac{\vdash \{A \wedge b\} c \{A\}}{\vdash \{A\} \text{ while } b \text{ do } c \{A \wedge \neg b\}}$$

But it is not compatible with the automatic transformation of predicates as performed in weakest precondition computation. Another way to address this issue is to unroll the loop:

$$\text{while b do c} \equiv \text{if b then c; while b do c else skip}$$

Then

$$\begin{aligned} & wp(\text{while } b \text{ do } c, B) \\ \triangleq\ & wp(\text{if } b \text{ then } c; \text{while } b \text{ do } c \text{ else skip}, B) \\ \triangleq\ & b \Rightarrow wp(c, wp(\text{while } b \text{ do } c, B)) \wedge \neg b \Rightarrow B \end{aligned}$$

Let us denote by $W = wp(\text{while } b \text{ do } c, B)$. We can use the loop unfolding to characterize recursively W:

$$W = (b \Rightarrow wp(c, W) \wedge \neg b \Rightarrow B)$$

Thanks to TARSKI's fixpoint theorem, considering the partial order induced by logical implication $\Rightarrow$, ie. $x \sqsubseteq y \triangleq y \Rightarrow x$, and the monotonic definition of W, this fixpoint exists. But this formula is difficult to compute and may not be representable with a finite set of atoms. If characterizable it captures precisely the loop semantics: the most precise loop invariant, the relationship between input and output, preserved by the loop body.

The solution proposed by DIJKSTRA [Dij76] is to provide, manually, a weaker loop invariant I, ie. such that $I \Rightarrow W$. The predicate transformation rule is then defined as

$$\begin{aligned} & wp(\text{while } b \text{ do } c, B) \\ \triangleq\ & I \wedge ((I \wedge b) \Rightarrow wp(c, I)) \wedge ((I \wedge \neg b) \Rightarrow B) \end{aligned} \tag{6}$$

As a result, any occurrence of loop in programs requires the definition of a loop invariant capturing the loop semantics.

Similarly, in order to prove termination, one needs to exhibit a loop variant, a decreasing sequence in a Noetherian relation, also called a well-founded relation. Typical implementations rely on a positive integer-valued function decreasing at each loop iteration.

2.3.2 *Memory model and low-level representation*

Until now all computations have been performed on a naive imperative language with real datatypes, without complex datastructure, memory allocation, or function calls.

Serious tools such, as Frama-C, handle all those constructs. Memory issues are a large part of them. Multiple choices could be made to represent the memory: from the simplest being the HOARE model without pointers or aliases, to a bit level representation. The more complex the memory model, the bigger the generated predicate.

Dedicated analyses such as separation logic [ORY01] can be used to detect aliases or guarantee that pointers x and y are separated, easing the later analyses. Tools such as the Verified Software Toolchain (VST) [App11] rely on such analysis.

2.3.3 *Underlying logic and automatic proof*

A last difficulty in realistic implementations is the expressivity and tools associated to the underlying logic. In Frama-C, the annotation language ACSL [Bau+08] (ANSI C Specification Language), is extremely rich and enables the definition of predicates or internal data structures in both functional or axiomatic ways. However, for the same concept, e.g. a linked list or a tree like structure, an integer valued function computing the size of a data structure,..., the generated predicate will widely differ and so do the results of the automatic solver to prove the final proof objective.

Efficient use of these techniques requires the understanding of solver capabilities and their efficiency on different kinds of modelings.

2.4 SMT-BASED MODEL CHECKING

While SMT-based model checking can be applied at code level, eg. the SPACER tool [KGC14; Kom+13], most applications are performed on earlier representations of the system, at model level.

In all cases, a logical representation of the denotational semantics is extracted from the model/code f. It can be as a single predicate associating outputs to inputs, or a more axiomatic definition, for example relying on a set of Horn clauses. In all cases, it characterizes a transition system with inputs In and outputs Out: $(\Sigma, Init \subseteq \Sigma, T)$ where $T(x, y) \equiv [\![f]\!]_{den}(x) = y$.

When considering functions with side effects, ie. depending on memory and modifying it through execution, the typical predicate is

$$T(in, out, mem\_pre, mem\_post)$$

We can also define the initial state of the memory with a predicate:

$$Init(mem)$$

These predicates are valid only for values that satisfy the program semantics. In the memoryless example of Fig. 2.1, we have $T(0, 42)$, $T(1, 46)$, $T(2, 46)$ since these values are valid pairs of input/output, but $T(1, 2)$ is false.

For models without complex datastructures this encoding can be rather straightforward. In case of a variety of datatypes, casts between values, complex control flow structures, the encoding can be less easy to define.

Once the encoding is available, one can reason about it. When relying on model-based development such as Matlab Simulink, ANSYS Scade, or Lustre, it is possible to extract such encoding. Since all those models denote synchronous dataflow languages, the semantics of a model is the infinite execution of the block semantics.

Let us consider a (possibly infinite) trace $s_0 \to \ldots \to s_i \to \ldots$ of such a system. It corresponds to the sequence of inputs $i_0 \to \ldots \to i_i \to \ldots$ and satisfies the following constraints:

$$\mathrm{Init}(s_0) \tag{7}$$
$$\forall i \geqslant 0, \exists o_i, \text{ s.t. } T(i_i, o_i, s_i, s_{i+1}) \tag{8}$$

generating the sequence of outputs $o_0 \to \ldots \to o_i \to \ldots$

Most SMT-based model checking techniques are based on the induction principle: a way to prove a property invariant over reachable states is to show it inductive over such states. Let $P(s)$ be the predicate encoding of this property.

We recall that the induction principle requires:

$$\forall s \in \Sigma, \quad \mathrm{Init}(s) \Rightarrow P(s) \qquad \text{(base case)} \tag{9}$$
$$\forall s_1, s_2 \in \Sigma, \quad P(s_1) \wedge T(s_1, s_2) \Rightarrow P(s_2) \qquad \text{(inductive case)} \tag{10}$$

However, while the property is inductive over reachable states $[\![f]\!]_{\text{coll}}$:

$$\forall s \in \Sigma, \quad \mathrm{Init}(s) \Rightarrow P(s) \qquad \text{(base case)} \tag{11}$$
$$\forall s_1, s_2 \in \Sigma \cap [\![f]\!]_{\text{coll}}, \quad P(s_1) \wedge T(s_1, s_2) \Rightarrow P(s_2) \qquad \text{(inductive case)} \tag{12}$$

The same property may not be inductive over some states $s \in \Sigma \setminus [\![f]\!]_{\text{coll}}$. Such a state would correspond to a spurious counter-example: a state s_1 unreachable but satisfying P such that its successor s_2 by the transition system semantics violates P:

$$(P(s_1) \wedge T(s_1, s_2)) \not\Rightarrow P(s_2)$$

Different approaches exist to attempt to address this issue, without guarantees of success since $[\![f]\!]_{\text{coll}}$ is not computable:

1. replace $[\![f]\!]_{\text{coll}}$ by some other invariant I of reachable states. The inductive case becomes[2]:

$$\forall s_1, s_2 \in \Sigma, \quad (P(s_1) \wedge T(s_1, s_2) \wedge I(s_1) \wedge I(s_2)) \Rightarrow P(s_2) \tag{13}$$

2. Improve the quality of the initial s_1 as part of reachable states: impose it to be part of a path of length k of the transition system. In that case, it is also required to update the base case in order to guarantee property P for the first k reachable states:

$$\forall l \leqslant k, \forall s_0, \ldots, s_l \in \Sigma, \quad \mathrm{Init}(s_0) \wedge \bigwedge_{0 \leqslant i \leqslant l-1} T(s_i, s_{i+1}) \Rightarrow \bigwedge_{0 \leqslant i \leqslant l} P(s_i) \tag{14}$$
$$\forall s_0, \ldots s_{k+1} \in \Sigma, \quad \bigwedge_{0 \leqslant i \leqslant k} (P(s_i) \wedge T(s_i, s_{i+1})) \Rightarrow P(s_{k+1}) \tag{15}$$

The second approach is known as k-induction and was first proposed for pure propositional properties and systems [SSS00] and then extended to more general systems using SMT [KT11]. This is typically the algorithm used in formal verifiers in ANSYS Scade or Matlab Simulink.

The first approach is quite natural: instead of looking for a general inductive property we focus on a restricted set of states. Multiple methods were proposed to synthesize the invariant I: simple patterns instantiation [KGT11], the use of abstract interpretation [GKT12], the use of quantifier elimination [CDD15], or the dynamic synthesis of property specific invariants in property-directed reachability (PDR/IC3) [Bra12].

As in deductive methods, the efficiency of the analysis depends on the encoding of the properties and the SMT solver abilities to prove the base and inductive cases.

2.5 Abstract interpretation (of collecting semantics)

The abstract interpretation framework proposed by Cousot and Cousot [CC77] provides a methodology in which analyses of semantics can be easily defined and proved correct. An essential step of that methodology is to characterize the semantics of interest as a fixpoint of a monotonic operator over a complete lattice. We refer the reader to Miné's PhD manuscript for a very good introduction to the theory [Min04].

For the moment, let us give the following definition.

Definition 2.2 (Abstract Interpretation) *Abstract Interpretation is a constructive and sound theory for the approximation of semantics expressed as fixpoint of monotonic operators in a complete lattice.*

While this formulation may seem unnatural to the newcomer, it is actually a simple step when it comes to collecting semantics. Collecting semantics is the semantics characterizing reachable states of a program or of a dynamical system. We recall that Σ is the set of all states. We are interested in characterizing all reachable states $s \in \Sigma$. All reachable states form a set of states $S \subseteq \Sigma$ and belongs to its powerset $S \in \wp(\Sigma)$. We would like to

2 Note that I may not be inductive with respect to T.

compute the most precise element of $\wp(\Sigma)$ denoting all reachable states.

Any powerset is a complete lattice. It is fitted with a partial order, the set inclusion $\subseteq$; any subset of elements admits a least upper bound, the set union $\cup$, and a greatest lower bound, the set intersection $\cap$. It is fitted with a lowest element $\emptyset$ and a greatest one Σ. Therefore, our element of interest denoting all reachable states, let's call it $\mathfrak{C}$, is one specific element of the complete lattice $\langle \wp(\Sigma), \subseteq, \cup, \cap, \emptyset, \Sigma \rangle$.

When one considers the underlying update function of the analyzed system - the transition relation of a dynamical system, or a function describing how each program point value is computed from its predecessors - it can be defined as an endomorphism of Σ. It maps a state to a new state. Let $f : \Sigma \to \Sigma$ be such a function. Note that this function does not need to be monotonic in any sense. In order to ease the later notations, we will indifferently denote by f the isomorphism of Σ or its lift $f^{\uparrow}$ to sets of states $\wp(\Sigma)$: $f^{\uparrow}(S) = \{f(s) \mid s \in S\}$.

Using f, we can derive the monotonic transfer function F of the collecting semantics. A classical definition of F is the endomorphism of $\wp(\Sigma)$ which accumulates states starting from an initial set of states $\mathit{Init} \in \wp(\Sigma)$:

$$\begin{array}{rcl} \wp(\Sigma) & \to & \wp(\Sigma) \\ S & \mapsto & \mathit{Init} \cup f(S) \end{array} \tag{16}$$

When one applies recursively this function to the empty set, the infimum $\bot$ of the lattice $\langle \wp(\Sigma), \subseteq, \cup, \cap, \emptyset, \Sigma \rangle$, we characterize the following sequence of sets of states:

$$\begin{array}{ll} S_0 = F(\bot) & = \mathit{Init} \\ S_1 = F(\mathit{Init}) & = \mathit{Init} \cup F(\mathit{Init}) \\ S_2 = F(\mathit{Init} \cup f(\mathit{Init})) & = \mathit{Init} \cup F(\mathit{Init}) \cup F^2(\mathit{Init}) \\ \dots & \end{array}$$

Theorem 2.3 (Tarski's fixpoint theorem) *Let D be a complete lattice $\langle D, \sqsubseteq, \sqcup, \sqcap, \bot, \top \rangle$ and $f : D \to D$ be an monotonic function. Then the set of fixed points of f in D is also a complete lattice, it admits a least (lfp) and a greatest (gfp) fixpoints.*

$$\mathrm{lfp} f = \sqcap \{X \mid F(X) \sqsubseteq X\}$$
$$\mathrm{gfp} f = \sqcup \{X \mid X \sqsubseteq F(X)\}$$

Since F is a monotonic operator of $\langle \wp(\Sigma), \subseteq, \cup, \cap, \emptyset, \Sigma \rangle$, by Tarski's fixpoint theorem, a least fixpoint exists. It is defined as the smallest postfixpoint. A postfixpoint is an element $X \in \wp(\Sigma)$ such that $F(X) \subseteq X$.

Then our set of reachable states, the collecting semantics, is *exactly* characterized by

$$\mathfrak{C} = \mathrm{lfp}_{\emptyset} F = \inf_{X \in \wp(\Sigma)} \{F(X) \subseteq X\} \tag{17}$$

Furthermore, the set of fixpoints is fitted with a complete lattice structure: it is closed by join and meet; its infimum is the least fixpoint; and its supremum the greatest one.

2.5.1 *Abstracting the fixpoint: fixpoint computation in abstract domains*

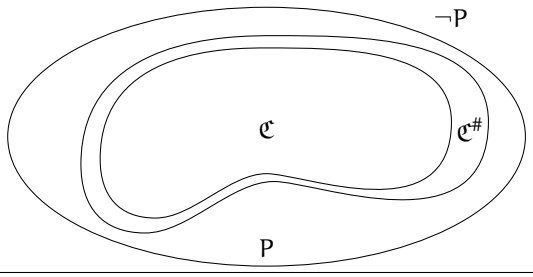
Soundness, incompleteness and alarms

Despite its proven existence, this exact set of reachable states is very hard to compute in general. The framework of abstract interpretation provides means to abstract it, that is, to compute another value $\mathfrak{C}^{\#}$ of $\wp(\Sigma)$ bigger than $\mathfrak{C}$ for the set inclusion, ie. containing more states. Some of those states are spurious, they are not reachable in practice, but will be considered as such by the abstraction computed. The validity of a property P is checked with respect to $\mathfrak{C}^{\#}$. P is characterized by the set of states satisfying it: $P = \{s \mid P(s)\}$. In case of success, we have all states in $\mathfrak{C}^{\#}$ satisfy the property, and therefore the subset $\mathfrak{C}$.

$$\begin{array}{ll} & \mathfrak{C}^{\#} \subseteq P \\ \Rightarrow & \mathfrak{C} \subseteq P \qquad \text{by inclusion } \mathfrak{C} \subseteq \mathfrak{C}^{\#} \end{array}$$

The figure 2.4 illustrates such inclusions.

Figure 2.4 Collecting semantics, abstraction and properties.



In case of failure, one cannot conclude since an erroneous state $s \in \mathfrak{C}^{\#} \setminus P$ could either be in $\mathfrak{C}$ or in spurious states introduced by the abstraction. We speak of an *alarm*. This characterizes the incompleteness of the approach.

3 In some cases, such as the one presented in Section 9.3.2, a complete lattice structure is not available. Proofs of convergence are then more complex to achieve.

Abstract domains

An abstraction is meant to approximate sets of states $\wp(\Sigma)$ and is defined by an abstract domain. An abstract domain represents a set of abstract states $D^\#$, fitted with a complete lattice structure[3]: $\langle D^\#, \sqsubseteq, \sqcup, \sqcap, \bot, \top \rangle$ where $\bot$ and $\top$ denotes infimum and supremum values, respectively.

It also provides means to abstract sets of states $\wp(\Sigma)$ to $D^\#$ and to compute a sound representation as set of states of its elements: those functions are called α and γ, the abstraction and the concretization functions:

$$\alpha : \wp(\Sigma) \to D^\# \qquad\qquad \gamma : D^\# \to \wp(\Sigma)$$

In order to fulfill the abstract interpretation framework methodology, in its most general setting, those abstraction and concretization functions should define a Galois connection:

$$\begin{cases} \text{monotonic } \alpha : \\ \forall s_1, s_2 \in \wp(\Sigma), s_1 \subseteq s_2 \Rightarrow \alpha(s_1) \sqsubseteq \alpha(s_2) \\ \text{monotonic } \gamma : \\ \forall s_1^\#, s_2^\# \in D^\#, s_1 \sqsubseteq s_2 \Rightarrow \gamma(s_1) \subseteq \gamma(s_2) \\ \text{reductivity of } \alpha \circ \gamma : \\ \forall s^\# \in D^\#, \alpha \circ \gamma(s^\#) \sqsubseteq s^\# \\ \text{extensivity of } \gamma \circ \alpha : \\ \forall s \in \wp(\Sigma), s \subseteq \gamma \circ \alpha(s) \end{cases} \tag{18}$$

An abstract domain is also fitted with means to compute, in the abstract, the operations that were performed in the concrete set of states Σ. This ranges from assignments of variables by a linear or polynomial expression, to comparison operations over values,.... We denote by $f^\# : D^\# \to D^\#$ the sound abstract counterpart of $f : \wp(\Sigma) \to \wp(\Sigma)$.

Soundness in abstract domains

Soundness is guaranteed with respect to the abstraction and concretization functions. We present here the classical definition on a unary operator fun.

$$\begin{aligned} &\forall S \in \wp(\Sigma), S^\# \in D^\#, \\ &S \subseteq \gamma(S^\#) \implies \text{fun}(S) \subseteq \gamma(\text{fun}^\#(S^\#)) \end{aligned} \tag{19}$$

Soundness could also be expressed relying on α. Intuitively this soundness requirement guarantees that all computations performed in the abstract will, at least, contain the real reachable states and values.

When the abstract domain is defined by a computable Galois connection (α, γ), one can derive automatically these abstract operators such that they compute a sound, yet most precise, solution:

$$\text{op}^\#(x) = \alpha \circ \text{op}(\gamma(x)) \tag{20}$$

In case of programs analyzed on their control flow graph representation, such as the ones of Figs. 2.1 and 2.3, (abstract) states of a node with multiple incoming edge, such as a loop head, or an instruction following a conditional statement, are the (abstract) join of the states available in each predecessors.

Using Tarski's theorem, one can associate to the concrete set of reachable states $\mathfrak{C}$ the fixpoint of an abstract function $F^\#$:

$$\mathfrak{C}^\# = \text{lfp}_\bot F^\# \tag{21}$$

$$= \inf_{X \in D^\#} \left\{ F^\#(X) \sqsubseteq X \right\} \tag{22}$$

where $F^\#(S) = \alpha(\text{Init}) \sqcup f^\#(S)$.

Fixpoint transfer

Thanks to the appropriate choice of α and γ functions, for example with a Galois connection, and with the additional constraint that the abstraction α commutes with F:

$$\alpha \circ F = F^\# \circ \alpha \tag{23}$$

We have:

	$\text{Init} \subseteq \gamma(\alpha(\text{Init}))$	(ext. of $\gamma \circ \alpha$)
$\Rightarrow$	$F(\text{Init}) \subseteq F(\gamma(\alpha(\text{Init})))$	(mon. F)
$\Rightarrow$	$F(\text{Init}) \subseteq F(\gamma(\alpha(\text{Init})))$	(mon. F)
$\Rightarrow$	$\alpha \circ F(\text{Init}) \subseteq \alpha \circ F(\gamma(\alpha(\text{Init})))$	(mon. α)
$\Rightarrow$	$\alpha \circ F(\text{Init}) \subseteq F^\# \circ \alpha \circ \gamma(\alpha(\text{Init}))$	(using 23)
$\Rightarrow$	$\alpha \circ F(\text{Init}) \subseteq F^\#(\alpha(\text{Init}))$	(red. of $\alpha \circ \gamma$ and mon. of $F^\#$)
$\Rightarrow$	$\gamma \circ \alpha \circ F(\text{Init}) \subseteq \gamma \circ F^\#(\alpha(\text{Init}))$	(mon. γ)
$\Rightarrow$	$F(\text{Init}) \subseteq \gamma \circ F^\#(\alpha(\text{Init}))$	(ext. of $\gamma \circ \alpha$)

Iterating over F, we obtain

$$\forall n, F^n(\text{Init}) \subseteq \gamma \circ F^{\#n}(\alpha(\text{Init})) \tag{24}$$

$$\text{lfp}_\emptyset F \subseteq \gamma(\text{lfp}_\bot F^\#) \tag{25}$$

and therefore

$$\mathfrak{C} \subseteq \gamma(\mathfrak{C}^\#)$$

2.5.2 *Effective computation:* Kleene *iterations and widening*

When the abstract domain is fitted with a complete lattice structure[4], this fixpoint could be accurately computed by Kleene iterations:

$$\mathfrak{C} = \text{lfp}_\emptyset F = \lim_{n \to \infty} F^n(\bot) \tag{26}$$

4 In theory, it is only required to admit least upper bound for ascending chains. In addition, $F^\#$ should be join complete on these chains, ie. upper continuous, ie. for all chain $w_0, w_i, \ldots, F^\#(\cup_i w_i)) \cup_i F^\#(w_i)$.

In case of infinite ascending chains of iterates, one relies on so-called *widening* operator to ensure convergence in a finite number of iterations. This operator acts as a rough join operator but has better convergence properties. It is however pessimistic since it introduces numerous spurious states in the abstract representation.

Remark 2 (Relative performance) *In general, SMT-based methods such as MC and DM perform better on disjunctive or integer based properties. SMT solvers are based on a SAT core and a set of solvers for axiomatized theories. These solvers perform generally well on at most linear properties and systems.*

AI typically performs better on the synthesis of numerical invariants because disjunctions are computed within the abstract representation, using the abstract join $\sqcup$, instead of being kept explicitly. Abstractions exist that postpone the interpretation of these disjunctions such as partitioned analyses [Fer05a; Gar08], disjunctive completion of domains [CC92; CC79], or delayed join [Min04] to regain precision but cost too much to be used in a systematic manner .

To summarize, for the most common setting, the effective use of abstract interpretation is the following:

1. Express the (collecting) semantics as a fixpoint of a monotonic function F over a lattice of properties. In our case, properties are sets of states.
2. Exhibit an abstract domain for set of states, defining abstraction and concretization functions, lattice operations such as join, and a sound abstract counterpart $F^\#$ of F.
3. Abstract initial states and compute with KLEENE iterations the least fixpoint in the abstract.
4. In case of convergence issue, rely on widening to converge to a bigger fixpoint in the abstract.
5. The concretization of this abstract fixpoint is a sound over-approximation of the concrete one.

In practice over-approximation is caused:

- by the set of properties represented or expressible in the abstract domain (linear relationships, intervals, ...); an abstract domain may be unable to represent or capture some specific property while another one will.
- by the abstraction function and the set of abstract counterparts of concrete functions. For example in case of difficult precise definitions of a function such as exp, one can approximate it soundly by a function returning the $\top = \mathbb{R}$ value. While sound, this definition is largely imprecise and will lead to more abstraction when this exp function is used. Another issue appears in presence of non linear expression analyzed with an abstract domain restricted to linear properties. In that case the non linear expression has to be soundly over-approximated, leading to additional imprecision. This imprecision is caused by the abstract transformers.
- by the use of widening introducing additional abstraction to the computed element.

2.6 NEED FOR INDUCTIVE INVARIANTS

Basically all formal methods rely on the expression of the property of interest as an inductive invariant over the system semantics. In practice all these techniques benefit from additionally provided invariants. We summarize the use of invariant in the different techniques.

2.6.1 *Loop invariants for deductive methods*

As mentioned in Section 2.3.1 the analysis of loop with deductive methods requires invariants to be provided to capture the loop semantics. While simple invariants may be easily provided, they may be too weak to capture precisely the loop semantics. For example the loop invariant expressed in Floyd euclidean division flowchart in Fig. 2.2 is extremely precise: $R, X, Q \geqslant 0, Y > 0, X = R + QY$.

If one considers Dijkstra's predicate transformer rule for loops in Eq. 6, one can see that the remaining property is essentially I, the invariant provided. Invariants for loops act as the cut-rule in proofs. A sound yet weak invariant will generate a weaker precondition that guarantees the post-condition, but not the weakest. The proof that the provided pre-condition imply the weaker precondition may be unfeasible.

These loop invariants are either manually provided [Wan+16a] or computed by other means such as abstract interpretation [Moy08].

2.6.2 *Inductive invariants to reinforce transition relation in SMT-based model-checking*

Similarly, SMT-based model-checking is essentially based on induction. As mentioned in Section 2.4 different approaches are used to address the lack of availability of the collecting semantics $[\![f]\!]_{coll}$. While looking different, k-induction, PDR or invariant injection all amount to the characterization of invariants of $[\![f]\!]_{coll}$.

2.6.3 *Inductive invariants to strengthen abstract interpretation fixpoint computation*

Abstract interpretation aims at computing inductive invariants. The definition of abstraction through sound abstract domains enables their composition to improve the analysis results. Since the Cartesian product of two complete lattices is also a complete lattice and since Galois connections can be similarly composed, one can easily define as a sound analysis an analysis that rely on multiple abstractions at the same time. Another interesting construct is the domain reduction: it enables multiple domains to communicate and refine their own (sound) properties. Let us illustrate that notion on a simple example.

Example 2 *Consider a set of integer values* $\Sigma = \mathbb{N}$ *and the three following abstractions: sign, interval, and parity.*

$$\alpha_{\mathtt{sign}}(S) = \begin{cases} \bot_{\mathtt{sign}} \textit{ when } S = \emptyset \\ 0 \textit{ when } S = \{0\} \\ + \textit{ when } \forall s \in S, s \geqslant 0 \\ - \textit{ when } \forall s \in S, s \leqslant 0 \\ \top_{\mathtt{sign}} \textit{ otherwise} \end{cases}$$

$$\alpha_{\mathtt{interval}}(S) = (\min S, \max S)$$

$$\alpha_{\mathtt{parity}}(S) = \begin{cases} \bot_{\mathtt{parity}} \textit{ when } S = \emptyset \\ \mathrm{Odd} \textit{ when } \forall s \in S, s \mod 2 = 1 \\ \mathrm{Even} \textit{ when } \forall s \in S, s \mod 2 = 0 \\ \top_{\mathtt{parity}} \textit{ otherwise} \end{cases}$$

Abstracting a set by the sign of its elements is always less precise than representing more finely the set of values by its lower and upper elements. But those two abstract representations are not comparable with the abstraction that determines whether all values are even or odd.

Abstractions could be however combined. If both intervals and parity are of interest to us, one can analyze the semantics of the program with both abstractions in the same computation and represent more precisely the interval and parity associated to the abstract set of values. This could lead to further improvements. For example, an interval abstraction may have identified a set $[1, 1000]$ *of reachable values while the parity abstraction guarantees that all values are odd. In that case the interval representation can be refined into* $[1, 9999]$.

One of the major application of abstract interpretation is the tool Astrée that was designed specifically for the analysis of the Airbus A3xx family control command systems. It combines numerous abstract domains with complex reductions [Cou+07]. One of these domains is specifically focused on second-order linear filters in order to bound their reachable states [Fer04].

3

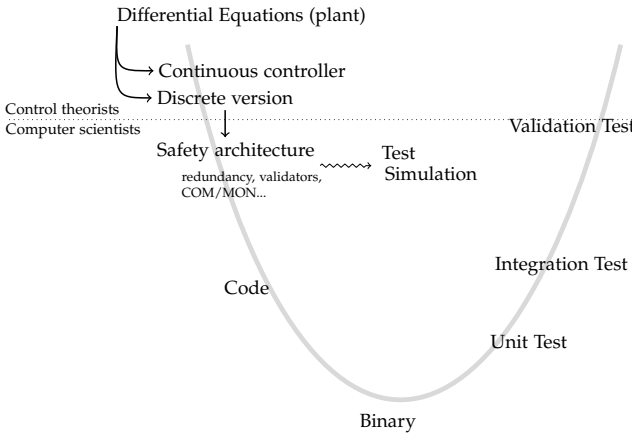
CONTROL SYSTEMS

All our analyses are focused on control systems. We sketch here their typical development and refer the reader to classical books such as Åström/Murray book [AM08] or Levine's control handbook [Lev96] for more details on control system design.

Historically control design started in the continuous world: a system had to be controlled, its dynamics was captured by the equations of physics, for example using ordinary differential equations (ODE). Then, control theory provides means to build a controller: another system that, was used in combination with the system to be controlled, is able to move the system in the requested state.

The Figure 3.1 presents a typical process leading to the development of a controller in the aerospace domain. We now give an idea of each steps.

Figure 3.1 Current development process



3.1 CONTROLLERS DEVELOPMENT PROCESS

Let us give a naive yet representative process leading to the definition of a control system.

SYSTEM DYNAMICS At first an identification phase is required to obtain the *plant* dynamics. This identification phase can be complex and rely on various means to describe the system dynamics: a finite element structural model relying on a precise modeling of the aircraft shape, or a rough point mass system with a given number of degrees of liberty. For example, for a system like an aircraft able to move in a volume, one can characterize roughly its dynamics by 12 equations defining its position and velocity in a an orthogonal basis as well as its angle and angular velocity along the three Euler angles (Yaw, Pitch, Roll).

Typically, one characterizes the sum of forces applied to the system (gravity, thrust, lift and drag in the case of an aircraft) as we learn in high school. This set of constraints defines the differential equations capturing the dynamics of the system.

LINEARIZATION – TRANSFER FUNCTIONS Controlling non linear dynamics is still an active area of research. In practice, in the conservative aerospace industry, most basic controllers are still defined with old-school linear methods. For these methods the dynamics has to be linear. Since linearized system are not fully representative far away from the linearization point, multiple such points are defined, leading to multiple linearized versions of the dynamics. This can be done using Taylor expansion for example. The general ODE can then be expressed, locally, as linear differential equation (LDE) expressed over a single input and a single output signals. The dynamics described by this LDE can be interpreted as a function mapping this input signal $x(t)$ in the output one $y(t)$. In this continuous setting, one defines this function as the linear mapping relating the Laplace transforms of $x(t)$ and $y(t)$. The transfer function is expressed to map those two Laplace transforms.

Control design then provides tools to build a feedback controller: another transfer function which, when associated to the initial transfer function, provides the expected behavior. Various techniques exist to synthesize such a controller: proportional, lead-lag, proportional-integral-derivative (PID), ...

ANALYSIS The produced controller can be evaluated with respect to control-level properties. A controller drives the plant in the desired state by minimizing the error between the controller command and the current plant state. This feedback system, the closed-loop system, is analyzed with respect to stability, robustness and

performance. Stability and robustness capture the damping of the system, its ability to converge to goal even in presence of noise in the feedback loop. Performance evaluates the speed of convergence and the shape of the feedback response (overshoot, number of oscillations, settling time, ...).

DISCRETIZATION This controller is meant to be embedded in an onboard computer and to interact with the system sensors and actuators. Depending on the speed of each of these devices, and the available computing resources, an appropriate rate of discretization is chosen. For example a typical control law for an aircraft runs at 100Hz. But a trajectory planning controller may run at a much lower speed.

COMPLETE CONTROLLERS Once a discrete controller has been obtained for a linearized version of the plant, a more global one is obtained by combining local controllers. One of the approaches is to synthesize a controller for each linearization point while keeping the same controller synthesis method. Since the previous steps characterized single input single output (SISO) subcontrollers, it is easy to switch the controller depending on the input value. When considering an intermediate value between two linearization, one can characterize the linear interpolation of controller gains, the coefficients synthesized for each local controller.

Moreover, additional constructs are introduced to account for divergence of integrators in case of a break in the closed-loop system. Saturations or Anti-windups (cf. § 12) act as such and enable the output to remain within given bounds.

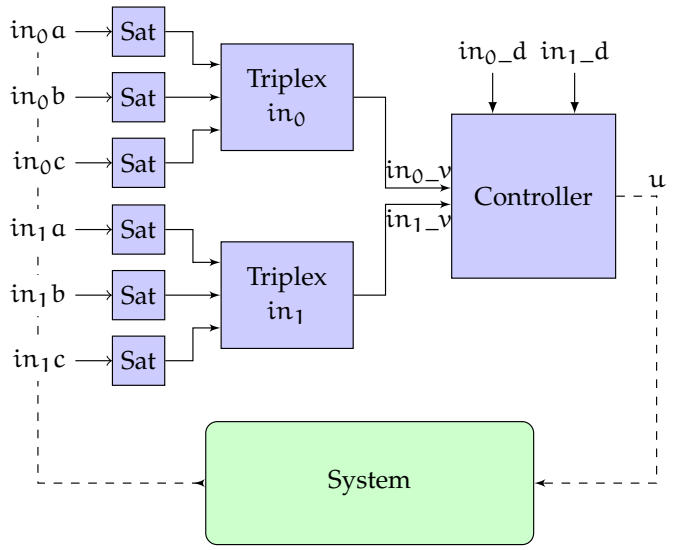
INTEGRATION: SAFETY ARCHITECTURE In critical applications the controller will not be directly embedded on the target platform but rather used in conjunction with a safety architecture used to obtain a fault tolerant system.

This safety architecture is usually identified before the design of the controller itself since it identifies early in the process development the potential causes of failure and their impact on the various systems. These failures can range from faulty parts such as sensors generating false data, a transient error such as a single event upset (SEU) or a multiple bits upset (MBU), or a bug in software.

This leads to local impacts at the control level with the fusion of input data in case of redundancy in the sensors: validators, alarm detection, voters, ... The alarm detection mechanisms typically check that the read value lies in an expected range and emit different kinds of alarm signals when a value outside the legal scope is detected.

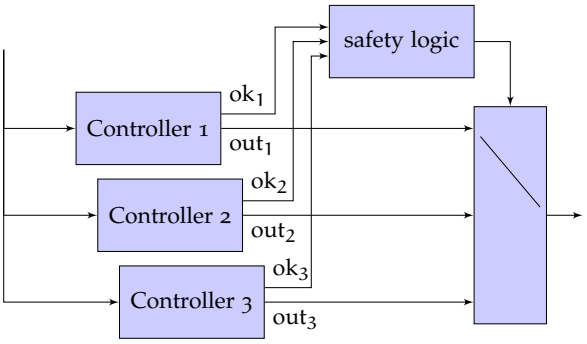
Figure 3.2 presents an example of such architecture with triplicated input sensors.

Figure 3.2 Example of a controller with two triplicated inputs



At the system level, more complex safety patterns allow the execution of the controller in a distributed fashion, on multiple computers. These different computers may also run different implementations, to account for hardware (CPU or RAM) and software errors. For example, a first pattern can sequence redundant implementations with only a single one in control as shown on Figure 3.3. Another one, called COM/MON for Command/Monitor is based on the notion of computer-local observers that detect whether the current output is valid or not. In case of local failure the primary computer leaves the command to the secondary one.

Figure 3.3 Triplication of the controller



CODE GENERATION AND V&V Last, once the complete design has been done, the final code is created. As developed later in Chapter. 11 the code can be automatically generated from model description, or directly coded in C code, for example.

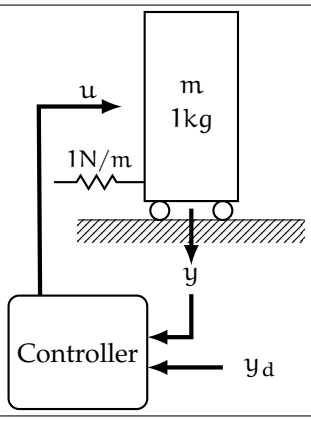
This code is very specific to control system. It consists mainly of an endless loop, acquiring input data, performing one step of computation, propagating orders to actuators and waiting the next clock tick.

```c
while (true) {
in = read_sensors (); // read input data
*state = ctl(*state, in);
actuators(*state); // send orders
wait_next_tick();
}
```

At the verification level, in addition to functional requirements such as the validity of the safety architecture or the alarm mechanism, one needs to prove that the generated code will satisfy the timing constraints imposed by the discretization, as well as prove the absence of runtime errors, such as overflows, that will impact drastically the global behavior of the controlled system, as it happened in the failure of the first Ariane 5 flight [SIAM96].

3.2 A SIMPLE LINEAR SYSTEM: SPRING-MASS DAMPER

Figure 3.4 Motivating example: a spring-mass damper



When considering linear systems: plant and controller, we reuse the running example of [Fér10; FWP90]. This dynamical system is composed of a single mass and a single spring.

1 This is explained with more details in Chapter. 7

3.2.1 *Continuous dynamics: plant and lead-lag controller*

First the plant dynamics is characterized by the following ODE:

$$\frac{d}{dt}x_p = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} x_p + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u \tag{27}$$

where x_p denotes $\begin{bmatrix} z & \dot{z} \end{bmatrix}$ the position and velocity of the mass with respect to the origin. The sensor of the plant provides the position z.

The control is performed by a lead-lag controller obtained through classical control recipes where the input y_c is defined as the saturation in the interval $[-1, 1]$ of $y - y_d$ with y the measure of the mass position and $|y_d| \leqslant 0.5$ a bounded command.

The transfer function of the synthesized controller is:

$$u(s) = -128 \cdot \frac{s+1}{s+0.1} \cdot \frac{s/5+1}{s/50+1} y_c(s) \tag{28}$$

The transfer can be expressed a continuous linear controller using a realization[1] of the above transfer function:

$$\begin{aligned} \frac{d}{dt}x_c &= \begin{bmatrix} -50.1 & 5.0 \\ 1.0 & 0.0 \end{bmatrix} x_c + \begin{bmatrix} 100 \\ 0 \end{bmatrix} \mathrm{SAT}(y_k - y_k^d) \\ u &= \begin{bmatrix} 564.48 & 0 \end{bmatrix} x_c - 1280 \end{aligned} \tag{29}$$

where $\mathrm{SAT}(x)$ denotes the saturation of signal x to 1:

$$\mathrm{SAT}(x) = \begin{cases} -1 & \text{when } x < -1 \\ 1 & \text{when } x > 1 \\ x & \text{otherwise} \end{cases}$$

3.2.2 *Discrete plant dynamics*

When producing the embedded controller, the continuous model is discretized at a given rate of execution. This leads to embedded runtime systems which are executed on a platform at the given rate. The rate is chosen according to both the requirements in terms of hardware – one cannot run heavy computation at 1GHz – and in terms of performance – a controller feedback every second may be too slow to control an unstable system such as an inverted pendulum. Typical rate to maintain an aircraft stability is 100Hz.

In order to enable the later analyses, we also provide a discretized version of the plant dynamics. Both controller and plant have been discretized at an execution rate of 100Hz.

The plant is described by a linear system over the state variables $p = [x_{p1}\ x_{p2}]^\intercal \in \mathbb{R}^2$, characterized by the matrices $A_P \in \mathbb{R}^{2\times2}$, $B_P \in \mathbb{R}^{1\times2}$ and $C_P \in \mathbb{R}^{2\times1}$ where u denotes the actuator command of the plant and y the projection of the plant state p over the y sensor:

$$\begin{aligned} p_{k+1} &= A_P p_k + B_P u_k \\ y_{k+1} &= C_P p_{k+1} \end{aligned} \tag{30}$$

with

$$A_P := \begin{bmatrix} 1 & 0.01 \\ -0.01 & 1 \end{bmatrix} \quad B_P := \begin{bmatrix} 0.00005 \\ 0.01 \end{bmatrix} \quad C_P := \begin{bmatrix} 1 & 0 \end{bmatrix}$$

3.2.3 *Discrete controller dynamics*

The controller without saturation is similarly described by a linear system over the state variables $c = [x_{c1}\ x_{c2}]^\intercal \in \mathbb{R}^2$, controlled by both the feedback from the plant sensors $y \in \mathbb{R}^{d_y}$ and the user command $y_d \in \mathbb{R}$, and parametrized by the four real matrices $A_C \in \mathbb{R}^{2\times2}$, $B_C \in \mathbb{R}^{1\times2}$, $C_C \in \mathbb{R}^{2\times1}$ and $D_C \in \mathbb{R}$:

$$\begin{aligned} c_{k+1} &= A_C c_k + B_C(y_k - y_{d,k}) \\ u_{k+1} &= C_C c_{k+1} + D_C(y_{k+1} - y_{d,k+1}) \end{aligned} \tag{31}$$

with

$$A_C := \begin{bmatrix} 0.4990 & -0.05 \\ 0.01 & 1 \end{bmatrix} \qquad B_C := \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

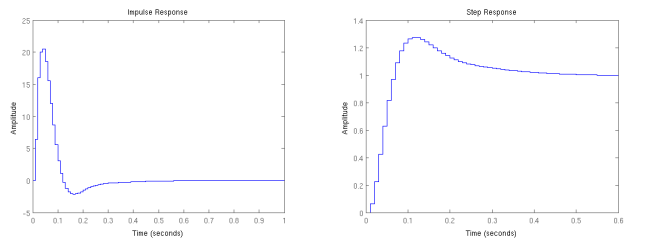
$$C_C := \begin{bmatrix} 564.48 & 0 \end{bmatrix} \qquad D_C := -1280$$

These numerical values have been obtained by a first-order Euler discretization of the continuous controller.

3.2.4 *Closed-loop system*

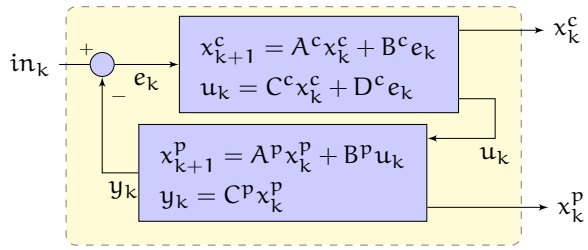
The closed-loop system can be characterized and evaluated. Fig. 3.5 presents the impulse and step response of the closed-loop system.

Figure 3.5 Impulse and Step response for the controlled spring mass damper.



Let us first consider a version of the closed-loop system, without saturation:

Figure 3.6 Closed-loop system.



System without saturation

The resulting closed-loop system is defined by considering Equations (30) and (31) at once. It can be expressed over the state space $x := [c\ p]^\intercal$ as

$$x_{k+1} = Ax_k + By_{d,k} \tag{32}$$

with

$$\begin{aligned} A &:= \begin{bmatrix} A_C & B_C C_P \\ B_P C_C & A_P + B_P D_C C_P \end{bmatrix} \\ &= \begin{bmatrix} 0.499 & -0.05 & 1 & 0 \\ 0.01 & 1 & 0 & 0 \\ 0.028224 & 0 & 0.936 & 0.01 \\ 5.6448 & 0 & -12.81 & 1 \end{bmatrix} \end{aligned}$$

$$B := \begin{bmatrix} -B_C \\ -B_P D_C \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 0.064 \\ 12.8 \end{bmatrix}$$

From that formalization, it is possible to characterize a virtual implementation of the closed system as a program. Figure 3.7 displays such code. A control flow graph analysis such as our Kleene based graph reconstruction abstract domains [RG13] can extract the associated system representation of Figure 3.8.

Figure 3.7 Analyzed code for the closed-loop system.

```
xc1 = xc2 = xp1 = xp2 = 0;
while (1) {
  yd = acquire_input();
  assert(yd >= -0.5 && yd <= 0.5);
  oxc1 = xc1; oxc2 = xc2; oxp1 = xp1; oxp2 = xp2;
  xc1 = 0.499 * oxc1 - 0.05 * oxc2 + (oxp1 - yd);
  xc2 = 0.01 * oxc1 + oxc2;
  xp1 = 0.028224 * oxc1 + oxp1 + 0.01 * oxp2
        - 0.064 * (oxp1 - yd);
  xp2 = 5.6448 * oxc1 - 0.01 * oxp1 + oxp2
        - 12.8 * (oxp1 - yd);
  wait_next_clock_tick();
}
```

Remark 3 *This corresponds to the system presented in Equation (32) with the input y_d bounded by 0.5 ($|y_{d,k}| \leqslant 0.5$ for all k).*

Figure 3.8 Control flow graph for code of Figure 3.7.

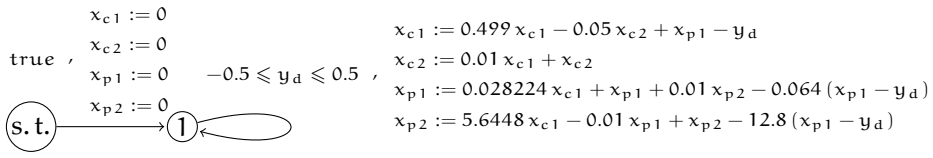
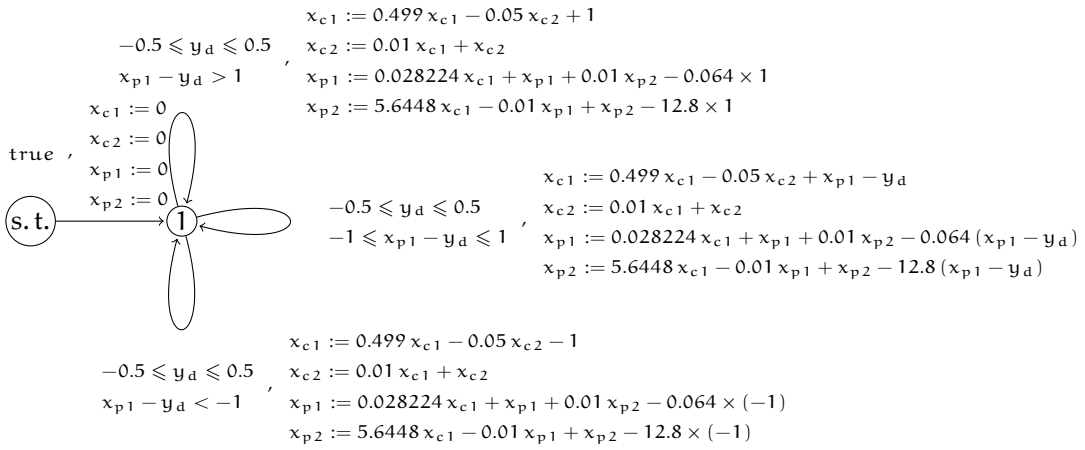


Figure 3.9 Control flow graph for the system with a saturation.



System with saturation

Similarly, the more realistic setting integrating the saturation over $(y - y_d)$ will be defined by the system:

$$x_{k+1} = Ax_k + B\,\mathrm{SAT}(Cx_k - y_{d,k}) \tag{33}$$

where

$$A := \begin{bmatrix} A_C & 0 \\ B_P C_C & A_P \end{bmatrix} = \begin{bmatrix} 0.499 & -0.05 & 0 & 0 \\ 0.01 & 1 & 0 & 0 \\ 0.028224 & 0 & 1 & 0.01 \\ 5.6448 & 0 & -0.01 & 1 \end{bmatrix}$$

$$B := \begin{bmatrix} B_C \\ B_P D_C \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ -0.064 \\ -12.8 \end{bmatrix} \qquad C := \begin{bmatrix} 0 \\ C_P \end{bmatrix}^T = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}^T$$

and SAT is defined as

$$\mathrm{SAT}(x) = \begin{cases} -1 & \text{if } x < -1 \\ x & \text{if } -1 \leqslant x \leqslant 1 \\ 1 & \text{if } x > 1 \end{cases}$$

The control flow graph extracted by our analysis is presented in Figure 3.9.

Part II

INVARIANT SYNTHESIS: CONVEX-OPTIMIZATION BASED ABSTRACT INTERPRETATION

4

DEFINITIONS – BACKGROUND

This part focuses on the computation of non linear numerical invariants for discrete controllers. As mentioned in the motivation part, controllers are usually designed in a continuous setting and then discretized; in both cases a semantics in the real field is assumed. The semantics of interest, in this context, is then a discrete dynamical system with a real semantics. Then, those simple controllers are combined with simple mechanisms such as switches, interpolation of gains, saturations, anti-windups, etc. In order to be able to analyze systems as complex as the ones embedded in aircraft, we extend the considered semantics to account for piecewise behaviors.

Regarding the analysis of those semantics, we were initially motivated by applying abstract interpretation on controller programs. We then experimented a long known result for control people: "stable linear controllers admit quadratic Lyapunov functions". However, most state-of-the-art abstract domains were abstracting states through linear properties. Furthermore, the current trend was to compute even weaker abstractions, such as octagons [Mino6], to control the complexity of the analyses and obtain non trivial results in reasonable time. When manipulating non linear abstractions, the classical Kleene based approach to fixpoint computation does not seem to be very efficient or appropriate: non linear subspaces were not easily fitted with a lattice structure – in other words a least upper bound operator was not as obvious as it is for finite sets of convex polyhedra or intervals. Following the path of control scientists we chose to rely on numerical tools, in our case convex optimization, to solve the so-called Lyapunov equations.

The current chapter presents our formalisms to describe discrete dynamical systems and gives an overview on the convex optimization tools and methods we used to compute our analyses. The following chapters develop our contributions.

4.1 DISCRETE DYNAMICAL SYSTEMS

A dynamical system is a typical object used in control systems or in signal processing. In some cases, it is eventually implemented in a program to perform the desired feedback control to a cyber physical system.

Definition 4.1 (State space) *Let Σ be the state space, a set of states. A dynamical system computes an infinite sequence of states Σ starting from an initial state $\mathtt{init} \in X^{\mathrm{Init}} \subseteq \Sigma$. The dynamics of the system is defined by a function $f : \Sigma \to \Sigma$. In some cases, the dynamics is also perturbed – or controlled, depending on the point of view – by an external signal, ie. sequences of values. Let us call them inputs $u \in X^{\mathrm{In}}$. The system map is then defined as $f : \Sigma \times X^{\mathrm{In}} \to \Sigma$. Let $\bar{\Sigma}$ be the state-input space defined as $\Sigma \times X^{\mathrm{In}}$.*

Definition 4.2 (Trajectory) *A trajectory of the system is defined by an initial state $\mathtt{init} \in X^{\mathrm{Init}}$ and an infinite sequence of inputs $(u_k)_{k \geqslant 1} \in X^{\mathrm{In}}$:*

$$x_0 = \mathtt{init} \qquad\qquad x_{n+1} = f(x_n, u_n)$$

Language-wise, model based languages such as Lustre [Hal+91], Ansys Scade, or Matlab Simulink provide primitives to build these dynamical systems or controllers relying on simpler constructs. In terms of programs, such dynamical systems can easily be implemented as a *while true loop* initialized by the initial state and performing the update f. The simplest systems are usually directly coded in the target language, eg. C code, while more advanced systems are *compiled* through autocoders: Lustre compilers, Scade KCG or Matlab Real Time Workshop (RTW).

Let us sketch a typical implementation: the variable u is being read from an external source, eg. as a mutable variable or an IO call.

C
```
x = i;
while true {
  u = read();
  x = f(x, in);
}
```

Most systems perform an action at each computation step. In case of controllers, the action typically moves some actuators in order to impact the controlled system. This generates an output signal , a sequence of produced values $y \in X^{\mathrm{Out}}$. This output is computed by a function $g : \Sigma \times X^{\mathrm{In}} \to X^{\mathrm{Out}}$.

```c
x = i;
while true {
  in = read();
  x = f(x, u);
  y = g(x, u);
}
```

A discrete dynamical system is then defined by the following sets $\Sigma, X^{Init}, X^{In}, X^{Out}$ and functions f, g.

In the following, we specialize this description depending on the considered sets and functions: linear systems, piecewise linear systems, and polynomial ones.

Again, these descriptions are provided at the model level or could be extracted from the implementation as we did for linear systems [RG13]. In order to simplify this extraction phase, or to understand it more easily, we assume without loss of generality that the analyzed programs are written in Static Single Assignment (SSA) form, that is each variable is initialized at most once.

As a last remark, since we are first interested only in the internal state x of the system, the output part is often neglected.

4.1.1 *Linear systems*

This simplest systems are composed of a single loop and a linear update. While they could seem over simple to the non expert, most controllers are linear, from rocket stabilization controllers, to aircraft controllers or satellite attitude and orbital control systems (AOCS).

The basic control literature mentions proportional controllers (P), proportional derivative (PD), or proportional-integral-derivative (PID) ones. In all cases, these are linear controllers. In order to obtain more precision, the *order* of the linear controller, ie. the size of its state space Σ could be extended, considering a more complex system.

A linear controller is typically implemented by the following code, for a system with $\Sigma = \mathbb{R}^2$ and $X^{In} = X^{Out} = \mathbb{R}$:

```c
x0 = i0;
x1 = i1;
while true {
  in = read();
  nx0 = a00 * x0 + a01 * x1 + b00 * in;
  nx1 = a10 * x0 + a11 * x1 + b10 * in;
  y = c00 * x0 + c10 * x1 + d00 * in;
  x0 = nx0;
  x1 = nx1;
}
```

In all systems, assignments of variables are performed using only *parallel assignments*. At the implementation level, this imposes to keep a copy of the variable values before the final updates $x_i = nx_i$.

Definition

In this first setting, a linear system is defined over a system state in Σ, represented as a vector of $\mathbb{R}^d$, with inputs in X^{In}, represented as a vector of $\mathbb{R}^m$, and by a pair of matrices $A \in \mathbb{R}^{d\times d}$ $B \in \mathbb{R}^{d\times m}$. Its output in X^{Out}, represented by a vector of $\mathbb{R}^o$, is computed similarly by a pair of matrices $C \in \mathbb{R}^{o\times d}$ $D \in \mathbb{R}^{o\times m}$.

Such system is defined by the two functions:

$$\begin{cases} f : \mathbb{R}^d \times \mathbb{R}^m \to \mathbb{R}^d \\ \qquad (x,u) \mapsto Ax + Bu \\ g : \mathbb{R}^d \times \mathbb{R}^m \to \mathbb{R}^o \\ \qquad (x,u) \mapsto Cx + Du \end{cases}$$

Linear Controller Example

Let us consider the following Linear Quadratic Gaussian (LQG) Regulator:

```c
double x[3] = {0, 0, 0};
double nx[3];
double in;
while (1) {
  in = acquire_input();
  nx[0] = 0.9379*x[0] - 0.0381*x[1] -
      0.0414*x[2] + 0.0237*in;
  nx[1] = -0.0404*x[0] + 0.968*x[1] -
      0.0179*x[2] + 0.0143*in;
  nx[2] = 0.0142*x[0] - 0.0197*x[1] +
      0.9823*x[2] + 0.0077*in;
  x[0] = nx[0]; x[1] = nx[1]; x[2] = nx
      [2];
  wait_next_clock_tick();  // a tick every
       10 ms for instance
}
```

This characterizes the following dynamical system.

Example 3 (Linear system example) *Let* $\Sigma = \mathbb{R}^3$, $X^{Init} = \{(0,0,0)\}$, *and* $X^{In} = \mathbb{R}$, *with the following matrices* A *and* B*:*

$$A := \begin{bmatrix} 0.9379 & -0.0381 & -0.0414 \\ -0.0404 & 0.968 & -0.0179 \\ 0.0142 & -0.0197 & 0.9823 \end{bmatrix} \quad B := \begin{bmatrix} 0.0237 \\ 0.0143 \\ 0.0077 \end{bmatrix}$$

4.1.2 *Switched linear systems: constrained piecewise affine discrete-time systems*

Most systems are not purely linear. The programs or systems we consider here are composed of a single loop with possibly a complicated switch-case type loop body.

Our switch-case loop body is supposed to be written as a nested sequence of *ite* statements, or as a switch:

C
```
x = i;
while true {
  in = read();
  switch
    c1 → x = f1(x, in);
    c2 → x = f2(x, in);
    c3 → x = f3(x, in);
    _  → x = f4(x, in);
}
```

Moreover, we suppose that the analyzed programs are written in affine arithmetic, both the switch conditions c_i and the associated update functions f_i. Consequently, the programs analyzed here can be interpreted as constrained piecewise affine discrete-time systems.

Polyhedral Partitioning of $\bar{\Sigma}$

The term piecewise affine means that there exists a polyhedral partition $\{X^i, i \in \mathbb{I}\}$ of the state-input space $\bar{\Sigma} \subseteq \mathbb{R}^{d+m}$ such that for all $i \in \mathbb{I}$, the dynamic of the system is affine and represented by the following relation for all $k \in \mathbb{N}$:

$$\text{if } (x_k, u_k) \in X^i,\ x_{k+1} = A^i x_k + B^i u_k + b^i, k \in \mathbb{N} \quad (34)$$

where $A^i \in \mathbb{R}^{d\times d}$, $B^i \in \mathbb{R}^{d\times m}$ and $b^i \in \mathbb{R}^d$. As in the linear case, the variable $x \in \mathbb{R}^d$ refers to the state variable and $u \in \mathbb{R}^m$ refers to some input variable.

We define a partition of the state-input space as a family of nonempty sets X^i such that:

$$\bigcup_{i\in\mathbb{I}} X^i = \bar{\Sigma},\ \forall i, j \in \mathbb{I},\ i \neq j, X^i \cap X^j = \emptyset. \quad (35)$$

In the current setting, since X^i are convex polyhedra, we characterize polyhedral partitions of the state-input space. From now on, we call X^i *cells*.

Affine conditions: strict and weak affine convex constraints

Cells $\{X^i\}_{i\in\mathbb{I}}$ are convex polyhedra which can contain both strict and weak inequalities.

Definition 4.3 (Cells as convex polyhedra) *Cells can be represented by a $n_i \times (d+m)$ matrix T^i and a vector $c^i \in \mathbb{R}^{n_i}$. We denote by $\mathbb{I}_s^i$ the set of indices which represent strict inequalities for the cell X^i, denote by T_s^i and c_s^i the parts of T^i and c^i corresponding to strict inequalities and by T_w^i and c_w^i the one corresponding to weak inequalities. Finally, we have the matrix representation given by Formula (36).*

$$X^i = \left\{ \begin{pmatrix} x \\ u \end{pmatrix} \in \mathbb{R}^{d+m} \,\middle|\, T_s^i \begin{pmatrix} x \\ u \end{pmatrix} \ll c_s^i,\ T_w^i \begin{pmatrix} x \\ u \end{pmatrix} \leqslant c_w^i \right\} \quad (36)$$

We use the following notations: $y \leqslant z$ is a partial order built as the piecewise lift of the total order over reals to vectors, meaning that for all coordinates l, $y_l \leqslant z_l$. The other relation $y \ll z$ is the strict version, meaning that for all coordinates l, $y_l < z_l$.

While the approach we propose can consider arbitrary partitioning of the system dynamics into convex cells, we infer automatically the cell's definition using the guards of the switch case constructs.

Homogenization: encoding affine system as a linear one

In order to simplify the following analyses, it is easier to consider a linear system rather than an affine one. Therefore, we define a homogeneous flavor of the system dynamics: instead of considering a system state in $\mathbb{R}^d$ with inputs in $\mathbb{R}^m$, we manipulate system states in $\mathbb{R}^{1+d+m}$. We will need homogeneous versions of update functions and thus introduce the $(1+d+m)\times(1+d+m)$ matrices F^i defined as follows:

$$F^i = \begin{pmatrix} 1 & 0_{1\times d} & 0_{1\times m} \\ b^i & A^i & B^i \\ 0 & 0_{m\times d} & \mathrm{Id}_{m\times m} \end{pmatrix} \quad (37)$$

where $\mathrm{Id}_{m\times m}$ denotes the identity matrix of dimension $m \times m$.

The system defined in Equation (34) can be rewritten as $(1, x_{k+1}, u_{k+1})^\intercal = F^i(1, x_{k+1}, u_k)^\intercal$. Note that, in order to obtain a square matrix, we introduce a "virtual" dynamic law $u_{k+1} = u_k$ on the input variable in Equation (37). It will not be used in the following analyses.

Let $p = \mathrm{card}(\mathbb{I})$, the global system can be defined as:

$$\begin{cases} (1, x_{k+1}, u_{k+1})^\intercal = F^1(1, x_k, u_k)^\intercal & \text{w. } (x_k, u_k) \in X^1 \\ (1, x_{k+1}, u_{k+1})^\intercal = F^2(1, x_k, u_k)^\intercal & \text{w. } (x_k, u_k) \in X^2 \\ \dots \\ (1, x_{k+1}, u_{k+1})^\intercal = F^p(1, x_k, u_k)^\intercal & \text{w. } (x_k, u_k) \in X^p \end{cases} \quad (38)$$

Piecewise Linear Discrete Dynamical System Example

Let us consider the following program. It is constituted by a single while loop with two nested conditional branches in the loop body, characterizing four cells.

```c
(x,y)∈ [−9,9] × [−9,9];
while(true)
ox=x;
oy=y;
read(u); \\u ∈ [−3,3]
if (-9*ox+7*y+6*u<5){
  if(-4*ox+8*oy-8*u<4){
    x=0.4217*ox+0.1077*oy+0.5661*u;
    y=0.1162*ox+0.2785*oy+0.2235*u-1;
  }
  else { \\4*ox-8*oy+8*u<-4
    x=0.4763*ox+0.0145*oy+0.9033*u;
    y=0.1315*ox+0.3291*oy+0.1459*u+9;
  }
}
else { \\9*ox-7*y-6*u<-5
  if(-4*ox+8*oy-8*u<4){
    x=0.2618*ox+0.1107*oy+0.0868*u-4;
    y=0.4014*ox+0.4161*oy+0.6320*u+4;
  }
  else { \\4*ox-8*oy+8*u<-4
    x=0.3874*ox+0.00771*oy+0.5153*u+10;
    y=0.2430*ox+0.4028*oy+0.4790*u+7;
  }
}
```

The initial condition of the piecewise affine system is $(x, y) \in [-9, 9] \times [-9, 9]$ and the polytope where the input variable u lives is $\mathcal{U} = [-3, 3]$.

We can rewrite this program as a piecewise affine discrete-time dynamical systems using our notations. We give details on the matrices T_s^i and T_w^i and vectors c_s^i and c_w^i (see Equation (36)) which characterize the cells and on the matrices F^i representing the homogeneous version (see Equation (37)) of affine laws in the cell X^i.

Example 4 (Piecewise linear system example) *Let* $\Sigma = \mathbb{R}^2$, $X^{Init} = [-9, 9] \times [-9, 9]$, $X^{In} = [-3, 3]$, $\mathrm{card}(\mathbb{I}) = 4$ *with the following matrices and vectors:*

$$F^1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0.4217 & 0.1077 & 0.5661 \\ -1 & 0.1162 & 0.2785 & 0.2235 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

$$\begin{cases} T_s^1 = \begin{pmatrix} -9 & 7 & 6 \\ -4 & 8 & -8 \end{pmatrix} \\ c_s^1 = (5\ 4)^\intercal \end{cases}, \quad \begin{cases} T_w^1 = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & -1 \end{pmatrix} \\ c_w^1 = (3\ 3)^\intercal \end{cases}$$

$$F^2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0.4763 & 0.0145 & 0.9033 \\ 9 & 0.1315 & 0.3291 & 0.1459 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

$$\begin{cases} T_s^2 = \begin{pmatrix} -9 & 7 & 6 \end{pmatrix} \\ c_s^2 = 5 \end{cases}, \quad \begin{cases} T_w^2 = \begin{pmatrix} 4 & -8 & 8 \\ 0 & 0 & 1 \\ 0 & 0 & -1 \end{pmatrix} \\ c_w^2 = (-4\ 3\ 3)^\intercal \end{cases}$$

$$F^3 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -4 & 0.2618 & 0.1177 & 0.0868 \\ 4 & 0.4014 & 0.4161 & 0.6320 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

$$\begin{cases} T_s^3 = \begin{pmatrix} -4 & 8 & -8 \end{pmatrix} \\ c_s^3 = 4 \end{cases}, \quad \begin{cases} T_w^3 = \begin{pmatrix} 9 & -7 & -6 \\ 0 & 0 & 1 \\ 0 & 0 & -1 \end{pmatrix} \\ c_w^2 = (-5\ 3\ 3)^\intercal \end{cases}$$

$$F^4 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 10 & 0.3874 & 0.0771 & 0.5153 \\ 7 & 0.2430 & 0.4028 & 0.4790 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

$$\begin{cases} T_w^4 = \begin{pmatrix} 9 & -7 & -6 \\ 4 & -8 & 8 \\ 0 & 0 & 1 \\ 0 & 0 & -1 \end{pmatrix} \\ c_w^4 = (-5\ -4\ 3\ 3)^\intercal \end{cases}$$

4.1.3 *Piecewise Polynomial Systems*

A last flavor of considered systems is the further extension to polynomial constraints and update: piecewise polynomial discrete-time dynamical systems. Let us first recall some definitions of polynomial functions in $\mathbb{R}^d$.

Definition 4.4 (Polynomial functions of $\mathbb{R}^d$) *A function* f *from* $\mathbb{R}^d$ *to* $\mathbb{R}$ *is a polynomial if and only if there exists* $k \in \mathbb{N}$, *a family* $\{c_\alpha \mid \alpha = (\alpha_1, \ldots, \alpha_d) \in \mathbb{N}^d, |\alpha| = \alpha_1 + \ldots + \alpha_d \leqslant k\}$ *such that for all* $x \in \mathbb{R}^d$, $f(x) = \sum_{|\alpha| \leqslant k} c_\alpha x_1^{\alpha_1} \ldots x_d^{\alpha_d}$. *By extension a function* $f : \mathbb{R}^d \mapsto \mathbb{R}^d$ *is polynomial if and only if all its coordinate functions are polynomials. Let* $\mathbb{R}[x]$ *stands for the set of* d*-variate polynomials.*

We focus now on programs composed of a single loop with a possibly complicated switch-case type loop body.

```
x ∈ X^Init;
while true {
  case (r^1_1(x)<#0 and ... and r^1_{n_1}(x)<#0):
            x = T^1(x);
  case ...
  case (r^1_i(x)<#0 and ... and r^1_{n_i}(x)<#0):
            x = T^i(x);
}
```

Basic semialgebraic set

In this setting, conditions are expressed as a conjunction of weak polynomial inequalities $r(x) \leqslant 0$ or strict polynomial inequalities $r(x) < 0$. These functions, describing guards, are real-valued polynomials of the state-input space: $\bar{\Sigma} \to \mathbb{R}$. Such conditions characterize a *basic semialgebraic set*. Recall that a set $C \subseteq \mathbb{R}^d$ is said to be a *basic semialgebraic set* if there exist $g_1, \ldots, g_m \in \mathbb{R}[x]$ such that $C = \{x \in \mathbb{R}^d \mid g_j(x) <^{\#} 0, \forall j = 1, \ldots, m\}$, where $<^{\#}$ is used to encode either a strict $<$ or a weak $\leqslant$ inequality.

Semialgebraic Partitioning of $\bar{\Sigma}$

These basic semialgebraic sets of $\bar{\Sigma} = \mathbb{R}^{d+m}$ act as the cells we used in piecewise affine systems. They characterize a partition of the state-input space $\bar{\Sigma}$. Let $\mathbb{I}$ be the set of cells X^i, ie. basic semialgebraic sets.

$$X^i = \left\{ x \in \mathbb{R}^{d+m} \;\middle|\; \bigwedge_{1 \leqslant j \leqslant ns_i} r^i_{s_j} < 0 \bigwedge_{1 \leqslant j \leqslant nw_i} r^i_{w_j} \leqslant 0 \right\} \tag{39}$$

where ns_i and nw_i denote, respectively, the number of strong and weak polynomial constraints in the semialgebraic set X^i.

Cells X^i satisfy Eq. 35: they form a semialgebraic partition of $\bar{\Sigma}$. As a result, any element of $\bar{\Sigma}$ belong to exactly one cell X^i.

Polynomial updates

Assignments associated to each cell X^i with $i \in \mathbb{I}$ are defined by polynomial functions T^i.

$$\text{if } x_k \in X^i,\ x_{k+1} = T^i(x_k)\,. \tag{40}$$

For systems without input, we have T^i a d-variate polynomial: $T^i \in \mathbb{R}[x]$; in case of systems with input in $\mathbb{R}^m$, T^i is a polynomial function $T^i \in \mathbb{R}^{d+m} \to \mathbb{R}^d$, where each coordinate function is a polynomial $\mathbb{R}^{d+m} \to \mathbb{R}^d$.

Definition of a Piecewise Polynomial System (PPS)

We assume that $\mathbb{I}$ is finite and that the initial condition x_0 belongs to some compact basic semialgebraic set X^{Init} satisfying Eq. (39). For the program, X^{Init} is the set where the variables are supposed to be initialized in.

Definition 4.5 (Piecewise Polynomial System) *A constrained polynomial piecewise discrete-time dynamical system (PPS) is the quadruple $(X^{Init}, X^{In}, \bar{\Sigma}, \mathcal{L})$ with:*

- *$X^{Init} \subseteq \Sigma \subseteq \mathbb{R}^d$ is the compact basic semialgebraic set of the possible initial conditions;*
- *$X^{In} \subseteq \mathbb{R}^m$ is the basic semialgebraic set where the input variable lives;*
- *$\bar{\Sigma} := \{X^i, i \in \mathbb{I}\}$ is a partition as defined in Equation (35);*
- *$\mathcal{L} := \{T^i, i \in \mathcal{I}\}$ is the family of the polynomials from $\mathbb{R}^{d+m}$ to $\mathbb{R}^d$, w.r.t. the partition $\bar{\Sigma}$ satisfying Equation (40).*

Piecewise Polynomial System Example

From now on, we associate a PPS representation to each program of the form described earlier.

Let us consider a concrete example. The program below involves four variables and contains an infinite loop with a conditional branch in the loop body. Each branch update is defined by a polynomial function. The parameters c_{ij} (resp. d_{ij}) are given parameters. During the analysis, we only keep the variables x_1 and x_2 since $oldx_1$ and $oldx_2$ are just memories.

```
x1, x2 ∈ [a1, a2] × [b1, b2];
oldx1 = x1;
oldx2 = x2;
while (-1 <= 0){
  oldx1 = x1;
  oldx2 = x2;
  case : oldx1^2 + oldx2^2 <= 1 :
    x1 = c11 * oldx1^2 + c11 * oldx2^3;
    x2 = c21 * oldx1^3 + c22 * oldx2^2;
  case :  -oldx1^2 - oldx2^2 < -1
    x1 = d11 * oldx1^3 + d12 * oldx2^2;
    x2 = d21 * oldx1^2 + d22 * oldx2^2;
}
```

Example 5 (Piecewise polynomial system example) *The associated PPS corresponds to the input-empty quadruple $(X^{Init}, \emptyset, \{X^1, X^2\}, \{T^1, T^2\})$. In this case $\bar{\Sigma} = \Sigma$. We have the set of initial conditions:*

$$X^{Init} = [a_1, a_2] \times [b_1, b_2]\,,$$

the partition verifying Equation (35) is:

$$X^1 = \{x \in \mathbb{R}^2 \mid x_1^2 + x_2^2 \leqslant 1\},$$
$$X^2 = \{x \in \mathbb{R}^2 \mid -x_1^2 - x_2^2 < -1\},$$

and the polynomials relative to the partition $\{X^1, X^2\}$ are:

$$T^1(x) = \begin{pmatrix} c_{11}x_1^2 + c_{12}x_2^3 \\ c_{21}x_1^3 + c_{22}x_2^2 \end{pmatrix}$$

and

$$T^2(x) = \begin{pmatrix} d_{11}x_1^3 + d_{12}x_2^2 \\ d_{21}x_1^2 + d_{22}x_2^2 \end{pmatrix} .$$

4.2 ELEMENTS OF (APPLIED) CONVEX OPTIMIZATION

This section intends to provide elements to the computer scientist to understand basic principles of convex optimization and the typical approaches to manipulate such optimization problems. We refer the interested reader to the excellent book "Convex Optimization" by BOYD and VANDENBERGHE [BV04] for a more thorough introduction. Other valuable references include Numerical Optimization by NOCEDAL and WRIGHT [NW06] and "Éléments d'optimisation différentiable" by GILBERT [Gil13].

In the following we focus on optimization problems of the form

$$\begin{aligned} \min \quad & f_0(x) \\ \text{s.t.} \quad & f_i(x) \leqslant b_i \text{ for } i \in [1, m] \end{aligned} \tag{41}$$

Here $x \in \mathbb{R}^n$ is the optimization variable. $f_0 \in \mathbb{R}^n \to \mathbb{R}$ denotes the objective function and the functions $f_i \in \mathbb{R}^n \to \mathbb{R}$, with associated bound b_i, the constraints.

A solution x of (41) is feasible if it satisfies all constraints. It is optimal if it is smallest of all feasible ones.

An optimization algorithm is a numerical tool that computes or approximate such feasible optimal solution.

4.2.1 *Convex Conic optimization*

In the case where the only part of the problem is the objective function, i.e. no constraint is provided, then classical methods such as gradient, conjugate gradient or NEWTON methods will iteratively approximate the solution. These algorithms compute a sequence of points of $\mathbb{R}^n$ by updating the previous point with a local descent direction d_k obtained by considering the derivative of the objective function (aka. the gradient).

$$x_{k+1} = x_k + \alpha_k d_k$$

Here α_k denotes the step size. Both α_k and d_k depend on the current point x_k and typically rely on $f_0'(x_k)$ (aka. $\nabla f(x_k)$). KANTOROVICH's theorem characterizes conditions imposed on f_0 to guarantee the existence of a unique solution and the convergence to it for NEWTON's method. These constraints amount to provide a bound on the variation of the function, its LIPSCHITZ constant.

Solving a general case of optimization problems with constraints is still an open question. However, a solution to guarantee the existence of such bound is to constrain the functions f_0, and $f_i, \forall i \in [1, m]$ to be convex. We recall that a function f is convex when $\forall \alpha, \beta \geqslant 0, \alpha + \beta = 1, f(\alpha x + \beta y) \leqslant \alpha f(x) + \beta f(y)$. In that case any local optimal point is also a global optimal one. An even stronger condition would be to require it to be linear, i.e. $f(\alpha x + \beta y) = \alpha f(x) + \beta f(y)$.

Convex optimization is then a restriction of general optimization to the following problem:

$$\begin{aligned} \min \quad & f_0(x) \\ \text{s.t.} \quad & f_i(x) \leqslant 0 \text{ for } i \in [1, m] \\ & a_j^T x = b_j \text{ for } j \in [1, p] \end{aligned} \tag{42}$$

where $f, 0$, f_i are convex functions. Note that equality constraints have to be affine: they correspond to a conjunction of two convex inequalities: $f(x) \leqslant 0 \wedge f(x) \geqslant 0$; the only solution is to require f to be affine or linear.

A well known case of this convex optimization problem is linear optimization or linear programming, in which a linear objective function f_0 is optimized while satisfying the linear constraints f_i.

This notion of convex optimization can be further extended to more general convex sets: convex cones. A cone $\mathcal{K}$ is a subset of $\mathbb{R}^n$ closed by positive scaling: $\forall x \in \mathcal{K}, \theta \geqslant 0, \theta x \in \mathcal{K}$. A convex cone satisfies: $\forall x, y \in \mathcal{K}, \theta_1, \theta_2 \geqslant 0, \theta_1 x + \theta_2 y \in \mathcal{K}$. Such convex cone can be fitted with partial order $\preceq$ such that $\forall x, y \in \mathcal{K}, (x \preceq y) \equiv (y - x \in \mathcal{K})$. By extension a function convex in the cone is $\mathcal{K}$-convex.

In that setting a convex conic optimization problem is defined as

$$\begin{aligned} \min \quad & f_0(x) \\ \text{s.t.} \quad & f_i(x) \preceq_{\mathcal{K}} 0 \text{ for } i \in [1, m] \\ & Ax = b \end{aligned} \tag{43}$$

where $f_0 \in \mathbb{R}^n \to \mathbb{R}$ is convex, $f_i \in \mathbb{R}^n \to \mathcal{K}$ are $\mathcal{K}$-convex functions, and $A \in \mathbb{R}^{p \times n}$.

As a first specialization, we speak about linear problems when f_0, f_i are linear. Each function f can then be described as a scalar product $\langle \cdot, \cdot \rangle$ when real valued, or as a product by a matrix when $\exists m, \mathcal{K} \subseteq \mathbb{R}^m$. In the following we denote the function f_0 by a constant vector

c: $f_0(x) = \langle c, x\rangle$, and the functions f_i by the pair A_i, b_i such that $f_i(x) = A_i x - b_i$.

$$\begin{aligned} \min \quad & \langle c, x\rangle \\ \text{s.t.} \quad & A_i x - b_i \preceq_{\mathcal{K}} 0 \text{ for } i \in [1, m] \\ & Ax = b \end{aligned} \tag{44}$$

Let us now focus on special cases depending on the cone $\mathcal{K}$ considered.

Polytopes

When $\mathcal{K} = \mathbb{R}^+$, a famous case is the optimization over closed polyhedra. Each constraint characterizes a subspace of $\mathbb{R}^n$. The feasible set being the intersection of these subspaces, a convex set. The goal is to optimize a linear function over this bounded convex set. In case of bounded feasible set, a finite number of vertices characterize the polytope. Since the optimal solution is necessarily on a vertex, the simplex method enumerates these vertices and compute the optimal one.

Positive semidefinite cone

Let us consider the set $\mathbb{S}^n$ of symmetric matrices of $\mathbb{R}^{n\times n}$:

$$\mathbb{S}^n = \left\{X \in \mathbb{R}^{x\times n} | X = X^\intercal\right\}$$

The set $\mathbb{S}^n_+$ of positive semi-definite matrices is the subset of matrices of $\mathbb{S}^n$ admitting only positive eigenvalues.

$$\mathbb{S}^n_+ = \{X \in \mathbb{S}^n | X \succeq 0\}$$

Equivalently, we have:

$$\mathbb{S}^n_+ = \{X \in \mathbb{S}^n | \forall x \in \mathbb{R}^n, x^\intercal X x \geqslant 0\}$$

This set is a convex cone: it is closed by addition and external multiplication by a positive scalar. Optimizing over this cone leads to problem of the form:

$$\begin{aligned} \min \quad & \langle c, x\rangle \\ \text{s.t.} \quad & \sum_{i\in[1,n]} x_i F_i + G \preceq 0 \\ & Ax = b \end{aligned} \tag{45}$$

Here x is a vector and matrices $G, F_1, \ldots, F_n \in \mathbb{S}^n_+$. The inequality is known as a *Linear Matrix Inequality* (LMI).

Indeed, we can easily have unknown matrices since a matrix $A \in \mathbb{R}^{n\times n}$ can be expressed as $\sum_{i=0,j=0}^{n-1,n-1} A_{i,j} E^{i,j}$, where $E^{i,j}$ is the matrix with zeros everywhere except a one at line i and column j. Likewise, multiple LMIs can be grouped into one since $A \succeq 0 \wedge B \succeq 0$ is equivalent to $\begin{bmatrix} A & 0 \\ 0 & B \end{bmatrix} \succeq 0$.

Efficient solvers for semidefinite programming (SDP), based on interior point method algorithms are available such as *Mosek* [AA00], *SDPA* [Yam+10] or *CSDP* [Bor99]. For more details about SDP, we refer the interested reader to [VB96].

Sum-of-square polynomials

Let $\mathbb{R}[x]$ be the set of multivariate polynomials of $\mathbb{R}^n$ and $\mathbb{R}[x]_{2m}$ its restriction to polynomials of degree at most $2m$. We denote by $\Sigma[x] \subset \mathbb{R}[x]$ the cone of sums-of-squares (SOS) polynomials, that is

$$\Sigma[x] := \left\{ \sum_i q_i^2, \text{ with } q_i \in \mathbb{R}[x] \right\} \tag{46}$$

The existence of an SOS representation for a given polynomial is an approach to *Positivestellensatz* witness, a sufficient condition to prove its global nonnegativity, ie. $\forall p(x) \in \Sigma[x], p(x) \geqslant 0$. The SOS condition (46) is equivalent to the existence of a positive semi-definite matrix Q such that

$$p(x) = Z^\intercal(x) Q Z(x) \tag{47}$$

where $Z(x)$ is a vector of monomials of degree less than or equal to $\deg(p)/2$.

Searching for a positive polynomial of a given degree $d = 2m$ amounts to solve a semi-definite optimization problem and synthesizing the matrix $Q \succeq 0$ satisfying the Eq. 47.

Example 6 *Consider the bi-variate polynomial* $q(x) := 1 + x_1^2 - 2x_1x_2 + x_2^2$. *With* $Z(x) = (1, x_1, x_2)$, *one looks for a semi-definite positive matrix* Q *such that the polynomial equality* $q(x) = Z(x)^\intercal\, Q\, Z(x)$ *holds for all* $x \in \mathbb{R}^2$.

The matrix

$$Q = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & -1 & 1 \end{pmatrix}$$

satisfies this equality and has three nonnegative eigenvalues, which are 0, 1, *and* 2, *respectively associated to the three eigenvectors* $e_0 := (0, 1, 1)^\intercal$, $e_1 := (1, 0, 0)^\intercal$ *and* $e_2 := (0, -1, 1)^\intercal$. *Defining the matrices* $L := (e_1\, e_2\, e_0) = \left(\begin{smallmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & -1 & 1 \end{smallmatrix}\right)$ *and* $D = \left(\begin{smallmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{smallmatrix}\right)$, *one obtains the decomposition* $Q = L\, D\, L^{-1}$ *and the equality* $q(x) = (L\, Z(x))\, D\, (L^{-1}\, Z(x)) = \sigma(x) = 1 + (x_2 - x_1)^2$, *for all* $x \in \mathbb{R}^2$. *The polynomial* σ *is called a* SOS certificate *and guarantees that* q *is nonnegative.*

A SOS optimization problem can be defined as

$$\begin{array}{ll} \min & \langle c, x\rangle \\ \text{s.t.} & p_i(x) \in \Sigma[x] \text{ for } i \in [1,m] \\ & Ax = b \end{array} \tag{48}$$

SOS programming solvers provide a front-end easing the translation of a SOS-based optimization problem into SDP. Each sum-of-square polynomial constraint p_i is associated to a symmetric positive semi-definite matrix Q_i. By identification, coefficients of the matrices Q_i are associated to equality constraints depending on the expression characterizing the polynomial p_i. Once the SDP problem is solved, its solution is used to rebuild the polynomial problem and provides the positive certificate of the SOS problem (48).

The Matlab toolbox Yalmip [Löfo4] provides such a frontend.

More references regarding SOS based polynomial optimization can be found in Parrilo [Paro3] and Lasserre [Laso9] works.

4.2.2 *Convex optimization tools*

When manipulating optimization problems, there are few standard operations that enable to relax a problem into a solvable one. We present here of few of those techniques.

Convexifying constraints: S-procedure, Lagrangian relaxation

When facing non convex constraints such as implication between positive definite matrices

$$P_1 \succeq 0 \implies P_2 \succeq 0$$

we can express a sufficient condition in a convex way. The S-Procedure provides such as relaxation.

Theorem 4.6 (S-Procedure) *For any* $P, P_1, \ldots, P_k \in \mathbb{R}^{n\times n}$ *and* $b, b_1, \ldots, b_k \in \mathbb{R}$ *and* $b, b' \in \mathbb{R}$*, the following*

$$\exists \tau_1, \ldots, \tau_k \in \mathbb{R}, \quad \left(\bigwedge_{i=1}^k \tau_i \geqslant 0\right) \wedge \begin{bmatrix} -P & 0 \\ 0 & b \end{bmatrix} - \sum_{i=1}^k \tau_i \begin{bmatrix} -P_i & 0 \\ 0 & b_i \end{bmatrix} \succeq 0 \tag{49}$$

is a sufficient condition for

$$\forall x \in \mathbb{R}^n, \left(\bigwedge_{i=1}^k x^T P_i\, x \leqslant b_i\right) \Rightarrow x^T P\, x \leqslant b. \tag{50}$$

More generally Lagrangian relaxation uses a similar approach to express a constraint in the objective function. It consists in adding to the objective function the inner product of the vector of constraints with a positive vector of the euclidean space whose dimension is the number of constraints. Let us consider the following simple linear problem

$$\begin{array}{ll} \min\limits_{x\in R} & \langle c, x\rangle \\ \text{s.t.} & ax \leqslant b \end{array} \tag{51}$$

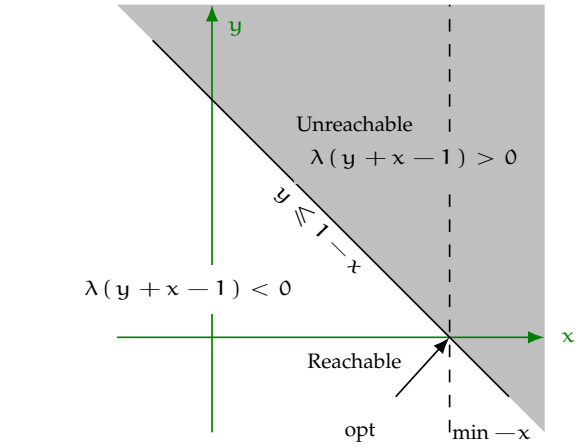
It is possible to express a second problem without constraints by introducing a nonnegative Lagrange multiplier $\lambda \in \mathbb{R}^+$:

$$\max_{\lambda\in\mathbb{R}^+} \min_{x\in\mathbb{R}} \langle c, x\rangle + \lambda(ax - b) \tag{52}$$

Since λ is nonnegative, any x satisfying the constraint $ax \leqslant b$ renders the term $\lambda(ax-b)$ negative. Trying to maximize the goal over variable λ, the optimum is obtained when $\lambda = 0$. Fixing x, any $\lambda' > \lambda$ will generate a solution $\langle c, x\rangle + \lambda'(ax-b) < \langle c, x\rangle + \lambda(ax-b)$. However, any x outside of the constraint will generate a positive term $\lambda(ax-b)$ that will be made arbitrarily large when trying to maximize it over $\lambda \geqslant 0$.

Example 7 *Figure 4.1 represents a simple Lagrangian relaxation in the linear case. The objective function is* $-x$ *when* $y \geqslant 0$ *and* $y \leqslant 1 - x$*. The optimal solution is* $(x, y) = (1, 0)$*. When maximizing over* λ *in term* $\lambda(y + x - 1)$*, one obtain* 0 *when satisfying the constraint, and* $+\infty$ *otherwise.*

Figure 4.1 Example of a Lagrangian relaxation



Similarly, a maximization problem can be reformulated as a inf sup when using Lagrangian relaxation to integrate a constraint in the objective.

$$\begin{array}{ll}\inf & f(x) \\ ax \leqslant b & \\ x \in \mathbb{R} & \end{array} \leqslant \sup_{\lambda \in \mathbb{R}^+} \inf_{x \in \mathbb{R}} f(x) - \lambda(ax - b) \tag{53}$$

$$\begin{array}{ll}\sup & f(x) \\ ax \leqslant b & \\ x \in \mathbb{R} & \end{array} \geqslant \inf_{\lambda \in \mathbb{R}^+} \sup_{x \in \mathbb{R}} f(x) - \lambda(ax - b) \tag{54}$$

In case of equality constraint, any sign before the $\lambda(ax - b)$ term is valid.

These notions are easily extended to LMI as long as the constraint to integrate into the objective is conic convex and can be expressed as $A_i x - b_i \preceq 0$.

SOS extensions: SOS reinforcement and relaxation

The SOS reinforcement of polynomial optimization problems consists of restricting polynomial nonnegativity to being an element of $\Sigma[x]$. In case of polynomial maximization problems, the SOS reinforcement boils down to computing an upper bound of the real optimal value. For example let $p \in \mathbb{R}[x]$ and consider the unconstrained polynomial maximization problem $\sup\{p(x) \mid x \in \mathbb{R}^d\}$. Applying SOS reinforcement, we obtain:

$$\begin{aligned}\sup\{p(x) \mid x \in \mathbb{R}^d\} &= \inf\{\eta \mid \forall x, \eta - p(x) \geqslant 0\} \\ &\leqslant \inf\{\eta \mid \eta - p \in \Sigma[x]\} .\end{aligned} \tag{55}$$

Now, let $p, q \in \mathbb{R}[x]$ and consider the constrained polynomial maximization problem:

$$\sup\{p(x) \mid \forall x \in \mathbb{R}^d, q(x) \leqslant 0\}$$

We can perform a Lagrangian relaxation but require λ to be a positive (SOS) polynomial instead of a positive scalar. Let $\lambda \in \Sigma[x]$, then:

$$\sup_{q(x) \leqslant 0,\ x \in \mathbb{R}^d} p(x) \leqslant \sup_{x \in \mathbb{R}^d} p(x) - \lambda(x) \cdot q(x) .$$

Indeed, suppose $q(x) \leqslant 0$, then $-\lambda(x)q(x) \geqslant 0$ and $p(x) \leqslant p(x) - \lambda(x)q(x)$. Finally, taking the supremum over $\{x \in \mathbb{R}^d \mid q(x) \leqslant 0\}$ provides the above inequality. Since $\sup\{p(x) - \lambda(x) \cdot q(x) \mid x \in \mathbb{R}^d\}$ is an unconstrained polynomial maximization problem then we apply an SOS reinforcement (as in Eq. (55)) and we obtain:

$$\begin{aligned}\sup_{q(x) \leqslant 0,\ x \in \mathbb{R}^d} p(x) &\leqslant \sup_{x \in \mathbb{R}^d} p(x) - \lambda(x) \cdot q(x) \\ &\leqslant \inf\{\eta \mid \eta - p - \lambda q \in \Sigma[x]\}.\end{aligned}$$

Finally, note that this latter inequality is valid whatever $\lambda \in \Sigma[x]$ and so we can take the infimum over $\lambda \in \Sigma[x]$ which leads to:

$$\begin{aligned}\sup_{q(x) \leqslant 0,\ x \in \mathbb{R}^d} p(x) &\leqslant \inf_{\lambda \in \Sigma[x]} \sup_{x \in \mathbb{R}^d} p(x) - \lambda(x) \cdot q(x) \\ &\leqslant \inf_{\substack{\eta - p - \lambda q \in \Sigma[x] \\ \lambda \in \Sigma[x]}} \eta .\end{aligned} \tag{56}$$

4.2.3 *Duality*

A last useful manipulation of optimization problems relies on topological duality in Banach spaces (vector spaces with good topological structures). We give here an incomplete and informal overview of duality theory, since it enables the characterization of the dual problem. The interested reader could find more details in [BV04, §5.2].

Any vector space E over the field $\mathbb{R}$ can be associated with its dual vector space $E^\dagger$ defined as the set of real-valued linear functionals on E. That is the set of functions $\phi : E \to \mathbb{R}$. For any element of E we can associate an element of its dual space. This is characterized by the duality bracket $\langle \phi_x, x \rangle_{E^\dagger, E}$.

Thanks to the Riesz representation theorem, this element is unique in Hilbert space and can be represented in the same space. Let consider for example the finite dimensional Hilbert space $\mathbb{R}^n$, and an element $c \in \mathbb{R}^n$. The dual space is the set of linear functionals over $\mathbb{R}^n$, that is the set of linear functions $\phi : \mathbb{R}^n \to \mathbb{R}$. Any such linear function can be defined by a scalar product. One can then build the linear functional associated to c: $\phi_c(x \in \mathbb{R}^n) = \langle x, c \rangle$ where $\langle \cdot, \cdot \rangle$ denotes the inner product of the Hilbert space E, in that case the scalar product of $\mathbb{R}^n$. Hilbert spaces are then auto-dual since a linear functional can be characterized by an element of the initial space.

Let us consider the general case of two Banach spaces E and F, $E^\dagger$ and $F^\dagger$ their topological dual, respectively, $K \subseteq E$ a convex cone (and $K^\dagger$ its dual), and the following optimizing problem:

$$\begin{array}{ll}\max & \langle c, x \rangle_{E^\dagger, E} \\ \text{s.t.} & Ax = b \text{ with } A : E \to F \\ & x \in K\end{array} \tag{57}$$

In the following, let us denote this problem as the *primal problem*. This form is equivalent to the earlier version of Eq. (44) (cf. [BV04] for more explanation):

$$\begin{array}{ll} \max & \langle c, x\rangle \\ \text{s.t.} & A_i x - b_i \preceq_{\mathcal{K}} 0 \text{ for } i \in [1,m] \\ & Ax = b \end{array} \tag{58}$$

The constraint $Ax = b$ is equivalent to $Ax - b = 0_F$. Then one can introduce a Lagrangian multiplier $y \in F^\dagger$ to express the constraint. We have the duality bracket $\langle y, Ax - b\rangle_{F^\dagger,F}$. Using linearity of the linear form, one has $\langle y, Ax - b\rangle_{F^\dagger,F} = \langle y, Ax\rangle_{F^\dagger,F} - \langle y, b\rangle_{F^\dagger,F}$.

We can introduce the adjoint $A' : F^\dagger \to E^\dagger$ of A as the unique linear application such that $\langle y, Ax\rangle_{F^\dagger,F} = \langle A'y, x\rangle_{E^\dagger,E}$.

The constraint can then be expressed as $\langle A'y, x\rangle_{E^\dagger,E} - \langle y, b\rangle_{F^\dagger,F}$.

Going back to the initial problem, we have

$$\begin{array}{lcll} & & & \forall y \in F^\dagger, \\ \max\limits_x \langle c, x\rangle_{E^\dagger,E} & \leqslant & \max\limits_x & \langle c, x\rangle_{E^\dagger,E} \\ \text{s.t} \begin{cases} Ax = b \\ x \in K \end{cases} & & & + \langle A'y, x\rangle_{E^\dagger,E} \\ & & & - \langle y, b\rangle_{F^\dagger,F} \end{array}$$

Since y is free in the left hand part, one can build the following inequality:

$$\begin{array}{lcll} \max\limits_x \langle c, x\rangle_{E^\dagger,E} & \leqslant & \min\limits_y \max\limits_x & \langle c + A'y, x\rangle_{E\hat{}\dagger,E} \\ \text{s.t} \begin{cases} Ax = b \\ x \in K \end{cases} & & & - \langle y, b\rangle_{F\hat{}\dagger,F} \end{array}$$

The maximum with respect to x depends only on $\langle c + A'y, x\rangle$. Let us first define the dual cone of K as the restriction of the topological dual of E to positive linear forms on K:

$$K^\dagger = \{f \in E^\dagger | \forall x \in K, \langle f, x\rangle \geqslant 0\}$$

Since $x \in K$, we have to make sure that $\langle c + A'y, x\rangle_{E^\dagger,E}$ will not diverge and corrupt the maximum of x when minimizing y. If we choose $c + A'y \in K^\dagger$, then the duality bracket is positive and will impact badly the maximum over x. Therefore, we have to choose $-c - A'y \in K^\dagger$.

We obtain the dual optimization problem:

$$\begin{array}{ll} \min_y & -\langle y, b\rangle_{F^\dagger,F} \\ \text{s.t.} & -c - A'y \in K^\dagger \end{array} \tag{59}$$

While this description is general and will be used later in Chap. 13, a simpler version on Hilbert spaces will be used in Chap. 6.

FEASIBILITY OF PRIMAL AND DUAL PROBLEMS A last remark concerns the feasibility of the two primal and dual problems. Thanks to the construction of the dual problem and the use of Lagrangian relaxation, we have the following inequality:

$$\begin{array}{lcll} \max\limits_x & \langle c, x\rangle_{E^\dagger,E} & \leqslant & \min_y \quad -\langle y, b\rangle_{F^\dagger,F} \\ \text{s.t.} & A : E \to F & & \text{s.t.} \quad -c - A'y \in K^\dagger \\ & Ax = b & & \\ & x \in K & & \end{array} \tag{60}$$

In the case where both optimization problems admit strict feasible solutions – we speak about primal and dual feasibility and in case of convex constraints, the inequality of Eq. (60) becomes an equality, without any duality gap. In other words solving any of the two problems gives the optimum solution. The conditions required are usually referred as the Slater's conditions.

In practice one can easily obtain cases where one of the problems has an empty interior and is not strictly feasible. In that case the numerical solutions of both problems are not the same; we speak about a duality gap between these two solutions. We will come back to that notion in Chapter 10.

5

INVARIANT SYNTHESIS VIA CONVEX OPTIMIZATION: POSTFIXPOINT COMPUTATION AS SEMIALGEBRAIC CONSTRAINTS

5.1 INVARIANTS, LYAPUNOV FUNCTIONS AND CONVEX OPTIMIZATION

This chapter focuses on the computation of invariant for a discrete dynamical system collecting semantics.

Invariants or collecting semantics properties are properties preserved along all executions of a system and verified in all reachable states. A subset of these invariants are defined as inductive. Inductive invariants are properties, or relationships between variables, that are inductively preserved by one transition of considered systems. As used in induction proofs, it is not required to consider a reachable state and all (or part of) its past while arguing about the validity of the invariant, but only a single state. Applying the induction principle we obtain that any state satisfying the property is mapped to a next state preserving that same property.

For example when one analyzes a geometric progression with a ratio r such that $|r| < 1$ then any invariant expressed as an interval can be easily proved: if $[a, b]$ contains both the initial state and the value 0, any element of the progression will belong to $[a, b]$. Note that, here, we are only focused in the invariant but not interested in characterizing the decay or growth rate of the progression.

Discrete dynamical systems admit an infinite behavior, it is therefore of utmost importance to be able to characterize their reachable states, for example proving the boundedness of such set. In the control community *lingua* a system is said stable if, without any input, it convergences to zero. This idea is captured by LYAPUNOV functions.

The current chapter proposes methods to compute dynamical systems invariants based on LYAPUNOV function synthesis using convex optimization. In this section we introduce these notions. The following sections develop different encodings to compute these invariants for a wide variety of settings and solve different kinds of optimization problems.

5.1.1 *Fixpoint characterization, Invariant and Inductive Invariants*

The motivation is to determine automatically if a given property holds for the analyzed program, or to compute precise bounds on reachable states. We are interested in numerical properties and more precisely in properties on the values taken by the d-uplet of the variables of the program.

According to the abstract interpretation framework outlined in Sec. 2.5, a semantics can be characterized by a set of elements; for collecting semantics that is the set of reachable states. Hence, in our point-of-view, as for the semantics characterization, a property is characterized by some set $P \subseteq \mathbb{R}^d$ of values satisfying the property.

Let us first recall the fixpoint characterization and instantiate it on our discrete dynamical system formalization.

Collecting Semantics as postfixpoint characterization

In Sec. 2.5 we introduced the collecting semantics map in Eq. 16 and the fixpoint characterization of the collecting semantics in Eq. 17.

$$\begin{array}{rcl} \wp(\Sigma) & \to & \wp(\Sigma) \\ S & \mapsto & I \cup f(S) \end{array} \tag{61}$$

$$\mathfrak{C} = \mathrm{lfp}_{\perp} F = \min_{X \in \wp(\Sigma)} \{F(X) \subseteq X\} \tag{62}$$

where f denotes the transition relation.

As a consequence any subset C of $\wp(\Sigma)$ verifying the condition $\{F(X) \subseteq X\}$ is a sound over-approximation of $\mathfrak{C}$ since all reachable states verify C: $\mathfrak{C} \subseteq C$.

Collecting Semantics of discrete dynamical systems

Let us consider now focus on a program of the forms presented in Sec. 4.1.1, 4.1.2 or 4.1.3. In the most general case, it is characterized by an initial set X^{Init}, and by a list of update functions f_i and associated conditions c_i.

In the following we assume that we are given with a set representation X^i of each condition c_i. We recall that X^i are assumed to form a partition of Σ, ie. for each element a unique update function is applicable.

Let C be set satisfying the previous equation, over-approximating reachable states $\mathfrak{C}$. With F a piecewise discrete dynamical system, we have the following constraints on P:

$$\begin{aligned} &\{F(C) \subseteq C\} \\ &= \left\{X^{Init} \cup f(C) \subseteq C\right\} \\ &= \left\{ C \;\middle|\; \begin{array}{l} X^{Init} \subseteq C \\ \text{for } i \in \mathcal{I}, f_i(C \cap X^i) \subseteq C \end{array} \right\} \end{aligned} \tag{63}$$

This equation can be further simplified in case of a single update function, ie. not disjunction and conditions c_i, X^i.

$$\left\{ C \;\middle|\; \begin{array}{l} X^{Init} \subseteq C \\ f(C) \subseteq C \end{array} \right\} \tag{64}$$

5.1.2 Lyapunov *functions*

In 1890, Alexander Lyapunov published his well know result stating that the differential equation $\frac{d}{dt}x = Ax(t)$ is stable if and only if there exists a positive-definite matrix P such that $A^\intercal P + PA \preceq 0$. Here both A and P are square matrices of $\mathbb{R}^{n\times n}$ and P is positive definite $P \succ 0$, ie. $\forall x \in \mathbb{R}^n, x^\intercal P x \geqslant 0$. Later this was formulated in a discrete-time setting over discrete linear systems:

$$x_{k+1} = Ax_k \text{ with } A \in \mathbb{R}^{n\times n}$$

as

$$\exists P \in\in \mathbb{R}^{n\times n}, \text{ s.t.} \begin{cases} \exists P \succ 0 \\ A^\intercal P A - P \preceq 0 \end{cases} \tag{65}$$

In both cases, P is the measure of energy of the system: the Lyapunov function $x \mapsto x^\intercal P x$. When *measuring* the energy of the image state Ax, we obtain $(Ax)^\intercal P(Ax) = x^\intercal A^\intercal P A x$.

Since P is positive definite, $\forall x \in \mathbb{R}^n, x^\intercal P x > 0$, and P denotes a norm over states. While, thanks to the second constraint, its sublevel sets are inductive over states: $\forall x \in \mathbb{R}^n, x^\intercal P x \geqslant x^\intercal A^\intercal P A x$. The inequality $A^\intercal P A - P \preceq 0$ encodes a kind of energy dissipation along trajectories. When the energy reaches 0, the state of the system is near 0. In this original setting, the considered system is closed, ie. it does not admit input. In case of linear systems with bounded inputs, one can rely on the same argument not to motivate asymptotic stability but to argue that the system will not diverge and remains within some bounds.

Simpler arguments do exist for the specific case of linear systems, eg. one can compute the eigenvalues of the matrix and check that the linear map A is contracting. However, this notion of Lyapunov function seems more extensible and was widely developed in the control community.

Let us consider numerical systems with $\Sigma = \mathbb{R}^d$. More formally, a Lyapunov function $V : \mathbb{R}^d \to \mathbb{R}^+$ for a discrete-time system is a positive real valued function over system states that should satisfy:

- Null at origin, positive elsewhere

$$\begin{cases} V(0) = 0 \\ \forall x \in \mathbb{R}^d \backslash \{0\}, V(x) > 0 \wedge \lim_{\|x\|\to\infty} V(x) = \infty \end{cases} \tag{66}$$

- Decreasing along trajectories

$$\forall x \in \mathbb{R}^d, V \circ f(x) - V(x) \leqslant 0. \tag{67}$$

Depending on the strictness of the $\leqslant$ operator, the Lyapunov function guaranties asymptotic stability and exponential convergence, or just boundedness of states.

It is shown for example in [HC08] that exhibiting such a function proves the Lyapunov stability of the system, meaning that its state variables will remain bounded through time. Equation (67) expresses the fact that the function $k \mapsto V(x_k)$ decreases, which, combined with (66), shows that the state variables remain in the bounded sublevel set $\{x \in \mathbb{R}^n | V(x) \leqslant V(x_0)\}$ at all instants $k \in \mathbb{N}$.

5.1.3 Lyapunov *functions as problem specific abstractions: semialgebraic template abstractions*

We saw that Lyapunov functions characterizes inductive sublevel sets for the considered discrete dynamical system semantics. Therefore, instead of approximating reachable states in the abstract interpretation framework using predefined numerical abstractions, such as intervals, octagons or convex polyhedra, we rather propose to rely on the Lyapunov function as the main mean of abstraction. This is a template abstraction [CS11; SG09].

A template is a real-valued function $t : \Sigma \to \mathbb{R}$.

Example 8 *A template is then a function over those state variables. For example, it can characterize the norm 2 of a state:*

$$t_1(s) = \|s\|_2 = \sqrt{(\Sigma_{v\in V} v^2)}$$

or just focus on the value of a single variable $x \in V$

$$t_2(s) = s(x) \qquad \textit{when } s \textit{ is a map or}$$
$$t_2(s) = x_i \qquad \textit{when } s \textit{ is a vector}$$

Templates allow to express intervals $t_1 = x, t_2 = -x$*, fixed shape polyhedra such as octagons* $\pm x_i \pm x_j$.

For a given template t, a sublevel-set abstraction can be defined by a given levelset λ:

$$\{s \mid t(s) \leqslant \lambda\}$$

In case of multiple templates $t_1, t_2, \ldots, t_n$ and associated bounds $\lambda_1, \lambda_2, \ldots, \lambda_n$, the interpretation of this abstract representation is the intersection of sub-level sets. In case of polynomial template functions t_i, this is a basic semi algebraic set.

$$\bigcap_i \{s \mid t_i(s) \leqslant \lambda_i\}$$

5.1.4 *Synthesis of templates using convex optimization*

In the early definition of LYAPUNOV functions, with quadratic properties and linear systems, the conditions defining the LYAPUNOV function P where characterized as a Linear Matrix Inequality (LMI):

$$\begin{cases} \exists P \succeq 0 \\ A^\intercal P A - P \preceq 0 \end{cases} \tag{68}$$

With the development of interior point algorithms [NN94] and convex optimization [BV04], the numerical resolution of these optimization problems becomes feasible in reasonable time.

Our approach is to guide the search for inductive invariants as LYAPUNOV-like constraints expressed as convex optimization problems.

Once a LYAPUNOV function is synthesized, as a kind of norm of a state, it can be used as a template abstraction and denote a relevant abstraction of reachable states. Depending on the encoding of the constraints, the results of the optimization step could either be a bound template, eg. $t(x) \leqslant 1$, or just a relevant unbounded template t.

In that second case, the template t has to be bounded by other means; for example using classical KLEENE iterations, or even using randomly large values. Thanks to the inductiveness property of the template with respect to the system semantics, any bound λ such that

$$t(f(x)) \leqslant \lambda$$

characterizes a sound postfixpoint (invariant):

$$\{x \,|\, t(x) \leqslant \lambda\}$$

5.2 QUADRATIC INVARIANTS

5.2.1 *Linear systems*

As mentioned above, in the simplest case of linear system the conditions over a quadratic LYAPUNOV function P are given by the LMI of Equation (65).

$$\exists P \in\in \mathbb{R}^{n\times n}, \text{ s.t.} \begin{cases} \exists P \succeq 0 \\ A^\intercal P A - P \preceq 0 \end{cases} \tag{69}$$

One can directly solve this LMI and obtain a valid quadratic template, relevant for the considered system. However, while inductive over system semantics, a sublevel set property characterized by such LYAPUNOV function P may not be the most precise with respect to the collecting semantics $\mathfrak{C}$ (c.f. §2.5):

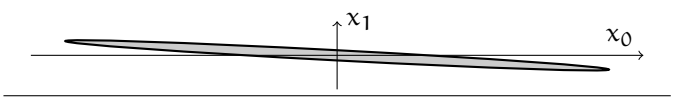
$$\mathfrak{C} \ll \left\{ x \in \mathbb{R}^d \middle| x^\intercal P x \leqslant \lambda \right\}$$

In order to synthesize a more precise invariant, one can further constrain the LMI.

Minimizing Condition Number

Graphically, the condition number of a positive definite matrix expresses a notion similar to that addressed by eccentricity for ellipses in dimension 2. It measures how 'close' to a circle (or its higher dimension equivalent) the resulting ellipsoid will be. Multiples of the identity matrix, which all represent a circle, have a condition number of 1. Thus one idea of constraint we can impose on P is to have its condition number as close to 1 as possible. A rationale for this is that 'flat' ellipsoids, *i.e.* having a large condition number, can yield a very bad bound on one of the variables, as illustrated on Figure 5.1.

Figure 5.1 'flat' ellipsoids can yield very large bounds on some variables.



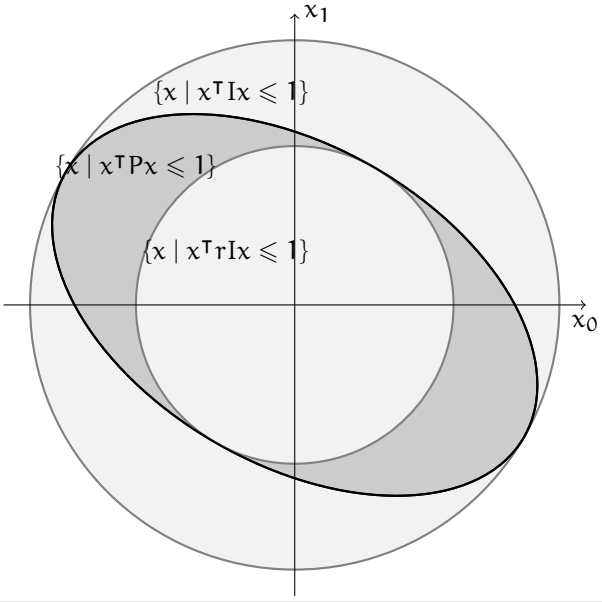
This is done [Boy+94] by minimizing a new variable, r, in the following matrix inequality

$$I \preceq P \preceq rI$$

Indeed, if a point x is in the ellipsoid P, then $x^\intercal P x \leqslant 1$ which implies $x^\intercal I x \leqslant 1$, *i.e.* x is in the sphere of radius 1. Thus, the ellipsoid P is included in the sphere of radius 1.

Similarly, P contains the sphere of radius $r^{-\frac{1}{2}}$. This way, P is sandwiched between these two spheres and making their radius as close as possible will make P as 'round' as possible, as depicted on Figure 5.2.

Figure 5.2 Making the ellipsoid P as 'round' as possible by sandwiching it between spheres of radius $r^{-\frac{1}{2}}$ and 1: $I \preceq P \preceq rI$ and minimizing r.



This constraint, along with the others (Lyapunov equation, symmetry and positive definiteness of P), can be expressed as a LMI, which is solved using the semi-definite programming techniques mentioned in Section 4.2.1:

Figure 5.3 Quadratic invariant for linear system minimizing the condition number of P.

$$\begin{aligned} &\text{minimize} && r \\ &\text{subject to} && A^T P A - P \prec 0 \\ &&& I \preceq P \preceq rI \\ &&& P^T = P. \end{aligned} \tag{70}$$

Example 9 *With the following matrix A of the running example*

$$A := \begin{bmatrix} 0.9379 & -0.0381 & -0.0414 \\ -0.0404 & 0.968 & -0.0179 \\ 0.0142 & -0.0197 & 0.9823 \end{bmatrix}$$

a semi-definite solver simply returns $r = 1$ and the identity matrix

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Preserving the Shape

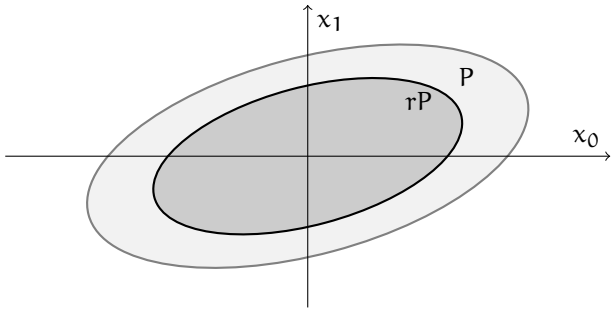
Another approach [Yan92] is to minimize $r \in (0, 1)$ in the following inequality

Figure 5.4 Quadratic invariant preserving shape of the ellipsoid P.

$$A^T P A - rP \preceq 0. \tag{71}$$

Intuitively, this corresponds to finding the shape of ellipsoid that gets 'preserved' the best when the update $x_{k+1} = Ax_k$ is applied, as depicted on Figure 5.5. r can be seen as the minimum contraction achieved by this update in the norm defined by P, hence the name *decay rate* given to this value by control theorists. This is the choice implicitly made in [Fer04] for a particular case of matrices A of order 2.

Figure 5.5 Choice of the ellipsoid whose shape is the best preserved.



With this technique however, the presence of a quadratic term rP in the equation prevents the use of usual LMI solving tools 'as is'. To overcome this, the following property enables the choice of an approach where the value for r is refined by dichotomy. Only a few steps are then required to obtain a good approximation of the optimal value.

Property 5.1 *If Equation 71 admits as solution a positive definite matrix P for a given r, then it is also the case for any $r' \geqslant r$.*

Example 10 *With the following matrix A of the running example:*

$$A := \begin{bmatrix} 0.9379 & -0.0381 & -0.0414 \\ -0.0404 & 0.968 & -0.0179 \\ 0.0142 & -0.0197 & 0.9823 \end{bmatrix},$$

looking for a small $r \in (0,1)$, the first value tested is $r = 0.5$, i.e. a solution to the following semi-definite program is looked for

$$\begin{aligned} &\textit{minimize} && 0 \\ &\textit{subject to} && A^T P A - 0.5P \preceq 0 \\ &&& P \succ 0 \\ &&& P^T = P. \end{aligned}$$

Since there is no solution, r is now looked for in interval $(0.5,1)$. $r = 0.75$ is tested, without more success, then $r = 0.875$, $r = 0.9375$, $r = 0.98675$ and $r = 0.984375$ are still unsuccessful. Finally, $r = 0.9921875$ yields the following solution (all figures being rounded to four digits):

$$P = \begin{bmatrix} 239.1338 & 37.5557 & 77.9203 \\ 37.5557 & 226.3640 & 65.8287 \\ 77.9203 & 65.8287 & 325.1628 \end{bmatrix}.$$

Stopping here leaves $r \in (0.984375, 0.9921875)$ and the above matrix P as solution for $r = 0.9921875$.

5.2.2 *Consider linear systems with inputs*

Most system trajectories are not purely characterized by their initial state: they have inputs.

$$x_{k+1} = Ax_k + Bu_k, \|u_k\|_\infty \leqslant 1. \tag{72}$$

In case of unbounded input the system is guaranteed to diverge. We are therefore interested in showing that, when the input values at bounded $\|u_k\|_\infty \leqslant 1$ (ie. $\max_k u_k \leqslant 1$), then the system still has a bounded behavior. This constraint over u_k is reasonable: most inputs come from sensor which themselves have physical limits. We can also choose the bound 1 without loss of generality since one can always alter the matrix B to account for different bounds.

Considering the inputs requires a slight reinforcement of Equation (65) into

$$A^T P A - P \prec 0 \tag{73}$$

We can still guarantee that the state variables of (72) will remain in a sublevel set $\{x \in \mathbb{R}^n \mid x^T P x \leqslant \lambda\}$ (for some $\lambda > 0$), which is an ellipsoid in this case.

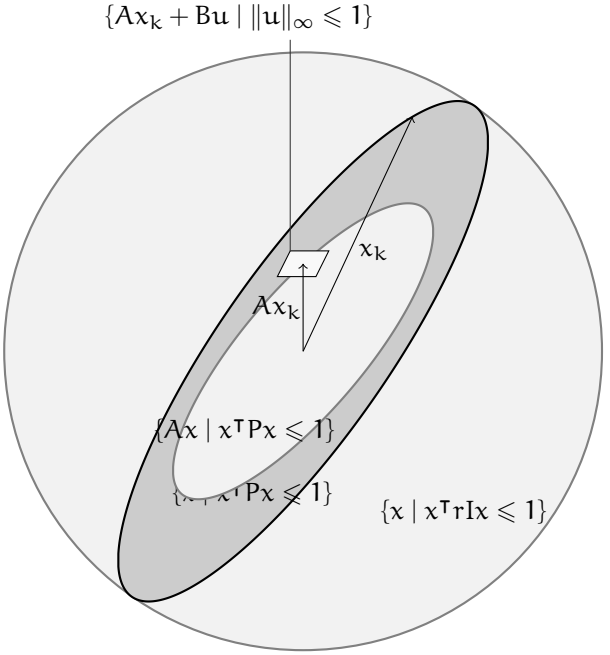
Quadratic invariant for bounded-input linear systems

The two previous methods were based only on A, completely abstracting B away, which could lead to rather coarse abstractions. We try here to take both A and B into account by finding the ellipsoid P included in the smallest possible sphere which is stable, *i.e.* such that

$$\begin{aligned} &\forall x, \forall u, \|u\|_\infty \leqslant 1 \wedge x^T P x \leqslant 1, \\ &(Ax + Bu)^T P (Ax + Bu) \leqslant 1. \end{aligned}$$

This is illustrated in Figure 5.6.

Figure 5.6 Looking for an invariant ellipsoid included in the smallest possible sphere by maximizing r.



The previous condition can be rewritten as

$$\begin{aligned} \forall x, \forall u, \left(\textstyle\bigwedge_{i=0}^{p-1} \left(e_i^T u\right)^2 \leqslant 1\right) \wedge x^T P x \leqslant 1 \\ \Rightarrow (Ax + Bu)^T P (Ax + Bu) \leqslant 1 \end{aligned}$$

where e_i is the i-th vector of the canonical basis (*i.e.* with all coefficients equal to 0 except the i-th one which is 1). This amounts to

$$\begin{aligned} \forall x, \forall u, \left(\bigwedge_{i=0}^{p-1} \begin{bmatrix} x \\ u \end{bmatrix}^T \begin{bmatrix} 0 & 0 \\ 0 & E^{i,i} \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} \leqslant 1\right) \\ \wedge \begin{bmatrix} x \\ u \end{bmatrix}^T \begin{bmatrix} P & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} \leqslant 1 \\ \Rightarrow \begin{bmatrix} x \\ u \end{bmatrix}^T \begin{bmatrix} A^T P A & A^T P B \\ B^T P A & B^T P B \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} \leqslant 1 \end{aligned}$$

where $E^{i,j}$ is the matrix with 0 everywhere except the coefficient at line i, column j which is 1. Using the S-procedure (Theorem 4.6, page 34), this holds when there are τ and $\lambda_0, \ldots, \lambda_{p-1}$ all nonnegatives such that

$$\begin{bmatrix} -A^TPA & -A^TPB & 0 \\ -B^TPA & -B^TPB & 0 \\ 0 & 0 & 1 \end{bmatrix} - \tau \begin{bmatrix} -P & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} - \sum_{i=0}^{p-1} \lambda_i \begin{bmatrix} 0 & 0 & 0 \\ 0 & -E^{i,i} & 0 \\ 0 & 0 & 1 \end{bmatrix} \succeq 0 \tag{74}$$

As in Section 5.2.1, this is not an LMI since τ and P are both variables. And again, there is a $\tau_{min} \in (0,1)$ such that this inequality admits as solution a positive definite matrix P if and only if $\tau \in (\tau_{min}, 1)$. This value τ_{min} can by the way be approximated thanks to the exact same procedure. Similarly to what was done in Section 5.2.1, P is forced to be contained in the smallest possible sphere by maximizing r in the additional constraint

$$P \succeq rI. \tag{75}$$

The function f is then defined as the function mapping $\tau \in (\tau_{min}, 1)$ to the optimal value of the following semi-definite program.

Figure 5.7 Quadratic template for bounded-input linear systems

$$\begin{aligned} &\text{maximize} && r \\ &\text{subject to} && (74) \\ &&& (75) \\ &&& P^T = P \\ &&& \bigwedge_{i=0}^{p-1} \lambda_i \geqslant 0 \end{aligned} \tag{76}$$

This function f can then be evaluated for a given input τ simply by solving the above semi-definite program. f seems concave which could enable a smart optimization procedure. However, in practice, it is enough to just sample f for some equally spaced values in the interval $(\tau_{min}, 1)$ and just keep the matrix P obtained for the value enabling the greatest r.

Example 11 *With the following matrices A and B of the running example:*

$$A := \begin{bmatrix} 0.9379 & -0.0381 & -0.0414 \\ -0.0404 & 0.968 & -0.0179 \\ 0.0142 & -0.0197 & 0.9823 \end{bmatrix} \quad B := \begin{bmatrix} 0.0237 \\ 0.0143 \\ 0.0077 \end{bmatrix},$$

according to Example 10, $\tau_{min} = 0.9921875$.

Then, f is evaluated on a few points between τ_{min} and 1 (rounded figures):

τ	$f(\tau)$
0.9928	1.6064
0.9935	1.4653
0.9941	1.3231
0.9948	1.1798
0.9954	1.0355
0.9961	0.8902

τ	$f(\tau)$
0.9967	0.7440
0.9974	0.5970
0.9980	0.4490
0.9987	0.3002
0.9993	0.1505

and the one giving the best value ($\tau = 0.9928$) is kept with the corresponding

$$P = \begin{bmatrix} 12.6465 & -14.1109 & -10.5402 \\ -14.1109 & 25.6819 & 3.06577 \\ -10.5402 & 3.06577 & 29.5981 \end{bmatrix}.$$

Optimize template for a given variable

If a tighter bound is required on one of the variables, the identity matrix I in inequality (75) can be replaced by a diagonal matrix with larger coefficients for variables of interest. For instance, to get a smaller bound on the first variable x_0, the matrix I can be replace by $\begin{bmatrix} 10 & 0 \\ 0 & I \end{bmatrix}$.

This intuitively corresponds to minimize the radius of an ellipsoid containing P flatter on the dimension of interest instead of a sphere. This is illustrated on Figures 5.8 and 5.9.

Figure 5.8 Constraining ellipsoid P to lie in a sphere.

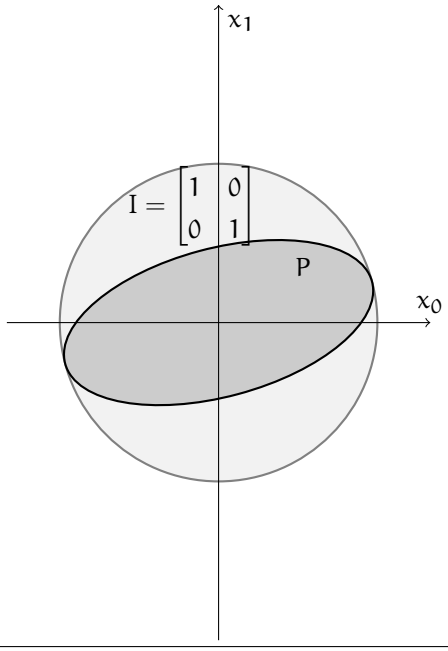
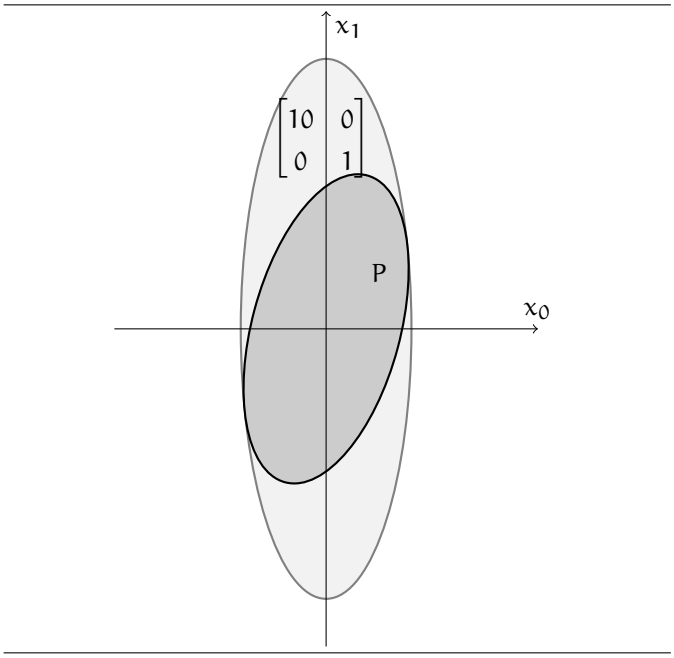


Figure 5.9 Constraining ellipsoid P to lie in an ellipsoid flatter in a given direction.



Example 12 *With the following matrices* A *and* B *of the running example:*

$$A := \begin{bmatrix} 0.9379 & -0.0381 & -0.0414 \\ -0.0404 & 0.968 & -0.0179 \\ 0.0142 & -0.0197 & 0.9823 \end{bmatrix} B := \begin{bmatrix} 0.0237 \\ 0.0143 \\ 0.0077 \end{bmatrix},$$

expressing a higher interest in the first variable as exposed above gives

$$P = \begin{bmatrix} 12.6465 & -14.1109 & -10.5402 \\ -14.1109 & 25.6819 & 3.06577 \\ -10.5402 & 3.06577 & 29.5981 \end{bmatrix}.$$

5.3 PIECEWISE QUADRATIC INVARIANTS

5.3.1 *Piecewise affine systems*

While strong results do exist for pure linear systems, most of them vanish in presence of non linearity such as switches between linear dynamics. As we saw in previous section, stable linear systems were guaranteed to admit a quadratic LYAPUNOV function and therefore a quadratic invariant. In switched linear systems, this property is undecidable [Blo+01, Theorem. 2]. The proposed methods are therefore meant to be understood as heuristics; trying to synthesize a meaningfully invariant for such systems.

As presented in Sec. 4.1.2 describing switched linear systems, these systems are composed by a set of linear updates A_i associated to conditions c_i. A common practice in control is to look for a common LYAPUNOV function: a quadratic LYAPUNOV function characterized by a positive definite matrix P such that

$$\begin{aligned} &A^{1\intercal}PA^1 - A^1 \preceq 0 \\ &A^{2\intercal}PA^2 - A^2 \preceq 0 \\ &\dots \\ &A^{n\intercal}PA^n - A^n \preceq 0 \end{aligned}$$

This common LYAPUNOV function decreases along trajectories regardless of the active cell (see Sec. 4.1.2). Note that, in this encoding, all information about the condition satisfied in each cells are ignored.

The main difficulty in the switched case is related to the change of dynamics: we must decrease whenever a transition from one cell to another is fired. Moreover, we only require the norm induced by the quadratic LYAPUNOV function P to be local i.e. positive only where the law is used.

Therefore, our main goal is to synthesize a LYAPUNOV function $V(x,u)$ and an associated bound α characterizing the invariant of reachable states as a sublevel-set S_α, such that

$$\forall i \in \mathcal{I}, \forall (x,u) \in X^i, V(x,u) \leqslant \alpha \tag{77}$$

$$\begin{aligned} &\forall i,j \in \mathcal{I}, \forall (x,u) \in X^i, \forall (x',u') \in X^j, \text{ s.t.} \\ &x' = A^i x + B^i u + b^i, V(x,u) \geqslant V(x',u') \end{aligned} \tag{78}$$

5.3.2 *Encode conditions and switches as quadratic constraints*

In equations (77) and (78), the inequalities on V are local on cells. In (77), the function has to decrease only on feasible transitions from cell X^i to cell X^j.

In order to encode the problem as a set of linear matrix inequalities, we need to express conditions associated to each cell in suitable form. For SDP, encoding constraints requires to be able to express cell membership or feasible transitions as quadratic constraints.

Quadratization of cells

We recall that for us local means that true on a cell and thus true on a polyhedron. Using the homogeneous version of a cell, we can define local positiveness on a polyhedral cone. Let Q be a $d \times d$ symmetric matrix and M be a $n \times d$ matrix. Local positivity in our case means that $My \geqslant 0 \implies y^\intercal Q y \geqslant 0$. The problem will be to write the local positivity as a constraint without implication. The problem is not new (e.g. the survey paper [ISoo]). [MJ81] proves that local positivity is equivalent, when M has a full row rank, to $Q - M^\intercal C M \succeq 0$ where C is a

copositive matrix i.e. $x^T C x \geqslant 0$ if $x \geqslant 0$. First in general (when the rank of M is not necessarily equal to its number of rows), note that if $Q - M^T C M \succeq 0$ for some copositive matrix C then Q satisfies $My \geqslant 0 \implies y^T Q y \geqslant 0$. Secondly every matrix C with nonnegative entries is copositive. Since copositivity seems to be as difficult as local positivity to handle, we will restrict copositive matrices to be matrices which nonnegative entries. The idea is instead of using cells as polyhedral cones, we use a quadratization of cells by introducing nonnegative entries and we will define the quadratization of a cell X^i by:

$$\overline{X^i} = \left\{ \begin{pmatrix} x \\ u \end{pmatrix} \in \mathbb{R}^{d+m} \;\middle|\; \begin{pmatrix} 1 \\ x \\ u \end{pmatrix}^T E^{iT} W^i E^i \begin{pmatrix} 1 \\ x \\ u \end{pmatrix} \geqslant 0 \right\} \quad (79)$$

where W^i is a $(1+n_i) \times (1+n_i)$ symmetric matrix with nonnegative entries and $E^i = \begin{pmatrix} E^i_s \\ E^i_w \end{pmatrix}$ with $E^i_s = \begin{pmatrix} 1 & 0_{1\times(d+m)} \\ c^i_s & -T^i_s \end{pmatrix}$ and $E^i_w = \begin{pmatrix} c^i_w & -T^i_w \end{pmatrix}$. Recall that n_i is the number of rows of T^i. The matrix E^i is thus of the size $n_i + 1 \times (1 + d + m)$. The goal of adding the row $(1, 0_{1\times(d+m)})$ is to avoid adding the opposite of a vector of X^i in $\overline{X^i}$. Indeed without this latter vector $\overline{X^i}$ would be symmetric. We illustrate this fact at Example 13. Note that during optimization process, matrices W^i will be decision variables.

Example 13 (Homogenization) *Let us take the polyhedron $X = \{x \in \mathbb{R} \mid x \leqslant 1\}$. Using our notations, we have $X = \{x \mid M(1\ x)^T \geqslant 0\}$ with $M = (1\ -1)$. Let us consider two cases, the first one without adding the row and the second one using it.*

Without any modification, the quadratization of X relative to a nonnegative real W is $X' = \{x \mid (1\ x)M^T W M(1\ x)^T \geqslant 0\}$. But $(1\ x)M^T W M(1\ x)^T = W(1\ x)(1\ -1)^T(1\ -1)(1\ x)^T = 2W(1-x)^2$. Hence, $X' = \mathbb{R}$ for all nonnegative real W.

Now let us take $E = \begin{pmatrix} 1 & 0 \\ 1 & -1 \end{pmatrix}$ defined as M with the additional row 1. The quadratization as defined by Equation (79) relative to a 2×2 symmetric matrix W with nonnegative coefficients is $\overline{X} = \{x \mid (1\ x)E^T W E(1\ x)^T \geqslant 0\}$. We have:

$$\begin{aligned}(1\ x)\begin{pmatrix} 1 & 1 \\ 0 & -1 \end{pmatrix}\begin{pmatrix} w_1 & w_3 \\ w_3 & w_2 \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 1 & -1 \end{pmatrix}(1\ x)^T \\ = w_1 + 2w_3(1-x) + w_2(1-x)^2\end{aligned}$$

To take a matrix W such that $w_2 = w_1 = 0$ and $w_3 > 0$ implies that $\overline{X} = X$.

Now we introduce an example of the quadratization of the cell X^1 for our running example, cf. §4.

Example 14 *Let us consider the running example and the cell X^1. We recall that X^1 is characterized by the matrices and vectors:*

$$\begin{cases} T^1_s = \begin{pmatrix} -9 & 7 & 6 \\ -4 & 8 & -8 \end{pmatrix} \\ c^1_s = (5\ 4)^T \end{cases},$$

$$\begin{cases} T^1_w = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & -1 \end{pmatrix} \\ c^1_w = (3\ 3)^T \end{cases}$$

$$\textit{and } E^1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 5 & 9 & -7 & -6 \\ 4 & 4 & -8 & 8 \\ 3 & 0 & 0 & -1 \\ 3 & 0 & 0 & 1 \end{pmatrix}$$

As suggested we have added the row $(1, 0_{1\times3})$. Take for example the matrix:

$$W^1 = \begin{pmatrix} 63.0218 & 0.0163 & 0.0217 & 12.1557 & 8.8835 \\ 0.0163 & 0.0000 & 0.0000 & 0.0267 & 0.0031 \\ 0.0217 & 0.0000 & 0.0000 & 0.0094 & 0.0061 \\ 12.1557 & 0.0267 & 0.0094 & 4.2011 & 59.5733 \\ 8.8835 & 0.0031 & 0.0061 & 59.5733 & 3.0416 \end{pmatrix}$$

We have

$$\begin{aligned}\overline{X^1} &= \{(x,y,u) \mid (1,x,y,u)E^1 W^1 E^1 (1,x,y,u)^T \geqslant 0\} \\ &\supseteq X^1\end{aligned}.$$

Local positivity of quadratic forms will also be used when a transition from a cell to an other is fired . For the moment, we are interested in the set of (x,u) such that $(x,u) \in X^i$ and whose the image is in X^j and we denote by X^{ij} the set:

$$\left\{ \begin{pmatrix} x \\ u \end{pmatrix} \in \mathbb{R}^{d+m} \;\middle|\; \begin{array}{l} \begin{pmatrix} x \\ u \end{pmatrix} \in X^i \text{ and} \\ (A^i x + B^i u + b^i, u) \in X^j \end{array} \right\}$$

for all pairs $i, j \in \mathcal{I}$. Note that in [MFMoo], the authors take into account all pairs (i,j) such that there exists a state x_k at moment k in X^i and the image of x_k that is x_{k+1} is in X^j. We will discuss in Subsection 5.3.2 the computation or a reduction to possible switches using linear programming as suggested in [Bis+05]. To construct a quadratization of X^{ij}, we use the same approach

than before by introducing a $(1+n_i+n_j)\times(1+n_i+n_j)$ symmetric matrix U^{ij} with nonnegative entries to get a set $\overline{X^{ij}}$ defined as:

$$\overline{X^{ij}} = \left\{ \begin{pmatrix} x \\ u \end{pmatrix} \in \mathbb{R}^{d+m} \;\middle|\; \begin{pmatrix} 1 \\ x \\ u \end{pmatrix}^T E^{ij\,T} U^{ij} E^{ij} \begin{pmatrix} 1 \\ x \\ u \end{pmatrix} \geqslant 0 \right\} \tag{80}$$

where $E^{ij} = \begin{pmatrix} E_s^{ij} \\ E_w^{ij} \end{pmatrix}$ with

$$E_s^{ij} = \begin{pmatrix} 1 & 0_{1\times(d+m)} \\ c_s^i & -T_s^i \\ c_s^j - T_s^j \begin{pmatrix} b^i \\ 0 \end{pmatrix} & -T_s^j \begin{pmatrix} A^i & B^i \\ 0_{d\times m} & \mathrm{Id}_{m\times m} \end{pmatrix} \end{pmatrix}$$

and

$$E_w^{ij} = \begin{pmatrix} c_w^i & -T_w^i \\ c_w^j - T_w^j \begin{pmatrix} b^i \\ 0 \end{pmatrix} & -T_w^j \begin{pmatrix} A^i & B^i \\ 0_{d\times m} & \mathrm{Id}_{m\times m} \end{pmatrix} \end{pmatrix} \tag{81}$$

Switching cells

We have to manage another constraint which comes from the cell switches. After applying the available law in cell X^i, we have to specify the reachable cells i.e. the cells X^j such that there exists (x,u) satisfying:

$$(x,u) \in X^i \text{ and } (A^i x + B^i u + b^i, u) \in X^j$$

We say that a switch from i to j is fireable iff:

$$\left\{ (x,u) \in \mathbb{R}^{d+m} \;\middle|\; \begin{array}{l} T_s^i (x,u)^T \ll c_s^i \\ T_s^j (A^i x + B^i u + b^i, u)^T \ll c_s^j \\ T_w^i (x,u)^T \leqslant c_w^i \\ T_w^j (A^i x + B^i u + b^i, u)^T \leqslant c_w^j \end{array} \right\} \neq \emptyset \tag{82}$$

We will denote by $i \to j$ if the switch from i to j is fireable. Recall that the symbol $<$ means that we can deal with both strict inequalities and inequalities. Problem (82) is a linear programming feasibility problem with both strict and weak inequalities. However, we only check whether the system is solvable and we can detect infeasibility by using Motzkin transposition theorem [Mot51]. Motzkin's theorem is an alternative type theorem, that is we oppose two linear systems such that exactly one of the two is feasible. To describe the alternative system, we have to separate strict and weak inequalities and use the matrices E_s^{ij} and E_w^{ij} defined at Equation (81). Problem (82) is equivalent to check whether the set $\{y = (z,x,u) \in \mathbb{R}^{1+d+m} \mid E_w^{ij} y \geqslant 0,\ E_s^{ij} y \gg 0\}$ is empty or not. To detect feasibility we test the infeasibility of the alternative system defined as:

$$\begin{cases} (E_s^{ij})^T p^s + (E_w^{ij})^T p = 0 \\ \sum_{k\in\mathbb{I}} p_k^s = 1 \\ p_k^s \geqslant 0,\ \forall k \in \mathbb{I} \\ p_i \geqslant 0,\ \forall i \notin \mathbb{I} \end{cases} \tag{83}$$

From Motzkin's transposition theorem [Mot51], we get the following proposition.

Proposition 1 *Problem (82) is feasible iff Problem (83) is not.*

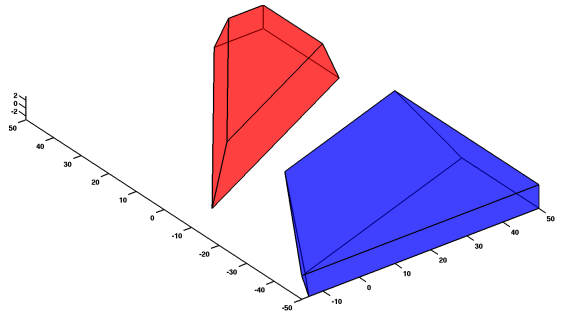
However, reasoning directly on the matrices can allow unfireable switches. For certain initial conditions, for all $k \in \mathbb{N}$, the condition $(x_k, u_k) \in X^i$ and $(A^i x_k + B^i u + b^i, u) \in X^j$ does not hold whereas Problem (82) is feasible. To avoid it, we must know all the possible trajectories of the system (which we want to compute) and remove all inactivated switches. A sound way to underapproximate unfireable transitions is to identify unsatisfiable sets of linear constraints.

Example 15 *We continue to detail our running example. More precisely, we consider the possible switches. We take for example the cell X^2. To switch from cell X^2 to cell X^1 is possible if the following system of linear inequalities has a solution:*

$$\begin{array}{rcl} -9x + 7y + 6u & < & 5 \\ -0.8532x + 2.5748y - 10.4460 & < & -68 \\ -3.3662x + 2.1732y - 1.1084u & < & -58 \\ 4x - 8y + 8u & \leqslant & -4 \\ u & \leqslant & 3 \\ -u & \leqslant & 3 \end{array} \tag{84}$$

The two first consists in constraining the image of (x,y,u) to belong to X^1 and the four last constraints correspond to the definition of X^2. The representation of these two sets (X^2 and the preimage of X^1 by the law defined in X^2) is given at Figure 5.10.

Figure 5.10 The truncated representation of X^2 in red and the preimage of X^1 by the law inside X^2 in blue



We see at Figure 5.10 that the system of inequalities defined at Equation (84) seems to not have solutions. We check that using Equation (83) and Proposition 1. The matrices E_s^{ij} *and* E_w^{ij} *of Equation (83) are in this example:*

$$E_s^{21} = \begin{pmatrix} 5 & 9 & -7 & -6 \\ -68 & 0.8532 & -2.5748 & 10.446 \\ -58 & 3.3662 & -2.1732 & 1.1084 \end{pmatrix}$$

$$\text{and } E_w^{21} = \begin{pmatrix} -4 & -4 & 8 & -8 \\ 3 & 0 & 0 & -1 \\ 3 & 0 & 0 & 1 \end{pmatrix}$$

We thus solve the linear program defined in Equation (83) (with Matlab and Linprog) and we found $p = (0.8735, 0.0983, 0.0282)^\intercal$ *and* $q = (0.3325, 14.2500, 7.8461)^\intercal$. *This means that the alternative system is feasible and consequently the initial is not from Proposition 1. Finally, the transition from* X^2 *to* X^1 *is not possible.*

5.3.3 *Local invariants with coupling conditions*

As in the linear case, we are relying here in SDP solver and LMI encoding, the unknowns of the optimization problems have to be at most quadratic.

Piecewise quadratic Lyapunov *function*

The Lyapunov function V is piecewise defined, relying on the partition of cells provided by the analyzed piecewise affine system. This V is defined as:

$$\begin{aligned} V(x,u) &= V^i(x,u), \text{ if } \begin{pmatrix} x \\ u \end{pmatrix} \in X^i \\ &= \begin{pmatrix} x \\ u \end{pmatrix}^\intercal P^i \begin{pmatrix} x \\ u \end{pmatrix} + 2q^{i\intercal} \begin{pmatrix} x \\ u \end{pmatrix}, \text{ if } \begin{pmatrix} x \\ u \end{pmatrix} \in X^i \end{aligned}$$

The function V^i is thus a local function only defined on X^i.

A sublevel set S_α of V of level $\alpha \in \mathbb{R}$ is represented as:

$$\begin{aligned} S_\alpha &= \bigcup_{i\in\mathcal{I}} S_{i,\alpha} \\ &= \bigcup_{i\in\mathcal{I}} \left\{ \begin{pmatrix} x \\ u \end{pmatrix} \in X^i \mid \begin{pmatrix} x \\ u \end{pmatrix}^\intercal P^i \begin{pmatrix} x \\ u \end{pmatrix} + 2q^{i\intercal} x \leqslant \alpha \right\} \\ &= \bigcup_{i\in\mathcal{I}} \left\{ \begin{pmatrix} x \\ u \end{pmatrix} \in X^i \mid \begin{pmatrix} 1 \\ x \\ u \end{pmatrix}^\intercal \begin{pmatrix} -\alpha & q^{i\intercal} \\ q^i & P^i \end{pmatrix} \begin{pmatrix} 1 \\ x \\ u \end{pmatrix} \leqslant 0 \right\} \end{aligned}$$

The set $S_{i,\alpha}$ is thus the local sublevel set of V^i associated to the level α.

So we are looking a family of pairs of a matrix and a vector $\{(P^i, q^i)\}_{i\in\mathcal{I}}$ and a real $\alpha \in \mathbb{R}$ such that S_α is invariant by the piecewise affine system. To obtain invariance property, we have to constraint S_α to contain the initial conditions of the system. Finally, to prove that the reachable set is bounded, we have to constraint S_α to be bounded.

Before deriving the semi-definite constraints, let us first state a useful result in Proposition 2. This result, which is a special case of the S-Procedure 4.6, allows to encode implications into semi-definite constraints in a safe way. The implication must involve quadratic inequalities on both sides.

Proposition 2 *Let* A, B, C *be* $d \times d$ *matrices. Then,* $C + A + B \succeq 0$ *holds implies that the implication* $(y^\intercal A y \leqslant 0 \wedge y^\intercal B y \leqslant 0) \implies y^\intercal C y \geqslant 0$ *holds.*

Writing invariance as semi-definite constraints

We assume that $(x,u) \in X^i \cap S_{i,\alpha}$ (this index i is unique). Invariance means that if we apply the available law to (x,u) and suppose that the image of (x,u) belongs to some cell X^j (notation $i \to j$), then the image of (x,u) belongs to $S_{j,\alpha}$. Note that $(x,u) \in X^i$ and its image is supposed to be in X^j then $(x,u) \in X^{ij}$. Let $(i,j) \in \mathcal{I}^2$ such that $i \to j$, invariance translated in inequalities and implication gives :

$$\begin{aligned} &\begin{pmatrix} x \\ u \end{pmatrix} \in X^{ij} \wedge \begin{pmatrix} x \\ u \end{pmatrix} \in S_{i,\alpha} \\ \implies &\begin{pmatrix} A^i x + B^i u + b^i \\ u \end{pmatrix} \in S_{j,\alpha} \end{aligned} \tag{85}$$

We can use the relaxation of Subsection 5.3.2 as representation of cells and use matrix variables W^i and U^{ij} to

encode their quadratization. We get, for $(i,j) \in \mathcal{I}^2$ such that $i \to j$:

$$\begin{aligned} & \begin{pmatrix} 1 \\ x \\ u \end{pmatrix}^\intercal E^{ij\intercal} U^{ij} E^{ij} \begin{pmatrix} 1 \\ x \\ u \end{pmatrix} \geqslant 0 \\ \wedge \quad & \begin{pmatrix} 1 \\ x \\ u \end{pmatrix}^\intercal \begin{pmatrix} -\alpha & q^{i\intercal} \\ q^i & P^i \end{pmatrix} \begin{pmatrix} 1 \\ x \\ u \end{pmatrix} \leqslant 0 \\ \implies \quad & \begin{pmatrix} 1 \\ x \\ u \end{pmatrix}^\intercal \left(F^{i\intercal} \begin{pmatrix} -\alpha & q^{j\intercal} \\ q^j & P^j \end{pmatrix} F^i \right) \begin{pmatrix} 1 \\ x \\ u \end{pmatrix} \leqslant 0 \end{aligned} \tag{86}$$

where E^{ij} is the matrix defined at Equation (80) and F^i is defined at Equation (37).

Finally, we obtain a stronger condition by considering semi-definite constraints such as Equation (87). Proposition 2 proves that if $(P^i, P^j, q^i, q^j, U^{ij})$ is a solution of Equation (87) then $(P^i, P^j, q^i, q^j, U^{ij})$ satisfies Equation (86). For $(i,j) \in \mathcal{I}^2$ such that $i \to j$:

$$-F^{i\intercal} \begin{pmatrix} 0 & q^{j\intercal} \\ q^j & P^j \end{pmatrix} F^i + \begin{pmatrix} 0 & q^{i\intercal} \\ q^i & P^i \end{pmatrix} - E^{ij\intercal} U^{ij} E^{ij} \succeq 0 \ . \tag{87}$$

Note that the symbol $-\alpha$ is canceled during the computation.

5.3.4 *Initialization and boundedness*

Integrating initial conditions

To complete the invariance property, the invariant set must contain initial conditions. Suppose that initial condition is a polyhedron $X^0 = \{(x,u) \in \mathbb{R}^{d+m} \mid T_w^0(x,u) \leqslant c_w^0,\ T_s^0(x,u) \ll c_s^0\}$. We must have $X^0 \subseteq S_\alpha$. But X^0 is contained in the union of X^i. Hence, X^0 is the union over $i \in \mathcal{I}$ of the sets $X^0 \cap X^i$. If, for all $i \in \mathcal{I}$, the set $X^0 \cap X^i$ is contained in $S_{i,\alpha}$ then $X^0 \subseteq S_\alpha$. We can use the same method as before to express that all sets $S_{i,\alpha}$ such that $X^0 \cap X^i \neq \emptyset$ must contain $X^0 \cap X^i$. In term of implications, it can be rewritten as for all $i \in \mathcal{I}$ such that $X^0 \cap X^i \neq \emptyset$:

$$(x,u) \in X^0 \cap X^i \implies (x,u) P^i (x,u)^\intercal + 2(x,u) q^i \leqslant \alpha \tag{88}$$

Since $X^0 \cap X^i$ is a polyhedron, it admits some quadratization that is: $\overline{X^0 \cap X^i} = \{(x,u) \in \mathbb{R}^{d+m} \mid (1,x,u) E^{0i\intercal} Z^i E^{0i} (1,x,u)^\intercal \geqslant 0\}$ where $E^{0i} = \begin{pmatrix} E_s^{0i} \\ E_w^{0i} \end{pmatrix}$ with:

$$E_w^{0i} = \begin{pmatrix} c_w^0 & -T_w^0 \\ c_w^i & -T_w^i \end{pmatrix} \text{ and } E_s^{0i} = \begin{pmatrix} 1 & 0_{1\times(d+m)} \\ c_s^0 & -T_s^0 \\ c_s^i & -T_s^i \end{pmatrix}$$

and Z^i is some symmetric matrix whose coefficients are nonnegative.

For all $i \in \mathcal{I}$ such that $X^0 \cap X^i \neq \emptyset$, we obtain a stronger notion by introducing semi-definite constraints:

$$-\begin{pmatrix} -\alpha & q^{i\intercal} \\ q^i & P^i \end{pmatrix} - E^{0i\intercal} Z^i E^{0i} \succeq 0 \tag{89}$$

Proposition 2 proves that if (P^i, q^i, Z^i) is a solution of Equation (89) then (P^i, q^i, Z^i) satisfies Equation (88).

Note since $X^0 \cap X^i$ is a polyhedron then its emptiness can be decided by checking the feasibility of the linear problem (90) and by using of same argument than Proposition 1.

$$\begin{cases} (E_s^{0i})^\intercal p^s + (E_w^{0i})^\intercal p = 0 \\ \sum_{k \in \mathbb{I}} p_k^s = 1 \\ p_k^s \geqslant 0,\ \forall k \in \mathbb{I} \\ p_i \geqslant 0,\ \forall i \notin \mathbb{I} \end{cases} \tag{90}$$

Linear program (90) is feasible iff $X^0 \cap X^i = \emptyset$.

Writing boundedness as semi-definite constraints

The sublevel S_α is bounded if and only if for all $i \in \mathcal{I}$, the sublevel $S_{i,\alpha}$ is bounded. The boundedness constraint in term of implications is, for all $i \in \mathcal{I}$, there exists $\beta \geqslant 0$:

$$(x,u) \in X^i \wedge \begin{pmatrix} x \\ u \end{pmatrix} \in S_{i,\alpha} \implies \|(x,u)\|_2^2 \leqslant \beta \tag{91}$$

where $\|\cdot\|_2$ denotes the Euclidean norm of $\mathbb{R}^{d+m}$.

As invariance, we use the quadratization of X^i and the definition of $S_{i,\alpha}$. We use the fact that $\|(x,u)\|_2^2 = \begin{pmatrix} x \\ u \end{pmatrix}^T \mathrm{Id}_{(d+m)\times(d+m)} \begin{pmatrix} x \\ u \end{pmatrix}$ and we get for all $i \in \mathcal{I}$:

$$\begin{pmatrix} 1 \\ x \\ u \end{pmatrix}^T E^{iT} W^i E^i \begin{pmatrix} 1 \\ x \\ u \end{pmatrix} \geqslant 0 \wedge \begin{pmatrix} 1 \\ x \\ u \end{pmatrix}^T \begin{pmatrix} -\alpha & q^{iT} \\ q^i & P^i \end{pmatrix} \begin{pmatrix} 1 \\ x \\ u \end{pmatrix} \leqslant 0 \implies \begin{pmatrix} 1 \\ x \\ u \end{pmatrix}^T \begin{pmatrix} -\beta & 0_{1\times(d+m)} \\ 0_{(d+m)\times 1} & \mathrm{Id}_{(d+m)\times(d+m)} \end{pmatrix} \begin{pmatrix} 1 \\ x \\ u \end{pmatrix} \leqslant 0 \tag{92}$$

where E^i is defined in Equation (79).

Finally, as invariance we obtain a stronger condition by considering semi-definite constraints such as Equation (93). Proposition 2 proves that (P^i, q^i, W^i) is a solution of Equation (93) the (P^i, q^i, W^i) satisfies Equation (92). For all $i \in \mathcal{I}$:

$$-E^{iT} W^i E^i + \begin{pmatrix} -\alpha & q^{iT} \\ q^i & P^i \end{pmatrix} + \begin{pmatrix} \beta & 0_{1\times(d+m)} \\ 0_{(d+m)\times 1} & -\mathrm{Id}_{(d+m)\times(d+m)} \end{pmatrix} \succeq 0 \tag{93}$$

5.3.5 *Overall method*

The following algorithm summarizes the method.

Figure 5.11 Algorithm to compute piecewise quadratic invariant for piecewise affine dynamical systems.

input : Piecewice affine system defined by $T^i_{s,w}, c^i_{s,w}, A^i, B^i, b^i, \forall i \in \mathcal{I}$
local : $E^i, E^{ij}, E^{0i}, \forall i,j \in \mathcal{I}$
output: $\alpha, \beta, P^i, q^i, Z^i, W^i, U^{ij}, \forall i,j \in \mathcal{I}$

1. Compute quadratization of cells E^i using Equation (79), $\forall i \in \mathcal{I}$;
2. Over-approximate feasible switches: compute possible switches $L \in \mathcal{I}^2$ using Equation (82);
3. Compute quadratization of switches E^{ij} using Equation (80), $\forall i,j \in L$;
4. Compute quadratization of initialization E^{0i} using Equation (89), $\forall i \in \mathcal{I}$;
5. Solve the SDP problem of Equation(94)
6. Invariants:
7. $\bigcup_{i\in\mathcal{I}} \left\{ (x,u) P^i_{opt} (x,u)^T + 2(x,u) q^i_{opt} \leqslant \alpha_{opt} \,\middle|\, (x,u) \in X^i \right\}$
8. $\|(x,u)\| \leqslant \beta_{opt}$

Figure 5.12 Summary of generated SDP problem for piecewise affine discrete systems

$$\begin{aligned} &\text{minimize } \alpha + \beta \\ &\text{st.} \forall i \in \mathcal{I}, (i,j) \in L, \\ &-F^{iT} \begin{pmatrix} 0 & q^{jT} \\ q^j & P^j \end{pmatrix} F^i + \begin{pmatrix} 0 & q^{iT} \\ q^i & P^i \end{pmatrix} - E^{ijT} U^{ij} E^{ij} \succeq 0 \ . \\ &- \begin{pmatrix} -\alpha & q^{iT} \\ q^i & P^i \end{pmatrix} - E^{0iT} Z^i E^{0i} \succeq 0 \\ &\qquad -E^{iT} W^i E^i + \begin{pmatrix} -\alpha & q^{iT} \\ q^i & P^i \end{pmatrix} \\ &+ \begin{pmatrix} \beta & 0_{1\times(d+m)} \\ 0_{(d+m)\times 1} & -\mathrm{Id}_{(d+m)\times(d+m)} \end{pmatrix} \succeq 0 \end{aligned} \tag{94}$$

5.3.6 *Example*

The method applied to our piecewise affine system defined in Sec. 4.1.2 computes the following values:

$$\begin{aligned} \alpha_{opt} &= 242.0155 \\ \beta_{opt} &= 2173.8501 \end{aligned}$$

This means that $\|(x,y,u)\|_2^2 = x^2 + y^2 + u^2 \leqslant \beta_{opt}$. We can conclude, for example, that the values taken by the variables x are between $[-46.6154, 46.6154]$.

Note that this specific example does not admit a common Lyapunov function.

The value α_{opt} gives the level of the invariant sublevel of our piecewise quadratic Lyapunov function where the local quadratic functions are characterized by the following matrices and vectors:

$$P^1 = \begin{pmatrix} 1.0181 & -0.0040 & -1.1332 \\ -0.0040 & 1.0268 & -0.5340 \\ -1.1332 & -0.5340 & -13.7623 \end{pmatrix}$$
$$q^1 = (0.1252, 1.3836, -29.6791)^T$$

$$P^2 = \begin{pmatrix} 9.1540 & -7.0159 & -2.6659 \\ -7.0159 & 9.5054 & -2.4016 \\ -2.6659 & -2.4016 & -8.9741 \end{pmatrix}$$
$$q^2 = (-21.3830, -44.6291, 114.2984)^T$$

$$P^3 = \begin{pmatrix} 1.1555 & -0.3599 & -2.6224 \\ -0.3599 & 2.4558 & -2.8236 \\ -2.6224 & -2.8236 & -2.3852 \end{pmatrix}$$
$$q^3 = (-5.3138, 6.7894, -40.5537)^T$$

$$P^4 = \begin{pmatrix} 3.7314 & -3.4179 & -3.1427 \\ -3.4179 & 6.1955 & 0.9499 \\ -3.1427 & 0.9499 & -10.6767 \end{pmatrix}$$
$$q^4 = \begin{pmatrix} 28.5011, -73.5421, 48.2153 \end{pmatrix}^T$$

Finally, for conciseness reason, we do not provide here the matrix certificates W^i for each cell X^i, nor the matrices U^{ij} encoding quadratization matrices of polyhedron Xij. These matrices are computed by the analysis but do not provide useful information with respect to bounds.

5.4 K-INDUCTIVE QUADRATIC INVARIANTS

5.4.1 *K-induction principle*

The principle behind all compute invariants up to now was the inductiveness of computed LYAPUNOV function $V(x)$ with respect to the system transition function f.

However, as mentioned in Sect. 5.1.1 a property could be valid, ie. an invariant, without being directly inductive. In SMT-based model-checking, a trade-off to prove the validity of a property for a given transition system $(\Sigma, I \subseteq \Sigma, T \in \Sigma^2)$ is to search for a k-induction proof [KT11; SSS00] instead of a 1-induction one.

In k-induction, the base step addresses the property verification on all traces of length up to k, rooted in an initial state, while the inductive step intends to show that any trace suffix of length k validating the property, preserves it in the $k+1$-th step.

Definition 5.2 (k-induction) *Let (Σ, I, T) be a transition system over states Σ with initial states $I \subseteq \Sigma$ and transition relation $T \subseteq \Sigma \times \Sigma$. A safety property $\mathrm{Prop} \subseteq \Sigma$ is said k-inductive with respect to the transition system iff*

- *for all system traces of length less than k, all reachable states verify* Prop

$$\forall j \leqslant k \in \mathbb{N}, \forall x_0, \ldots, x_j \in \Sigma, \quad x_0 \in I \wedge \bigwedge_{i \in [0,j-1]} (x_i, x_{i+1}) \in T \implies x_j \in \mathrm{Prop} \tag{95}$$

- *for all system subtraces of length k satisfying* Prop *then the next state satisfies* Prop *as well*

$$\forall x_0, \ldots, x_k \in \Sigma, \quad \bigwedge_{i \in [0,k-1]} x_i \in \mathrm{Prop} \wedge (x_i, x_{i+1}) \in T \implies x_k \in \mathrm{Prop} \tag{96}$$

In our fixpoint characterization, this amounts to substitute

$$\{F(C) \subseteq C\} = \left\{X^{Init} \cup f(C) \subseteq C\right\}$$

by

$$\begin{aligned} & \{F^k(C) \subseteq C\} \\ = \; & \left\{X^{Init} \cup \bigcup_{1 \leqslant i \leqslant k} f^i(Init) \cup f^k(C) \subseteq C\right\} \\ = \; & \left\{ C \;\middle|\; \begin{array}{l} X^{Init} \subseteq C \\ f(X^{Init}) \subseteq C \\ f^2(X^{Init}) \subseteq C \\ \ldots f^k(X^{Init}) \subseteq C \\ f(C) \subseteq C \end{array} \right\} \end{aligned}$$

5.4.2 k-*inductive* LYAPUNOV *function*

We recall that we consider a piecewise system composed of cells X^i indexed by a set $_{\mathrm{ind}}$ of partition labels, such that $\Sigma = \bigcup_{i \in \mathcal{I}} X^i$, and which transition relation is piecewise defined with transitions T^i. The k-inductive property Prop denotes here a boundedness property represented by a sublevel set S^i_α of a LYAPUNOV function V. Then, a k-induction proof amounts to find this function V such that:

$$\forall j < k \in \mathbb{N}, \forall i_0, \ldots i_j \in \mathcal{I}, \forall x_0, \ldots, x_j \in \Sigma, \quad x_0 \in (I \cap X^0) \wedge \bigwedge_{i \in [0,j-1]} x_i \in X^i \wedge (x_i, x_{i+1}) \in T^i \implies x_j \in S_\alpha \tag{97}$$

$$\forall i_0, \ldots i_k \in \mathcal{I}, \forall x_0, \ldots, x_k \in \Sigma, \quad \bigwedge_{i \in [0,k-1]} x_i \in (X^i \cap S_\alpha) \wedge (x_i, x_{i+1}) \in T^i \implies x_k \in S_\alpha \tag{98}$$

Let I^* be the set of finite words of the letters in I, and I^*_k its restriction to words of length exactly k. In the following, we denote by $|w|$ the length of word w, by $a \cdot b$ the concatenation of the words a and b into ab and by $\mathrm{tl}(w)$ the tail of a non empty word w, i.e. w without its first letter. For example $\mathrm{tl}(i \cdot w) = w$.

Following LEE and DULLERUD approach [LD11; LD07; LDK07], we reinforce the equations (97)-(98) and search for a quadratic LYAPUNOV function V^w for each non empty sequence of switches $w = i_0 \cdot \ldots \cdot i_{k-1} \in \mathcal{I}^*_k$:

$$V^w \begin{pmatrix} x \\ u \end{pmatrix} = \begin{pmatrix} x \\ u \end{pmatrix}^t P^w \begin{pmatrix} x \\ u \end{pmatrix}$$

Let $S_{w \cdot i, \alpha}$ be the local quadratic sublevel set associated to the non empty path $w \cdot i$ and the level α:

$$S_{w\cdot i,\alpha} = \left\{ \begin{pmatrix} x \\ u \end{pmatrix} \in X^i \;\middle|\; \begin{array}{l} V^{w\cdot i}(x,u) \leqslant \alpha \wedge \exists \begin{pmatrix} x' \\ u' \end{pmatrix} \text{ s.t.} \\ \left(\begin{pmatrix} x' \\ u' \end{pmatrix}, \begin{pmatrix} x \\ u \end{pmatrix} \right) \in T^j \wedge \\ \begin{pmatrix} x' \\ u' \end{pmatrix} \in S_{w,\alpha} \\ \text{when } w = w' \cdot j \end{array} \right\}$$

Let us consider a non empty finite path w, the sublevel $S_{w,\alpha}$ denotes that the $|w|$ predecessors of $\begin{pmatrix} x \\ u \end{pmatrix}$ belong to the sublevel associated to the path prefixes.

E.g.

$$S_{123,\alpha} = \left\{ \begin{pmatrix} x \\ u \end{pmatrix} \in X^3 \;\middle|\; \begin{array}{l} V^{123}(x,u) \leqslant \alpha \wedge \exists \begin{pmatrix} x' \\ u' \end{pmatrix} \\ T \begin{pmatrix} x' \\ u' \end{pmatrix} = \begin{pmatrix} x \\ u \end{pmatrix} \wedge \begin{pmatrix} x' \\ u' \end{pmatrix} \in S_{12,\alpha} \end{array} \right\}$$

$$S_{12,\alpha} = \left\{ \begin{pmatrix} x \\ u \end{pmatrix} \in X^2 \;\middle|\; \begin{array}{l} V^{12}(x,u) \leqslant \alpha \wedge \exists \begin{pmatrix} x' \\ u' \end{pmatrix} \\ T \begin{pmatrix} x' \\ u' \end{pmatrix} = \begin{pmatrix} x \\ u \end{pmatrix} \wedge \begin{pmatrix} x' \\ u' \end{pmatrix} \in S_{1,\alpha} \end{array} \right\}$$

$$S_{1,\alpha} = \left\{ \begin{pmatrix} x \\ u \end{pmatrix} \in X^1 \;\middle|\; V^1(x,u) \leqslant \alpha \right\}$$

The equations can be rephrased as:

$$\forall\, w \in \mathcal{I}_1^*, \forall x \in \Sigma, x \in \mathcal{I} \cap X^i \implies x \in S_{w,\alpha} \tag{99}$$

$$\forall\, 1 \leqslant j < k, \forall w \cdot i \in \mathcal{I}_j^*, \forall x, y \in \Sigma, \quad (x,y) \in T^i \wedge x \in S_{w\cdot i,\alpha} \implies y \in S_{w\cdot i\cdot j,\alpha} \tag{100}$$

$$\forall\, w \cdot i \in \mathcal{I}_k^*, \forall x, y \in \Sigma, \quad (x,y) \in T^i \wedge x \in S_{w\cdot i,\alpha} \implies y \in S_{tl(w\cdot i)\cdot j,\alpha} \tag{101}$$

Proposition 3 *Any solution* $\{P^w | \forall 1 \leqslant j \leqslant k, w \in \mathcal{I}_j^*\}$ *of equations (99-101) satisfies (97-98) with* S^i_α *defined as*

$$S^i_\alpha = \left\{ \begin{pmatrix} x \\ u \end{pmatrix} \;\middle|\; \max_{w\cdot i \in \mathcal{I}^*} V^{w\cdot i} \begin{pmatrix} x \\ u \end{pmatrix} \leqslant \alpha \right\}.$$

We now adapt the semi-definite constraints of previous section to satisfy the k-inductive based constraints. While it is possible to target directly the synthesis of a k-inductive piecewise quadratic sublevel set, the approach typically starts from $k = 1$ and increase to $k+1$ in case of failure to find a minimal k-inductive piecewise quadratic invariant.

5.4.3 *Associate quadratic invariants to path suffixes*

We adapt the previous method to express properties over bounded sequence of past transitions.

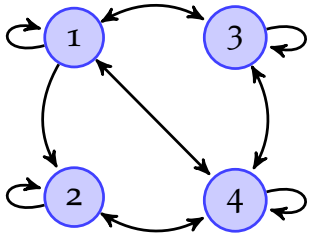
Characterizing the graph of possible switches – enumerating the paths.

As a first step, we compute the set of possible paths of given length up to k. First a graph $\mathcal{G} = (I, \mathtt{Init}, \mathit{Switches})$ denoting possible switches between cells $i \in \mathcal{I}$ is computed using the approach presented in Sect. 5.3.2.

$\mathtt{Init} = \{i \in \mathcal{I} | X^0 \cap X^i \neq \emptyset\}$ denotes the subset of cells $i \in \mathcal{I}$ that verify the initial conditions. This characterizes a set of polyhedral constraints which vacuity is computed using the method presented in Sect. 5.3.2.

We then enumerate the possible paths in the graph using classical graph algorithms. Let $\mathtt{Paths}^k$ be such set of paths of length up to k.

Figure 5.13 Switch graph of the running example



Example 16 *The figure 5.13 presents the possible transitions as over-approximated by our method presented in Sec. 5.3.2. Depending on the target length the following paths are generated:*

length	
1	*1,2,3,4*
2	*11, 12, 13, 14, 22, 24, 31, 33, 34, 41, 42, 43, 44*
3	*111, 112, 113, 114, 122, 124, 131, 133, 134, 141, 142, 143, 144, 222, 224, 241, 242, 243, 244, 311, 312, 313, 314, 331, 333, 334, 341, 342, 343, 344, 411, 412, 413, 414, 422, 424, 431, 433, 434, 441, 442, 443, 444*
4	...

Integrating initial conditions.

The initial condition only applies for the quadratic sublevel associated to initial cells. Let $\mathtt{Init}$ be the set of cells admitting initial elements, as defined in the graph construction.

By construction of the set of paths $\mathtt{Paths}^k$, it contains the single letter words denoting initial cells $\{i \mid i \in$

$\mathtt{Init}\} \subseteq \mathtt{Paths}^k$. The set of initial constraints only apply for these one letter word satisfying the initial condition:

$$(x,u) \in X^0 \cap X^i \implies (x,u)P^i(x,u)^\intercal + 2(x,u)q^i \leqslant \alpha \tag{102}$$

We can rely on the same stronger encoding as a semi-definite constraints, using the quadratization of the condition $X^0 \cap X^i$ as the matrix E^{0i}:

$$-\begin{pmatrix} -\alpha & q^{i\intercal} \\ q^i & P^i \end{pmatrix} - E^{0i\intercal} Z^i E^{0i} \succeq 0 \tag{103}$$

Note that, independently of the value of k, a system with n cells is parameterized by at most n Z^i variables.

Expressing transitions in initial and inductive cases as semi-definite constraints.

The equations (100) and (101) denoting a transition X^{ij} from cell X^i to cell X^j can be defined as:

$$\begin{aligned} &\begin{pmatrix} x \\ u \end{pmatrix} \in X^{ij} \wedge \begin{pmatrix} x \\ u \end{pmatrix} \in S_{w\cdot i,\alpha} \\ \implies &\begin{pmatrix} A^i x + B^i u + b^i \\ u \end{pmatrix} \in S_{w\cdot i\cdot j,\alpha} \end{aligned} \tag{104}$$

$$\begin{aligned} &\begin{pmatrix} x \\ u \end{pmatrix} \in X^{ij} \wedge \begin{pmatrix} x \\ u \end{pmatrix} \in S_{w\cdot i,\alpha} \wedge |w\cdot i| = k \\ \implies &\begin{pmatrix} A^i x + B^i u + b^i \\ u \end{pmatrix} \in S_{\mathtt{tl}(w\cdot i)\cdot j,\alpha} \end{aligned} \tag{105}$$

As before, these constraints are first relaxed with the use of quadratization of cell transitions E^{ij}, and then expressed as semi-definite constraints using Prop. 2.

when $|i\cdot w| = k$:

$$\begin{aligned} &-F^{i\intercal}\begin{pmatrix} 0 & q^{\mathtt{tl}(w\cdot i)\cdot j\intercal} \\ q^{\mathtt{tl}(w\cdot i)\cdot j} & P^{\mathtt{tl}(w\cdot i)\cdot j} \end{pmatrix} F^i \\ &+\begin{pmatrix} 0 & q^{w\cdot i\intercal} \\ q^{w\cdot i} & P^{w\cdot i} \end{pmatrix} - E^{ij\intercal} U^{w\cdot i,j} E^{ij} \succeq 0 \ . \end{aligned} \tag{106}$$

when $|i\cdot w| < k$:

$$\begin{aligned} &-F^{i\intercal}\begin{pmatrix} 0 & q^{w\cdot i\cdot j\intercal} \\ q^{w\cdot i\cdot j} & P^{w\cdot i\cdot j} \end{pmatrix} F^i \\ &+\begin{pmatrix} 0 & q^{w\cdot i\intercal} \\ q^{w\cdot i} & P^{w\cdot i} \end{pmatrix} - E^{ij\intercal} U^{w\cdot i,j} E^{ij} \succeq 0 \ . \end{aligned} \tag{107}$$

Note that we have $|\mathtt{Paths}^k|$ variables q^w, P^w and $|\mathtt{Paths}^k| \times |I|$ variables $U^{w,j}$.

Expressing boundedness.

The boundedness constraint expressed as a semi-definite constraint is straightforward. We require that all path-associated quadratic sublevel is bounded by the same scalar β.

For all $w\cdot i \in \mathtt{Paths}^k$, there exists $\beta \geqslant 0$:

$$(x,u) \in X^i \wedge \begin{pmatrix} x \\ u \end{pmatrix} \in S_{w\cdot i,\alpha} \implies \|(x,u)\|_2^2 \leqslant \beta \tag{108}$$

The associated semi-definite constraints is:

$$\begin{aligned} &-E^{i\intercal} W^{w\cdot i} E^i + \begin{pmatrix} -\alpha & q^{w\cdot i\intercal} \\ q^{w\cdot i} & P^{w\cdot i} \end{pmatrix} \\ &+\begin{pmatrix} \beta & 0_{1\times(d+m)} \\ 0_{(d+m)\times 1} & -\mathrm{Id}_{(d+m)\times(d+m)} \end{pmatrix} \succeq 0 \end{aligned} \tag{109}$$

We have here $|\mathtt{Paths}^k|$ variables W^w.

Remark: special case of length 1.

When one consider the equations (103), (106), (107), (109) with the set of paths $\mathtt{Paths}^1$ of length up to 1, we obtain exactly the equations (89), (87), (93). In that case, the equation (107) does not hold since no non empty word of length strictly less than 1 exists.

Overall method

The following algorithm summarizes the method.

Figure 5.14 Algorithm to compute piecewise k-inductive quadratic invariant for piecewise affine dynamical systems.

input : Piecewise affine system defined by $T^i_{s,w}, c^i_{s,w}, A^i, B^i, b^i, \forall i \in \mathcal{I}$
local : $E^i, E^{ij}, E^{0i}, \forall i,j \in \mathcal{I}$
output: $\alpha, \beta, P^{w\cdot i}, q^{w\cdot i}, Z^i, W^{w\cdot i}, U^{w\cdot i,j}, \forall i,j \in \mathcal{I}, w \in \mathtt{Paths}^k$

1 Compute quadratization of cells E^i using Equation (79), $\forall i \in \mathcal{I}$;
2 Over-approximate feasible switches: compute possible switches $L \in \mathcal{I}^2$ using Equation (82);
3 Compute $\mathtt{Paths}^k$ list of paths of length $\leqslant k$;
4 Compute quadratization of switches E^{ij} using Equation (80), $\forall i,j \in L$;
5 Compute quadratization of initialization E^{0i} using Equation (89), $\forall i \in \mathcal{I}$;
6 Solve the SDP problem of Equation (110)
7 Invariants:
8 $\bigcup_{w\cdot i \in \mathtt{Paths}^k} \left\{ \begin{array}{l} (x,u)P^{w\cdot i\,\mathtt{doti}}_{\mathrm{opt}}(x,u)^\intercal + 2(x,u)q^{w\cdot i}_{\mathrm{opt}} \leqslant \alpha_{\mathrm{opt}} \\ \text{when } (x,u) \in X^i \end{array} \right\}$
9 $\|(x,u)\| \leqslant \beta_{\mathrm{opt}}$

Figure 5.15 Summary of generated SDP problem for k-inductive piecewise quadratic invariant for piecewise affine discrete systems

$$
\begin{aligned}
&\text{minimize } \alpha+\beta\\
&\text{st.}\forall i\in\mathcal{I},(i,j)\in L,\\
&\quad\forall w\in \mathrm{Paths}^k,\ \text{s.t. } |i\cdot w|=k\\
&\quad -F^{i\intercal}\begin{pmatrix}0 & q^{tl(w\cdot i)\cdot j\intercal}\\ q^{tl(w\cdot i)\cdot j} & P^{tl(w\cdot i)\cdot j}\end{pmatrix}F^i\\
&\quad +\begin{pmatrix}0 & q^{w\cdot i\intercal}\\ q^{w\cdot i} & P^{w\cdot i}\end{pmatrix}-E^{ij\intercal}U^{w\cdot i,j}E^{ij}\succeq 0\ .\\
&\quad\forall w\in \mathrm{Paths}^k,\ \text{s.t. } |i\cdot w|<k\\
&\quad -F^{i\intercal}\begin{pmatrix}0 & q^{w\cdot i\cdot j\intercal}\\ q^{w\cdot i\cdot j} & P^{w\cdot i\cdot j}\end{pmatrix}F^i\\
&\quad +\begin{pmatrix}0 & q^{w\cdot i\intercal}\\ q^{w\cdot i} & P^{w\cdot i}\end{pmatrix}-E^{ij\intercal}U^{w\cdot i,j}E^{ij}\succeq 0\ .\\
&\quad -E^{i\intercal}W^{w\cdot i}E^i+\begin{pmatrix}-\alpha & q^{w\cdot i\intercal}\\ q^{w\cdot i} & P^{w\cdot i}\end{pmatrix}\\
&\quad +\begin{pmatrix}\beta & 0_{1\times(d+m)}\\ 0_{(d+m)\times 1} & -\mathrm{Id}_{(d+m)\times(d+m)}\end{pmatrix}\succeq 0
\end{aligned}
\tag{110}
$$

5.4.4 *Example*

The analysis of the running example with increased length generates the following results:

length	$\beta(\sqrt{\beta})$	α	$\lvert\mathrm{Paths}^k\rvert$
1	2173 (46.6154)	242.0155	4
2	2133 (46.1844)	233.0847	17
3	1652 (40.6448)	220.8596	73
4	1574 (39.6737)	228.5051	314

Note that the bound α on the piecewise quadratic sublevel applies on different sets of such local Lyapunov function. Their comparison is meaningless.

5.5 POLYNOMIAL INVARIANTS

5.5.1 *Fixpoints expression using polynomial Lyapunov functions*

We focus here on a more general family of problems: piecewise polynomial systems. We also rely on more general optimization problems: the cone of of positive polynomials and its relaxation/reinforcement as the cone of sum-of-squares polynomials (SOS).

Instead of expressing constraints as linear matrix inequalities (LMI), we can here express constraints as positive polynomial constraints. These constraints will be further reinforced by requiring them to be SOS polynomials.

Let us consider again the Equation 63 defining inductiveness of computed property with respect to the system semantics.

$$
\begin{cases}
X^{Init}\subseteq P,\\
\forall i\in\mathcal{I},\ T^i\left(P\cap X^i\right)\subseteq P\,.
\end{cases}
\tag{111}
$$

Encoding property P as the sublevel set of a polynomial p, we obtain the following problem:

$$
\begin{cases}
p(x)\leqslant 0\,, & \forall x\in X^{Init},\\
\forall i\in\mathcal{I},\ \ p\left(T^i(x)\right)\leqslant 0\,, & \forall x\in P\cap X^i\,.
\end{cases}
\tag{112}
$$

5.5.2 *Property-driven analysis*

As for the linear case, the previous equation only captures the inductiveness of the sublevel set induced by the polynomial Lyapunov function synthesized. However, no constraint encodes the need to obtain a precise (hence small) invariant. The expressivity of sum-of-square optimization enables us to encode a target property represented as a sublevel set of a polynomial and require the polynomial Lyapunov function to implies this property.

Considered properties: sublevel properties $\mathcal{P}_{\kappa,\alpha}$

We restrict our encoding to sublevel properties: those defined as sublevel sets of a given polynomial function.

Definition 5.3 (Sublevel property) *Given a polynomial function $\kappa\in\mathbb{R}[x]$ and $\alpha\in\mathbb{R}\cup\{+\infty\}$, we define the sublevel property $\mathcal{P}_{\kappa,\alpha}$ as follows:*

$$\mathcal{P}_{\kappa,\alpha}:=\{x\in\mathbb{R}^d\mid \kappa(x)\ll\alpha\}\,.$$

where $\ll$ denotes $\leqslant$ when $\alpha\in\mathbb{R}$ and denotes $<$ for $+\infty$. The expression $\kappa(x)<+\infty$ expresses the boundedness of $\kappa(x)$ without providing a specific bound α.

Example 17 (Sublevel property examples)
Boundedness. *When one wants to bound the reachable values of a system, we can try to bound the l_2-norm of the system: $\mathcal{P}_{\|\cdot\|_2^2,\infty}$ with $\kappa(x)=\|x\|_2^2$. The use of $\alpha=\infty$ does not impose any bound on $\kappa(x)$.*
Safe set. *Similarly, it is possible to check whether a specific bound is matched. Either globally using the l_2-norm and a specific α: $\mathcal{P}_{\|\cdot\|_2^2,\alpha}$, or bounding the reachable values of each variable: $\mathcal{P}_{\kappa_i,\alpha_i}$ with $\kappa_i:x\mapsto x_i$ and $\alpha_i\in\mathbb{R}$.*
Avoiding bad regions. *If the bad region can be encoded as a sublevel property $k(x)\leqslant 0$ then its negation $-k(x)\leqslant 0$ characterize the avoidance of that bad zone. Eg. if one wants to*

prove that the square norm of the program variables is always greater than 1, then we can consider the property $\mathcal{P}_{\kappa,\alpha}$ with $\kappa(x) = 1 - \|x\|_2^2$ and $\alpha = 0$.

A sublevel property is called *sublevel invariant* when this property is an inductive invariant of the discrete dynamical system collecting semantics $\mathfrak{C}$. In that case, the sublevel property itself would be an appropriate abstraction of the system. However, this is not the case in general. We rather propose to constrain the search for an inductive polynomial invariant guided by this sublevel property.

$\mathcal{P}_{\kappa,\alpha}$-driven inductive polynomial invariant

In this subsection, we explain how with adapt the constraints of Equation (112) to compute a d-variate polynomial $p \in \mathbb{R}[x]$ and a bound $w \in \mathbb{R}$, such that the polynomial sublevel sets $P := \{x \in \mathbb{R}^d \mid p(x) \leqslant 0\}$ and $\mathcal{P}_{\kappa,w}$ satisfy:

$$\mathfrak{C} \subseteq P \subseteq \mathcal{P}_{\kappa,w} \subseteq \mathcal{P}_{\kappa,\alpha}\,. \tag{113}$$

The first (from the left) inclusion forces P to be valid for the whole reachable values set. The second inclusion constraints all elements of P to satisfy the given sublevel property for a certain bound w. The last inclusion requires that the bound w is smaller than the desired level α. When $\alpha = \infty$, any bound w ensures the sublevel property.

We derive sufficient conditions on p and w to satisfy Equation (113). Thanks Equation (112), the first inclusion holds: $\mathfrak{C} \subseteq P$.

Now, we are interested in the second and third inclusions at Equation (113) that is the sublevel property satisfaction. The condition $P \subseteq \mathcal{P}_{\kappa,w} \subseteq \mathcal{P}_{\kappa,\alpha}$ can be formulated as follows:

$$\kappa(x) \leqslant w \leqslant \alpha\,, \quad \forall x \in P\,. \tag{114}$$

We recall that we have supposed that P is written as $\{x \in \mathbb{R}^d \mid p(x) \leqslant 0\}$ where $p \in \mathbb{R}[x]$. Finally, we provide sufficient conditions to satisfy both (112) and (114). Consider the following optimization problem:

$$\begin{cases} \inf_{p \in \mathbb{R}[x], w \in \mathbb{R}} \quad w, \text{ s.t.} \\ \qquad p(x) \leqslant 0\,, & \forall x \in X^{Init}\,, \\ \forall i \in \mathcal{I}, p\,(T^i(x)) \leqslant p(x)\,, & \forall x \in X^i\,, \\ \qquad \kappa(x) \leqslant w + p(x)\,, & \forall x \in \mathbb{R}^d\,. \end{cases} \tag{115}$$

We remark that α is not present in Problem (115). Indeed, since we minimize w, either there exists a feasible w such that $w \leqslant \alpha$ and we can exploit this solution or such w is not available and we cannot conclude. However, from Problem (115), we can extract (p, w) and in the case where the optimal bound w is greater than α, we could use this solution in conjunction with other abstractions as presented in the following chapter.

Lemma 5.4 *Let (p, w) be any feasible solution of Problem (115) with $w \leqslant \alpha$ or $w < \infty$ in the case of $\alpha = \infty$. Then, (p, w) satisfies both (112) and (114) with $P := \{x \in \mathbb{R}^d \mid p(x) \leqslant 0\}$. Finally, P and $\mathcal{P}_{\kappa,w}$ satisfy Equation (113).*

In practice, we rely on sum-of-squares programming to solve a relaxed version of Problem (115).

5.5.3 *SOS relaxed semantics*

One way to strengthen the three nonnegativity constraints of Problem (115) is to take $\lambda^i = 1$, for all $i \in \mathcal{I}$, $\nu = 1, \alpha = w$, then to consider the a *hierarchy* of SOS programs, parameterized by the integer m representing the half of the degree of p. All positivity constraints are expressed using sum-of-square polynomials of fixed degrees. As for the k-induction proof, one can increase the degree of such polynomial to consider more general problems. Equation (116) details the SOS problem submitted to the solver.

Figure 5.16 Property-driven polynomial invariants using SOS programming.

$$\begin{cases} \inf_{p \in \mathbb{R}[x]_{2m}, w \in \mathbb{R}} \quad w, \text{ s.t.} \\ \quad -p = \sigma_0 - \sum_{j=1}^{n_{in}} \sigma_j r_j^{in}\,, \\ \quad \forall i \in \mathcal{I},\ p - p \circ T^i = \sigma^i - \sum_{j=1}^{n_i} \mu_j^i r_j^i\,, \\ \quad w + p - \kappa = \psi\,, \\ \\ \quad \forall j = 1, \ldots, n_{in}\,,\ \sigma_j \in \Sigma[x]\,,\ \deg(\sigma_j r_j^{in}) \leqslant 2m\,, \\ \quad \sigma_0 \in \Sigma[x]\,,\ \deg(\sigma_0) \leqslant 2m\,, \\ \quad \forall i \in \mathcal{I}\,,\ \sigma^i \in \Sigma[x]\,,\ \deg(\sigma^i) \leqslant 2m \deg T^i\,, \\ \quad \forall i \in \mathcal{I}\,,\ \forall j = 1, \ldots, n_i\,,\ \mu_j^i \in \Sigma[x]\,, \\ \qquad \deg(\mu_j^i r_j^i) \leqslant 2m \deg T^i\,, \\ \quad \psi \in \Sigma[x]\,,\ \deg(\psi) \leqslant 2m\,. \end{cases} \tag{116}$$

where $\forall j \in [1, n_{in}], r_j^{in} \leqslant 0$ denotes the initial semi-algebraic set, and for all partition i, $\forall j \in [1, n_i], r_j^i \leqslant 0$ denotes the constraints describing the partition.

Proposition 4 *For a given $m \in \mathbb{N}$, let (p_m, w_m) be any feasible solution of Problem (116). Then, (p_m, w_m) is also a feasible solution of Problem (115). Moreover, if $w_m \leqslant \alpha$*

then both $P_m := \{x \in \mathbb{R}^d \mid p_m(x) \geqslant 0\}$ *and* $\mathcal{P}_{\kappa,w_m}$ *satisfy Equation* (113).

5.5.4 *Examples*

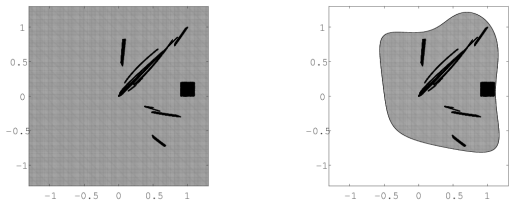
Here, we perform some numerical experiments while solving Problem (116) (given in Section 4.2.1) on several examples. In Section 5.5.4, we verify that the program of Example 5 satisfies some boundedness property. We also provide examples involving higher dimensional cases. Then, Section 5.5.4 focuses on other properties, such as checking that the set of variable values avoids an unsafe region.

We rely on the SDP solver Mosek to perform the computation.

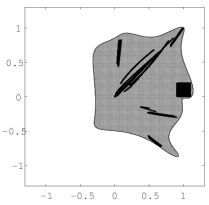
Checking boundedness of the set of variables values

Example 18 *Following Example 5, we consider the constrained piecewise discrete-time dynamical system* $\mathcal{S} = (X^{Init},\{X^1,X^2\},\{T^1,T^2\})$ *with* $X^{Init} = [0.9,1.1]\times[0,0.2]$, $X^1 = \{x \in \mathbb{R}^2 \mid r^1(x) \leqslant 0\}$ *with* $r^1 : x \mapsto \|x\|^2 - 1$, $X^2 = \{x \in \mathbb{R}^2 \mid r^2(x) < 0\}$ *with* $r^2 = -r^1$ *and* $T^1 : (x_1,x_2) \mapsto (c_{11}x_1^2 + c_{12}x_2^3, c_{21}x_1^3 + c_{22}x_2^2)$, $T^2 : (x_1,x_2) \mapsto (d_{11}x_1^3 + d_{12}x_2^2, d_{21}x_1^2 + d_{22}x_2^2)$. *We are interested in showing that the boundedness property* $\mathcal{P}_{\|\cdot\|_2^2,\alpha}$ *holds for some positive* α.

Figure 5.17 A hierarchy of sublevel sets P_m for Example 18



(a) $m = 3$ **(b)** $m = 4$



(c) $m = 5$

Here we illustrate the method by instantiating the program of Example 5 with the following input: $a_1 = 0.9$, $a_2 = 1.1$, $b_1 = 0$, $b_2 = 0.2$, $c_{11} = c_{12} = c_{21} = c_{22} = 1$, $d_{11} = 0.5$, $d_{12} = 0.4$, $d_{21} = -0.6$ and $d_{22} = 0.3$. We represent the possible initial values taken by the program variables (x_1,x_2) by picking uniformly N points $(x_1^{(i)},x_2^{(i)})$ $(i = 1,\ldots,N)$ inside the box $X^{Init} = [0.9,1.1]\times[0,0.2]$ (see the corresponding square of dots on Figure 5.17). The other dots are obtained after successive updates of each point $(x_1^{(i)},x_2^{(i)})$ by the program of Example 5. The sets of dots in Figure 5.17 are obtained with $N = 100$ and six successive iterations.

At step $m = 3$, Program (116) yields a solution $(p_3,w_3) \in \mathbb{R}_6[x]\times\mathbb{R}$ together with SOS certificates, which guarantee the boundedness property, that is $x \in \mathfrak{C} \implies x \in P_3 := \{p_3(x) \leqslant 0\} \subseteq \mathcal{P}_{\|\cdot\|_2^2,w_3} \implies \|x\|_2^2 \leqslant w_3$. One has $p_3(x) := -2.510902467 - 0.0050x_1 - 0.0148x_2 + 3.0998x_1^2 - 0.8037x_2^3 - 3.0297x_1^3 + 2.5924x_2^2 + 1.5266x_1x_2 - 1.9133x_1^2x_2 - 1.8122x_1x_2^2 + 1.6042x_1^4 + 0.0512x_1^3x_2 - 4.4430x_1^2x_2^2 - 1.8926x_1x_2^3 + 0.5464x_2^4 - 0.2084x_1^5 + 0.5866x_1^4x_2 + 2.2410x_1^3x_2^2 + 1.5714x_1^2x_2^3 - 0.0890x_1x_2^4 - 0.9656x_2^5 + 0.0098x_1^6 - 0.0320x_1^5x_2 - 0.0232x_1^4x_2^2 + 0.2660x_1^3x_2^3 + 0.7746x_1^2x_2^4 + 0.9200x_1x_2^5 + 0.6411x_2^6$ (for the sake of conciseness, we do not display p_4 and p_5).

Figure 5.17 displays in light gray outer approximations of the set of possible values X_1 taken by the program of Example 18 as follows: (a) the degree six sublevel set P_3, (b) the degree eight sublevel set P_4 and (c) the degree ten sublevel set P_5. The outer approximation P_3 is coarse as it contains the box $[-1.5,1.5]^2$. However, solving Problem (116) at higher steps yields tighter outer approximations of $\mathfrak{C}$ together with more precise bounds w_4 and w_5 (see the corresponding row in Table 5.4).

We also succeeded to certify that the same property holds for higher dimensional programs, described in Example 19 ($d = 3$) and Example 20 ($d = 4$).

Example 19 *Here we consider* $X^{Init} = [0.9,1.1]\times[0,0.2]^2$, $r^0 : x \mapsto -1$, $r^1 : x \mapsto \|x\|_2^2 - 1$, $r^2 = -r^1$, $T^1 : (x_1,x_2,x_3) \mapsto 1/4(0.8x_1^2 + 1.4x_2 - 0.5x_3^2, 1.3x_1 + 0.5x_3^2, 1.4x_2 + 0.8x_3^2)$, $T^2 : (x_1,x_2,x_3) \mapsto 1/4(0.5x_1 + 0.4x_2^2, -0.6x_2^2 + 0.3x_3^2, 0.5x_3 + 0.4x_1^2)$ *and* $\kappa : x \mapsto \|x\|_2^2$.

Example 20 *Here we consider* $X^{Init} = [0.9,1.1]\times[0,0.2]^3$, $r^0 : x \mapsto -1$, $r^1 : x \mapsto \|x\|_2^2 - 1$, $r^2 = -r^1$, $T^1 : (x_1,x_2,x_3,x_4) \mapsto 0.25(0.8x_1^2 + 1.4x_2 - 0.5x_3^2, 1.3x_1 + 0.5, x_2^2 - 0.8x_4^2, 0.8x_3^2 + 1.4x_4, 1.3x_3 + 0.5x_4^2)$, $T^2 : (x_1,x_2,x_3,x_4) \mapsto 0.25(0.5x_1 + 0.4x_2^2, -0.6x_1^2 + 0.3x_2^2, 0.5x_3 + 0.4x_4^2, -0.6x_3 + 0.3x_4^2)$ *and* $\kappa : x \mapsto \|x\|_2^2$.

Tables 5.1,5.2,5.3 report several data obtained while solving Problem (116) at step m, $(2 \leqslant m \leqslant 5)$, either for Example 18, Example 19 or Example 20. Each instance of Problem (116) is recast as an SDP program, involving a total number of "Nb. vars" SDP variables, with an SDP matrix of size "Mat. size". We indicate the CPU time required to compute the optimal solution of each SDP program with Mosek.

The symbol "–" means that the corresponding SOS program could not be solved within one day of computation. These benchmarks illustrate the computational considerations mentioned in Section 4.2.1 as it takes more CPU time to analyze higher dimensional programs. Note that it is not possible to solve Problem (116) at step 5 for Example 20. A possible workaround to limit this computational blow-up would be to exploit the sparsity of the system.

Table 5.1: Comparison of timing results for Example 18

deg 2m	Ex. 18		
	# vars	SDP size	time
4	1513	368	0.82 s
6	5740	802	1.35 s
8	15705	1404	4 s
10	35212	2174	9.86 s

Table 5.2: Comparison of timing results for Example 19

deg 2m	Ex. 19		
	# vars	SDP size	time
4	2115	628	0.84 s
6	11950	1860	2.98 s
8	46461	4132	21.4 s
10	141612	7764	109 s

Table 5.3: Comparison of timing results for Example 20

deg 2m	Ex. 20		
	# vars	SDP size	time
4	7202	1670	2.85 s
6	65306	6622	57.3 s
8	18480	373057	1534 s
10	–	–	–

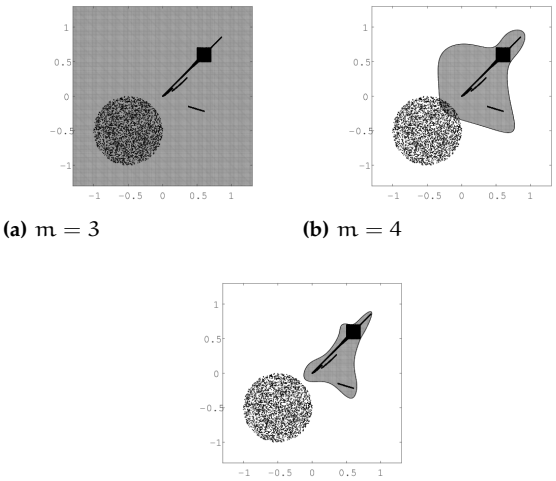
Other properties

Here we consider the program given in Example 21. One is interested in showing that the set X_1 of possible values taken by the variables of this program does not meet the ball B of center $(-0.5,-0.5)$ and radius 0.5.

Example 21 *Let consider the piecewise polynomial system* $\mathcal{S} = (X^{Init},\{X^1,X^2\},\{T^1,T^2\})$ *with* $X^{Init} = [0.5,0.7]\times[0.5,0.7]$, $X^1 = \{x\in\mathbb{R}^2 \mid r^1(x)\leqslant 0\}$ *with* $r^1 : x\mapsto \|x\|_2^2 - 1$, $X^2 = \{x\in\mathbb{R}^2 \mid r^2(x)\leqslant 0\}$ *with* $r^2 = -r^1$ *and* $T^1 : (x_1,x_2)\mapsto (x_1^2+x_2^3, x_1^3+x_2^2)$, $T^2 : (x,y)\mapsto (0.5x_1^3+0.4x_2^2, -0.6x_1^2+0.3x_2^2)$. *With* $\kappa : (x_1,x_2)\mapsto 0.25-(x_1+0.5)^2-(x_2+0.5)^2$, *one has* $B := \{x\in\mathbb{R}^2 \mid 0\leqslant\kappa(x)\}$. *Here, one shall prove* $x\in\mathfrak{C} \implies \kappa(x) < 0$ *while computing some negative* α *such that* $\mathfrak{C}\subseteq\mathcal{P}_{\kappa,\alpha}$. *Note that* κ *is not a norm, by contrast with the previous examples.*

At step $m = 3$ (resp.$m = 4$), Program (116) yields a non-negative solution w_3 (resp. w_4). Hence, it does not allow to certify that $\mathfrak{C}\cap B$ is empty. This is illustrated in both Figure 5.18 (a) and Figure 5.18 (b), where the light gray region does not avoid the ball B. However, solving Program (116) at step $m = 5$ yields a negative bound w_5 together with a certificate that $\mathfrak{C}$ avoids the ball B (see Figure 5.18 (c)). The corresponding values of w_m ($m = 3,4,5$) are given in Table 5.4.

Figure 5.18 A hierarchy of sublevel sets P_m for Example 21



(a) $m = 3$ **(b)** $m = 4$

(c) $m = 5$

Finally, one analyzes the program given in Example 22.

Example 22 *(adapted from Example 3 in [AJ13])*
Let $\mathcal{S}$ *be the piecewise polynomial system* $(X^{Init},\{X^1,X^2\},\{T^1,T^2\})$ *with* $X^{Init} = [-1,1]\times[-1,1]$, $X^1 = \{x\in\mathbb{R}^2 \mid r^1(x)\leqslant 0\}$ *with* $r^1 : x\mapsto x_2 - x_1$, $X^2 = \{x\in\mathbb{R}^2 \mid r^2(x)\leqslant 0\}$ *with* $r^2 = -r^1$ *and* $T^1 : (x_1,x_2)\mapsto (0.687x_1 + 0.558x_2 - 0.0001 * x_1x_2, -0.292x_1 + 0.773x_2)$, $T^2 : (x,y)\mapsto (0.369x_1 + 0.532x_2 - 0.0001x_1^2, -1.27x_1 + 0.12x_2 - 0.0001x_1x_2)$. *We consider the boundedness property* $\kappa_1 := \|\cdot\|_2^2$ *as well as* $\kappa_2(x) := \|T^1(x) - T^2(x)\|_2^2$. *The function* κ_2 *can be viewed as the absolute error made by updating the variable* x *after a possibly "wrong" branching. Such behaviors could occur while computing wrong values for the conditionals (e.g.* r^1*) using floating-point arithmetics. Table 5.4 indicates the hierarchy of bounds obtained after solving Problem (116) with* $m = 3,4,5$, *for both properties. The*

bound $w_5 = 2.84$ *(for* κ_1*) implies that the set of reachable values may not be included in the initial set* X^{Init}*. A valid upper bound of the error function* κ_2 *is given by* $w_5 = 2.78$*.*

	Ex. 18	Ex. 21	Ex. 22	
κ	$\|\cdot\|_2^2$	κ	$\|\cdot\|_2^2$	κ_2
w_2	639	0.25	10.2	5.66
w_3	17.4	0.249	2.84	2.81
w_4	2.44	0.0993	2.84	2.78
w_5	2.02	-0.0777	2.84	2.78

with Ex.21 $\kappa = x \mapsto 0.25 - \|x + 0.5\|_2^2$ and Ex. 22 $\kappa_2 = x \mapsto \|T^1(x) - T^2(x)\|_2^2$.

Table 5.4: Hierarchies of bounds obtained for various properties

5.6 RELATED WORKS

Automated non linear analyses are not very common in formal verification. A line of works[MS02; SSM04] rely on iterative computations using GRÖBNER bases to synthesize polynomial equality invariants. Similarly to KARR's domain representing affine relationships among variables [Kar76], these domains extract polynomial relationship between variables. More recent work [Cac+14] rely on a kind of weakest precondition computation to synthesize these polynomial equalities. All these works cannot, in the current state, express semi-algebraic sets nor capture the stability of a linear controller.

Regarding quadratic invariants, ie. ellipsoids, they are not fitted with a lattice structure since there is no unique smallest ellipsoids containing a set of ellipsoids. Unrolling techniques such as [Fer05b; Fer04; Mon07; SB13] enable the precise analysis of linear systems by solving mathematically these dynamical systems. While more precise than the approach we proposed, these techniques can hardly handle disjunction or saturations. Their use could however be used locally to improve the precision of the analyses. The use of convex optimization such as SDP or SOS to synthesize sublevel set properties was proposed by COUSOT in [Cou05] providing simple inductive templates and without addressing methods to check the soundness of the result. Other recent approaches such as [OV15] proposed a classical Kleene iterations-based abstract domain for ellipsoids in which the join operator is implemented as the call to an SDP solvers synthesizing the minimal volume ellipsoid. The interesting approach of [All+15] proposed an algebraic method to manipulate a specific class of ellipsoids (zero-centered ellipsoids), without the need to call a numerical tool such as an SDP solver to compute such minimal volume ellipsoids.

A last related category of analyses is the computation of non convex properties. Non convex properties were also used to express disjunctions as holes in a given more classical convex abstraction: difference-bound matrices (DBM), the underlying domain of octagons, with disequality constraints [PH07], or the Donut domain [Gho+12].

6

TEMPLATE BASED ANALYSES AND MIN-POLICY ITERATION

While the previous chapter addressed the direct synthesis of invariants as bound templates, there are other configurations in which with are interested in bounding provided templates.

A first case arises when the previous method, as in Equations (70), and (71), only synthesize the template but not the bound. A second appear when one want to analyze a system with multiple templates. Typically, we are interested by bounds on each variable and want to consider the templates $p(x) = x_i^2$ for each variable x_i in state characterization $x \in \Sigma$. The current chapter proposes a policy iteration algorithm, based on SOS optimization, to refine such template bounds. In practice, we use it by combining a LYAPUNOV based template obtained using one of the previous method with additional template encoding bounds on some variables or property specific templates.

6.1 TEMPLATE BASED ABSTRACT DOMAINS

Let us now assume that the abstraction is based on a template abstraction. We recall that a template is a real-valued function $p : \Sigma \to \mathbb{R}$. For the rest of the chapter, we assume that these templates are given.

For each template p, one can characterize an abstract domain $D_t^\#$ as presented in Sec. 2.5. We also denote by $\bar{\mathbb{R}} = \mathbb{R} \cup \{-\infty, +\infty\}$ the extension of $\mathbb{R}$ with infinite values and by $\dot{\leqslant}$ the extension of $\leqslant$ to those values.

As for the characterization of the fixpoint presented earlier, this abstraction also defines a complete lattice. The order relation $\dot{\leqslant}$ is total and relies on the real number order applied to the level sets. The join and meet of two abstract values, ie. the two scalars representing sublevel sets, are computed with max and min.

$$D_p^\# = \langle \bar{\mathbb{R}}, \dot{\leqslant}, \max, \min, -\infty, +\infty \rangle$$

The abstraction and concretization functions are defined as:

$$\alpha_p : S \mapsto \max\{p(s) | s \in S\}$$
$$\gamma_p : \lambda \mapsto \{s \in S | p(s) \leqslant \lambda\}$$

Multiple templates could be considered at once. Let $\mathbb{P}$ be a finite family of templates $(p_i)_{0 \leqslant i < n}$. And $\mathcal{F}\left(\mathbb{P}, \overline{\mathbb{R}}\right)$ be the set of functions from $\mathbb{P}$ to $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, +\infty\}$. We fit $\mathcal{F}\left(\mathbb{P}, \overline{\mathbb{R}}\right)$ with the functional partial order $\leqslant_{\mathbb{F}}$ i.e. $v \leqslant_{\mathbb{F}} w$ iff $v(p) \leqslant w(p)$ for all $p \in \mathbb{P}$. This defines our abstract domain, the lattice

$$D_{\mathbb{P}}^\# = \langle \mathcal{F}\left(\mathbb{P}, \overline{\mathbb{R}}\right), \leqslant_{\mathbb{F}}, \max_{\mathbb{F}}, \min_{\mathbb{F}}, (-\infty)_{\mathbb{F}}, (+\infty)_{\mathbb{F}} \rangle$$

where the functions $\max_{\mathbb{F}}, \min_{\mathbb{F}}$ are lift of max and min to functions. $(\pm\infty)_{\mathbb{F}}$ denote the functions $p \in \mathbb{P} \mapsto \pm\infty$.

We characterize the abstraction $\star$ and concretization $\dagger$ functions. Let $w \in \mathcal{F}\left(\mathbb{P}, \overline{\mathbb{R}}\right)$ and $X \in \mathbb{R}^n$. The concretization of w to sets gives the set $w^\star$:

$$w^\star = \{x \in \mathbb{R}^d \mid p(x) \leqslant w(p), \forall\, p \in \mathbb{P}\}. \tag{117}$$

While the abstraction of X to $\mathcal{F}\left(\mathbb{P}, \overline{\mathbb{R}}\right)$ is defined by the abstract element $X^\dagger$:

$$X^\dagger(p) := \sup_{x \in X} p(x) \tag{118}$$

6.2 TEMPLATE ABSTRACTION FIXPOINT AS AN OPTIMIZATION PROBLEM

Let us summarize the current definitions:

- The collecting semantics of a system is defined using Equation (17) as the least fixpoint of an endomorphism over set of states; and is characterized by the minimum set $S \in \wp(\Sigma)$ of the postfixpoints $F(S) \subseteq S$.
- A possible set of abstractions is defined by the templates abstract domains. An abstract domain is specified by a finite family of templates, real valued function over system states. An abstract value is a vector of real values characterizing of sublevel sets of templates.

Then computing an inductive invariant in the templates domain boils down to providing, for each template p, a bound $w(p)$ such that the intersection over the templates p of sublevel sets $\{x \in \mathbb{R}^d \mid p(x) \leqslant w(p)\}$ is an inductive invariant. We recall, that, in our context, a template is simply an *a-priori* fixed multivariate polynomial.

We need to express the inductiveness of the sets $w^\star$ into inequalities on w. By definition the set $w^\star$ is an inductive invariant iff $F(w^\star) \subseteq w^\star$, that is:

$$\bigcup_{i\in\mathcal{I}} T^i(w^\star \cap X^i) \cup X^{Init} \subseteq w^\star \ .$$

By definition, $w^\star$ is an inductive invariant iff:

$$\forall p \in \mathbb{P},\ \forall x \in \bigcup_{i\in\mathcal{I}} T^i(w^\star \cap X^i) \cup X^{Init},\ p(x) \leqslant w(p) \ .$$

Using the definition of the supremum, $w^\star$ is an inductive invariant iff:

$$\forall p \in \mathbb{P},\quad \sup_{x \in \bigcup_{i\in\mathcal{I}} T^i(w^\star \cap X^i) \cup X^{Init}} p(x) \leqslant w(p) \ .$$

Now, let consider $p \in \mathbb{P}$. Using the fact that for all $A, B \subseteq \mathbb{R}^d$ and for all functions f, $\sup_{A\cup B} f = \sup\{\sup_A f, \sup_B f\}$:

$$\sup_{x\in\bigcup_{i\in\mathcal{I}} T^i(w^\star\cap X^i)\cup X^{Init}} p(x) =$$

$$\sup\left\{\sup_{i\in\mathcal{I}} \sup_{x\in T^i(w^\star\cap X^i)} p(x),\ \sup_{x\in X^{Init}} p(x)\right\} \ .$$

By definition of the image:

$$\sup_{x\in\bigcup_{i\in\mathcal{I}} T^i(w^\star\cap X^i)\cup X^{Init}} p(x) =$$

$$\sup\left\{\sup_{i\in\mathcal{I}} \sup_{y\in w^\star\cap X^i} p(T^i(y)),\ \sup_{x\in X^{Init}} p(x)\right\} \ .$$

Let us introduce the following notation to denote the image of a set $w^\star$ by an guarded update function T^i, for all $p \in \mathbb{P}$:

$$F^\sharp_i(w)(p) := \sup_{x\in w^\star\cap X^i} p(T^i(x))$$

We also recall the definition of abstraction applied on initial state:

$$X^{Init^\dagger}(p) := \sup_{x\in X^{Init}} p(x) \ .$$

Finally, we define the function from $\mathcal{F}(\mathbb{P}, \overline{\mathbb{R}})$ to itself, for all $w \in \mathcal{F}(\mathbb{P}, \overline{\mathbb{R}})$:

$$F^\sharp(w) := \sup\left\{\sup_{i\in\mathcal{I}} F^\sharp_i(w), X^{Init^\dagger}\right\}$$

By construction, we obtain the following proposition:

Proposition 5 *Let $w \in \mathcal{F}(\mathbb{P}, \overline{\mathbb{R}})$. Then $w^\star$ is an inductive invariant (i.e. $F(w^\star) \subseteq w^\star$) iff $F^\sharp(w) \leqslant_{\mathbb{F}} w$.*

From Prop. 5, $\inf\{w \in \mathcal{F}(\mathbb{P}, \overline{\mathbb{R}})^n \mid F^\sharp(w) \leqslant_{\mathbb{F}} w\}$ identifies the smallest inductive invariant $w^\star$ of the form (117).

Example 23 *Let us consider the system defined at Example 22. Let us consider the same templates basis $\mathbb{P} = \{q_1, q_2, p\}$ where $q_1(x) = x_1^2$, $q_2(x) = x_2^2$ and p is a well-chosen polynomial of degree 6. Let $w \in \mathcal{F}(\mathbb{P}, \overline{\mathbb{R}})$. For $i = 1$ and the templates q_1, we have:*

$$F^\sharp_1(w)(q_1) = \sup_{\substack{-x_1^2+1\leqslant 0\\ x_1^2\leqslant w(q_1),\\ x_2^2\leqslant w(q_2),\\ p(x)\leqslant w(p)}} (0.687x_1 + 0.558x_2 - 0.0001x_1x_2)^2$$

Indeed, $X^1 = \{x \in \mathbb{R}^2 \mid -x_1^2 + 1\}$ and the dynamics associated with X^1 is the polynomial function T^1 defined for all $x \in \mathbb{R}^2$ by: $T^1(x) = \begin{pmatrix} 0.687x_1+0.558x_2-0.0001x_1x_2 \\ -0.292x_1+0.773x_2 \end{pmatrix}$ and thus since q_1 computes the square of the first coordinates $q_1(T^1(x)) = (0.687x_1 + 0.558x_2 - 0.0001x_1x_2)^2$.

With $w \in \mathcal{F}(\mathbb{P}, \overline{\mathbb{R}})$, computing $F^\sharp(w)$ boils down to solving a finite number of nonconvex polynomial optimization problems. General methods do not exist to solve such problems. In Section 6.3, we propose a method based on Sums-of-Squares (SOS) to over-approximate $F^\sharp(w)$.

6.3 SOS-RELAXED SEMANTICS

In this section, we introduce the relaxed functional on which we will compute a fixpoint, yielding a further over-approximation of the set $\mathcal{R}$ of reachable values. This relaxed functional is constructed from a LAGRANGE relaxation of maximization problems involved in the evaluation of $F^\sharp$ and Sums-of-Squares strengthening of polynomial nonnegativity constraints.

6.3.1 *Relaxed semantics*

The computation of $F^\sharp$ as a polynomial maximization problem cannot be directly performed using numerical solvers. We use the SOS reinforcement mechanisms described above to relax the computation and characterize an abstraction of $F^\sharp$.

We still assume the knowledge of the template basis $\mathbb{P}$, involving polynomials of degree at most $2m$. Let us define $\mathcal{F}(\mathbb{P}, \mathbb{R}_+)$ the set of nonnegative functions over $\mathbb{P}$ i.e. $g \in \mathcal{F}(\mathbb{P}, \mathbb{R}_+)$ iff for all $p \in \mathbb{P}$, $g(p) \in \mathbb{R}_+$. Let $p \in \mathbb{P}$

and $w \in \mathcal{F}(\mathbb{P}, \overline{\mathbb{R}})$. Starting from the definition of $F_i^\sharp$, one obtains the following:

$$
\begin{aligned}
&\left(F_i^\sharp(w)\right)(p)\\
&= \sup_{\substack{q(x)\leqslant w(q),\ \forall q\in\mathbb{P}\\ r_j^i(x)\leqslant 0,\ \forall j\in \mathbf{In}_i}} p(T^i(x))\\
&\leqslant \inf_{\substack{\lambda\in\mathcal{F}(\mathbb{P},\mathbb{R}_+)\\ \sigma\in\Sigma[x],\mu_l\in\Sigma[x]\\ \deg(\sigma)\leqslant 2m\deg T^i\\ \deg(\mu_l r_l^i)\leqslant 2m\deg T^i}} \sup_{x\in\mathbb{R}^d} p(T^i(x)) + \sum_{q\in\mathbb{P}}\lambda(q)(w(q)-q(x)) - \sum_{l=1}^{n^i}\mu_l(x)r_l^i(x)\\
&\leqslant \inf_{\lambda,\sigma,\mu_l,\eta} \eta\\
&\quad \text{s.t.} \begin{cases} \eta - p\circ T^i - \sum_{q\in\mathbb{P}}\lambda(q)(w(q)-q) + \sum_{l=1}^{n^i}\mu_l r_l^i = \sigma,\\ \lambda\in\mathcal{F}(\mathbb{P},\mathbb{R}_+),\ \sigma\in\Sigma[x],\\ \mu_l\in\Sigma[x],\ \eta\in\mathbb{R},\\ \deg(\sigma)\leqslant 2m\deg T^i,\\ \deg(\mu_l r_l^i)\leqslant 2m\deg T^i \end{cases}\\
&\quad \text{(using an SOS reinforcement to remove the sup)}
\end{aligned}
$$

We denote by $\Sigma[x]^n$ the set of n-tuples of SOS polynomials. For clarity purpose, the dependency on i is omitted within the notations of the multipliers μ_l. Moreover, let us write $\sum_{l=1}^{n^i}\mu_l r_l^i$ as $\langle \mu, r^i\rangle$. Finally, we write $\left(F_i^{\mathcal{R}}(w)\right)(p)$ the over-approximation of $\left(F_i^\sharp(w)\right)(p)$, defined as follows:

$$
\begin{aligned}
&\left(F_i^{\mathcal{R}}(w)\right)(p) = \inf_{\lambda,\sigma,\mu,\eta} \eta\\
&\quad \text{s.t.} \begin{cases} \eta - p\circ T^i - \sum_{q\in\mathbb{P}}\lambda(q)(w(q)-q) + \langle \mu, r^i\rangle = \sigma\\ \lambda\in\mathcal{F}(\mathbb{P},\mathbb{R}_+),\ \sigma\in\Sigma[x],\ \mu\in\Sigma[x]^{n_i},\\ \eta\in\mathbb{R},\\ \deg(\sigma)\leqslant 2m\deg T^i,\\ \deg(\langle\mu, r^i\rangle)\leqslant 2m\deg T^i. \end{cases}
\end{aligned}
\tag{119}
$$

We conclude that, for all $i\in\mathcal{I}$, the evaluation of $F_i^{\mathcal{R}}$ can be done using SOS programming, since it is reduced to solve a minimization problem with a linear objective function and linear combination of polynomials constrained to be sum-of-squares.

Example 24 *We still consider the running example defined at Example 22 and take the following templates basis:* $q_1 : x \mapsto x_1^2$, $q_2 : x \mapsto x_2^2$, *and a well-chosen polynomial* p *of degree 6. For the index of the partition* $i = 1$. *Recall that* $T^1(x) = \begin{pmatrix} 0.687x_1+0.558x_2-0.0001x_1x_2 \\ -0.292x_1+0.773x_2 \end{pmatrix}$ *and* $X^1 = \{x\in\mathbb{R}^2 \mid -x_1^2+1\leqslant 0\}$ *and thus* $r_1^1(x) = -x_1^2+1$. *Let* $w\in\mathcal{F}(\mathbb{P},\overline{\mathbb{R}})$, *then:*

$$
\begin{aligned}
&\left(F_1^{\mathcal{R}}(w)\right)(q_1) =\\
&\inf_{\lambda,\sigma,\mu,\eta} \eta\\
&\text{s.t.} \begin{cases} \eta - (0.687x_1+0.558x_2-0.0001x_1x_2)^2\\ -\lambda(q_1)(w(q_1)-x_1^2) - \lambda(q_2)(w(q_2)-x_2^2)\\ -\lambda(p)(w(p)-p(x)) + \mu(x)(1-x_1^2) = \sigma(x)\\ \lambda\in\mathcal{F}(\mathbb{P},\mathbb{R}_+),\ \sigma\in\Sigma[x],\ \mu\in\Sigma[x],\ \eta\in\mathbb{R},\\ \deg(\sigma)\leqslant 6,\ \deg(\mu)\leqslant 6. \end{cases}
\end{aligned}
$$

In practice, one cannot find any feasible solution of degree less than 6, thus we replace the degree constraint by the more restrictive one: $\deg(\sigma)\leqslant 6$, $\deg(\mu)\leqslant 6$.

The computation of $F^\sharp$ requires the approximation of $X^{Init\dagger} := \sup\{p(x), x\in X^{Init}\}$. Since X^{Init} is a basic semi-algebraic set and each template p is a polynomial, then the evaluation of $X^{Init\dagger}$ boils down to solving a polynomial maximization problem. Next, we use SOS reinforcement described above to over-approximate $X^{Init\dagger}$ with the set $X^{Init\mathcal{R}}$, defined as follows:

$$
X^{Init\mathcal{R}}(p) := \inf\left\{ \eta \;\middle|\; \begin{array}{c} \eta - p + \langle \nu^{n_{in}}, r^{n_{in}}\rangle = \sigma_0,\\ \eta\in\mathbb{R},\ \sigma_0\in\Sigma[x], \nu^{in}\in\Sigma[x]^{n_{in}},\\ \deg(\sigma_0)\leqslant 2m, \deg(\langle\nu^{n_{in}}, r^{n_{in}}\rangle)\leqslant 2m \end{array} \right\}.
$$

Thus, the value of $X^{Init\mathcal{R}}(p)$ is obtained by solving an SOS optimization problem. Since X^{Init} is a nonempty compact basic semi-algebraic set, this problem has a feasible solution (see the proof of [Laso1, Th. 4.2]), ensuring that $X^{Init\mathcal{R}}(p)$ is finite valued.

Example 25 *The initialization set* X^{Init} *of Example 22 is* $[-1,1]\times[-1,1]$. *It can be written as:* $\{(x_1,x_2)\in\mathbb{R}^2 \mid x_1^2-1\leqslant 0,\ x_2^2-1\leqslant 0\}$. *Then, considering the same template basis of Example 24 and the template* q_1*:*

$$
X^{Init\mathcal{R}}(q_1) := \inf\left\{ \eta \;\middle|\; \begin{array}{c} \eta - x_1^2 + \nu_1^{n_{in}}(x)(x_1^2-1)\\ +\nu_2^{n_{in}}(x)(x_2^2-1) = \sigma_0(x),\\ \eta\in\mathbb{R},\ \sigma_0\in\Sigma[x], \nu_1^{in},\nu_2^{in}\in\Sigma[x],\\ \deg(\sigma_0)\leqslant 6, \deg(\langle\nu_1^{n_{in}})\leqslant 6,\\ \deg(\langle\nu_2^{n_{in}})\leqslant 6 \end{array} \right\}.
$$

It is easy to see that taking for all $x\in\mathbb{R}^2$, $\nu_1^{n_{in}}(x) = 1$ *and for all* $x\in\mathbb{R}^2$, $\nu_2^{n_{in}}(x) = 0$ *leads to* $\eta - x_1^2 + \nu_1^{n_{in}}(x)(x_1^2-1) + \nu_2^{n_{in}}(x)(x_2^2-1) = \eta - 1 = \sigma_0(x)$. *Thus for* $\eta = 1$ *and for all* $x\in\mathbb{R}^2$, $\sigma_0(x) = 0$, *we obtain* $X^{Init\mathcal{R}}(q_1) = 1$.

Finally, we define the relaxed functional $F^{\mathcal{R}}$ for all $w \in \mathcal{F}(\mathbb{P}, \overline{\mathbb{R}})$ and for all $p \in \mathbb{P}$ as follows:

$$\left(F^{\mathcal{R}}(w)\right)(p) = \sup\left\{\sup_{i\in\mathcal{I}}\left(F_i^{\mathcal{R}}(w)\right)(p), X^{Init^{\mathcal{R}}}(p)\right\} . \quad (120)$$

By construction, the relaxed functional $F^{\mathcal{R}}$ provides a safe over-approximation of the abstract semantics $F^{\sharp}$.

Proposition 6 (Safety) *The following statements hold:*

1. $X^{Init^{\dagger}} \leqslant_{\mathbb{F}} X^{Init^{\mathcal{R}}}$;
2. *For all* $i \in \mathcal{I}$, *for all* $w \in \mathcal{F}(\mathbb{P}, \overline{\mathbb{R}})$, $F_i^{\sharp}(w) \leqslant_{\mathbb{F}} F_i^{\mathcal{R}}(w)$;
3. *For all* $w \in \mathcal{F}(\mathbb{P}, \overline{\mathbb{R}})$, $F^{\sharp}(w) \leqslant_{\mathbb{F}} F^{\mathcal{R}}(w)$.

An important property that we will use to prove some results on policy iteration algorithm is the monotonicity of the relaxed functional.

Proposition 7 (Monotonicity)

1. *For all* $i \in \mathcal{I}$, $w \mapsto F_i^{\mathcal{R}}(w)$ *is monotone on* $\mathcal{F}(\mathbb{P}, \overline{\mathbb{R}})$;
2. *The function* $w \mapsto F^{\mathcal{R}}(w)$ *is monotone on* $\mathcal{F}(\mathbb{P}, \overline{\mathbb{R}})$.

From the third assertion of Prop. 6, if w satisfies $F^{\mathcal{R}}(w) \leqslant_{\mathbb{F}} w$ then $F^{\sharp}(w) \leqslant_{\mathbb{F}} w$ and from Prop. 5, $w^{\star}$ is an inductive invariant and thus $\mathcal{R} \subseteq w^{\star}$. This result is formulated as the following corollary.

Corollary 1 (Over-approximation) *For all* $w \in \mathcal{F}(\mathbb{P}, \overline{\mathbb{R}})$ *such that* $F^{\mathcal{R}}(w) \leqslant_{\mathbb{F}} w$ *then* $\mathcal{R} \subseteq w^{\star}$.

6.3.2 *Policy Iteration in Polynomial Templates Abstract Domains*

We are interested in computing the least fixpoint $\mathcal{R}^{\mathcal{R}}$ of $F^{\mathcal{R}}$, $\mathcal{R}^{\mathcal{R}}$ being an over-approximation of $\mathcal{R}$ (least fixpoint of F). As for the definition of $\mathcal{R}$, it can be reformulated using Tarski's theorem as the minimal post-fixpoint:

$$\mathcal{R}^{\mathcal{R}} = \min\{w \in \mathcal{F}(\mathbb{P}, \overline{\mathbb{R}}) \,|\, F^{\mathcal{R}}(w) \leqslant_{\mathbb{F}} w\}.$$

The idea behind policy iteration is to over-approximate $\mathcal{R}^{\mathcal{R}}$ using successive iterations which are composed of

- the computation of polynomial template bounds using linear programming,
- the determination of new policies using SOS programming,

until a fixpoint is reached. Policy iteration navigates in the set of post-fixpoints of $F^{\mathcal{R}}$ and needs to start from a post-fixpoint w^0 know *a-priori*. It acts like a narrowing operator and can be interrupted at any time. For further information on policy iteration, the interested reader can consult [Cos+05; Gau+07].

6.3.3 *Policies*

Policy iteration can be used to compute a fixpoint of a monotone self-map defined as an infimum of a family of affine monotone self-maps. We propose to design a policy iteration algorithm to compute a fixpoint of $F^{\mathcal{R}}$. In this subsection, we give the formal definition of policies in the context of polynomial templates and define the family of affine monotone self-maps. We do not apply the concept of policies on $F^{\mathcal{R}}$ but on the functions $F_i^{\mathcal{R}}$ exploiting the fact that for all $i \in \mathcal{I}$, $F_i^{\mathcal{R}}$ is the optimal value of a minimization problem.

Policy iteration needs a *selection property*, that is, when an element $w \in \mathcal{F}(\mathbb{P}, \overline{\mathbb{R}})$ is given, there exists a policy which achieves the infimum. In our context, since we apply the concept of policies to $F_i^{\mathcal{R}}$, it means that the minimization problem involved in the computation of $F_i^{\mathcal{R}}$ has an optimal solution. In our case, for $w \in \mathcal{F}(\mathbb{P}, \overline{\mathbb{R}})$ and $p \in \mathbb{P}$, an optimal solution is a vector $(\lambda, \sigma, \mu) \in \mathcal{F}(\mathbb{P}, \mathbb{R}_+) \times \Sigma[x] \times \Sigma[x]^{n_i}$ such that, using (119), we obtain:

$$\begin{array}{l}\left(F_i^{\mathcal{R}}(w)\right)(p) = \\ p \circ T^i + \sum_{q\in\mathbb{P}} \lambda(q)(w(q) - q) - \langle \mu, r^i\rangle + \sigma \\ \text{and } \deg(\sigma) \leqslant 2m \deg T^i, \\ \deg(\langle \mu, r^i\rangle) \leqslant 2m \deg T^i\end{array} . \quad (121)$$

Observe that in Eq. (121), $\left(F_i^{\mathcal{R}}(w)\right)(p)$ is a scalar whereas the right-hand-side is a polynomial. The equality in this equation means that this polynomial is a constant polynomial. Then we introduce the set of feasible solutions for the SOS problem $\left(F_i^{\mathcal{R}}(w)\right)(p)$:

$$\mathrm{Sol}(w, i, p) = \left\{ \begin{array}{l} (\lambda, \sigma, \mu) \in \mathcal{F}(\mathbb{P}, \mathbb{R}_+) \times \Sigma[x] \times \Sigma[x]^{n_i} \\ \text{s.t. Eq. (121) holds} \end{array} \right\} \quad (122)$$

Since policy iteration algorithm can be stopped at any step and still provides a sound over-approximation, we stop the iteration when $\mathrm{Sol}(w, i, p) = \emptyset$. Now, we are interested in the elements $w \in \mathcal{F}(\mathbb{P}, \mathbb{R})$ such that $\mathrm{Sol}(w, i, p)$ is non-empty:

$$\mathcal{FS}(\mathbb{P}, \overline{\mathbb{R}}) = \left\{ w \in \mathcal{F}(\mathbb{P}, \overline{\mathbb{R}}) \;\middle|\; \begin{array}{l} \forall i \in \mathcal{I},\ \forall p \in \mathbb{P}, \\ \mathrm{Sol}(w, i, p) \neq \emptyset \end{array} \right\} . \quad (123)$$

The notation $\mathcal{FS}(\mathbb{P}, \overline{\mathbb{R}})$ was introduced in [AGG10] to define the elements $w \in \mathcal{F}(\mathbb{P}, \overline{\mathbb{R}})$ satisfying $\mathrm{Sol}(w, i, p) \neq \emptyset$. In [AGG10, Section 4.3], we could ensure that $\mathrm{Sol}(w, i, p) \neq \emptyset$ using Slater's constraint qualification condition. In the current nonlinear setting, we cannot

use the same condition, which yields a more complicated definition for $\mathcal{FS}\left(\mathbb{P},\overline{\mathbb{R}}\right)$.

Finally, we can define a policy as a map which selects, for all $w \in \mathcal{FS}\left(\mathbb{P},\overline{\mathbb{R}}\right)$, for all $i \in \mathcal{I}$ and for all $p \in \mathbb{P}$ a vector of $\mathrm{Sol}(w,i,p)$. More formally, we have the following definition:

Definition 6.1 (Policies in the SOS policy iteration) *A policy is a map* $\pi : \mathcal{FS}\left(\mathbb{P},\overline{\mathbb{R}}\right) \mapsto \left((\mathcal{I}\times\mathbb{P}) \mapsto \mathcal{F}\left(\mathbb{P},\mathbb{R}_+\right) \times \Sigma[x] \times \Sigma[x]^{n_i} \times \Sigma[x]^{n_0}\right)$ *such that:* $\forall w \in \mathcal{FS}\left(\mathbb{P},\overline{\mathbb{R}}\right)$, $\forall i \in \mathcal{I}$, $\forall p \in \mathbb{P}$, $\pi(w)(i,p) \in \mathrm{Sol}(w,i,p)$.

We denote by Π the set of policies. For $\pi \in \Pi$, let us define π_λ as the map from $\mathcal{FS}\left(\mathbb{P},\overline{\mathbb{R}}\right)$ to $(\mathcal{I}\times\mathbb{P}) \mapsto \mathcal{F}\left(\mathbb{P},\mathbb{R}_+\right)$ which associates with $w \in \mathcal{FS}\left(\mathbb{P},\overline{\mathbb{R}}\right)$ and $(i,p) \in \mathcal{I}\times\mathbb{P}$ the first tuple element of $\pi(w)(i,p)$ i.e. if $\pi(w)(i,p) = (\lambda,\sigma,\mu)$ then $\pi_\lambda(w)(i,p) = \lambda$.

As said before, policy iteration exploits the linearity of maps when a policy is fixed. We have to define the affine maps we will use in a policy iteration step. With $\pi \in \Pi$, $w \in \mathcal{FS}\left(\mathbb{P},\overline{\mathbb{R}}\right)$, $i \in \mathcal{I}$ and $p \in \mathbb{P}$ and $\lambda = \pi_\lambda(w)(i,p)$, let us define the map $\phi^\lambda_{w,i,p} : \mathcal{F}\left(\mathbb{P},\overline{\mathbb{R}}\right) \mapsto \overline{\mathbb{R}}$ as follows:

$$\begin{aligned} &v \mapsto \phi^\lambda_{w,i,p}(v) = \\ &\textstyle\sum_{q\in\mathbb{P}} \lambda(q)v(q) + \left(F^{\mathcal{R}}_i(w)\right)(p) - \sum_{q\in\mathbb{P}} \lambda(q)w(q) \end{aligned} \tag{124}$$

Then, for $\pi \in \Pi$, we define for all $w \in \mathcal{FS}\left(\mathbb{P},\overline{\mathbb{R}}\right)$, the map $\Phi^{\pi(w)}_w$ from $\mathcal{F}\left(\mathbb{P},\overline{\mathbb{R}}\right) \mapsto \mathcal{F}\left(\mathbb{P},\overline{\mathbb{R}}\right)$. Let $v \in \mathcal{F}\left(\mathbb{P},\overline{\mathbb{R}}\right)$ and $p \in \mathbb{P}$:

$$\Phi^{\pi(w)}_w(v)(p) = \sup\left\{ \sup_{i\in\mathcal{I}} \phi^\lambda_{w,i,p}(v), X^{\mathrm{Init}^{\mathcal{R}}}(p) \right\} \tag{125}$$

Example 26 *Let us consider Example 24 and the function* $w^0(q_1) = w^0(q_2) = 2.1391$ *and* $w^0(p) = 0$. *Then there exists two SOS polynomials* $\overline{\mu}$ *and* $\overline{\sigma}$ *such that, for all* $x \in \mathbb{R}^d$: $\left(F^{\mathcal{R}}_1(w)\right)(q_1) = (0.687x_1 + 0.558x_2 - 0.0001x_1x_2)^2 + \overline{\lambda}(q_1)(2.1391 - x_1^2) + \overline{\lambda}(q_2)(2.1391 - x_2^2) - \overline{\lambda}(p)p(x) - \overline{\mu}(x)(1-x_1^2) + \overline{\sigma}(x) = 1.5503$ *with* $\overline{\lambda}(q_1) = \overline{\lambda}(q_2) = 0$ *and* $\overline{\lambda}(p) = 2.0331$. *It means that* $\overline{\lambda}$, $\overline{\mu}$ *and* $\overline{\sigma}$ *are computed such that* $(0.687x_1 + 0.558x_2 - 0.0001x_1x_2)^2 + \overline{\lambda}(q_1)(2.1391 - x_1^2) + \overline{\lambda}(q_2)(2.1391 - x_2^2) - \overline{\lambda}(p)p(x) - \overline{\mu}(x)(1-x_1^2) + \overline{\sigma}(x)$ *is actually a constant polynomial.*

Then $(\overline{\lambda},\overline{\mu},\overline{\sigma}) \in \mathrm{Sol}(w^0,1,q_1)$ *and we can define a policy* $\pi(w^0)$ *such that* $\pi(w^0)(1,q_1) = (\overline{\lambda},\overline{\mu},\overline{\sigma})$ *and thus* $\pi_\lambda(w^0)(1,q_1) = (0,0,2.0331)$. *We can thus define for* $v \in \mathcal{F}\left(\mathbb{P},\mathbb{R}\right)$, *the affine mapping:* $\phi^\lambda_{w^0,1,q_1}(v) = \lambda(q_1)v(q_1) + \lambda(q_2)v(q_2) + \lambda(p)v(p) + \left(F^{\mathcal{R}}_1(w)\right)(q_1) - \lambda(q_1)w(q_1) - \lambda(q_2)w(q_2) - \lambda(p)w(p) = 2.1391v(p) + 1.5503$.

Let us denote by $\mathcal{F}\left(\mathbb{P},\mathbb{R}\right)$ the set of finite valued function on $\mathbb{P}$ i.e $g \in \mathcal{F}\left(\mathbb{P},\mathbb{R}\right)$ iff $g(p) \in \mathbb{R}$ for all $p \in \mathbb{P}$.

Proposition 8 (Properties of $\phi^\lambda_{i,w,p}$) *Let* $\pi \in \Pi$, $w \in \mathcal{FS}\left(\mathbb{P},\overline{\mathbb{R}}\right)$ *and* $(i,p) \in \mathcal{I}\times\mathbb{P}$. *Let us write* $\lambda = \pi_\lambda(w)(i,p)$. *The following properties are true:*

1. $\phi^\lambda_{w,i,p}$ *is affine on* $\mathcal{F}\left(\mathbb{P},\mathbb{R}\right)$;
2. $\phi^\lambda_{w,i,p}$ *is monotone on* $\mathcal{F}\left(\mathbb{P},\overline{\mathbb{R}}\right)$;
3. $\forall v \in \mathcal{F}\left(\mathbb{P},\overline{\mathbb{R}}\right)$, $F^{\mathcal{R}}_i(v)(p) \leqslant \phi^\lambda_{w,i,p}(v)$;
4. $\phi^\lambda_{w,i,p}(w) = F^{\mathcal{R}}_i(w)(p)$.

The properties presented in Prop. 8 imply some useful properties for the maps $\Phi^{\pi(w)}_w$.

Proposition 9 (Properties of $\Phi^{\pi(w)}_w$) *Let* $\pi \in \Pi$ *and* $w \in \mathcal{FS}\left(\mathbb{P},\overline{\mathbb{R}}\right)$. *The following properties are true:*

1. $\Phi^{\pi(w)}_w$ *is monotone on* $\mathcal{F}\left(\mathbb{P},\overline{\mathbb{R}}\right)$;
2. $F^{\mathcal{R}} \leqslant_{\mathbb{F}} \Phi^{\pi(w)}_w$;
3. $\Phi^{\pi(w)}_w(w) = F^{\mathcal{R}}(w)$;
4. *Suppose that the least fixpoint of* $\Phi^{\pi(w)}_w$ *is* $L \in \mathcal{F}\left(\mathbb{P},\mathbb{R}\right)$. *Then* L *can be computed as the unique optimal solution of the linear program:*

$$\inf\left\{ \sum_{p'\in\mathbb{P}} v(p') \;\middle|\; \begin{array}{l} \forall (i,p) \in \mathcal{I}\times\mathbb{P}, \\ \phi^{\pi_\lambda(w)(i,p)}_{i,w,p}(v) \leqslant v(p), \\ \forall q \in \mathbb{P}, \\ X^{\mathrm{Init}^{\mathcal{R}}}(q) \leqslant v(q) \end{array} \right\}. \tag{126}$$

Recall that a function $g : \mathbb{R}^d \mapsto \mathbb{R}$ is upper-semicontinuous at x iff for all $(x_n)_{n\in\mathbb{N}}$ converging to x, then $\limsup_{n\to+\infty} g(x_n) \leqslant g(x)$.

Proposition 10 *Let* $p \in \mathbb{P}$. *Then* $w \mapsto F^{\mathcal{R}}(w)(p)$ *is upper-semicontinuous on* $\mathcal{FS}\left(\mathbb{P},\overline{\mathbb{R}}\right) \cap \mathcal{F}\left(\mathbb{P},\mathbb{R}\right)$.

6.3.4 *Policy Iteration*

Now, we describe the policy iteration algorithm. We suppose that we have a post-fixpoint w^0 of $F^{\mathcal{R}}$ in $\mathcal{F}\left(\mathbb{P},\mathbb{R}\right)$.

Figure 6.1 SOS-based policy iteration algorithm for PPS programs.

input : $w^0 \in \mathcal{F}(\mathbb{P}, \mathbb{R})$, a post-fixpoint of $F^{\mathcal{R}}$
output: a fixpoint $w = F^{\mathcal{R}}(w)$ if $\forall k \in \mathbb{N}$, $w^k \in \mathcal{FS}(\mathbb{P}, \overline{\mathbb{R}})$ or a post-fixpoint otherwise

1 k=0;
2 **while** *fixpoint not reached* **do**
3 **begin** compute the next policy π for the current iterate w^k
4 Compute $F^{\mathcal{R}}(w^k)$ using Eq. (120) and Eq. (119);
5 **if** $w^k \in \mathcal{FS}(\mathbb{P}, \overline{\mathbb{R}})$ **then**
6 Define $\pi(w^k)$;
7 **else**
8 return w^k;
9 **end**
10 **end**
11 **begin** compute the next iterate w^{k+1}
12 Define $\Phi_{w^k}^{\pi(w^k)}$ and compute the least fixpoint w^{k+1} of $\Phi_{w^k}^{\pi(w^k)}$ from Problem (126);
13 k=k+1;
14 **end**
15 **end**

Now we detail step by step Algorithm 6.1. At Line 1, Algorithm 6.1 is initialized and thus $k = 0$. At Line 4, we compute $F^{\mathcal{R}}(w^k)$ using Eq. (120) and solve the SOS problem involved in Eq. (119). At Line 6, if for all $i \in \mathcal{I}$ and for $p \in \mathbb{P}$, the SOS problem involved in Eq. (119) has an optimal solution, then a policy π is available and we can choose any optimal solution of SOS problem involved in Eq. (119) as policy. If an optimal solution does not exist then Algorithm 6.1 stops and return w^k. Now, if a policy π has been defined, Algorithm 6.1 goes to Line 12 and we can define $\Phi_{w^k}^{\pi(w^k)}$ following Eq. (125). Then, we solve LP problem (126) and define the new bound on templates w^{k+1} as the smallest fixpoint of $\Phi_{w^k}^{\pi(w^k)}$. Finally, at Line 13, k is incremented.

If for some $k \in \mathbb{N}$, $w^k \notin \mathcal{FS}(\mathbb{P}, \overline{\mathbb{R}})$ and $w^{k-1} \in \mathcal{FS}(\mathbb{P}, \overline{\mathbb{R}})$ then Algorithm 6.1 stops and returns w^k. Hence, we set for all $l \geqslant k$, $w^l = w^k$.

Theorem 6.2 (Convergence result of Algorithm 6.1) *The following statements hold:*

1. *For all* $k \in \mathbb{N}$, $w^k \in \mathcal{F}(\mathbb{P}, \mathbb{R})$ *and* $F^{\mathcal{R}}(w^k) \leqslant w^k$
2. *The sequence* $(w^k)_{k \geqslant 0}$ *generated by Algorithm 6.1 is decreasing and converges;*
3. *Let* $w^\infty = \lim_{k \to +\infty} w^k$, *then* $F^{\mathcal{R}}(w^\infty) \leqslant w^\infty$. *Furthermore, if for all* $k \in \mathbb{N}$, $w^k \in \mathcal{FS}(\mathbb{P}, \overline{\mathbb{R}})$ *and if* $w^\infty \in \mathcal{FS}(\mathbb{P}, \overline{\mathbb{R}})$ *then* $F^{\mathcal{R}}(w^\infty) = w^\infty$.

6.4 EXAMPLE.

Recall that our running example is given by the following piecewise polynomial system: $(X^{Init}, \{X^1, X^2\}, \{T^1, T^2\})$, where:

$$X^{Init} = [-1, 1] \times [-1, 1]$$
$$\begin{cases} X^1 = \{x \in \mathbb{R}^2 \mid -x_1^2 + 1 \leqslant 0\} \\ X^2 = \{x \in \mathbb{R}^2 \mid x_1^2 - 1 < 0\} \end{cases}$$

and the functions relative to the partition $\{X^1, X^2\}$ are:

$$T^1(x_1, x_2) = \begin{pmatrix} 0.687x_1 + 0.558x_2 - 0.0001x_1x_2 \\ -0.292x_1 + 0.773x_2 \end{pmatrix}$$
and
$$T^2(x_1, x_2) = \begin{pmatrix} 0.369x_1 + 0.532x_2 - 0.0001x_1^2 \\ -1.27x_1 + 0.12x_2 - 0.0001x_1x_2 \end{pmatrix}$$

The first step consists in constructing the template basis and compute the template p and bound w on the reachable values as a solution of Problem (116). We fix the degree of p to 6. The template p generated from Matlab is of degree 6 and is defined as follows:

$$-1.931348006 + 3.5771x_1^2 + 2.0669x_2^2 + 0.7702x_1x_2 - (2.6284e{-}4)x_1^3 - (5.5572e{-}4)x_1^2x_2 + (3.1872e{-}4)x_1x_2^2 + 0.0010x_2^3 - 2.4650x_1^4 - 0.5073x_1^3x_2 - 2.8032x_1^2x_2^2 - 0.5894x_1x_2^3 - 1.4968x_2^4 + (2.7178e{-}4)x_1^5 + (1.2726e{-}4)x_1^4x_2 - (3.8372e{-}4)x_1^3x_2^2 + (6.5349e{-}5)x_1^2x_2^3 + (5.7948e{-}6)x_1x_2^4 - (6.2558e{-}4)x_2^5 + 0.5987x_1^6 - 0.0168x_1^5x_2 + 1.1066x_1^4x_2^2 + 0.3172x_1^3x_2^3 + 0.8380x_1^2x_2^4 + 0.0635x_1x_2^5 + 0.4719x_2^6.$$

The upper bound w is equal to 2.1343. In order to compute bounds per variable, we can take the template basis $\mathbb{P} = \{p, x \mapsto x_1^2, x \mapsto x_2^2\}$. We write q_1 for $x \mapsto x_1^2$ and q_2 for $x \mapsto x_2^2$. The basic semi-algebraic $\{x \in \mathbb{R}^2 \mid p(x) \leqslant 0,\ q_1(x) \leqslant 2.1343,\ q_2(x) \leqslant 2.1343\}$ is an inductive invariant and the corresponding bounds function is $w^0 = (w^0(q_1), w^0(q_2), w^0(p)) = (2.1343, 2.1343, 0)$.

As in Line 4 of Algorithm 6.1, we compute the image of w^0 by $F^{\mathcal{R}}$ using SOS (Eq. (119)). We found that

$$F^{\mathcal{R}}(w^0)(q_1) = 1.5503,$$
$$F^{\mathcal{R}}(w^0)(q_2) = 1.9501$$
$$F^{\mathcal{R}}(w^0)(p) = 0\ .$$

Since $w^0 \in \mathcal{FS}(\mathbb{P}, \overline{\mathbb{R}})$, Algorithm 6.1 goes to Line 6 and the computation of $F^{\mathcal{R}}(w^0)$ permits to determine a new policy $\pi(w^0)$. The important data is the vector λ. For example, for $i = 1$ and the template q_1, the vector λ is $(0, 0, 2.0331)$. It means that we associate for each template q a weight $\lambda(q)$. In the case of $\lambda = (0, 0, 2.0331)$,

$\lambda(q_1) = 0$, $\lambda(q_2) = 0$ and $\lambda(p) = 2.0332$. For $i = 1$, the template q_1 and the bound vector w^0, the function $\phi^{\lambda}_{w^0,1,q_1}(v) = 2.0331v(p) + 1.5503$.

To get the new invariant, Algorithm 6.1 goes to Line 12 and we compute a bound vector w^1 solution of Linear Program (126). In this case, it corresponds to the following LP problem:

$$\min v(q_1) + v(q_2) + v(p)$$

s.t.

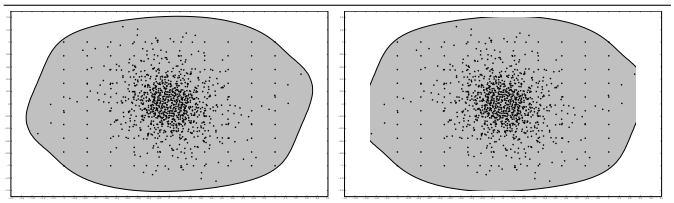
$$\begin{cases} 1 \leqslant v(q_1),\ 1 \leqslant v(q_2),\ 0 \leqslant v(p) & \text{(init)} \\ 0.4578v(p) + 0.8843 \leqslant v(q_1), & \\ 0.2048v(p) + 1.9501 \leqslant v(q_2), & (i = 1) \\ 0.9985v(p) - 3.4691e{-}7 \leqslant v(p) & \\ 2.0331v(p) + 1.5503 \leqslant v(q_1), & \\ 1.0429v(p) + 1.2235 \leqslant v(q_2), & (i = 2) \\ 0.9535v(p) - 0.0248 \leqslant v(p) & \end{cases}$$

We obtain:

$$w^1(q_1) = 1.5503,\ w^1(q_2) = 1.9501 \text{ and } w^1(p) = 0$$

We then come back to Line 4 of Algorithm 6.1 and we compute $F^{\mathcal{R}}(w^1)$ using the SOS program Eq. (119). The implemented stopping rule is $\|F^{\mathcal{R}}(w^k) - w^k\|_\infty \leqslant$ 1e–6 and since $\|F^{\mathcal{R}}(w^1) - w^1\|_\infty \leqslant$ 1e–6, Algorithm 6.1 terminates. The two successive inductive invariants are depicted at Figure 6.2.

Figure 6.2 Successive sets computed from Policy Iterations



In gray, on the left, the set $w^{0\star}$ computed from Problem 116, while, on the right, the set $w^{1\star}$ computed from Policy Iterations. In both figures, black points represents a discretized version of $\mathcal{R}$.

6.5 RELATED WORKS

As presented in the Formal Methods introduction, in Chapter 2, the classical framework for abstract interpretation is a fixpoint over-approximation through a KLEENE fixpoint computation using widening to ensure convergence. Another mechanism, narrowing, enables to recover precision once a postfixpoint has been obtained through widening.

In static analysis, the more recent approach of policy[1] iterations [Cos+05; Gau+07; GS07a] attempts to solve exactly the fixpoint equation for a given abstract domain when specific conditions are satisfied using appropriate mathematical solvers. While this chapter addressed a rather large set of programs – piecewise polynomial systems – using SOS optimization, related (and previous) works were considering simpler classes of programs and of convex optimizations. For example when both the abstract domain and the fixpoint equation use linear equations, then linear programming could be used to compute the exact solution without the need of widening and narrowing [Gau+07; GS07a]. Similarly, when the function and the abstract domain are at most quadratic, semi-definite programming (SDP) could be used [AGG10; GS10; Gaw+12]. In all cases, these analyses are performed on template-based abstract domains, representing the abstract elements as sublevel-sets; optimization techniques being used to bound these templates.

Regarding policy iterations related works, two different "schools" exist in the static analysis community. The "French school" [AGG10; Cos+05; Gau+07; Gaw+12] offers to iterate on min-policies, starting from an over-approximation of a fixpoint and decreasing the bounds until the fixpoint is reached. The "German school" [Gaw+12; GS07a; GS07b] in contrary operates on max-policies, starting from bottom and increasing the bounds until a fixpoint is reached. While the first can be interrupted at any point leaving a sound over-approximation, the second approach requires waiting until the fixpoint is reached to provide its result. Note that a first valid postfixpoint is required in the first case.

Min-Policy Iterations

To some extent, Min-Policy iterations [AGG10] can be seen as a very efficient *narrowing*, since they perform descending iterations from a postfixpoint towards some fixpoint, working in a way similar to the NEWTON-RAPHSON method. Iterations are not guaranteed to reach a fixpoint but can be stopped at any time leaving an overapproximation thereof. Moreover, convergence is usually fast.

Writing a system of equations $b = F(b)$ with $b = (b_i)_{i\in[\![1,n]\!]}$ and $F : \overline{\mathbb{R}}^n \to \overline{\mathbb{R}}^n$ (n being the number of templates), a min-policy is defined as follows: $\underline{F}$ is a min-policy for F if for every $b \in \overline{\mathbb{R}}^n$, $F(b) \leqslant \underline{F}(b)$ and there exist some $b_0 \in \overline{\mathbb{R}}^n$ such that $\underline{F}(b_0) = F(b_0)$.

The following theorem can then be used to compute the least fixpoint of F.

Theorem 6.3 *Given a (potentially infinite) set $\underline{\mathcal{F}}$ of min-policies for F. If for all $b \in \overline{\mathbb{R}}^n$ there exist a policy $\underline{F} \in \underline{\mathcal{F}}$*

1 The word *strategy* is also used in the literature for *policy*, with equivalent meaning.

interpolating F *at point* b *(i.e.* $\underline{F}(b) = F(b)$*) and if each* $\underline{F} \in \underline{\mathcal{F}}$ *has a least fixpoint* $\mathrm{lfp}\underline{F}$*, then the least fixpoint of* F *satisfies*

$$\mathrm{lfp}F = \bigwedge_{\underline{F} \in \underline{\mathcal{F}}} \mathrm{lfp}\underline{F}.$$

Iterations are done with two main objects: a min-policy σ and a tuple β of values for variables b_i of the system of equations. The following policy iteration algorithm starts from some postfixpoint β_0 of F and aims at refining it to produce a better overapproximation of a fixpoint of F. Policy iteration algorithms always proceed by iterating two phases: first a policy σ_i is selected, then it is solved giving some β_i. More precisely in our case:

- find a linear min-policy σ_{i+1} being tangent to F at point β_i, this can be done thanks to a semi definite programming solver and an appropriate relaxation;
- compute the least fixpoint β_{i+1} of policy σ_{i+1} thanks to a linear programming solver.

Iterations can be stopped at any point (for instance after a fixed number of iterations or when progress between β_i and β_{i+1} is considered small enough) leaving an overapproximation β of a fixpoint of F.

Max-Policy Iterations

Behaving somewhat as a super *widening*, Max-Policy iterations [GS10] work in the opposite direction compared to Min-Policy iterations. They start from bottom and iterate computations of greatest fixpoints on so called max-policies until a global fixpoint is reached. Unlike the previous approach, the algorithm terminates with a *theoretically* precise fixpoint, but the user has to wait until the end since intermediate results are not overapproximations of a fixpoint.

Max-policies are the dual of min-policies: $\overline{F}$ is a max-policy for F if for every $b \in \overline{\mathbb{R}}^n$, $\overline{F} \leqslant F(b)$ and there exist some $b_0 \in \overline{\mathbb{R}}^n$ such that $\overline{F}(b_0) = F(b_0)$. In particular, the choice of one term in each equation is a max-policy.

Iterations are done with two main objects: a max-policy σ and a tuple β of values for variables $b_{i,j}$ of the system of equations. Considering that computing a fixpoint on a given policy reduces to a mathematical optimization problem and that a fixpoint of the whole equation system is also a fixpoint of some policy, the following policy iteration algorithm aims at finding such a policy by solving optimization problems. To initiate the algorithm, a term $-\infty$ is added to each equation, the initial policy σ_0 is then $-\infty$ for each equation and the initial value β_0 is the tuple $(-\infty, \ldots, -\infty)$. Then policies are iterated:

- find a policy σ_{i+1} improving policy σ_i at point β_i, i.e. that reaches (strictly) greater values evaluated at point β_i; if none is found, exit;
- compute the greatest fixpoint β_{i+1} of policy σ_{i+1}.

The last tuple β is then a fixpoint of the whole system of equations.

The Max-Policy iteration builds an ascending chain of abstract elements similarly to KLEENE iterations elements. However, it is guaranteed to be finite, bounded by the number of policies σ, while KLEENE iterations require the use of widening to ensure termination. Since there are exponentially many max-policies in the number of templates and since each policy can be an improving one only once, we have an exponential bound on the number of iterations. But in practice, only a small number of policies are usually considered and the number of iterations remains reasonable. One of the approach to select a good policy is to rely on SMT-solvers to find a matching policy [MS14].

Last, recent works [KMW16] relied on Max-policies based on linear problems and Linear programming solvers to compute efficiently local invariants on large programs. This work is applied in a completely different context than ours: targeting general C programs rather than critical controllers, with linear properties rather than expressive semialgebraic ones. It shows the applicability of the approach to a larger set of programs than numerical controllers.

Part III

SYSTEM-LEVEL ANALYSIS AT MODEL AND CODE LEVEL

7

SYSTEM-LEVEL PROPERTIES AS NUMERICAL INVARIANTS

All numerical tools presented in previous chapters were focused on the precise over-approximation of reachable states. In terms of properties addressed, we can argue about simple properties: e. g.the state space is bounded, the reachable states avoid a bad region, etc.

We believe that it is however important to be able to express higher level properties than just bounding reachable states.

The idea that drove our invariants and template synthesis was this notion of LYAPUNOV functions and of LYAPUNOV stability. Assuming a control level property, it would be extremely interesting to be able to express this property over the code or model artifact.

A main limitation for the study of these control level properties is the need for the plant description, which is generally not available when considering code artifact. In the following we assume the plant semantics is provided in a discrete fashion and therefore amenable to code level description as presented in Chapter 3.

We summarize here are first attempts to express classical notions of control theory such as stability or robustness using our invariant-based tools.

NOTATIONS. Let us first recall the notations of Chapter 3: we focus on linear systems *i.e.* a linear plant with a linear controller feedback. Both the controller and the plant dynamics are expressed as discrete linear systems. Let (A^c, B^c, C^c, D^c) and (A^p, B^p, C^p) the matrices defining the controller and plant dynamics, respectively; e denotes the input of the controller, often referred to as the error *i.e.* the distance to the target reference in; u denotes both the output of the controller and the input of the plant, such as the effect of actuator commands; y denotes the measure of the plant state *i.e.* the feedback e. g.as obtained by sensors:

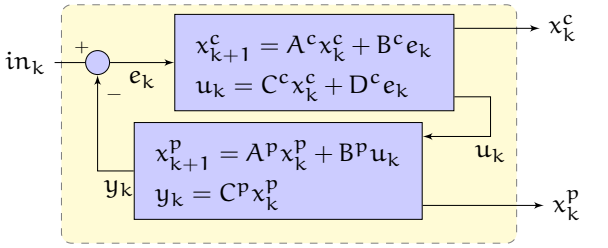
$$\begin{cases} x^c_{k+1} = A^c x^c_k + B^c e_k \\ u_k = C^c x^c_k + D^c e_k \end{cases} \quad \begin{cases} x^p_{k+1} = A^p x^p_k + B^p u_k \\ y_k = C^p x^p_k \end{cases} \tag{127}$$

A closed-loop representation of the system is given in Fig. 7.1; it is expressed over the state space defined by vectors $\begin{pmatrix} x^c & x^p \end{pmatrix}^\intercal$. Let $\mathbf{x}$ be such vectors. The error e is computed using a reference command in and the feedback y obtained from the plant.

$$e_k = in_k - y_k$$

One can consider in as the input of the closed-loop system, and $\mathbf{x}$ as its output.

Figure 7.1 Closed-loop system.



7.1 OPEN-LOOP AND CLOSED-LOOP STABILITY

As mentioned in Chapter 3, the notion of stability for a dynamical systems captures both the boundedness of reachable states and a notion of convergence. A stable system guarantees that a small change in the input will not produce a large change in the output. Mathematically speaking, the notion of asymptotic stability ensures that with a null input, the system converges to zero. This stability can be studied in two ways: open loop stability and closed-loop stability. In the open loop setting the stability of the controller itself is studied while in the closed-loop setting the complete system integrating the feedback interconnection of controller and plant is addressed.

While closed-loop stability is the main stability property of interest – that is, the controlled system will have a stable behavior – ensuring open-loop stability avoids the undesirable situation where the feedback interconnection is stable, while the controller alone is intrinsically unstable. In terms of system implementation, an open-loop stable controller has a reasonable behavior on its own, e. g. assuming only bounded input, it will pro-

vide a bounded output. This is called the bounded input bounded output (BIBO) property.

Stability properties can be assessed in different ways. A system's dynamics are expressed as transfer functions mapping inputs to outputs. These are obtained by taking the FOURIER or LAPLACE transform of the impulse response of a system. This so-called frequency domain approach is commonly used for linear systems, along with graphical tools such as BODE plots or NYQUIST diagrams. An alternative approach, temporal domain analysis, is performed on the state-space representation, and is based on LYAPUNOV functions. As mentioned in the previous parts, LYAPUNOV functions express a notion of positive energy that decreases along the trajectories of the system and captures its asymptotic stability. For linear systems, such functions are usually defined using a positive definite matrix $P \succeq 0$ such that:

$$A^\intercal P A - P \prec 0, \tag{128}$$

where A is the state matrix of the system.

7.1.1 LYAPUNOV *function computation*

Using the tools proposed in Chapters 5 we can compute inductive numerical invariants, such as positive definite matrices P^o and P^c denoting LYAPUNOV functions for these open- and closed-loop systems.

For the open-loop system, the LYAPUNOV function P^o is used to express a BIBO property of the controller alone: to bound reachable states x^c assuming a bounded input e:

$$\|e\|_\infty \leqslant 1 \implies x^{c\intercal} P^o x^c \leqslant 1$$

For the closed-loop system, integrating the feedback of the plant in the controller input, a similar property is expressed. For a bounded target reference in, the closed-loop system will admit only bounded reachable states $\mathbf{x}$:

$$\|in\|_\infty \leqslant 1 \implies \mathbf{x}^\intercal P^c \mathbf{x} \leqslant 1$$

These boundedness properties may seem weak to control engineers compared to the asymptotic stability properties expressed by the LYAPUNOV functions. However, they are of extreme importance to guarantee that the implementation will behave properly, without diverging and causing runtime errorse. g.producing numerical overflows. Once provided with a quadratic bound on reachable states using the LYAPUNOV function characterizing the stability of the controller, static analyses of the discrete model and the code can rely on policy iterations, cf. Chap. 6, to infer bounds on x^c and $\mathbf{x}$.

7.1.2 *Stability of Closed-loop system without saturation*

Recall that the closed-loop system example presented in Chapter. 3 was presented in two flavors. The first one considered a simple feedback between the linear controller and the linear plant. This global linear system is exactly described by Figure. 7.1.

Figure 7.2 displays the analyzed code for the closed-loop system described in the previous section. From such a code, our analyzer extracts the control flow graph of Figure 7.3.

Figure 7.2 Analyzed code for the closed-loop system.

```
xc1 = xc2 = xp1 = xp2 = 0;
while (1) {
  yd = acquire_input();
  assert(yd >= -0.5 && yd <= 0.5);
  oxc1 = xc1; oxc2 = xc2; oxp1 = xp1; oxp2 = xp2;
  xc1 = 0.499 * oxc1 - 0.05 * oxc2 + (oxp1 - yd);
  xc2 = 0.01 * oxc1 + oxc2;
  xp1 = 0.028224 * oxc1 + oxp1 + 0.01 * oxp2
        - 0.064 * (oxp1 - yd);
  xp2 = 5.6448 * oxc1 - 0.01 * oxp1 + oxp2
        - 12.8 * (oxp1 - yd);
  wait_next_clock_tick();
}
```

Our analysis then synthesizes a quadratic LYAPUNOV-based template P, inductive over system transitions:

$$A^\intercal P A - P \prec 0, \tag{129}$$

where A denotes the closed-loop system discrete dynamics.

Let P be the matrix obtained:

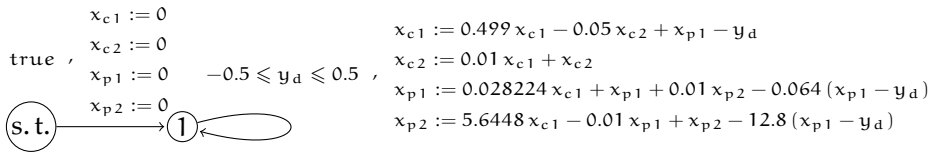
$$P := \begin{bmatrix} 1.7776 & 1.3967 & -0.6730 & 0.1399 \\ 1.3967 & 1.1163 & -0.4877 & 0.1099 \\ -0.6730 & -0.4877 & 0.3496 & -0.0529 \\ 0.1399 & 0.1099 & -0.0529 & 0.0111 \end{bmatrix},$$

From the extracted control flow graph and a set of expressions t_i on program variables, called *templates*, policy iterations techniques, cf Chapter 6 compute, for each graph vertex, bounds b_i such that $\bigwedge_i t_i \leqslant b_i$ is an invariant.

Given the templates $t_1 := x^\intercal P x$, $t_2 := x_{c1}^2$, $t_3 := x_{c2}^2$, $t_4 := x_{p1}^2$ and $t_5 := x_{p2}^2$ where x is the vector $[x_{c1}\ x_{c2}\ x_{p1}\ x_{p2}]^\intercal$ and (rounded to four digits) policy iterations compute the invariant

$$t_1 \leqslant 0.2302 \wedge t_2 \leqslant 51.0162 \wedge t_3 \leqslant 15.4720 \\ \wedge t_4 \leqslant 10.1973 \wedge t_5 \leqslant 1767.75$$

Figure 7.3 Control flow graph for code of Figure 7.2.



which implies

$$|x_{c1}| \leqslant 7.1426 \wedge |x_{c2}| \leqslant 3.9334 \wedge |x_{p1}| \leqslant 3.1933 \wedge |x_{p2}| \leqslant 42.0446.$$

Our static analyzer took 0.76s to produce this template and 1.28 to bound it on an Intel Core2 @ 1.2GHz, hence a fully automatic computation in a total of 2.19s. This shows the existence of a LYAPUNOV function, bounds reachable states and proves stability.

7.1.3 *Closed-loop system with saturations*

Realistic controllers usually contain saturations to bound the values read from sensors or sent to actuators, in order to ensure that these values remain in the operating ranges of those devices. With such a saturation on its input, the control flow graph of our running example changes to the one shown in Figure 7.4.

Unfortunately the previous method does not readily apply for the system flavor with saturation.

A first idea could be to try to generate, as previously described, a quadratic template P for each edge of the control flow graph of Figure 7.4. This approach sometimes proves successful but fails on our running example. Indeed, only one of the edges of the graph on Figure 7.4 leads to a template P (for other edges, the LYAPUNOV equation has no solution) and this template does not allow policy iterations to compute a worthwhile invariant on the whole program.

Using common LYAPUNOV functions constitutes a second idea. That is, looking for a solution to the conjunction of LYAPUNOV equations for each edge. Again, this fails since LYAPUNOV equations have no solution for some edges. This is due to the fact that the closed-loop system is not globally stable. Indeed, intuitively, when its input is saturated, the controller is not able to stabilize any arbitrary state of the plant.

Last, other approaches such as piecewise LYAPUNOV functions, cf. Sect. 5.3, admit similar limits: they require strict inequalities to ensure soundness of the analysis.

The following two Sections 7.1.3 and 7.1.3 offer two alternative ways to generate a template $x^T P x$ such that $x^T P x \leqslant r$ is an invariant of the closed-loop system with saturation for some r. Both methods manage to produce such a template but more investigations are needed to determine their relative advantages and drawbacks.

Linearizing the Saturation

One solution in this case, strongly inspired from [Fér10], provides a heuristic that can be used on systems with saturations, such as the one described in equation (33). Indeed, let P be a candidate matrix describing an invariant ellipsoid for the system. We try to characterize P as closely as possible while keeping the solving process tractable:

Assuming $x_k^T P x_k \leqslant 1$, a bound on $|Cx_k|$ is given by $\gamma := \sqrt{CP^{-1}C^T}$. Since $|y_{d,k}| \leqslant 0.5$, the constant $\tilde{\gamma} := \gamma + 0.5$ is an upper bound on $|Cx_k - y_{d,k}|$. Letting $y_{c,k} := \mathrm{SAT}(Cx_k - y_{d,k})$, we have the following sector bound:

$$\left(y_{c,k} - \frac{1}{\tilde{\gamma}}(Cx_k - y_{d,k})\right)(y_{c,k} - (Cx_k - y_{d,k})) \leqslant 0. \quad (130)$$

Figure 7.5 illustrates the reason for this inequality. With the added bound $\tilde{\gamma}$ on $|Cx_k - y_{d,k}|$, we see that $y_{c,k}$ necessarily lies between $Cx_k - y_{d,k}$ and $\frac{1}{\tilde{\gamma}}(Cx_k - y_{d,k})$. Then $y_{c,k} - \frac{1}{\tilde{\gamma}}(Cx_k - y_{d,k})$ and $y_{c,k} - (Cx_k - y_{d,k})$ must be of opposite signs, hence the inequality.

Figure 7.5 Illustration of the sector bound relationship. The equality $y_c = \mathrm{SAT}(Cx - y_d)$ (thick line) is abstracted by the inequalities $(Cx - y_d)/\tilde{\gamma} \leqslant y_c \leqslant Cx - y_d$ (gray area).

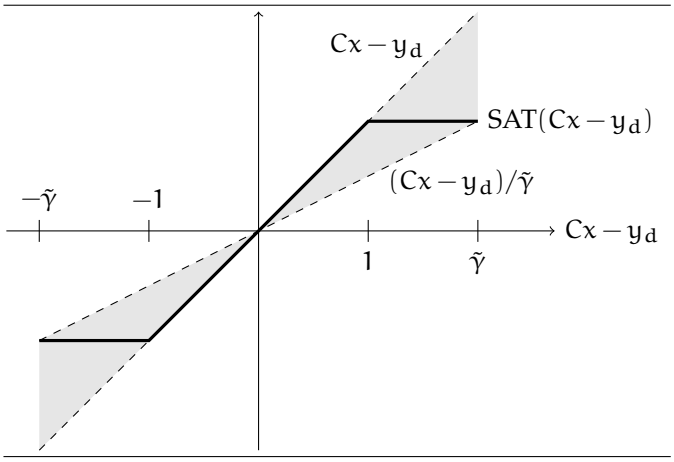
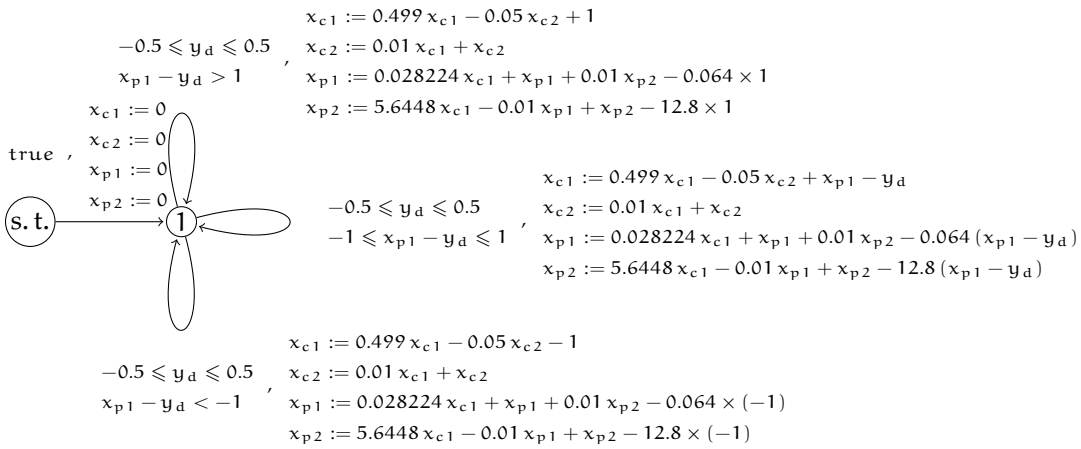


Figure 7.4 Control flow graph for the system with a saturation.



We thus look for a matrix P such that

$$\sqrt{CP^{-1}C^T} \leqslant \gamma \tag{131}$$

and

$$\left(x_k^T P x_k \leqslant 1 \wedge y_{d,k}^2 \leqslant 0.5^2 \wedge (130)\right) \implies x_{k+1}^T P x_{k+1} \leqslant 1. \tag{132}$$

Defining an extended state vector $\epsilon_k := [x_k\ y_{c,k}\ y_{d,k}\ 1]^T$ and the matrices

$$\mathcal{U} := \begin{bmatrix} A^TPA & A^TPB & 0_{4\times1} & 0_{4\times1} \\ B^TPA & B^TPB & 0 & 0 \\ 0_{1\times4} & 0 & 0 & 0 \\ 0_{1\times4} & 0 & 0 & -1 \end{bmatrix}$$

$$\mathcal{V} := \begin{bmatrix} P & 0_{4\times1} & 0_{4\times1} & 0_{4\times1} \\ 0_{1\times4} & 0 & 0 & 0 \\ 0_{1\times4} & 0 & 0 & 0 \\ 0_{1\times4} & 0 & 0 & -1 \end{bmatrix},$$

$$\mathcal{W} := \begin{bmatrix} \frac{2}{\tilde\gamma}C^TC & -\left(1+\frac{1}{\tilde\gamma}\right)C^T & -\frac{2}{\tilde\gamma}C^T & 0_{4\times1} \\ -\left(1+\frac{1}{\tilde\gamma}\right)C & 2 & 1+\frac{1}{\tilde\gamma} & 0 \\ -\frac{2}{\tilde\gamma}C & 1+\frac{1}{\tilde\gamma} & \frac{2}{\tilde\gamma} & 0 \\ 0_{1\times4} & 0 & 0 & 0 \end{bmatrix},$$

$$\mathcal{Y} := \begin{bmatrix} 0_{4\times4} & 0_{4\times1} & 0_{4\times1} & 0_{4\times1} \\ 0_{1\times4} & 0 & 0 & 0 \\ 0_{1\times4} & 0 & 1 & 0 \\ 0_{1\times4} & 0 & 0 & -0.5^2 \end{bmatrix},$$

we can rewrite equation (132) as

$$\left(\epsilon_k^T\mathcal{V}\epsilon_k \leqslant 0 \wedge \epsilon_k^T\mathcal{Y}\epsilon_k \leqslant 0 \wedge \epsilon_k^T\mathcal{W}\epsilon_k \leqslant 0\right) \implies \epsilon_k^T\mathcal{U}\epsilon_k \leqslant 0.$$

CONVEXIFICATION Equation (132) can then be relaxed by S-procedure: it will hold if there exists positive coefficients λ, μ, and ν, such that

$$\mathcal{U} - \lambda\mathcal{V} - \mu\mathcal{W} - \nu\mathcal{Y} \preceq 0. \tag{133}$$

Equation (131) can be rewritten using SCHUR complement:

$$\begin{bmatrix} \gamma^2 & C \\ C^T & P \end{bmatrix} \succeq 0. \tag{134}$$

Note that for fixed λ and γ, equations (133) and (134) form a Linear Matrix Inequality (LMI) in P, μ and ν, which means it can be solved by an SDP solver. $\tilde\gamma = \gamma + 0.5$ is expected to be larger than 1 (otherwise the saturation would never be activated), moreover since the saturation should somewhat "bound" this value, we can expect it not to span over multiple orders of magnitude. We also know that $\lambda \in (0,1)$ thanks to the bottom right coefficient of the LMI (133) (since $\nu > 0$). One possible strategy is then to iterate on potential values of λ and γ, and solving the corresponding LMI at each iteration. If a solution exists, it will provide the invariant $x^TPx \leqslant 1$ for the system with saturation.

For our running example, we generated a suitable template in 279s on an Intel Core2 @ 2.4GHz. Values for λ are chosen by exploring $(0,1)$ with numbers of the form $\frac{k}{2^i}$ for increasing values of $i \geqslant 1$, and $k < 2^i$. For each choice of λ, the LMI is solved with values of $\tilde\gamma$ ranging from 1 to 5 by increments of .1. The solution is found for $\lambda = \frac{63}{64}$ and $\tilde\gamma = 3.1$, which amounts to 2605 calls to the LMI solver.

First Abstracting the Disturbance

In the previous approach, the method used was mainly based on an abstraction of the saturation. This second one exposes an alternative method in which the disturbance y_d, rather than the saturation, is abstracted.

Let us first neglect the disturbance y_d and look for a LYAPUNOV function for the following system:

$$x_{k+1} = \begin{cases} Ax_k - B & \text{if } Cx_k \leqslant -0.5 \\ (A+BC)x_k & \text{if } -1.5 \leqslant Cx_k \leqslant 1.5 \\ Ax_k + B & \text{if } Cx_k \geqslant 0.5 \end{cases} \quad (135)$$

where A, B and C are the matrices given in (33).

Remark 4 *y_d is abstracted in the sense that the term $(A+BC)x - By_d$ of (33) is replaced by $(A+BC)x$ in (135). Similarly, guards such as $Cx - y_d \leqslant -1$ are replaced by $Cx \leqslant -0.5$ (since $|y_d| \leqslant 0.5$).*

Remark 5 *In case $0.5 \leqslant \pm Cx_k \leqslant 1.5$, the system non deterministically takes one of the two available transitions, the transition taken by the actual system (33) being determined by the value of the abstracted variable y_d.*

A quadratic LYAPUNOV function $x \mapsto x^T P x$ for this system must then satisfy $x_{k+1}^T P x_{k+1} \leqslant x_k^T P x_k$ for all $x_k \in \mathbb{R}^4$ and all possible transitions from x_k to x_{k+1}. Hence, for all $x \in \mathbb{R}^4$

$$\begin{cases} Cx \leqslant -0.5 \Rightarrow (Ax-B)^T P(Ax-B) \leqslant x^T P x \\ -1.5 \leqslant Cx \leqslant 1.5 \\ \qquad \Rightarrow ((A+BC)x)^T P((A+BC)x) \leqslant x^T P x \\ Cx \geqslant 0.5 \Rightarrow (Ax+B)^T P(Ax+B) \leqslant x^T P x. \end{cases}$$

It is worth noting that we can get rid of the first constraint by a symmetry argument. Indeed, the first constraint holds for some x if and only if the third one holds for $-x$. Similarly, we can remove the left part of the implication in the second constraint. Indeed, the right part of the implication holds for some x if and only if it holds for αx and, for α small enough, αx will satisfy the left part of the implication. Thus $x \mapsto x^T P x$ is a LYAPUNOV equation for (135) if and only if for all $x \in \mathbb{R}^4$

$$\begin{cases} ((A+BC)x)^T P((A+BC)x) \leqslant x^T P x \\ Cx \geqslant 0.5 \Rightarrow (Ax+B)^T P(Ax+B) \leqslant x^T P x. \end{cases} \quad (136)$$

By defining the vector $x' := [x^T \; 1]^T$, this can be rewritten

$$\begin{cases} x^T(A+BC)^T P(A+BC)x \leqslant x^T P x \\ [C \; 0]\, x' \geqslant 0.5 \\ \qquad \Rightarrow x'^T [A \; B]^T P [A \; B]\, x' \leqslant x'^T [I_4 \; 0]^T P [I_4 \; 0]\, x'. \end{cases}$$

By a Lagrangian relaxation, this holds when there exists a $\lambda \geqslant 0$ such that

$$\begin{cases} P - (A+BC)^T P(A+BC) \succeq 0 \\ [I_4 \; 0]^T P [I_4 \; 0] - [A \; B]^T P [A \; B] - \lambda \begin{bmatrix} 0 & C^T \\ C & -1 \end{bmatrix} \succeq 0 \end{cases}$$

where $M \succeq 0$ means that the matrix M is positive semi-definite (*i.e.* for all x, $x^T P x \geqslant 0$).

We eventually want the template $x^T P x$ to provide an invariant for the original system with the disturbance y_d. For that purpose, we not only want $(A+BC)^T P(A+BC)$ in the first inequality to be less than P but rather the least possible, in order to leave some room to later reintroduce y_d. That is, we look for τ_{min}, the least possible $\tau \in (0,1)$ satisfying

$$\tau P - (A+BC)^T P(A+BC) \succeq 0$$

for some positive definite matrix P. For any given value of τ, this is a LMI and an SDP solver can be used to decide whether a P satisfying it exists or not. Thus, τ_{min} can be efficiently approximated by a bisection search in the interval $(0,1)$.

Remark 6 *τ_{min} is also called minimum* decay rate *[Yan92].*

We are thus looking for a positive definite matrix P satisfying

$$\begin{cases} \tau_{min} P - (A+BC)^T P(A+BC) \succeq 0 \\ [I_4 \; 0]^T P [I_4 \; 0] - [A \; B]^T P [A \; B] - \lambda \begin{bmatrix} 0 & C^T \\ C & -1 \end{bmatrix} \succeq 0. \end{cases}$$

This is a LMI and could then be fed to an SDP solver. Unfortunately, it has no solution. Indeed, A has eigenvalues larger than 1 and taking x large enough can break the second constraint in (136) for any value of P.

However, x is saturated when $Cx \geqslant 1.5$ and it is then reasonable to expect Cx not to go to far beyond this threshold. We thus need to add a constraint $Cx \leqslant \gamma$ for some $\gamma > 1.5$, in the hope that the generated invariant will eventually satisfy it. This results in the following LMI

$$\begin{cases} \tau_{min} P - (A+BC)^T P(A+BC) \succeq 0 \\ [I_4 \; 0]^T P [I_4 \; 0] - [A \; B]^T P [A \; B] - \lambda D \succeq 0 \end{cases} \quad (137)$$

where $D := [C \; -0.5]^T [-C \; \gamma] + [-C \; \gamma]^T [C \; -0.5]$.

Finally, for a solution P of the above LMI, $x^T P x \leqslant r_{max}$ should be a good candidate invariant for the original system (33), with $r_{max} := \frac{\gamma^2}{CP^{-1}C^T}$ the largest r such that $x^T P x \leqslant r$ implies $Cx \leqslant \gamma$.

On our running example, 15 bisection search iterations first enable to compute $\tau_{min} = 0.9804$ (rounded to four

digits). Then, the values 2, 3, 4,... are successively tried for γ in (137). The LMI appears to have a solution for $\gamma = 2$ and $\gamma = 3$ but not for $\gamma = 4$. The value of P obtained for the last succeeding value of γ ($\gamma = 3$) is then kept as a template and fed to policy iterations along with $r_{max} = 0.26$. All these computations (bisection search for τ_{min}, tests for γ and computation of r_{max}) took 0.83s on an Intel Core2 @ 1.2GHz.

Relying on computed template

We use the second method to compute a matrix P:

$$P := \begin{bmatrix} 0.2445 & 0.3298 & -0.0995 & 0.0197 \\ 0.3298 & 1.0000 & -0.0672 & 0.0264 \\ -0.0995 & -0.0672 & 0.0890 & -0.0075 \\ 0.0197 & 0.0264 & -0.0075 & 0.0016 \end{bmatrix},$$

Then the previous steps can be performed:

Given the templates $t_1 := x^T P x$, $t_2 := x_{c1}^2$, $t_3 := x_{c2}^2$, $t_4 := x_{p1}^2$ and $t_5 := x_{p2}^2$ where x is the vector $[x_{c1}\ x_{c2}\ x_{p1}\ x_{p2}]^T$ and (rounded to four digits) policy iterations compute the invariant

$$t_1 \leqslant 0.1754 \wedge t_2 \leqslant 6.1265 \wedge t_3 \leqslant 0.3505 \wedge t_4 \leqslant 4.1586 \wedge t_5 \leqslant 1705.1748$$

which implies

$$|x_{c1}| \leqslant 2.4752 \wedge |x_{c2}| \leqslant 0.5921 \wedge |x_{p1}| \leqslant 2.0393 \wedge |x_{p2}| \leqslant 41.2938.$$

Our static analyzer took 1.39s to produce this result on an Intel Core2 @ 1.2GHz.

Remark 7 *Despite the fact that the disturbance y_d was abstracted to generate P, it is worth noting that policy iterations are performed on the complete system, with y_d.*

Remark 8 *Although quite heuristic, the choice for γ does not seem that difficult since any value in the interval $(2.40, 3.85)$ would also have led to a good template.*

7.2 ROBUSTNESS WITH VECTOR MARGIN

Beyond stability, an important property which needs to be verified is the robustness of the controller. The property of robustness is necessary in practice as there are many sources of imperfection in the feedback loop. It characterizes "how much" the closed-loop system is stable and which kind of perturbations or uncertainty can be sustained without losing stability. These imperfections can include errors in modeling the plant, uncertainties in the plant that cannot be captured by the model, noises in the sensors and actuators, limitations of the controller design, *i.e.* , not accounting for the complete range of behaviors of the system, non-linearities in the actuators, faulty actuators, etc.

The standard metric used in the industry to gauge the robustness of linear SISO[1] controllers consists of phase and gain margins. While these notions are now overseen by more modern techniques such as IQC [MR95] or μ-analysis [Doy82], there are still widely used in the industry as measures to be guaranteed for a controlled system.

However, theses margins are never analyzed or computed on the code artifact, taking into account the real implementation using floating-point arithmetic.

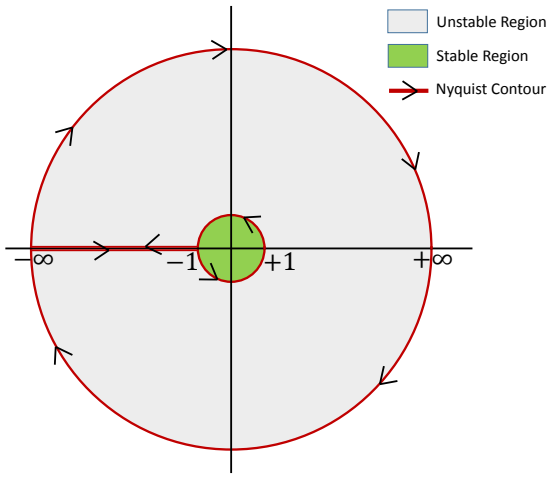
We propose here to rely on the notion of Vector margins to characterize such robustness and characterize it as a numerical invariant property over system states, and therefore amenable to code level analysis.

Let us first give an informal overview of classical frequency-based robustness analysis, using NYQUIST plots, then present our use of vector margins to bound robustness.

7.2.1 NYQUIST *Plot and Stability Criterion*

The NYQUIST plot is the frequency response (magnitude and phase) of the loop transfer function to a sinusoidal input displayed using a polar coordinate system. To construct the NYQUIST plot, the loop transfer function $L(z)$ is evaluated along the NYQUIST contour Γ. The NYQUIST contour, shown in Fig. 7.6, encircles the region outside of the unit disk (OUD) centered at the origin.

Figure 7.6 Nyquist contour in discrete-time.



1 SISO stands for "Single Input Single Output"

An example NYQUIST plot for the loop transfer function

$$L(z) := \frac{7.552\times10^{-5}z^3 - 7.583\times10^{-5}z^2 - 7.454\times10^{-5}z + 7.488\times10^{-5}}{z^4 - 3.979z^3 + 5.937z^2 - 3.937z + 0.979} \tag{138}$$

is shown in Fig. 7.7.

We now introduce the NYQUIST stability criterion which uses the Nyquist plot to determine the closed-loop stability of the system. Let Z_i be the number of OUD zeros of $L(z)+1$ and let P_i be the number of OUD poles of $L(z)+1$. By CAUCHY's principle of argument, the NYQUIST plot should encircle clockwise[2] the $-1+0j$ point N_i number of times where

$$N_i = Z_i - P_i. \tag{139}$$

Using (139) and the NYQUIST plot in Fig. 7.7, we can conclude the stability of the closed-loop system $\frac{L(z)}{1+L(z)}$ in the following way. First we know the loop transfer function $L(z)$ in (138) is stable*i.e.* $L(z)+1$ has 0 OUD poles, which means $P_i = 0$. Since the Nyquist plot in Fig. 7.7 does not encircle $-1+0j$*i.e.* the critical point, we can conclude that Z_i or the number of OUD zeros of $L(z)+1$ is also 0. Since OUD zeros of $L(z)+1$ are also the OUD poles of the closed-loop transfer function, we can conclude that the closed-loop system is also stable.

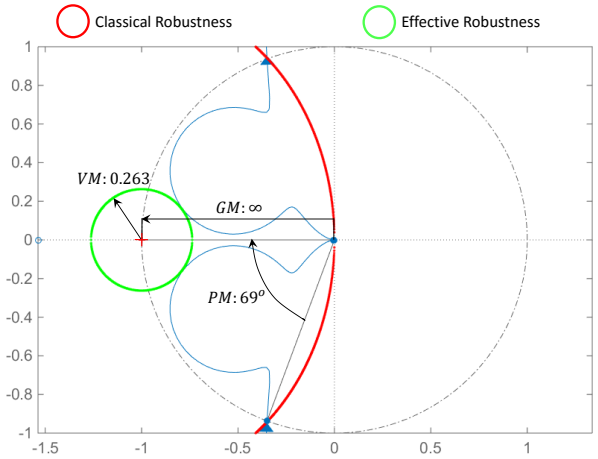
7.2.2 *Phase and Gain Margins*

From the NYQUIST stability criterion, one can infer that a possible robustness metric would be the size of the gap between the NYQUIST plot and the $-1+0j$ point. In fact, phase and gain margins are two different approximations of the "distance" from the NYQUIST plot to the critical point.

The first approximation, phase margin, measures how much phase lag the system can tolerate. A phase lag of $\frac{\pi}{2}$ or 90° corresponds to a delay of a quarter of a period. Geometrically speaking, introducing a phase lag of Δ_ω in the feedback loop results in the original NYQUIST plot rotated clockwise by Δ_ω *i.e.* $L(z) \to e^{j\Delta_\omega}L(z)$.

2 Counter-clockwise encirclement counts as negative.

Figure 7.7 Classical margins versus vector margin shown on the NYQUIST plot of (138).



The phase margin represents the amount of clockwise rotation that can be applied to the NYQUIST plot before it hits the critical point. As shown in Fig. 7.7, the phase margin (PM) is precisely the clockwise angle between the point where the unit circle, centered at the origin, intersects with the NYQUIST plot and $-1+0j$.

The second approximation, gain margin, measures how much feedback gain the system can tolerate*i.e.* how much one can scale up the NYQUIST plot radially before it intersects with the $-1+0j$ point. As shown in Fig. 7.7, the gain margin (GM) is precisely $20\log_{10}\frac{1}{x}$ where x is the magnitude of the NYQUIST plot at the phase angle of π. For good robustness, a typical requirement is a phase margin of at least 30° and a gain margin of at least 3db.

7.2.3 *Vector Margin computation*

Uncertainties in the feedback loop can introduce simultaneous phase lags and increases in the feedback gain. In those cases, interpreting the phase and gain margins could produce an overly optimistic view of the robustness of the feedback system. For example, a small phase lag combined with a small gain change would destabilize the system in Fig. 7.7, while a pure increase in gain would never do so and it would take a large phase lag alone to destabilize the system. To give a better indication of the robustness of the system, we look at the distance between $-1+0j$ and the NYQUIST plot induced by the complex modulus*i.e.* $\min_{z\in\Gamma}|L(z)+1|$. In this following, we call this robustness measure the vector margin. By plotting a circle of radius equal to the vector margin centered at the $-1+0j$ point, we get the effective robustness envelope in Fig. 7.7, which for this example, is far more pessimistic than the robustness envelope formed

by the classical measures. There are several advantages to using the vector margin.

1. It is a more faithful measure of the robustness.
2. It can be translated into the time-domain and then expressed on the code as a quadratic invariant.
3. It readily extends to MIMO systems [GVP00; Vin01].

The vector margin can be computed by finding the inverse of the maximum modulus of the sensitivity function $S(z) := \frac{1}{1+L(z)}$ over the NYQUIST contour Γ. This can be seen by noting that

$$\min_{z\in\Gamma} |L(z)+1| = \frac{1}{\max_{z\in\Gamma} \frac{1}{|L(z)+1|}} = \frac{1}{\max_{z\in\Gamma} \left|\frac{1}{L(z)+1}\right|}.$$

The sensitivity function is a first-order approximation of the change in the output over the change in the input for the closed-loop system. The state-space representation of the sensitivity function $S(z) := \frac{1}{1+L(z)}$ where $L(z) := P(z)C(z)$ can be expressed in terms of the matrices which form the state-space realization of the plant $P(z)$ and the controller $C(z)$. For the example in Fig. 7.1, the sensitivity transfer function has the following state-space realization

$$\begin{aligned} x_{k+1} &= A_s x_k + B_s in_k \\ e_k &= C_s x_k + D_s in_k \end{aligned} \tag{140}$$

where

$$A_s := \begin{bmatrix} A_c & -B_c C_p \\ B_p C_c & A_p - B_p D_c C_p \end{bmatrix} \quad B_s := \begin{bmatrix} B_c \\ B_p D_c \end{bmatrix} \\ C_s := \begin{bmatrix} 0 & -C_p \end{bmatrix} \quad D_s := \begin{bmatrix} I \end{bmatrix}. \tag{141}$$

By the application of the bounded real lemma [HC08, pg.821], we have the following result.

Property 7.1 *If there exists a positive-definite matrix P and $\gamma > 0$, such that*

$$x_{k+1}^T P x_{k+1} - x_k^T P x_k \leqslant \gamma^2 \|in_k\|_2^2 - \|e_k\|_2^2 \tag{142}$$

then $\max_{z\in\Gamma} |S(z)| \leqslant \gamma$.

The inequality in (142) is a dissipativity condition [Wil72] and can be checked efficiently by solving a linear matrix inequality (LMI) [Boy+94]. We have the following proposition.

Property 7.2 *The previous inequality (142) can be written as the following LMI*

$$\begin{pmatrix} A_s^T P A_s - P + C_s^T C_s & A_s^T P B_s + C_s^T D_s \\ B_s^T P A_s + D_s^T C_s & D_s^T D_s + B_s^T P B_s - \gamma^2 I \end{pmatrix} \prec 0. \tag{143}$$

By Proposition 7.2, for any $P \succ 0$ and $\gamma > 0$ satisfying (143), we have $\max_{z\in\Gamma} |S(z)| \leqslant \gamma$. By minimizing γ in (143), we get the vector margin $\delta = \frac{1}{\gamma}$.

Thus, summing (142) from time 0 to any time T, we get

$$x_{T+1}^T P x_{T+1} - x_0^T P x_0 \leqslant \gamma^2 \left(\sum_{k=0}^{T} \|in_k\|_2^2\right) - \sum_{k=0}^{T} \|e_k\|_2^2$$

and since P is positive definite, assuming $x_0 = 0$

$$\sum_{k=0}^{T} \|e_k\|_2^2 \leqslant \gamma^2 \left(\sum_{k=0}^{T} \|in_k\|_2^2\right). \tag{144}$$

7.2.4 *Relationship with Phase and Gain Margins*

While vector margins could be computed automatically on the linear system, including its implementation, the use of phase and gain margins is often required to interact with control engineers. We propose here classical projections of vector margins onto a safe approximation of their associated phase and gain margins.

PHASE MARGINS As explained in Sec 7.2.2, the phase margin denotes the angle between the intersection of the NYQUIST plot of the transfer function with the unit circle and the point $-1 + 0j$.

This angle is necessary larger than the angle between the intersection of the computed safe circle of radius δ with the unit circle and the point $-1 + 0j$ (*cf.* Fig. 7.7, where $\delta = VM$).

In that case a direct projection of vector margins to phase margins is

$$\phi_\delta = 2 \arcsin(\delta/2)$$

GAIN MARGINS Similarly a safe gain margin can be obtained by projecting the vector margin. Gain margin denotes the acceptable scale of the NYQUIST plot to avoid intersection with the point $-1 + 0j$.

We can approximate the gain margin associated to the vector margin δ:

$$\Theta_\delta = \frac{1}{1-\delta}$$

This gain is usually reported in *dB*:

$$\Theta_\delta = 20 \cdot \log 10 \left(\frac{1}{1-\delta}\right)$$

7.2.5 *Spring Mass Damper analysis*

When analyzing the example provided in Chapter 3, considering the closed-loop system, we obtain, with classical methods, the following phase and gain margins: $\Theta = 17dB$ and $\phi = 49°$. Note that we assume a system without saturations since saturation can not be considered when computing NYQUIST plot analysis.

From the discrete plant and controller description, the sensitivity system is automatically built and analyzed with the LMI (143), we obtain $\gamma = 1.4914$ and

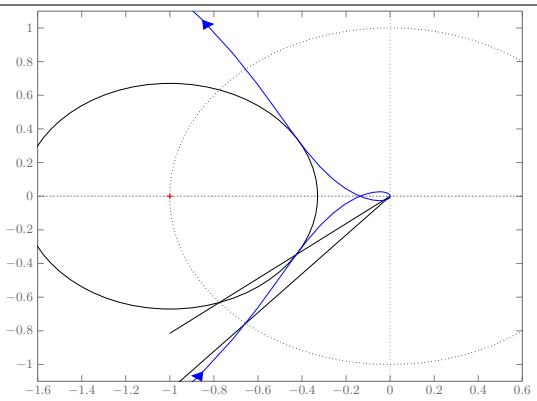
$$P = \begin{bmatrix} 111.8330 & 88.4842 & -48.4990 & 8.8432 \\ 88.4842 & 278.5963 & -20.2482 & 6.9605 \\ -48.4990 & -20.2482 & 28.7964 & -3.7961 \\ 8.8432 & 6.9605 & -3.7961 & 0.7013 \end{bmatrix}.$$

The resulting vector margin is $\delta = 1/\gamma = 0.6705$. And its projection to conservative gain and phase margins returns:

$$\Theta_\delta = 10dB \qquad \phi_\delta = 39°$$

Fig. 7.8 presents the NYQUIST plot and the vector margin.

Figure 7.8 NYQUIST plot of the spring mass damper system with vector margin



7.3 RELATED WORK

Few analyses addressed this issue of closed-loop stability in settings comparable to ours, and none the robustness as code level compatible analysis.

At control level, both stability and robustness properties were historically the earliest considered. Lots of techniques address them through different means, we refer the interested (computer scientist) reader to an introductory lecture on control theory [Lev96]. In control theory, two main approaches exist to analyze systems. Either the temporal domain, mentioned above, or the frequency domain, more commonly used. In the frequency domain, stability is usually analyzed by studying the pole placement of the transfer function, either on the Laplace transform of the signal (negative-real part), or on its Z-transform (within the unit circle). In both cases, the system has to be fully linearized (ie removing saturation around the linearization point) and the analysis assumes a real semantics, without considering floating-point computations.

Even in the temporal domains analyses, as computed by control theorists, the effect of floating-point computations performed at the controller level and those potentially done during the analysis itself are typically forgotten. These will be address by in Chapter 9.

On the static analysis side, few existing analyses are able to express the simple property of stability. As mentioned earlier, most of the existing abstract domains, used to compute an over-approximation of reachable states, rely on linear approximations. The methods proposed in Chapters 5 and 6, providing, respectively, the synthesis of non linear sublevel-set invariants and narrowing of them using min-policy iterations, addressed this issue. However, as developed in Chapter 9, because of floating point errors a strict inductive condition is enforced, preventing the analysis of saturating controllers. The proposed methods provide additional means to handle this large set of systems.

Finally, a last line of work has to be mentioned: the vast set of work focusing on hybrid systems [MT13; PW07; PS13; RS10; ST11]. It is difficult to summarize in a few words those analyses. We could however say that usually (1) they address systems of a somewhat different nature with a central continuous behavior described by differential equations and few discrete events (for instance a bouncing ball or an overflowing water tank) whereas controllers perform discrete transitions on a periodical basis, and (2) they focus on bounded time properties rather than invariant generation. These two points can be major obstacles to the adaptation of these very interesting techniques to our setting.

For instance, although bounded-time analyses (simulation) do not provide invariants, they enable the use of techniques directly analyzing the continuous plant, such as guaranteed integration [Bou+09]. This avoids discretizing the plant, as we do here, which can introduce additional conservatism in the analysis.

8

VALIDATION OF SYSTEM-LEVEL PROPERTIES AT CODE LEVEL

All previous analyzes were performed on models, a discrete dynamical systems. As mentioned in Section 4.1, these models can be either provided as early design artifacts or extracted for more concrete representation such as models or code.

However, once the control-level properties have been expressed and analyzed at model level, we would like to assert their validity on the code artifact extracted from the model.

Luckily this extraction of code from models is largely automatized thanks to autocoding framework generating embedded code from dataflow models such as MATLAB SIMULINK, ESTEREL SCADE or the academic language LUSTRE [Cas+87]. Code generation from dataflow languages [Bie+08] is now effective and widely used in the industry, supported by tools such as MATLAB REAL TIME WORKSHOP, ESTEREL KCG or Lustre compilers, eg. [GTK12].

We claim that code generation can be adapted to enable the expression of system-level properties at code level, and be later proved with respect to the code semantics.

The current chapter addresses this issue. We first give an overview of the modeling framework, enabling the expression of properties at model and code level. A second part explains our generation of such code annotations while a last part focuses on their verification.

8.1 AXIOMATIC SEMANTICS OF CONTROL PROPERTIES THROUGH SYNCHRONOUS OBSERVERS AND HOARE TRIPLES

The framework of control software credible autocoding using control semantics is summarized in figure 8.1. The framework provides a conduit that allows the domain expert e.g. the control engineer to more efficiently produce code with automatically verifiable guarantees of safety and high-level functional properties. A credible autocoding framework adds, on top of the basic model-based development cycle, an additional layer for the generation, translation, and verification of the control semantics of the system. By control semantics of the system, we mean precisely the closed-loop stability and performance properties of the control system and their accompanying proofs. In this framework, the control engineer can choose either to provide manually the properties and proofs as part of the specifications of the controller, or leave it to an automated analyzer that generates the proofs of stability and performance from the controller specifications as presented in Chapter 7.

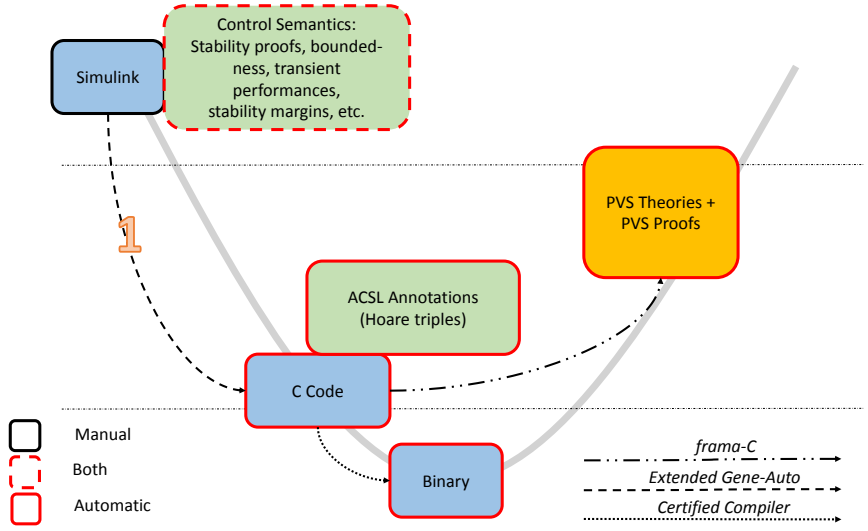
The framework is split into two nominally independent self-contained halves, in which, each corresponds to a side in the classic V-diagram of the software development cycle. The left half of the credible autocoding cycle automatically transforms the model of control system into a compilable code annotated with a collection of HOARE logic statements. Taken as a whole, the collection of logic statements i.e. the annotations along with the code, form a claim of proof that the code satisfies certain closed-loop stability and performance properties of the control system.

The right half of the framework performs an automatic deductive verification of the annotated code with respect to the control system properties expressed by the annotations. The deductive verification process, while assisted by the proof information provided within the generated annotations, is nonetheless independent of the credible autocoding process. It is independence can be seen as follows: analogously, the deductive verification of a program with respect to a property is similar to proving a theorem in mathematics; within the same analogy, the annotations generated by the credible autocoding process form the steps of a proof, and what the right half of the framework does is to automatically check the correctness of the provided proof using a computerized proof system that is sound; just like in mathematics, the proofs provided by one author can be independently verified by other parties; the main difference between that and verifying a proof on the code is that the proof statements are formalized in a such way that it can be checked automatically by a computer proof system.

8.1.1 *Languages and Tools*

For the framework, the input language could be any graphical data-flow modeling language such as

Figure 8.1 Automated Credible Autocoding and Verification Framework for Control Systems



Simulink or Scicos [CCN06] that is suited for controller design. The input language should have formal and precise semantics so the process for the generation of the code and the control semantics can be formally verified. The exact choice for the input language is dependent on the domain experts' preference and does not affect the utility of the framework as it can be adapted to other modeling languages such as Scade.

For this specific work the experimental tool-chain we developed accepts a subset of the Simulink language, since it relies on an existing open-source compiler for Simulink: GeneAuto. Likewise, for the output language, the choice is likely to depend on the preferences of the industry and the certification authority. We choose the C language because of its industrial popularity and the wide availability of static analyzers tailored for C code, including the Frama-C platform [Cuo+12] from CEA and its weakest precondition analyzer: WP.

The set of annotations in the output source code contains both the functional properties inserted by the domain expert and the proofs that can be used to automatically prove these properties. For the analysis of the annotated output, we built a prototype annotation checker that is based on the static analyzer Frama-C and the theorem prover PVS. For automating the proof-checking of the annotated output, a set of linear algebra definitions and theories were integrated into the standard NASA PVS library [Her+12].

In this chapter, the fully automated process from the input model to the verified output is showcased for the property of close-loop stability, but the expression of other functional properties on the model are also discussed in Chapter 12. At this point, we restrict the input to only linear controllers with possible saturations in the loop. For this presentation, example are related to the Spring-Mass damper introduced in Section. 3.2. However, we have applied the approach to several other much larger systems, which include the Quanser 3-degree-of-freedom Helicopter, an industrial F/A-18 UAV controller system and a FADEC control system for a small twin jet turbofan engine [Pak+13]. The state-space size of the engine FADEC, for example, is 15.

In the next three sections, we develop our approach on expressing and manipulating control-level semantics at the model in Simulink (see 8.1.2), at the code level in C (see 8.1.3), and as PVS predicates for the proof part (see 8.1.4).

8.1.2 *Control Semantics in Simulink: Ellipsoid-based synchronous observers*

When considering synchronous dataflow language a convenient approach to formalize their intend semantics is to rely on synchronous observers [HLR93; Rus12]. These observers are defined using the modeling language and are included in the description of the system. These are blocks that, depending on internal signals of the system compute a boolean output. A valid observation will produce a positive output while a violation of the encoded property will produce a negative one. These blocks characterize a projection, an observation, of the behavior of the system and therefore denote an axiomatic semantics. Synchronous observers can range from simple observation to complex systems in which the block defining the observer is itself defined by numerous subsystems including memories.

These observers can be used for verification and validation. When relying on tests, one can evaluate the output of these block, acting as a test oracle for the encoded

property. When performing proof, the goal is to prove that the output of the block is always positive, for all reachable states.

Concerning system-level properties, as we saw in previous chapter, both boundedness and stability can be expressed using a synchronous observer with inputs $x_i, i = 1, \ldots, n$, and the boolean-valued function

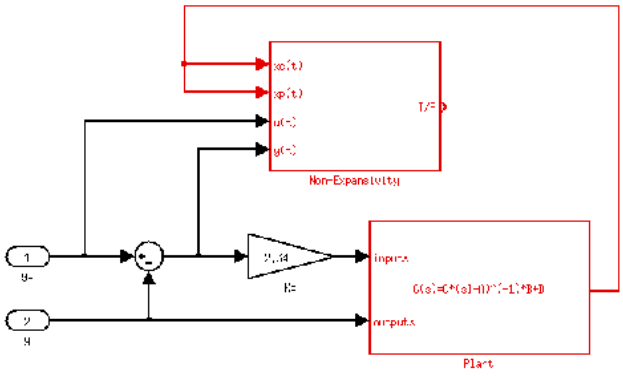
$$x \to \sum_{i,j=1,\ldots,n} x_i P_{ij} x_j \leqslant \mu. \tag{145}$$

This synchronous observer is parametrized by a symmetric matrix P and a multiplier μ.

For expressing the ellipsoid observers on the Simulink model, we constructed a custom S block denoted as *Ellipsoid* to represent the ellipsoid observer. Additionally, for expressing the operational semantics of the plant, we constructed a custom S block denoted as *Plant*. Its semantics is similar to Simulink's discrete-time state-space block with two key differences. One is that the input to the *Plant* block contains both the input and output of the plant. The other is that the output from the *Plant* block are the internal states of the plant. These variables are used to characterize the current state and are therefore inputs to the ellipsoid observer.

Other properties can also be expressed such as non-expansivity from the dissipativity framework. The *Non-Expansivity* block, when connected with the appropriate inputs and outputs, can be used to express a variety of performance measures such as the H_∞ characteristic of the system or the closed-loop vector margin of the control system. An example of such usage is shown in Figure 8.2 where the closed-loop vector margin of a constant gain controller is expressed using a combination of the *Plant* block and the *Non-Expansivity* block.

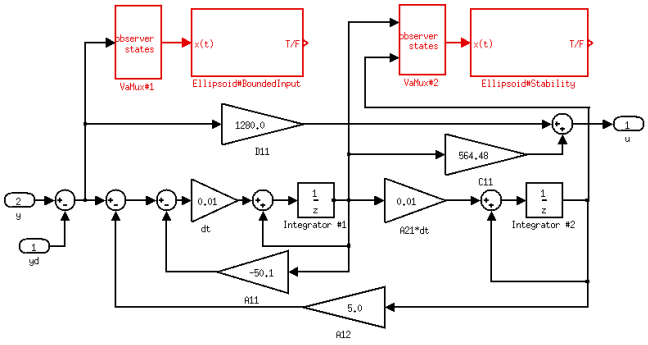
Figure 8.2 Expressing Vector Margin of the Closed-Loop System



In this chapter, we focus on the current fully-automated treatment of the open-loop stability properties, hence we will not consider the semantics displayed in Figure (8.2) beyond the description here.

For the running example, we have a Simulink model connected with two synchronous observers. The observers are displayed in red for clarity's purpose.

Figure 8.3 Running Example with Synchronous Observers



Recall that we made the following assumption for the quantity $y - y_d$:

$$\|y - y_d\| \leqslant 0.5, \tag{146}$$

which is expressed in Figure 8.3 by the Ellipsoid block *BoundedInput* with the parameters $P = 0.5$ and multiplier $\mu = 0.0009$. The stability proof is expressed in Figure 8.3 by the Ellipsoid block *Stability* with the parameters $P = \begin{bmatrix} 6.742 \times 10^{-4} & 4.28 \times 10^{-5} \\ 4.28 \times 10^{-5} & 2.4651 \times 10^{-3} \end{bmatrix}$ and $\mu = 0.9991$. The observer blocks in Figure 8.3 are connected to the model using the *VaMux* block. The role of the *VaMux* block is to concatenate a set of scalar signal inputs into a single vector output. This special block was constructed because the *Ellipsoid* observer block can accept only a single vector input.

8.1.3 *Control semantics at C code-level*

For the specific problem of open loop stability, the expressiveness needed at the C code level is twofold. On the one hand, one needs to express that a vector composed of program variables belongs to an ellipsoid: this amounts to a proof of a loop invariant; in our case, this entails a number of underlying linear algebra concepts. On the other hand, one needs to provide the static analysis tools with indications on how to proceed with the proof of correctness.

The ANSI/ISO C Specification Language (ACSL), is an annotation language for C [Bau+08]. It is expressive enough to fulfill our needs, and its associated verification tool, Frama-C [Cuo+12], offers a wide variety of backend provers which can be used to establish the correctness of the annotated code.

Linear Algebra in ACSL

A library of ACSL symbols has been developed to express concepts and properties pertaining to linear algebra. In particular, types have been defined for matrices and vectors, and predicates expressing that a vector of variables is a member of the ellipsoid $\mathcal{E}_P$ defined by $\{x \in \mathbb{R}^n : x^T P x \leqslant 1\}$, or the ellipsoid $\mathcal{G}_X$ defined by $\left\{ x \in \mathbb{R}^n : \begin{bmatrix} 1 & x^T \\ x & X \end{bmatrix} \geqslant 0 \right\}$. For example, expressing that the vector composed of program variables v_1 and v_2 is in the set $\mathcal{E}_P$ where $P = \begin{pmatrix} 1.53 & 10.0 \\ 10.0 & 507 \end{pmatrix}$, can be done with our ACSL extensions using the following annotations:

ACSL
```
/*@ logic matrix P = mat_of_2x2_scalar(1.53
    ,10.0,10.0,507);
  @ assert in_ellipsoid(P,vect_of_2_scalar(
      v_1,v_2)); */
```

The invariance of ellipsoid $\mathcal{E}_P$ throughout any program execution can be expressed by the following *loop invariant*:

C+ACSL
```
//@ loop invariant in_ellipsoid(
    P,vect_of_2_scalar(v_1,v_2));
while (true)\{
  //loop body
\}
```

This annotation expresses that before and after every execution of the loop, the property $\begin{bmatrix} v_1 & v_2 \end{bmatrix}^T \in \mathcal{E}_P$ will hold. In terms of expressiveness, it is all that is required to express open loop stability of a linear controller.

However, in order to facilitate the proof, intermediate annotations are added within the loop to propagate the ellipsoid through the different variable assignments, as suggested in [Fér10]. For this reason, a loop body instruction can be annotated with a local contract, like so:

C+ACSL
```
/*@ requires in_ellipsoid(P,vect_of_2_scalar
    (v_1,v_2));
  @ ensures in_ellipsoid(Q,vect_of_3_scalar(
      v_1,v_2,v_3));*/
\{
// assignment of v_3
\}
```

Including proof elements

An extension to ACSL, as well as a plugin to Frama-C, have been developed. They make it possible to indicate the proof steps needed to show the correctness of a contract, by adding extra annotations. For example, the following syntax:

C+ACSL
```
/*@ requires in_ellipsoid(P,vect_of_2_scalar
    (v_1,v_2));
  @ ensures in_ellipsoid(Q,vect_of_3_scalar(
      v_1,v_2,v_3));*/
  @ PROOF_TACTIC (use_strategy (
      AffineEllipsoid));
\{
// assignment of v_3
\}
```

advices Frama-C to use the strategy `AffineEllipsoid` to prove the correctness of the local contract considered.

closed-loop semantics: expressing plant dynamics in code

In order to express properties pertaining to the closed-loop behavior of the system, one needs to be able to refer to the plant variables. This is achieved through the use of ACSL *ghost* variables. These variables can be initialized and updated like regular C ones, but they only exist in the annotations of the code, and cannot influence the outcome of actual code computations.

This is possible since all our closed-loop analysis, stability and robustness through vector margins, were based on a discrete (linear) representation of the plant semantics. Therefore, the discrete block defining the plant could be used to generate ghost C code.

At the end of the control loop, we use these variables to express the state update of the plant that will result from the computed control signal value. We also enforce axiomatically the fact that the input read from the sensors equals the output of the plant.

8.1.4 *Control semantics in PVS*

Through a process described in Section 8.3, verifying the correctness of the annotated C code is done with the help of the interactive theorem prover PVS. This type of prover normally relies on a human in the loop to provide the basic steps required to prove a theorem. In order to reason about control systems, linear algebra theories have been developed. General properties of vectors and matrices, as well as theorems specific to this endeavor have been written and proven manually within the PVS environment.

Basic types and theories

Introduced in [Her+12] as part of the larger NASA PVS library, the PVS linear algebra library allows one to reason about matrix and vector quantities, by defining relevant types, operators and predicates, and proving major properties. To name a few:

- A vector type.
- A matrix type, along with all operations relative to the algebra of matrices.
- Various matrix subtypes such as square, symmetric and positive definite matrices.
- Block matrices
- Determinants
- High level results such as the link between SCHUR's complement and positive definiteness

Theorems specific to control theory

In [Her+12], a theorem was introduced, named the ellipsoid theorem. A stronger version of this theorem, along with a couple other useful results in proving open loop stability of a controller, have been added to the library. The first theorem, presented in Fig. 8.4, expresses in the PVS syntax how a generic ellipsoid $\mathcal{G}_Q$ is transformed into $\mathcal{G}_{MQM^T}$ by the linear mapping $x \mapsto Mx$. A second theorem, presented in Fig. 8.5, expresses how, given 2 vectors x and y in 2 ellipsoids $\mathcal{G}_{Q_1}$ and $\mathcal{G}_{Q_2}$, and multipliers $\lambda_1, \lambda_2 > 0$, such that $\lambda_1 + \lambda_2 \leqslant 1$, it can always be said that $\begin{pmatrix} x & y \end{pmatrix}^T \in \mathcal{G}_Q$, where $Q = \begin{pmatrix} \frac{Q_1}{\lambda_1} & 0 \\ 0 & \frac{Q_2}{\lambda_2} \end{pmatrix}$.

These 2 theorems are used heavily in Section 8.3 to prove the correctness of a given HOARE triple. While they are not particularly novel, their proof in PVS was no trivial process and required close to 10000 manual proof steps.

Figure 8.4 Ellipsoid theorem in PVS

PVS

```
ellipsoid_general: THEOREM
 \(\forall\) (n:posnat,m:posnat, Q:SquareMat
     (n),
           M: Mat(m,n), x:Vector[n], y:
               Vector[m]):
              in_ellipsoid_Q?(n,Q,x)
              AND y = M*x
        IMPLIES
        in_ellipsoid_Q?(m,M*Q*transpose(M),y
            )
```

Figure 8.5 Ellipsoids combination theorem in PVS

PVS

```
ellipsoid_combination: THEOREM
 \(\forall\) (n,m:posnat, lambda_1, lambda_2
     : posreal, Q_1: Mat(n,n),
           Q_2: Mat(m,m), x:Vector[n], y:
               Vector[m], z:Vector[m+n]):
              in_ellipsoid_Q?(n,Q_1,x)
              AND in_ellipsoid_Q?(m,Q_2,y)
              AND lambda_1+ lambda_2 <= 1
              AND z = Block2V(V2Block(n,m)(x
                  ,y))
        IMPLIES
        in_ellipsoid_Q?(n+m,Block2M(M2Block(
            n,m,n,m)(1/lambda_1*Q_1,
                        Zero_mat(m,n),
                            Zero_mat(n,m),1/
                            lambda_2*Q_2)),z)
```

8.2 GENERATING ANNOTATIONS: A STRONGEST POSTCONDITION PROPAGATION ALGORITHM

If provided such powerful tools one would only need to annotate the loop body with a single loop invariant: the sublevel set μ of the ellipsoid $\mathcal{E}_P$. Unfortunately tools, at that time, were not able to handle such an annotation and prove it inductive.

Therefore, we relied on intermediate annotations to express locally the impact of the computations to the loop invariant.

Remark 9 *Ellipsoids can be characterized by positive definite matrices* P*:* $\{x \mid x^tPx \leqslant 1\}$*. When* $P \succeq 0$*,* P *is invertible and then necessarily full rank. As a consequence it cannot contain any dependencies between the variables and the intermediate computation may lead the ellipsoid into a degenerate form. A degenerate ellipsoid* P *will characterize a set* $\{x \mid x^tPx \leqslant 1\}$ *of lower dimension, for example it can be visualized in two dimensions as a line segment.*

An alternative way is to manipulate ellipsoids through their Q-form representation, an application of SCHUR complement.

Definition 8.1 (Q-form) *If matrix* P^{-1} *exists and let* $Q = P^{-1}$*, then* $x^TPx \leqslant 1$ *is equivalent to* $\begin{bmatrix} 1 & x^T \\ x & Q \end{bmatrix} \geqslant 0$ *which is the Q-form*

Our framework reads the provided ellipsoids, convert it in its Q-form and annotate the loop body – usually a function – with it a an inductive contract.

Then Q-form ellipsoids are injected between each instruction to express the local invariant.

The following figures show a portion of the autocoded output of the running example.

The three ACSL annotations in Figure 8.6 defines the matrix variables **QMat_0**, **QMat_1** and **QMat_2**. All three matrix variables parametrize the same ellipsoid as the one obtained from the stability analysis and inserted into the Simulink model as the *Ellipsoid#Stability* observer.

Figure 8.6 Definition of ellipsoids as annotations

C+ACSL
```
/*@ logic matrix QMat_0 =
  mat_of_2x2_scalar(1484.8760396857954,-25
      .780980284188082,
       -25.780980284188082,406
           .11067541120576);
*/
/*@ logic matrix QMat_1 =
  mat_of_2x2_scalar(1484.8760396857954,-25
      .780980284188082,
       -25.780980284188082,406
           .11067541120576);
*/
/*@  logic matrix QMat_2 =
  mat_of_2x2_scalar(1484.8760396857954,-25
      .780980284188082,
      -25.780980284188082,406
          .11067541120576);
*/
```

Using the ACSL keywords *requires* and *ensures*, we can express pre and post-conditions for lines of code as well as functions. The ACSL function contract in Figure 8.7 expresses the inserted ellipsoid pre and post-conditions for the state-transition function of the controller: **discrete_timeg_no_plant_08b_compute**.

Figure 8.7 Loop body function contract

C+ACSL
```
/*@
        requires in_ellipsoidQ(QMat_0,
  vect_of_2_scalar(
      _state_->Integrator_1_memory,
      _state_->Integrator_2_memory));
        requires \valid(_io_) && \valid(
            _state_);
        ensures in_ellipsoidQ(QMat_1,
  vect_of_2_scalar(
      _state_->Integrator_1_memory,
  _state_->Integrator_2_memory));
*/
void discrete_timeg_no_plant_08b_compute(
    t_discrete_timeg_no_plant_08b_io *_io_,
    t_discrete_timeg_no_plant_08b_state *
        _state_);
```

Within the body of this loop function, ACSL annotations, such as the one of Figure 8.8, manipulate intermediate ellipsoids. In this case, it contains a copy of the pre-condition from the function contract annotation. This is the inserted ellipsoid pre-condition for the beginning of the function body.

Figure 8.8 Statement level annotation

C+ACSL
```
/*@
            behavior ellipsoid0_0:
            requires in_ellipsoidQ(QMat_2,
      vect_of_2_scalar(
          _state_->Integrator_1_memory,
      _state_->Integrator_2_memory));
            ensures in_ellipsoidQ(QMat_3,
      vect_of_3_scalar(
          _state_->Integrator_1_memory,
      _state_->Integrator_2_memory,x1));
            @ PROOF_TACTIC (use_strategy (
                AffineEllipsoid));
    */
{
    x1 = _state_->Integrator_1_memory;
}
```

8.2.1 *Invariant Forward Propagation*

The manual forward propagation of ellipsoid invariants was described in [Fér10]. We recall here the basic principles. More details are available in [Wan+16a] or [Wan15].

Affine Transformation

For the linear propagation of ellipsoids, the *AffineEllipsoid* rule is defined. It is used for linear assignments such as $[\![y]\!] \vDash Lx$, where $x \in \mathbb{R}^m$ is vector of program states and $L \in \mathbb{R}^{1\times m}$. It is applied to an existing ellipsoid in Q-form, Q_n, and characterizes the next ellipsoid Q_{n+1}.

Recall that Q_n is defined by $\begin{bmatrix} 1 & x^\intercal \\ x & Q \end{bmatrix} \geqslant 0$ which is equivalent to the more classical inequality $x^\intercal Q^{-1} x \leqslant 1$.

The formal definition of Q_{n+1} involves mechanisms to adapt the dimension of the ellipsoid depending whether y belong or not to the original set of variables in x. However, the main idea is the following: the ellipsoid information obtained after this assignment can be represented by $\begin{bmatrix} 1 & y^\intercal \\ y & LQ_nL^\intercal \end{bmatrix} > 0$. Forward propagation of linear assignment is then easily applicable to Q-form representations.

S-Procedure

The *SProcedure* rule is used to combine ellipsoids. We introduced it in Theorem 4.6, page 34. In case of multi-

ple ellipsoid invariants to be merged in a single one, we rely on the S-Procedure to compute the multiplies and generate the stronger one. This typically arises after a disjunction in the control flow graph such as a saturation.

8.2.2 *Verification of the Generated Post-condition*

After the invariant propagation step, we obtain the latest ellipsoid post-condition. It is necessary to check if this new post-condition implies the inserted ellipsoid pre-condition defined at the function contract level. Currently, we can do a conservative numerical verification by using a safe CHOLESKY decomposition. This is further developed in Chapter 10.

8.3 DISCHARGING PROOF OBJECTIVES USING PVS

Once the annotated C code has been generated, it remains to be proven that the annotations are correct with respect to the code. This is achieved by checking that each of the individual HOARE triples hold. Figure 8.9 presents an overview of the checking process. First, the WP plugin of Frama-C generates verification conditions for each HOARE triple, and discharges the trivial ones with its internal prover QeD. Then, the remaining conditions are translated into PVS theorems for further processing. It is then necessary to match the types and predicates introduced in ACSL to their equivalent representation in PVS. This is done through theory interpretation [OS01] and outlined in subsection 8.3.2. Once interpreted, the theorems can be generically proven thanks to custom PVS strategies, as described in subsection 8.3.3. In order to automatize these various tasks and integrate our framework within the Frama-C platform, which provides graphical support to display the status of a verification condition (proved/unproved), we wrote a Frama-C plugin named pvs-ellipsoid, described in subsection 8.3.4. Finally, the last ellipsoid inclusion check does not fall under either `AffineEllipsoid` of `SProcedure` strategies. It is discussed in subsection 8.3.5.

8.3.1 *From C code to PVS theorems*

The autocoder described in the previous Section generates two C functions. One of them is an initialization function, the other implements one execution of the loop that acquires inputs and updates the state variables and the outputs. It is left to the implementer to write the main function combining the two, putting the latter into a loop, and interfacing with sensors and actuators to provide inputs and deliver outputs. Nevertheless, the properties of open and closed loop stability, as well as state-boundedness, can be established by solely considering the update function, which this section now focuses on.

Let us consider the following annotated code:

Figure 8.10 Typical example of an ACSL HOARE Triple

C+ACSL

```
/*@
requires in_ellipsoidQ(QMat_4,
 vect_of_3_scalar(
   _state_->Integrator_1_memory,
   _state_->Integrator_2_memory,
   Integrator_1));
ensures in_ellipsoidQ(QMat_5,
 vect_of_4_scalar(
   _state_->Integrator_1_memory,
   _state_->Integrator_2_memory,
   Integrator_1,
   C11));
PROOF_TACTIC(use_strategy (AffineEllipsoid))
    ;
*/
\{
    C11 = 564.48 * Integrator_1;
\}
```

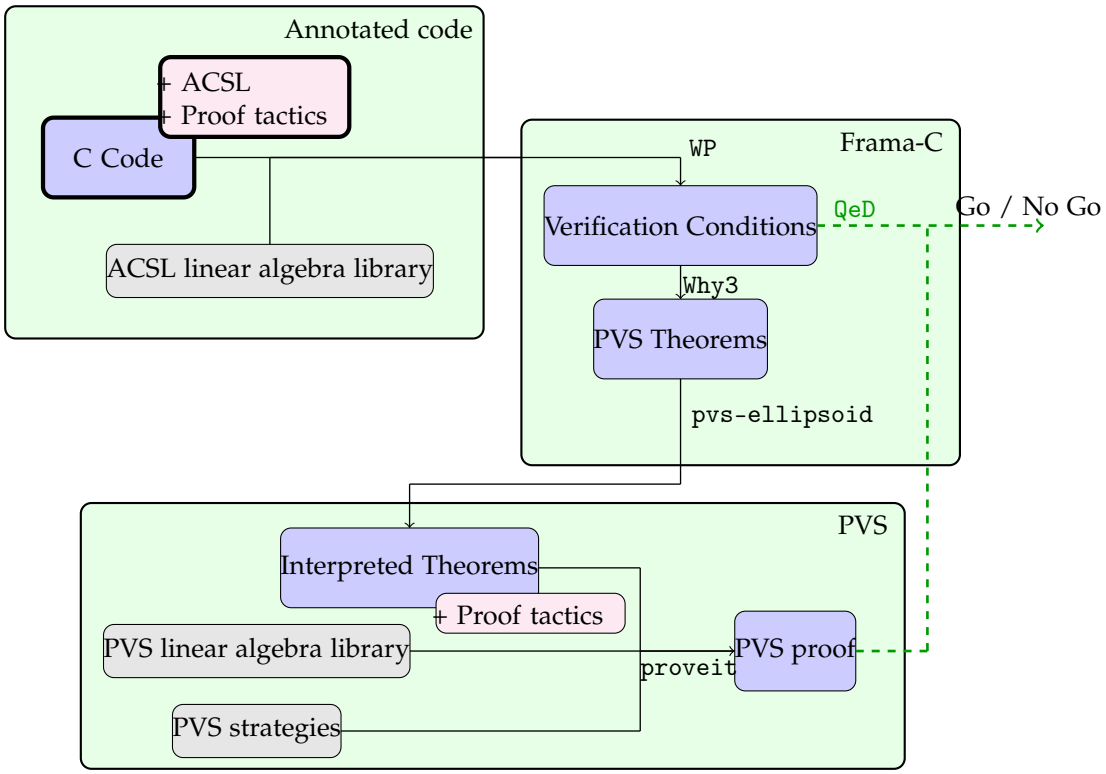
Frama-C/WP is able to characterize the weakest pre-condition of the ensures statement and to build the proof objective $pre \implies wp(code, post)$. Through the Why3 platform it is able to express it as a PVS theorem. For example, the ACSL/C triple shown in Figure 8.10, taken directly from our running example, becomes the theorem shown in Figure 8.11.

Figure 8.11 Excerpt of the PVS translation of the triple shown in Figure 8.10

PVS

```
wp: THEOREM
FORALL (integrator_1_0: real):
FORALL (malloc_0: [int -> int]):
FORALL (mflt_2: [addr -> real],
        mflt_1: [addr -> real],
        mflt_0: [addr -> real]):
FORALL (io_2: addr, io_1: addr,
        io_0: addr, state_2: addr,
        state_1: addr, state_0: addr):
...
=> p_in_ellipsoidq(l_qmat_4,
  l_vect_of_3_scalar(
    mflt_2(shift (state_2, 0)),
    mflt_2(shift (state_2, 1)),
    integrator_1_0))
=> p_in_ellipsoidq(l_qmat_5,
  l_vect_of_4_scalar(
    mflt_2(shift (state_2, 0)),
    mflt_2(shift (state_2, 1)),
    integrator_1_0,
    (14112/25 * integrator_1_0)))
```

Figure 8.9 General view of the automated verification process. The contribution of this section of the article lies in the domain specific libraries that have been developed at the different layers of description of the code, as well as in the generic proof strategies and the custom Frama-C plugin pvs-ellipsoid



Note that, for the sake of readability, part of the hypotheses of this theorem, including hypotheses on the nature of variables, as well as hypotheses stemming from Hoare triples present earlier in the code, are omitted here. Note also that in the translation process, functions like `malloc_0` or `mflt_1` have appeared. They describe the memory state of the program at different execution points.

C+ACSL

```
/*@ axiom mat_of_2x2_scalar_row:
  @ {\textbackslash}forall matrix A, real
      x0101, x0102, x0201, x0202;
  @ A == mat_of_2x2_scalar(x0101, x0102,
      x0201, x0202) ==>
  @ mat_row(A) == 2; /*
```

becomes, after translation to PVS:

PVS

```
q_mat_of_2x2_scalar_row:
 AXIOM FORALL (x0101_0:real, x0102_0:real,
     x0201_0:real, x0202_0:real):
         FORALL (a_0:a_matrix):
   (a_0 = l_mat_of_2x2_scalar(x0101_0,
       x0102_0, x0201_0, x0202_0)) =>
   (2 = l_mat_row(a_0))
```

8.3.2 *Theory interpretation*

At the ACSL level, a minimal set of linear algebra symbols has been introduced, along with axioms defining their semantics. Section 8.1.4 describes a few of them. Each generated PVS theorem is written within a theory that contains a translation 'as is' of these definitions and axioms, along with some constructs specific to handling the semantics of C programs. For example, the ACSL axiom expressing the number of rows of a 2 by 2 matrix:

In order to leverage the existing results on matrices and ellipsoids in PVS, theory interpretation is used. It is a logical technique used to relate one axiomatic theory to another. It is used here to map types introduced in ACSL, such as vectors and matrices, to their counterparts in PVS, as well as the operations and predicates on these types. To ensure soundness, PVS requires that what was

written as axioms in the ACSL library be proven in the interpreted PVS formalism.

The interpreted symbols and soundness checks are the same for each proof objective, facilitating the mechanization of the process. Syntactically, a new theory is created, in which the theory interpretation is carried out, and the theorem to be proven is automatically rewritten by PVS in terms of its own linear algebra symbols. These manipulations on the generated PVS code are carried out by a Frama-C plugin called pvs-ellipsoid, which is described below.

8.3.3 *Automatizing proofs in PVS*

Once the theorem is in its interpreted form, all that remains to do is to apply the proper lemma to the proper arguments. Since we now the theorems used to generate the annotation, as presented in Section 8.2, we provide PVS with strategies of theorems to apply. Without detailing all the issues met when dealing with PVS, its typecheck constraints (tccs) it introduces in the proof elements and the ability to identify arguments in a proof pattern, we develop a PVS strategy for each of our proof annotations, both for the affine combination of ellipsoids and the S-Procedure strategy.

8.3.4 *The `pvs-ellipsoid` plugin to Frama-C*

The pvs-ellipsoid plugin to Frama-C automatizes the steps mentioned in the previous subsections. It calls the WP plugin on the provided C file, then, whenever QeD fails to prove a step, it creates the PVS theorem for the verification condition through Why3 and modifies the generated code to apply theory interpretation. It extracts the proof tactic to be used on this specific verification condition, and uses it to signify to the next tool in the chain, `proveit`, what strategy to use to prove the theorem at hand. `proveit` is a command line tool that can be called on a PVS file and attempts to prove all the theories in it, possibly using user guidance such as the one just discussed. When the execution of `proveit` terminates, a report is produced, enabling the plugin to decide whether the verification condition is discharged or not. If it is, a proof file is generated, making it possible for the proof to be replayed in PVS.

8.3.5 *Checking inclusion of the propagated ellipsoid*

One final verification condition falls under neither `AffineEllipsoid` nor `S-Procedure` categories. It expresses that the state remains in the initial ellipsoid $\mathcal{G}_P$. Thanks to a number of transformations, we have proved that the state lies in some ellipsoid $\mathcal{G}'_P$. The conclusion of the verification lies in the final test $P - P' \geqslant 0$. The current state of the linear algebra library in PVS does not permit to make such a test, however a number of very reliable external tools, like the INTLAB package of the MATLAB software suite, can operate this check. In the case of our framework, the pvs-ellipsoid plugin intercepts this final verification condition before translating it to PVS, and uses custom code from [Rou+12] to ensure positive definiteness of the matrix, with the added benefit of soundness with respect to floating-point computations. These techniques are further developed in Chapter 10.

Part IV

NUMERICAL ISSUES

9

FLOATING-POINT SEMANTICS OF ANALYZED PROGRAMS

When it comes to implementation in a computer, one is limited by the finite bit-level representation of real numbers. A real number is then represented in the computer by a representation in a fixed number of bits. There exists mainly two families of such representations: fixed-point and floating-point representations. Their use largely depends on the application and industrial context and floating-point arithmetics is the main representation used in aerospace applications. This can however lead to strange non expected behaviors, see e.g. [Mono8] for detailed examples.

Until now all computations and formalizations have assumed real computations. In this chapter we revisit previous results and adapt them to account for numerical imprecision. A first part outlines floating-point semantics. A second part revisits previous results and adapts them to account for floating-point computations, assuming a bound on the rounding error is provided. A last part focuses on the approaches to bound these imprecisions, over-approximating the floating-point errors.

9.1 FLOATING-POINT SEMANTICS

9.1.1 *IEEE 754 floating-point representation*

The norm IEEE-754 defines the floating-point representation and operations. In the following we denote by *float* an IEEE 754 floating-point representation. Its implementation can minimally vary but, for the context of this work, we assume that the C level or machine level implementation faithfully respect the norm. On a fixed number of bits – typically 64bits – one can only represent, as IEEE-754 floating-point values, a finite (but large) number of real numbers. These values are defined as follows:

$$f = (-1)^s \cdot man \cdot 2^{exp} \tag{147}$$

where s is bit of sign, man the significand, also known as *mantissa*, relies on a bit representation over n bits of a positive integer, denoting a fractional part, and exp, the exponent, depends on a unsigned integer representation of k bits. In the following we denote by exp_b the k bits describing the exponent and by man_b the n bits describing the significand. We also denote by $[\![b]\!]^{ui}$ the unsigned integer interpretation of a bit word b.

As an unsigned integer, $[\![exp_b]\!]^{ui}$ ranges in $[0; 2^k - 1]$. Its bounds 0 and $2^k - 1$ denote special values: 0 is used to manipulate floats near zero, we speak about denormalized numbers, while $2^k - 1$ is used for infinities and not-a-number (NaN) values. For all other values of exp_b, the exponent is defined as $[\![exp_b]\!]^{ui} - 2^{k-1} + 1$; we speak about normalized numbers.

Let us first focus on these normalized numbers. In this case, the exponent lives within $[-2^{k-1}, 2^{k-1} + 1]$. In IEEE 754 normalized numbers significand are implicitly prefixed with a leading 1 bit. These $n + 1$ bits characterize the word $[\![1man_b]\!]^{ui} \in [2^n, 2^{n+1} - 1]$.

Then any normalized number can be written as $(-1)^s * [\![1man_b]\!]^{ui} * 2^{[\![exp_b]\!]^{ui} - 2^{k-1} - n + 1}$. The biggest representable value is then $(2^{n+1} - 1) * 2^{2^{k-1} - n + 1} = (1 - 2^{-n-1}) * 2^{2^{k-1}}$. For 32 bits simple precision float, $k = 8$ and $n = 24$, this gives the maximum value $(1 - 2^{-24}) * 2^{128}$. The smallest positive normal value is obtained with $man_b = 10\ldots0$ and $[\![exp_b]\!]^{ui} = 1$ which evaluates to $(2^n) * 2^{1 - 2^{k-1} + 1 - n} = 2^{2 - 2^{k-1}}$. With simple precision float, this is 2^{-126}.

In case of a tiny value near zero, a rounding error may also produce a false zero value. This is called an underflow. Denormalized numbers are a way to increase the number of floating-point values around zero. They occur when $[\![exp_b]\!]^{ui} = 0$; there is a 0 prefix for significand. In that case the number is written as $(-1)^s * [\![0man_b]\!]^{ui} * 2^{2 - 2^{k-1} - n}$. The smallest positive denormalized value is obtained with $man_b = 0\ldots01$: $2^{2 - 2^{k-1} - n}$. Let `eta` be such value; with simple precision float, this is $2^{2 - 128 - 23} = 2^{-149}$.

A useful notion is the unit-in-the-last-place (ulp): this function characterizes the distance between two consecutive floats and therefore the maximum imprecision caused by rounding errors. The ulp depends on the exponent part of the representation. For a normalized float value represented as $x = m * 2^e$, with $m = 1, \ldots$, the minimal distance between two floats is 2^{1-n+e} or 2^{-n+e}. So the ulp of 1 for simple precision is either $2^{-n} = 2^{-23}$ or $2^{1-n} = 2^{-22}$. For double precision, this

is 2^{-53} or 2^{-52}. Since it depends on the scale of the value, $\text{ulp}(x) = 2^{e-n} = 2^e * \text{ulp}(1) < |x| * \text{ulp}(1)$ and can then be directly over-approximated by $|x| * \text{ulp}(1)$. In the following we denote by eps such value $\text{ulp}(1)$.

The following table summarizes $\text{ulp}(1)$ and minimum positive values for single and double type of floats:

type	bits	k	n	eps $= \text{ulp}(1)$	eta
float	32	8	23	$\approx 10^{-7}$	$\approx 10^{-45}$
double	64	11	52	$\approx 10^{-16}$	$\approx 10^{-324}$

9.1.2 *Floating-point errors*

While computing in floats, rounding errors are introduced and accumulated.

Definition 9.1 *Let $\mathbb{F} \subset \mathbb{R}$ be the set of floating-point values and $\text{fl}(e) \in \mathbb{F}$ represents the floating-point evaluation of expression e with any rounding mode and any order of evaluation*[1]*.*

Example 27 *Depending on the order of evaluation, the value $\text{fl}(0.1 + 0.21 + 0.3)$ can be either $\text{round}(0.1 + \text{round}(0.21 + 0.3))$ or $\text{round}(\text{round}(0.1 + 0.21) + 0.3)$ with round any valid rounding mode (toward $+\infty$ or to nearest for instance), leading to different final values.*

Value error

Some simple values such as 0.1 are not exactly representable in floats since they cannot be expressed with a finite number of bits. Each use of a constant may therefore introduce basic errors. The error associated to the constant c is bounded by the minimal distance to the nearest floating-point value, and therefore bounded by $1/2\,\text{ulp}(x) < 1/2|x|\,\text{ulp}(1)$.

Numerical operations: addition and multiplication.

Similarly, each numerical operation introduces errors. For example, since the addition of two floats may not be exactly representable in floats, the result is rounded to a floating-point value, depending on the rounding mode.

Let $e^+(u, v)$ be such additive error. In case of u and v associated to their existing error e^u and e^v, the computation of $(u + e^u) + (v + e^v)$ returns $(u + v) + (e^u + e^v + e^+(u, v))$.

More practically it is possible to provide a bound for such error using eps:

Theorem 9.2 *Let $\texttt{eps} = \text{ulp}(1)$ is the precision of the floating-point format $\mathbb{F}$. We have for all $x, y \in \mathbb{F}$*

$$\begin{aligned} &\exists \delta \in \mathbb{R}, |\delta| \leqslant \texttt{eps} \\ &\text{fl}(x + y) = (1 + \delta)(x + y) \end{aligned} \tag{148}$$

Concerning multiplication, similar errors are introduced. We denote by $e^\times(u, v)$ such multiplicative error. When considering input with existing errors e^u and e^v, the computation of $(u + e^u) * (v + e^v)$ returns $(u * v) + (e^u * v + e^v * u + e^*(u, v))$.

Similarly, this error can be bounded using both the floating-point precision eps and the underflow constant eta[2]:

Theorem 9.3 *Let* eps *be the precision of the floating-point format $\mathbb{F}$ and* eta *the precision in case of underflows. In particular, we have for all $x, y \in \mathbb{F}$*

$$\begin{aligned} &\exists \delta, \eta \in \mathbb{R}, |\delta| \leqslant \texttt{eps} \wedge |\eta| \leqslant \texttt{eta} \\ &\text{fl}(x \times y) = (1 + \delta)(x \times y) + \eta. \end{aligned} \tag{149}$$

Remark 10 *Those are fairly classic notations and results [Hig96; Rum10].*

9.2 REVISITING INDUCTIVENESS CONSTRAINTS

Taking floating-point arithmetic into account, one needs to reevaluate the constraints used in the previous chapters to account for floating-point noise.

Let us be given a function that, provided bounds over input variables x, compute a safe-overapproximation of the floating-point error $\text{err}_f > 0$ when computing, in floating-point arithmetics, $\text{fl}(f(x))$.

The semantics of our discrete dynamical systems, as introduced in Section 4.1, were defined using a linear function $f(x, u) = Ax + Bu + b$, in Sections 4.1.1 and 4.1.2, or a polynomial function $T(x)$ for polynomial systems in Section 4.1.3. In addition, piecewise systems, linear or polynomial ones, were constrained by guards: $\bigwedge_i r^i(x) \leqslant b^i$ with r^i at most quadratic for piecewise linear systems.

9.2.1 LYAPUNOV *conditions with floats*

The LYAPUNOV condition (67), page 38 imposes that $V \circ f(x) - V(x) \leqslant 0$. When considering a floating-point implementation, we need to check the more constrained version $V \circ (\text{fl}(f(x)) - V(x) \leqslant 0$ considering an upper bound $\text{err}_{V \circ f}$ on the floating-point errors resulting from $\text{fl}(f(x))$ and propagated through function V, such that

$$(V \circ f(x) - V(x) + \text{err}_{V \circ f} \leqslant 0) \Rightarrow (V \circ (\text{fl}(f(x)) - V(x) \leqslant 0)$$

1 Order of evaluation matters since floating-point addition is not associative.

2 In case of simpler implementation of floating point numbers without denormalized numbers, the η term vanishes.

. Note that, since $V(x) = p(x)$ or $V(x) = x^\intercal P x$ are at least quadratic polynomials, the error obtained is not linear in the initial floating-point error of $\mathrm{fl}(f(x))$. Let $err_{V \circ f}$ be such error.

FLOATING-POINT ASSIGNMENTS. All inductiveness equations of Chapter 5 can be adapted to account for these floating-point errors: linear systems (69) and (70), preserving the shape (71), considering inputs (74) while bounding $f(x) = Ax + Bu$, with both x and u bounded; piecewise linear systems with their invariance constraint (87) where the term $-F^{i\intercal} \begin{pmatrix} 0 & q^j{}^\intercal \\ q^j & P^j \end{pmatrix} F^i$ shall be replaced by $-F^{i\intercal} \begin{pmatrix} 0 & q^j{}^\intercal \\ q^j & P^j \end{pmatrix} F^i - err_{F^{i\intercal} P^j F^i}$, and similarly for their k-inductive extension: Eqs (106) and (107); lastly, polynomial systems with the constraint $p - p \circ T^i$ in Eq. (116) are substituted by $p - p \circ T^i - err_{p \circ T^i}$[3].

FLOATING-POINT GUARDS. All methods proposed for piecewise systems integrate the condition in the constraints. Both quadratic constraints in the linear and more general polynomial ones are encoded as negativity constraints and introduced in the inductiveness constraint through a Lagrangian or SOS relaxation. Similarly to the assignment computed in floating-point arithmetic, the evaluation of the guard $r(x) \leqslant 0$ becomes $\mathrm{fl}(r(x)) \leqslant 0 \equiv r(x) \pm err_r \leqslant 0$ with err_r the actual floating point error. In case of a bound on variable x this quantity err_r can be over-approximated. Let $\overline{err_r}$ be such upper-bound. One can therefore encode the constraint by the stronger condition $r(x) - \overline{err_r} \leqslant 0$. If this condition holds, then the computation with floats of the guard $r(x) \leqslant 0$ will also hold. Note that, because of this stronger encoding, these sound guards do not characterize anymore a partitioning of the state space: multiple transitions could be computed from the same value. This would correspond to a non deterministic abstract of the initial deterministic partitioned dynamical system.

In both cases, assignments and guards, a coarse over-approximation of the floating-point error err will only over-constrain the result: more feasible transitions, and stricter inductiveness conditions.

9.2.2 *Policy iteration*

In the policy iteration algorithm presented in Chapter 6, the computation of an individual policy, in Eq. (119) and Eq. (121) for the SOS version, admits similar issues: $p \circ T^i$ should account for floating-point errors in the computation of T^i while $\langle \mu, r^i \rangle$ should consider floating-point errors in guards. Thanks to the available templates $q \in \mathbb{P}$, one can bound the variable x and over-approximate these two floating errors: $err_{p \circ T^i}$ and err_{r^i}. The constraint can be weakened on the guards side and strengthened on the assignment side:

$$\begin{aligned} \left(F_i^{\mathcal{R}}(w)\right)(p) = {} & p \circ T^i + err_{p \circ T^i} + \sum_{q \in \mathbb{P}} \lambda(q)(w(q) - q) \\ & - \langle \mu, r^i - err_{r^i} \rangle + \sigma \\ \text{and } \deg(\sigma) \leqslant 2m \deg T^i, & \deg(\langle \mu, r^i \rangle) \leqslant 2m \deg T^i \end{aligned} \quad . \tag{150}$$

9.2.3 *System-level analyses*

Closed-loop system analyses could be similarly analyzed. However, the plant part is not supposed to generate any floating-point error and could be omitted when computing the bound on the error. Analyses performed at code level (see Chapter 8) should compute separately the transformation of the ellipsoid invariants in Q-form, assuming a real semantics, and the approximation of the floating-point errors on the loop body. Then, when checking the final inductiveness of the final ellipsoid in P-form with respect to the initial one, the error has to be considered to further constrain the implication check.

Concerning robustness analysis and vector margins, as presented in Section 7.2, what we need is not exactly the inequality (142) describing the dissipativity condition but rather[4]

$$\mathrm{fl}(x_{k+1})^\intercal P \, \mathrm{fl}(x_{k+1}) - x_k^\intercal P x_k \leqslant \gamma^2 \|in_k\|_2^2 - \|\mathrm{fl}(e_k)\|_2^2.$$

Again, bounding by the positive upper bound ϵ the floating point error occurring when computing the linear update x_{k+1} evaluated through the LYAPUNOV function, one can express the stronger condition:

$$x_{k+1}{}^\intercal P \, x_{k+1} - x_k^\intercal P x_k + \epsilon \leqslant \gamma^2 \|in_k\|_2^2 - \|e_k\|_2^2.$$

Thus, instead of checking (143), we have to check

$$\begin{pmatrix} A_s{}^\intercal P A_s - P + C_s{}^\intercal C_s + \epsilon I & A_s{}^\intercal P B_s + C_s{}^\intercal D_s \\ B_s{}^\intercal P A_s + D_s{}^\intercal C_s & D_s{}^\intercal D_s + B_s{}^\intercal P B_s - \gamma^2 I + \epsilon I \end{pmatrix} \prec 0.$$

This robustness property is still difficult to prove since the inequality (142) is not absolute but relative to the difference between the norm 2 of the input and the error. The inequality should then hold for in_k and e_k as small as possible. However, because of finiteness of floating representation and underflow mechanism, the error they induce is no longer relative but absolute and we can only prove:

3 Recall that except the definitions of classical LYAPUNOV functions for pure linear systems, all our other convexification represent inductiveness as a positive constraints (rather than a negative one).

4 x_{k+1} and e_k both incur floating-point computations in the controller (*cf.* (140)) whereas in_k is just a real number.

$$\begin{aligned}&\mathrm{fl}(x_{k+1})^\intercal P\,\mathrm{fl}(x_{k+1}) - x_k^\intercal P x_k\\ &\leqslant \gamma^2\|in_k\|_2^2 - \|\mathrm{fl}(e_k)\|_2^2 + \eta\end{aligned}$$

where η is a constant, depending neither on x, nor on in. Thus, instead of (144), we get

$$\sum_{k=0}^{T}\|e_k\|_2^2 \leqslant \gamma^2\left(\sum_{k=0}^{T}\|in_k\|_2^2\right) + (T+1)\eta.$$

However, in practice η is tiny ($\eta \simeq 10^{-324}$) so that it can remain negligible in front of the input in as long as the number T of iterations of the system remains bounded (for instance, the flight commands of a plane typically operate at 100Hz and certainly no longer that 100 hours [Cou+05], meaning less than $T := 100\times 3600\times 100 \simeq 10^8$ iterations).

9.2.4 *Checking soundness of invariant with floating-point error noise*

In presence of simple linear updates and guards, assuming conditions on the dimension of the considered systems, and assuming a specific evaluation order, the error on the computation of linear update Ax can be bounded. We refer the reader to the work of Pierre Roux [Rou15; Rou13] for the characterization of such bounds and the formal proof the results in Coq.

However, in presence of a more complex dynamic, static analysis provide more flexible means to compute the floating-point error bound. Assuming such analysis is available, we propose the following algorithm to perform this invariant soundness check.

Figure 9.1 Algorithm to compute invariant soundness check with respect to floating-point semantics.

input : Dynamical system defined by function $f(x)$
output: Go/ NoGo
1 Perform a first analysis: synthesize non linear template;
2 Rely on policy iteration to compute a bound per local variable;
3 Over-approximate floating-point error on a single loop body evaluation: $\mathrm{err}_{\mathrm{body}}$;
4 Perform a second analysis: synthesize non linear template based on $f(x) + \mathrm{err}_{\mathrm{body}}$;
5 Rely on policy iteration to compute a bound per local variable;
6 Check inclusion in the initial bounds;

9.3 BOUND FLOATING-POINT ERRORS: TAYLOR-BASED ABSTRACTIONS AKA ZONOTOPIC ABSTRACT DOMAINS

Provided bounds on each variable, the floating-point error can be computed with classical interval-based analysis. Kleene based iterations with the interval abstract domain provide the appropriate framework to compute such bounds. In our setting this is even simpler since we are interested in bounding the floating-point error on a single call of our dynamic system transition function, that is a single loop body execution without internal loops.

9.3.1 *Interval based analysis*

Classical interval based abstract domains can be easily adapted to soundly over-approximate floating-point computation by rounding appropriately each operation:

Injection of a constant is expressed with the $\mathrm{ulp}(1)$ constant:

$$\alpha_{\mathcal{M}}(r) = [\mathrm{fl}(r) - |r| * \mathrm{ulp}(1), \mathrm{fl}(r) + |r| * \mathrm{ulp}(1)]$$

While compilers interpret directly constants through their floating point approximation, this safer interval contains the constant r that was written in the source code.

Note that it is difficult to detect whether a provided value is exactly representable within a given floating representation. The proposed encoding may introduce spurious noise with the $\pm|r| * \mathrm{ulp}(1)$ terms. For example the exact integer 1 will be encoded by $[r - \mathrm{ulp}(1), r + \mathrm{ulp}(1)]$. A more precise alternative could evaluate the obtained float and the other float obtained with a much larger floating-point representation such as a MPFR float over 1000 bits. In case of similar value, the number is likely (but not proven) to be exactly representable as a float.

Each numerical operation can be adapted to deal with floating-point rounding, rounding the operator to $-\infty$ for the lower bound and to $+\infty$ for the upper bound. We define by $\uparrow_\circ(x)$ the rounding of the real value x towards $\circ \in \{0, +\infty, -\infty, \sim\}$ denoting respectively the rounding towards zero, $+\infty$, $-\infty$ and to the nearest value.

Let us consider a binary operator $\Diamond$ and the resulting interval $[r_1, r_2] = [x_1, x_2]\Diamond[y_1, y_2]$ computed with a real semantics. A sound floating-point interval would then be $[\uparrow_{-\infty}(r_1), \uparrow_{+\infty}(r_2)]$.

The rounding error mode in C can be easily changed but would not be efficient if it has to be changed twice for each operation. One usually choose the rounding to $+\infty$ and rely on a negation to manipulate lower bounds. We rather compute $[-\uparrow_{-\infty}(-r_1), \uparrow_{+\infty}(r_2)]$.

For example, for addition we obtain

$$\begin{aligned}&[x_1, x_2] + [y_1, y_2] =\\ &[-\uparrow_{+\infty}(-x_1 - y_1), \uparrow_{+\infty}(x_2 + y_2)]\end{aligned} \tag{151}$$

Remember that unary negation only change the sign bit but do not introduce imprecision. In the following we will rely both on $\uparrow_{+\infty}(x)$ and $\uparrow_{-\infty}(x)$ in the notations, but recall that $\uparrow_{-\infty}(x)$ is implemented as $-\uparrow_{+\infty}(-x)$.

While the previous method compute a sound approximation of floating-point computation it does not enable the identification of the floating-point error part. An alternative abstraction would rely on rounding to the nearest, and represent a set of floating-point value by a pair of intervals $[f_1, f_2] + [e_1, e_2]$ where the first interval $[f_1, f_2]$ denotes the incorrect bounds obtained when manipulating floats as reals, and the second interval $[e_1, e_2]$ is used to carry on errors. In the following we denote such element by the 4-tuple (f_1, f_2, e_1, e_2). This new abstract domain is defined over the cartesian product of two lattices and therefore satisfies all abstract interpretation requirements for an abstract domain: structure lattice, existence of a Galois connection between subset of $\mathbb{R}$ and the domain, etc. We refer the reader to [Gou01; GP11; Mar05] for more details on means to bound these floating point accumulated rounding errors.

Let us recall the characterization of floating-point values for addition and multiplication presented in Section 9.1.2:

$$(u + e^u) + (v + e^v) = (u + v) + (e^u + e^v + e^+(u, v)) \tag{152}$$

$$(u + e^u) * (v + e^v) = (u * v) + (e^u * v + e^v * u + e^*(u, v)) \tag{153}$$

with $|e^+(u, v)| \leqslant |u + v|\,\texttt{eps}$ and $|e^*(u, v)| \leqslant |u * v|\,\texttt{eps} + \texttt{eta}$.

These equations can be adapted to pairs of intervals representation as detailed in Figure 9.2.

This method allows to characterize both the actual values obtained by floating-point computation in the value part and a safe error term. In case of a deterministic loopless code computing an expression exp, one would obtain the abstract value $[x, x] + [e_1, e_2]$ where the singleton interval for the value part denotes exactly the value x that would have been obtained when computing $\mathrm{fl}(\mathrm{exp})$. Thanks to the handling of floating-point errors the computation of exp with reals is guaranteed to lie within $[\uparrow_{-\infty}(x + e_1), \uparrow_{+\infty}(x + e_2)]$.

9.3.2 *Affine arithmetic*

Affine arithmetic was introduced in the 90s by COMBA and STOLFI [CS93] as an alternative to interval arithmetics, allowing to avoid some pessimistic computation like the cancellation:

$$x - x = [a, b] -_{\mathcal{I}} [a, b] = [a - b, b - a] \neq [0, 0]$$

It relies on a representation of convex subsets of $\mathbb{R}$ keeping dependencies between variables: e.g. $x \in [-1, 1]$ will be represented as $0 + 1 * \epsilon_1$ while another variable $y \in [-1, 1]$ will be represented by another ϵ term: $y = 0 + 1 * \epsilon_2$. Therefore $x - x$ will be precisely computed as $\epsilon_1 - \epsilon_1 = 0$ while $x - y$ will result in $\epsilon_1 - \epsilon_2$, i.e. denoting the interval $[-2, 2]$.

Affine arithmetics and variants of it have been studied in the area of applied mathematics community and global optimization. In global optimization, the objective is to precisely compute the minimum or maximum of a non convex function, typically using branch and bound algorithms. In most settings the objective function co-domain is $\mathbb{R}$ and interval or affine arithmetics allow to compute such bounds. Bisection, ie. branch and bound algorithm, improves the precision by considering subcases. The work of [MT06] introduced a quadratic extension of affine forms allowing, to express terms in $\epsilon_i \epsilon_j$.

In static analysis, affine forms lifted to abstract environments, as vectors of affine forms, are an interesting alternative to costly relational domains. They provide cheap and scalable relational abstractions: their complexity is linear in the number of error terms – the ϵ_i – while most relational abstract domains have a complexity at least cubic. Since their geometric concretization characterizes a zonotope, i.e. a symmetric convex polytope, they are commonly known as zonotopic abstract domains.

However, since zonotopes are not equipped with a lattice structure, their use in pure abstract interpretation using a Kleene iteration schema is not common. The definition of an abstract domain based on affine forms requires the definition of an upper bound and lower bound operators since no least upper bound and greatest lower bound exist in general. Choices vary from the computation of a precise minimal upper bound to a coarser upper bound that tries to maintain relationship among variables and error terms. For example, the choices of [GGP09] try to compute such bounds while preserving the error terms of the operands, as much as possible, providing a precise way to approximate a functional. In practice convergence of the fixpoint computation is not as easy to guarantee as it is for (join-)complete lattice. This is however not an issue in our context since we do not allow loops within the discrete dynamical system transfer function.

Affine arithmetics: affine forms.

An affine form is characterized by a pair $(c, (b)_m) \in \mathbb{R} \times \mathbb{R}^m$. It defines a map $a \in [-1, 1]^m \to \mathbb{R}$ such that

$$a(\epsilon) = c + b^\intercal \epsilon$$

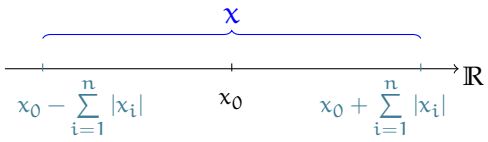
Figure 9.2 Addition and multiplication on intervals with floating-point errors.

$$(x_1,x_2,e_1,e_2)+(y_1,y_2,f_1,f_2)=\begin{pmatrix}\uparrow_\sim(x_1+y_1),\\ \uparrow_\sim(x_2+y_2),\\ \uparrow_{-\infty}(e_1+f_1-e^+(x_1,y_1)),\\ \uparrow_{+\infty}(e_2+f_2+e^+(x_2,y_2))\end{pmatrix} \tag{154}$$

$$(x_1,x_2,e_1,e_2)*(y_1,y_2,f_1,f_2)=$$
$$\begin{pmatrix}\min(\uparrow_\sim(x_1y_1),\uparrow_\sim(x_1y_2),\uparrow_\sim(x_2y_1),\uparrow_\sim(y_1y_2)),\\ \max(\uparrow_\sim(x_1y_1),\uparrow_\sim(x_1y_2),\uparrow_\sim(x_2y_1),\uparrow_\sim(y_1y_2)),\\ \uparrow_{-\infty}\begin{pmatrix}\min(\uparrow_{-\infty}(e_1y_1),\uparrow_{-\infty}(e_1y_2),\uparrow_{-\infty}(e_2y_1),\uparrow_{-\infty}(e_2y_2))\\ +\min(\uparrow_{-\infty}(x_1f_1),\uparrow_{-\infty}(x_1f_2),\uparrow_{-\infty}(x_2f_1),\uparrow_{-\infty}(x_2f_2))\\ -\min(e^*(x_1,y_1),e^*(x_1,y_2),e^*(x_2,y_1),e^*(x_2,y_2))\end{pmatrix},\\ \uparrow_{+\infty}\begin{pmatrix}\max(\uparrow_{+\infty}(e_1y_1),\uparrow_{+\infty}(e_1y_2),\uparrow_{+\infty}(e_2y_1),\uparrow_{+\infty}(e_2y_2))\\ +\min(\uparrow_{+\infty}(x_1f_1),\uparrow_{+\infty}(x_1f_2),\uparrow_{+\infty}(x_2f_1),\uparrow_{+\infty}(x_2f_2))\\ +\min(e^*(x_1,y_1),e^*(x_1,y_2),e^*(x_2,y_1),e^*(x_2,y_2))\end{pmatrix}\end{pmatrix} \tag{155}$$

where $e^+(a,b)$ is defined as $(|a|+|b|)\texttt{eps}$ and $e^*(a,b)$ as$|a*b|\ \texttt{eps}+\texttt{eta}$.

Figure 9.3 Range of an affine form.



In the following we denote by $\mathbf{A}=\mathbb{R}\times\mathbb{R}^n$ the set of affine forms. The variables $\epsilon_i\in[-1;1], i\in[1;n]$ are called the error terms.

Geometric interpretation.

An affine form $(c,(b)_n)$ denotes a subset of $\mathbb{R}$. We introduce a concretization function denoting the geometric interpretation of an affine form.

Definition 9.4 *Let $\gamma_{\mathbf{A}}:\mathbf{A}\to\wp(\mathbb{R})$ be the concretization function of $\mathbf{A}$ defined as:*

$$\begin{array}{rccl}\gamma_{\mathbf{A}}: & \mathbf{A} & \to & \wp(\mathbb{R})\\ & (c,(b)_n) & \mapsto & \left\{x\in\mathbb{R}\;\middle|\;\begin{array}{l}\exists\epsilon\in[-1;1]^n,\\ x=c+b^T\epsilon\end{array}\right\}\end{array}$$

When considering an affine form $a\in\mathbf{A}$, one can obtain the set of values $\gamma_{\mathbf{A}}\in\wp(\mathbb{R})$ it represents by considering only the absolute values of the coefficients, as illustrated in Fig. 9.3:

$$\gamma_{\mathbf{A}}(c,(b)_m)=\left[c-\Sigma_{i\in[1;m]}|b_i|\,;c+\Sigma_{i\in[1;m]}|b_i|\right]$$

Arithmetics.

The set $\mathbf{A}$ is fitted with arithmetic operators: addition, negation, multiplication and scalar multiplication.

Definition 9.5

$$\begin{array}{lcl}(c,(b)_n)+_{\mathbf{A}}(c',(b')_n) & = & (c+c',(b+b')_n)\\ -_{\mathbf{A}}(c,(b)_n) & = & (-c,(-b)_n)\\ (c,(b)_n)\times_{\mathbf{A}}(c',(b')_n) & = & (c\times c',(b'')_{n+1})\\ \quad \textit{where}\ \begin{cases}(b'')_n=c\times(b')_n+c'\times(b'')_n\\ b''_{n+1}=\sum_{i=1}^{n}\sum_{i=j}^{n}|b_i\times b'_j|\end{cases} & & \\ \lambda*_{\mathbf{A}}(c,(b)_n) & = & (\lambda\times c,\lambda\times(b)_n)\end{array}$$

Note that non linear operations introduce a new occurrence of an error term while others are exact.

One can inject an interval into an affine form by introducing a fresh error term ϵ_f:

Definition 9.6 *Let $x=[x^-,x^+]$ be an interval of $\mathbb{R}$. Let $\epsilon_f\in[-1;1]$ be a fresh error term. We define the affine form associated to x as:*

$$x=\frac{x^-+x^+}{2}+\frac{x^--x^+}{2}\epsilon_f$$

It characterizes the abstraction function $\alpha_{\mathbf{A}}:\mathbb{R}^2\to\mathbf{A}$:

$$\alpha_{\mathbf{A}}(x)=\left(\frac{x^-+x^+}{2},(\frac{x^--x^+}{2})_1\right)$$

Poset structure.

While hardly used in the global optimization community where affine forms are used, let us consider a partial order over affine sets. We rely on the geometrical interpretation given by the concretization function $\gamma_{\mathbf{A}}$, and fit the set $\mathbf{A}$ with a poset structure. We define the partial order $\sqsubseteq_{\mathbf{A}}$ as follows:

Definition 9.7 *We define the partial order* $\sqsubseteq_{\mathbf{A}}: \mathbf{A} \times \mathbf{A} \to \mathbb{B}$ *such that:*

$$\forall x, y \in \mathbf{A}, x \sqsubseteq_{\mathbf{A}} y \triangleq \gamma_{\mathbf{A}}(x) \subseteq \gamma_{\mathbf{A}}(y)$$

We also introduce a safe meet operator. Since $\langle \mathbf{A}, \sqsubseteq_{\mathbf{A}} \rangle$ is not fitted with a meet-complete structure, we cannot provide an exact meet operator computing the greatest lower bounds of two affine forms. However, we can rely on an abstract meet which provides a safe but imprecise upper bound of maximal lower bounds.

The following function performs such computation: it projects each affine form to its interval representation on which it performs the meet computation, before abstracting again to a fresh affine form.

Definition 9.8 *Let* $(c, (b)_n)$ *and* $(c', (b')_n) \in \mathbf{A}$. *We define the meet operator* $\sqcap_{\mathbf{A}} : \mathbf{A} \times \mathbf{A} \to \mathbf{A}$ *such that*

$$(c, (b)_n) \sqcap_{\mathbf{A}} (c', (b')_n) =$$

$$\begin{cases} (c, (b)_n) \text{ when } (c, (b)_n) = (c', (b')_n) \\ \alpha_{\mathbf{A}}(\gamma_{\mathbf{A}}(c, (b)_n) \cap \gamma_{\mathbf{A}}(c', (b')_n)) \text{ otherwise} \end{cases}$$

floating-point error computation

floating-point errors can be carried in identified error terms. floating-point error terms can be merged in order to make sure that the number of associated error terms does not increase significantly. However, some specific floating-point errors can be identified such as second order combinations of floating-point errors terms.

9.3.3 *Quadratic extension of zonotopes*

Quadratic forms.

A (not so) recent extension of affine arithmetics is quadratic arithmetics [MT06]. It is a comparable representation of values fitted with similar arithmetics operators but quadratic forms also considers products of two errors terms $\epsilon_i \epsilon_j$. A quadratic form is also parameterized by additional error terms used to encode non linear errors: $\epsilon_\pm \in [-1, 1], \epsilon_+ \in [0, 1]$ and $\epsilon_- \in [-1, 0]$. Let us define the set $\mathbf{C}^m \triangleq [-1, 1]^m \times [-1, 1] \times [0, 1] \times [-1, 0]$. A quadratic form on m noise symbols is a function q from $\mathbf{C}^m$ to $\mathbb{R}$ defined for all $t = (\epsilon, \epsilon_\pm, \epsilon_+, \epsilon_-) \in \mathbf{C}^m$ by $q(t) = c + b^\intercal \epsilon + \epsilon^\intercal A \epsilon + c_\pm \epsilon_\pm + c_- \epsilon_- + c_+ \epsilon_+$. The A term will generate the quadratic expressions in $\epsilon_i \epsilon_j$. A quadratic form is thus characterized by a 6-tuple $(c, (b)_m, (A)_{m^2}, c_\pm, c_+, c_-) \in \mathbb{R} \times \mathbb{R}^m \times \mathbb{R}^{m \times m} \times \mathbb{R}_+ \times \mathbb{R}_+ \times \mathbb{R}_+$. Without loss of generality, the matrix A can be assumed symmetric. To simplify, we will use the terminology quadratic form for both the function defined on $\mathbf{C}^m$ and the 6-tuple.

Let $\mathcal{Q}^m$ denote the set of quadratic forms.

Geometric interpretation.

Let $q \in \mathcal{Q}^m$. Since q is continuous, the image of $\mathbf{C}^m$ by q is a closed bounded interval. In our context, the image of $\mathbf{C}^m$ by q defines its geometric interpretation.

Definition 9.9 *The concretization map of a quadratic form* $\gamma_{\mathcal{Q}} : \mathcal{Q}^m \to \wp(\mathbb{R})$ *is defined by:*

$$\gamma_{\mathcal{Q}}(q) = \{x \in \mathbb{R} \mid \exists t \in \mathbf{C}^m \text{ s.t. } x = q(t)\}$$

Remark 11 *We can have* $\gamma_{\mathcal{Q}}(q) = \gamma_{\mathcal{Q}}(q')$ *with* $q \neq q'$ *e.g.* $q = \epsilon_1^2$ *and* $q' = \epsilon_2^2$. *Therefore,* $\gamma_{\mathcal{Q}}$ *could not characterize an antisymmetric relation on* $\mathcal{Q}^m$ *and therefore not a partial order. We could consider equivalence classes instead to get an order but we would loose the information that* q_1 *and* q_2 *are not correlated.*

The projection of q to intervals consists in computing the infimum and the supremum of q over $\mathbf{C}^m$ i.e. the values:

$$\mathbf{b}^q \triangleq \inf\{q(x) \mid x \in \mathbf{C}^m\} \tag{156}$$

$$\mathbf{B}^q \triangleq \sup\{q(x) \mid x \in \mathbf{C}^m\} \tag{157}$$

Computing $\mathbf{b}^q$ and $\mathbf{B}^q$ is reduced to solving a non-convex quadratic problem which is NP-hard [Vav90]. The approach described in [MT06] uses simple inequalities to give a safe over-approximation of $\gamma_{\mathcal{Q}}(q)$. The interval provided by this approach is $[\mathbf{b}^q_{MT}, \mathbf{B}^q_{MT}]$ which is defined as follows:

$$\begin{cases} \mathbf{b}^q_{MT} \triangleq & c - \sum_{i=1}^{m} |b_i| - \sum_{\substack{i,j=1,\dots,m \\ j \neq i}} |A_{ij}| + \sum_{i=1}^{m} [A_{ii}]^- \\ & -c_- - c_\pm \\ \mathbf{B}^q_{MT} \triangleq & c + \sum_{i=1}^{m} |b_i| + \sum_{\substack{i,j=1,\dots,m \\ j \neq i}} |A_{ij}| + \sum_{i=1}^{m} [A_{ii}]^+ \\ & +c_+ + c_\pm \end{cases} \tag{158}$$

where for all $x \in \mathbb{R}$, $[x]^+ = x$ if $x > 0$ and 0 otherwise and $[x]^- = x$ if $x < 0$ and 0 otherwise.

In practice, we use the interval projection operator $\mathcal{P}^{MT}_{\mathcal{Q}}(q) \triangleq [\mathbf{b}^q_{MT}, \mathbf{B}^q_{MT}]$ instead of $\gamma_{\mathcal{Q}}(q)$, since $\gamma_{\mathcal{Q}}(q) \subseteq \gamma_{\mathcal{I}}\left(\mathcal{P}^{MT}_{\mathcal{Q}}(q)\right)$ where $\gamma_{\mathcal{I}}$ denotes the concretization of intervals. In [AGW15], we presented a tighter over-approximation of $\gamma_{\mathcal{Q}}(q)$ using SDP.

We will need a "reverse" map to the concretization map $\gamma_{\mathcal{Q}}$: a map which associates a quadratic form to an interval. We call this map the *abstraction map*. Note that the abstraction map produces a fresh noise symbol.

First, we introduce some notations for intervals. Let $\mathcal{I}$ be the set of closed bounded real intervals i.e. $\{[a,b] \mid a,b \in \mathbb{R}, a \leqslant b\}$ and $\bar{\mathcal{I}}$ its unbounded extension, i.e. $a \in \mathbb{R} \cup \{-\infty\}, b \in \mathbb{R} \cup \{+\infty\}$. $\forall [a,b] \in \mathcal{I}$, we define two functions $\lg([a,b]) = (b-a)/2$ and $\mathrm{mid}([a,b]) = (b+a)/2$. Let $\sqcup_{\mathcal{I}}$ be the classic join of $\mathcal{I}$ that is $[a,b] \sqcup_{\mathcal{I}} [c,d] \triangleq [\min(a,c), \max(b,d)]$. Let $\sqcap_{\bar{\mathcal{I}}}$ be the classic meet of intervals.

Definition 9.10 *The abstraction map $\alpha_{\mathcal{Q}} : \mathcal{I} \to \mathcal{Q}^1$ is defined by:*

$$\alpha_{\mathcal{Q}}([a_1,a_2]) = (c, (b)_1, (0)_1, 0, 0, 0)$$

where $c = \mathrm{mid}([a_1,a_2])$ and $b = \lg([a_1,a_2])$.

Property 9.11 (Concretization of abstraction)

$$\gamma_{\mathcal{Q}}(\alpha_{\mathcal{Q}}([a_1,a_2])) \supseteq [a_1,a_2]$$

Arithmetic operators.

Quadratic forms are equipped with arithmetic operators whose complexity is quadratic in the number of error terms. We give here the definitions of the arithmetics operators:

Definition 9.12 *Addition, negation, multiplication by scalar are defined by:*

$$\begin{aligned} &(c,(b)_m,(A)_{m^2},c_\pm,c_+,c_-) \\ &+_{\mathcal{Q}}(c',(b')_m,(A')_{m^2},c'_\pm,c'_+,c'_-) = \\ &\left(\begin{array}{l} c+c',(b+b')_m,(A+A')_{m^2}, \\ c_\pm+c'_\pm,c_++c'_+,c_-+c'_- \end{array}\right) \end{aligned} \tag{159}$$

$$\begin{aligned} &-_{\mathcal{Q}}(c,(b)_m,(A)_{m^2},c_\pm,c_+,c_-) = \\ &(-c,(-b)_m,(-A)_{m^2},c_\pm,c_-,c_+) \end{aligned} \tag{160}$$

$$\begin{aligned} &\lambda *_{\mathcal{Q}} (c,(b)_m,(A)_{m^2},c_\pm,c_+,c_-) = \\ &(\lambda c,\lambda(b)_m,\lambda(A)_{m^2},|\lambda|c_\pm,|\lambda|c_+,|\lambda|c_-) \end{aligned} \tag{161}$$

The multiplication is more complex since it introduces additional errors.

$$\begin{aligned} &(c,(b)_m,(A)_{m^2},c_\pm,c_+,c_-) \\ &\times_{\mathcal{Q}}(c',(b')_m,(A')_{m^2},c'_\pm,c'_+,c'_-) = \\ &\left\{\begin{array}{l} \left(\begin{array}{l} cc',c'(b)_m+c(b')_m, \\ c'(A)_{m^2}+c(A')_{m^2}+(b)_m(b')^{\mathsf{T}}_m, \\ c''_\pm,c''_+,c''_- \end{array}\right) \text{ with} \\ c''_x = c''_{x_1}+c''_{x_2}+c''_{x_3}+c''_{x_4}, \forall x \in \{+,-,\pm\} \end{array}\right. \end{aligned} \tag{162}$$

Each c''_{x_i} accounts for multiplicative errors with more than quadratic degree, obtained in the following four sub terms: (1) $\epsilon^{\mathsf{T}}A\epsilon \times \epsilon^{\mathsf{T}}A'\epsilon$ (2) $b^{\mathsf{T}}\epsilon \times \epsilon^{\mathsf{T}}A'\epsilon$ and $b'^{\mathsf{T}}\epsilon \times \epsilon^{\mathsf{T}}A\epsilon$ (3) multiplication of a matrix element in A, A' times an error term in $\pm,+,-$ (4) multiplication between error terms or with constant c, c'. Their precise definition can be found in [MT06, §3].

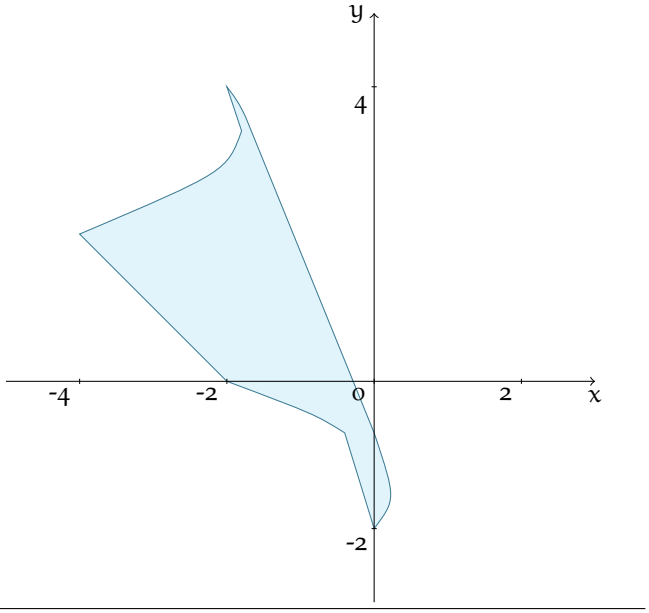
Quadratic Zonotopes: a zonotopic extension of quadratic forms to environments

Quadratic vectors are the lift to environments of quadratic forms. They provide a p-dimensional environment in which each dimension/variable is associated to a quadratic form. As for the affine sets used in zonotopic domains [GP09], the different variables share (some) error terms, this characterizes a set of relationships between variables, when varying the values of ϵ within $[-1,1]^m$. The geometric interpretation of quadratic vectors are non convex non symmetric subsets of $\mathbb{R}^p$. In the following, we call them Quadratic Zonotopes to preserve the analogy with affine sets and zonotopes.

Example 28 (quadratic vector) *Let us consider the following quadratic vector q:*

$$q = \begin{cases} x = -1 + \epsilon_1 - \epsilon_2 - \epsilon_{1,1} \\ y = 1 + 2\epsilon_2 + \epsilon_{1,2} \end{cases}$$

Figure 9.4 Zonotopic concretization of the quadratic vector $q \in \mathcal{Z}^p_{\mathcal{Q}^m}$ of Ex. 28: $\gamma_{\mathcal{Z}_{\mathcal{Q}}}(q)$



Note that it corresponds to the following vector of tuples defined over the sequence (ϵ_1, ϵ_2) of error terms:

$$\begin{cases} x = \left(-1, (1,-1)^\intercal, \begin{pmatrix} -1 & 0 \\ 0 & 0 \end{pmatrix}, 0, 0, 0\right) \\ y = \left(1, (0,2)^\intercal, \begin{pmatrix} 0 & 1/2 \\ 1/2 & 0 \end{pmatrix}, 0, 0, 0\right) \end{cases}$$

Fig. 9.4 represents its associated geometric interpretation, a quadratic zonotope.

Let $\mathcal{Z}^p_{\mathbb{Q}^m}$ be the set of quadratic vectors of dimension p: $(q^p) \in \mathcal{Z}^p_{\mathbb{Q}^m} = \left(c^p, (b)^p_m, (A)^p_{m^2}, c^p_\pm, c^p_+, c^p_-\right) \in \mathbb{R}^p \times \mathbb{R}^{p\times m} \times \mathbb{R}^{p\times m\times m} \times \mathbb{R}^p_+ \times \mathbb{R}^p_+ \times \mathbb{R}^p_+$.

The Zonotope domain is then a parametric relational abstract domain, parameterized by the vector of m error terms. In practice, its definition mimics a non relational domain based on an abstraction $\mathcal{Z}^p_{\mathbb{Q}^m}$ of $\wp(\mathbb{R}^p)$. Operators are (i) assignment of a variable of the zonotope to a new value defined by an arithmetic expression, using the semantics evaluation of expressions in $\mathbb{Q}$ and the substitution in the quadratic vector; (ii) guard evaluation, i.e. constraint over a zonotope, using the classical combination of forward and backward evaluations of expressions [Min04, §2.4.4].

1. Geometric interpretation and box projection. One can consider the geometric interpretation as the concretization of a quadratic vector to a quadratic zonotope.

   From now on, for all $n \in \mathbb{N}$, $[n]$ denotes the set of integers $\{1, \dots, n\}$.

   Definition 9.13 *The concretization map* $\gamma_{\mathcal{Z}_\mathbb{Q}} : \mathcal{Z}^p_{\mathbb{Q}^m} \mapsto \wp(\mathbb{R}^p)$ *is defined for all* $q = (q_1, \dots, q_p) \in \mathcal{Z}^p_{\mathbb{Q}^m}$ *by:*

   $$\gamma_{\mathcal{Z}_\mathbb{Q}}(q) = \left\{ x \in \mathbb{R}^p \;\middle|\; \begin{array}{l} \exists t \in \mathbf{C}^m \text{ s.t.} \\ \forall k \in [p],\ x_k = q_k(t) \end{array} \right\} .$$

   Remark 12 *Characterizing such subset of* $\mathbb{R}^p$ *explicitly as a set of constraints is not easy. A classical (affine) zonotope is the image of a polyhedron (hypercube) by an affine map, hence it is a polyhedron and can be represented by a conjunction of affine inequalities. For quadratic vectors a representation in terms of conjunction of quadratic or at most polynomial inequalities is not proven to exist*[5]*. This makes the concretization of a quadratic set difficult to compute precisely.*

   To ease the latter interpretation of computed values, we rely on a naive projection to boxes: each quadratic form of the quadratic vector is concretized as an interval using $\gamma_\mathbb{Q}$.

2. Preorder structure.

   We equip quadratic vectors with a preorder relying on the geometric inclusion provided by the map $\gamma_{\mathcal{Z}_\mathbb{Q}}$.

   Definition 9.14 *The preorder* $\sqsubseteq_{\mathcal{Z}_\mathbb{Q}}$ *over* $\mathcal{Z}^p_{\mathbb{Q}^m}$ *is defined by:*

   $$x \sqsubseteq_{\mathcal{Z}_\mathbb{Q}} y \iff \gamma_{\mathcal{Z}_\mathbb{Q}}(x) \subseteq \gamma_{\mathcal{Z}_\mathbb{Q}}(y) .$$

   Remark 13 *Since* $\gamma_{\mathcal{Z}_\mathbb{Q}}$ *is not computable,* $x \sqsubseteq_{\mathcal{Z}_\mathbb{Q}} y$ *is not decidable. Note also that, from Remark 11, the binary relation* $\sqsubseteq_{\mathcal{Z}_\mathbb{Q}}$ *cannot be antisymmetric and thus cannot be an order.*

   Remark 14 *The least upper bound of* $Z \subseteq \mathcal{Z}^p_{\mathbb{Q}^m}$ *i.e. an element* z' *s.t.*

   $$\begin{pmatrix} \forall z \in Z, z \sqsubseteq_{\mathcal{Z}_\mathbb{Q}} z' \wedge \\ \forall z \in Z, \forall z'' \in \mathcal{Z}^p_{\mathbb{Q}^m} \\ z \sqsubseteq_{\mathcal{Z}_\mathbb{Q}} z'' \end{pmatrix} \implies z' \sqsubseteq_{\mathcal{Z}_\mathbb{Q}} z''$$

   does not necessarily exist.

   Related work [GGP10; GGP09; GLP12; GP09; GPV12] addressed this issue by providing various flavors of join operator computing a safe upper bound or a minimal upper bound. The classical Kleene iteration scheme was adapted[6] to fit this loose framework without (efficient) least upper bound computation. Note that, in general, the aforementioned zonotopic domains do not rely on the geometric interpretation as the concretization to $\wp(\mathbb{R})$.

   We now detail a join operator used in our implementation. It is the lifting of the operator proposed in [GP09] to quadratic vectors. The motivation of this operator is to provide an upper bound while minimizing the set of error terms lost in the computation.

   First we introduce a useful function *argmin*: it cancels values of opposite sign but provides the argument with the minimal absolute value when provided with two values of the same sign:

   Definition 9.15 *We define for all* $a \in \mathbb{R}$, $\mathrm{sgn}(a) = 1$ *if* $a \geqslant 0$ *and -1 otherwise. The* argmin *function,* $\mathrm{argmin} : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ *is defined as:* $\forall a, b \in \mathbb{R}, \mathrm{argmin}(a, b) = \mathrm{sgn}(a) \min(|a|, |b|)$ *if* $ab \geqslant 0$ *and o otherwise.*

5 While it can expressed as a combination of conjunctions and disjunctions of quadratic inequalities. For very simple examples this can be computed through quantifier eliminations with cylindrical algebraic decompositions. The complexity and therefore efficiency is however terrible.

6 Typically this involves a large number of loop unrolling, trying to minimize the number of actual uses of join/meet.

We also need the projection map which selects a specific coordinate of a quadratic vector.

Definition 9.16 $\forall k \in [p]$, *the family of projection maps* $\pi_k : \mathcal{Z}^p_{\mathcal{Q}^m} \to \mathcal{Q}^m$ *is defined by:* $\forall q = (q_1, \ldots, q_p) \in \mathcal{Z}^p_{\mathcal{Q}^m}$, $\pi_k(q) = q_k$.

When a quadratic form q is defined before a new noise symbol is created, we have to extend q to take into account this fresh noise symbol. We introduce an extension map operator that extend the size of the error term vector considered. Informally, $\mathrm{ext}_{i,j}(q)$ adds i null error terms at the beginning of the error term vector and j at its tail, while keeping the existing symbols in the middle.

Definition 9.17 *Let* $i, j \in \mathbb{N}$. *The extension map* $\mathrm{ext}_{i,j} : \mathcal{Q}^m \to \mathcal{Q}^{i+j+m}$ *is defined by:* $\forall q = (c, (b)_m, (A)_{m^2}, c_\pm, c_+, c_-) \in \mathcal{Q}^m, \mathrm{ext}_{i,j}(q) = (c, (b')_{i+j+m}, (A')_{(i+j+m)^2}, c_\pm, c_+, c_-) \in \mathcal{Q}^{i+j+m}$ *where* $b'_k = b_{k-i}$ *if* $i+1 \leqslant k \leqslant m+i$ *and o otherwise and* $A'_{k,l} = A_{k-i,l-i}$ *if* $i+1 \leqslant k, l \leqslant m+i$ *and o otherwise.*

Property 9.18 (Extension properties) *Let* $i, j \in \mathbb{N}$.

a) Let $t = (\epsilon, \epsilon_\pm, \epsilon_+, \epsilon_-) \in \mathbf{C}^m$ *and* $t' = (\epsilon', \epsilon_\pm, \epsilon_+, \epsilon_-) \in \mathbf{C}^{m+i+j}$ s.t. $\forall i+1 \leqslant k \leqslant m+i$, $\epsilon'_k = \epsilon_{k-i}$. *Then* $q(\epsilon', \epsilon_\pm, \epsilon_+, \epsilon_-) = \mathrm{ext}_{i,j}(q)(\epsilon, \epsilon_\pm, \epsilon_+, \epsilon_-)$.

b) For all $q \in \mathcal{Q}^m$, $\gamma_{\mathcal{Q}}(q) = \gamma_{\mathcal{Q}}(\mathrm{ext}_{i,j}(q))$.

Now, we can give a formal definition of the upper bound of two quadratic vectors.

Definition 9.19 *The upper bound* $\sqcup_{\mathcal{Z}_\mathcal{Q}} : \mathcal{Z}^p_{\mathcal{Q}^m} \times \mathcal{Z}^p_{\mathcal{Q}^m} \to \mathcal{Z}^p_{\mathcal{Q}^{m+p}}$ *is defined, for all* $q = (c, b, A, c_\pm, c_+, c_-), q' = (c', b', A', c'_\pm, c'_+, c'_-) \in \mathcal{Z}^p_{\mathcal{Q}^m}$ *by:*

$$q \sqcup_{\mathcal{Z}_\mathcal{Q}} q' = \left(\mathrm{ext}_{0,p}(q''_k)\right)_{k\in[p]} + q^e \in \mathcal{Z}^p_{\mathcal{Q}^{m+p}}$$

where $q'' = (c'', (b'')^p_m, (A'')^p_{m^2}, c''^p_\pm, c''^p_+, c''^p_-) \in \mathcal{Z}^p_{\mathcal{Q}^m}$ *with, for all* $k \in [p]$*:*

- $(c'')_k = \mathrm{mid}(\gamma_\mathcal{Q}(\pi_k(q)) \cup \gamma_\mathcal{Q}(\pi_k(q')))$;
- $\forall t \in \{\pm, +, -\}, c''_{t,k} = \mathrm{argmin}(c_{t,k}, c'_{t,k})$;
- $\forall i \in [m]$, $(b'')_{k,i} = \mathrm{argmin}(b_{k,i}, b'_{k,i})$;
- $\forall i, j \in [m]$, $(A'')_{k,i,j} = \mathrm{argmin}(A_{k,i,j}, A'_{k,i,j})$;

and $\forall k \in [p]$, $q^e_k = \mathrm{ext}_{(m+k-1),(p-k)}\left(\alpha_\mathcal{Q}\left(C_k \sqcup_\mathcal{J} C'_k\right)\right)$ *with* $C_k = \gamma_\mathcal{Q}(\pi_k(q) - \pi_k(q''))$ *and* $C'_k = \gamma_\mathcal{Q}(\pi_k(q') - \pi_k(q''))$.

Let us denote the Minkowski sum and the cartesian product of sets, respectively, by $D_1 \oplus D_2 = \{d_1 + d_2 \mid d_1 \in D_1, d_2 \in D_2\}$ and $\prod_i^n D_i = \{(d_1, \ldots, d_n) \mid \forall i \in [n], d_i \in D_i\}$. We have the nice characterization of the concretization of the upper bound given by Lemma 9.20.

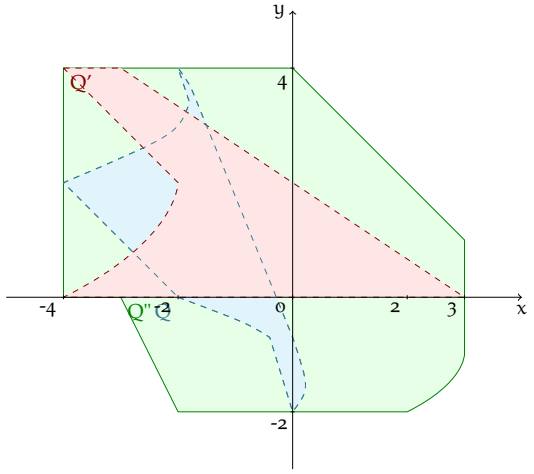
Lemma 9.20 *By construction of* q'' *and* q^e *previously defined:*

$$\gamma_{\mathcal{Z}_\mathcal{Q}}\left(\left(\mathrm{ext}_{0,p}(q''_k)\right)_{k\in[p]} + q^e\right) = \gamma_{\mathcal{Z}_\mathcal{Q}}(q'') \oplus \prod_{k=1}^{p} \gamma_{\mathcal{Q}^{m+p}}(q^e_k)$$

Now, we state in Theorem 9.21 that the $\sqcup_{\mathcal{Z}_\mathcal{Q}}$ operator computes an upper bound of its operands with respect to the preorder $\sqsubseteq_{\mathcal{Z}_\mathcal{Q}}$.

Theorem 9.21 *For all* $q, q' \in \mathcal{Z}^p_{\mathcal{Q}^m}$, $q \sqsubseteq_{\mathcal{Z}_\mathcal{Q}} q \sqcup_{\mathcal{Z}_\mathcal{Q}} q'$ *and* $q' \sqsubseteq_{\mathcal{Z}_\mathcal{Q}} q \sqcup_{\mathcal{Z}_\mathcal{Q}} q'$.

Figure 9.5 Upper bound computation



Example 29 *Let* Q *and* Q' *be two quadratic vectors:*

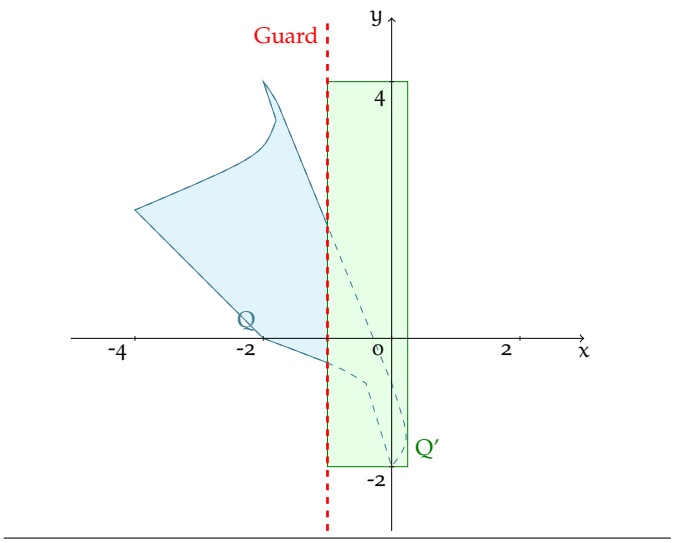
$$Q = \begin{cases} x = -1 + \epsilon_1 - \epsilon_2 - \epsilon_{1,1} \\ y = 1 + 2\epsilon_2 + \epsilon_{1,2} \end{cases}$$

$$Q' = \begin{cases} x = -2\epsilon_2 - \epsilon_{1,1} + \epsilon_+ \\ y = 1 + \epsilon_1 + \epsilon_2 + \epsilon_{1,2} \end{cases}$$

The resulted quadratic vector $Q'' = Q \sqcup_{\mathcal{Z}_\mathcal{Q}} Q'$ *is*

$$Q'' = \begin{cases} x = -\epsilon_2 - \epsilon_{1,1} + 2\epsilon_3 \\ y = 1 + \epsilon_2 + \epsilon_{1,2} + \epsilon_4 \end{cases}$$

Figure 9.6 Guard evaluation



3. Transfer functions.

   The two operators `guard` and `assign` over the expressions `RelExpr` and `Expr` are defined like in a non relational abstract domain, as described in [Mino4, §2.4.4]. Each operator relies on the forward semantics of numerical expressions, computed within arithmetics operators in $\mathcal{Q}$:

   Definition 9.22 *Let $\mathcal{V}$ be a finite set of variables. Let $[\![\cdot]\!]_{\mathcal{Q}}(\mathcal{V} \to \mathcal{Q}) \to \mathcal{Q}$ be the semantics evaluation of an expression in an environment mapping variables to quadratic forms.*

$$\begin{aligned} [\![v]\!]_{\mathcal{Q}}(Env) &= \pi_k(Env) \text{ where } k \in [p] \\ &\quad \text{is the index of } v \in \mathcal{V} \text{ in } Env \\ [\![e_1 \text{ bop } e_2]\!]_{\mathcal{Q}}(Env) &= [\![e_1]\!]_{\mathcal{Q}}(Env) \\ &\quad \text{bop}_{\mathcal{Q}} \ [\![e_2]\!]_{\mathcal{Q}}(Env) \\ [\![\text{uop } e]\!]_{\mathcal{Q}}(Env) &= \text{uop}_{\mathcal{Q}}[\![e]\!]_{\mathcal{Q}}(Env) \end{aligned}$$

   Guards, i.e. tests, are enforced through the classical combination of forward and backward operators. Backward operators are the usual fallback operators, e.g. $[\![x + y]\!]^{\leftarrow} = (x \sqcap_{\mathcal{Q}} ([\![x+y]\!] -_{\mathcal{Q}} y), y \sqcap_{\mathcal{Q}} ([\![x+y]\!] -_{\mathcal{Q}} x))$ where $\sqcap_{\mathcal{Q}}$ denotes the meet of quadratic forms. As for upper bound computation, no best lower bound exists and such meet operator in $\mathcal{Q}$ has to compute a safe but imprecise upper bound of maximal lower bounds.

   The meet over $\mathcal{Q}^m$ works as follows: it projects each argument to intervals using $\gamma_{\mathcal{Q}}$, (i) performs the meet computation and (ii) reinjects the resulting closed bounded interval to $\mathcal{Q}$ using $\alpha_{\mathcal{Q}}$, (iii) introducing it through a fresh noise symbol.

   The meet over $\mathcal{Z}^p_{\mathcal{Q}^m}$ is defined as the lift of$\mathcal{Q}^m$ meet to quadratic vectors. Formally:

   Definition 9.23 *The meet $\sqcap_{\mathcal{Q}} : \mathcal{Q}^m \times \mathcal{Q}^m \to \mathcal{Q}^1$ is defined by:*

$$\forall x, y \in \mathcal{Q}^m, x \sqcap_{\mathcal{Q}} y \triangleq \alpha_{\mathcal{Q}} (\gamma_{\mathcal{Q}}(x) \sqcap_{\mathcal{I}} \gamma_{\mathcal{Q}}(y)).$$

   The meet $\sqcap_{\mathcal{Z}_{\mathcal{Q}}} : \mathcal{Z}^p_{\mathcal{Q}^m} \times \mathcal{Z}^p_{\mathcal{Q}^m} \to \mathcal{Z}^p_{\mathcal{Q}^p}$ is defined, for all $x, y \in \mathcal{Z}^p_{\mathcal{Q}^m}$ by $z = x \sqcap_{\mathcal{Z}_{\mathcal{Q}}} y \in \mathcal{Z}^p_{\mathcal{Q}^p}$ where:

$$\forall i \in [p], z_i = \pi_i(x) \sqcap_{\mathcal{Q}} \pi_i(y)$$

when $\pi_i(x) \neq \pi_i(y), \pi_i(x)$ otherwise.

Example 30 *Let Q be the following quadratic vector. The meet with the constraint $x + 1 \geqslant 0$ produces the resulting quadratic vector Q':*

$$Q = \begin{cases} x = -1 + \epsilon_1 - \epsilon_2 - \epsilon_{1,1} \\ y = 1 + 2\epsilon_2 + \epsilon_{1,2} \end{cases}$$

$$Q' = \begin{cases} x = -\frac{3}{8} + \frac{5}{8}\epsilon_3 \\ y = 1 + 2\epsilon_2 + \epsilon_{1,2} \end{cases}$$

floating-point computations

In the specific case of quadratic forms, the term in $\epsilon_{\pm}$ is used to accumulate floating-point errors: the number of error terms does not increase due to floating-point computation. The generalization to zonotopes is straightforward since numerical operations are evaluated at form level.

We illustrate here these principles on the addition and external multiplication operators.

To summarize, all arithmetic operation are provided in Messine and Touhami [MTo6]. Our implementation with floating point semantics gathers the additive and multiplicative errors of each operator and accumulate them in $\epsilon_{\pm}$ terms, following [SF97] methodology.

1. Addition.

   We consider the addition of two quadratic forms $x = (x_0, (x_i), (x_{ij}), x_{\pm}, x_+, x_-)$ and $y = (y_0, (y_i), (y_{ij}), y_{\pm}, y_+, y_-)$. The addition of x and y is modified to consider these generated errors:

$$\begin{aligned} &(x_0, (x_i), (x_{ij}), x_{\pm}, x_+, x_-) \\ &+_{\mathcal{Q}} (y_0, (y_i), (y_{ij}), y_{\pm}, y_+, y_-) = \\ &\begin{pmatrix} x_0 + y_0, (x_i + y_i), (x_{ij} + y_{ij}), \\ x_{\pm} + y_{\pm} + r\_err, x_+ + y_+, x_- + y_- \end{pmatrix} \end{aligned}$$

   where

   - $r\_err = max(|\uparrow_{+\infty}(err)|, |\uparrow_{-\infty}(err)|)$
   - $err = \sum_{i,j=1}^{n} e^+(x_{ij}, y_{ij}) + \sum_{i=0}^{n} e^+(x_i, y_i) + e^+(x_{\pm}, y_{\pm}) + e^+(x_+, y_+) + e^+(x_-, y_-)$.

2. External multiplication.

   The operator $*_Q$ is modified to account for multiplicative errors:

$$\lambda *_Q (x_0, (x_i), (x_{ij}), x_\pm, x_+, x_-) = (\lambda x_0, \lambda(x_i), \lambda(x_{ij}), |\lambda| x_\pm + r\_err, |\lambda| x_+, |\lambda| x_-)$$

   where

   - $r\_err = max(|\uparrow_{-\infty}(err)|, |\uparrow_{+\infty}(err)|)$.
   - $err = \sum_{i=1}^{n} e^\times(\lambda, x_i) + \sum_{i,j=1}^{n} e^\times(\lambda, x_{ij}) + e^\times(\lambda, x_\pm) + e^\times(\lambda, x_-) + e^\times(\lambda, x_+) + \texttt{eta}$.

Figure 9.7 Arctan program

```
if (x > 1.) {
 y = 1.5708 - 1/x*(1-C1/x^2+C2/x^4+C3/x^6+
  C4/x^8+C5/x^10+C6/x^12+C7/x^14+C8/x^16)
}
if (x < 1.) {
 y = -1.5708 - 1/x*(1-C1/x^2+C2/x^4+C3/x^6+
  C4/x^8+C5/x^10+C6/x^12+C7/x^14+C8/x^16)
}
else {
 y = x*(1-C1*x^2+C2*x^4+C3*x^6+
  C4*x^8+C5*x^10+C6*x^12+C7*x^14+C8*x^16)
}
```

with the constants defined as:

C_1	0.0028662257	C_2	−0.0161657367
C_3	0.0429096138	C_4	−0.0752896400
C_5	0.1065626393	C_6	−0.1420889944
C_7	0.1999355085	C_8	−0.3333314528

Evaluation

All presented materials have been implemented in an open-source tool written in OCaml[7]. This tool is used for teaching purpose and only consider simple imperative programs without function calls. It implements interval analysis, affine and quadratic zonotopes.

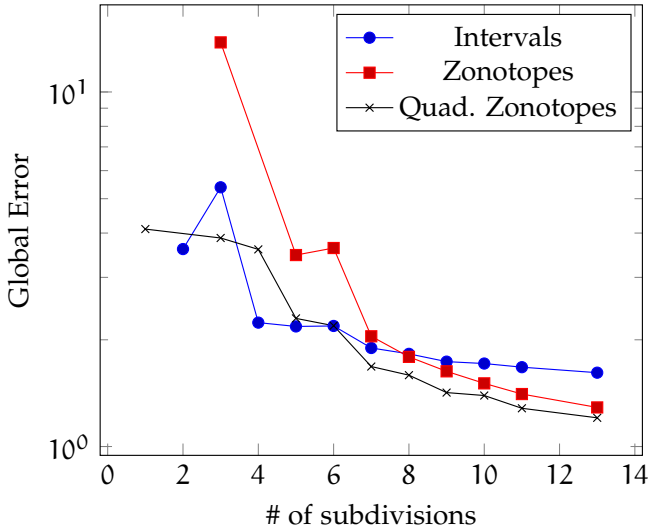
The quadratic zonotope domain has been evaluated on examples bundled in APRON library, or Fluctuat distribution, as well as simple iterative schemes. We present here the results obtained on an arctan function, the example of [CS93] and the Householder function analyzed in [GGP09]. Note that we present here the global value instead on focusing only on the floating-point error.

Let us first consider the arctan function defined in Figure 9.7 and the analysis results in Table 9.1. We can see the dramatic increase in precision obtained with our quadratic extension. This is particularly visible on this example which relies widely on multiplication and division. As a reference the maximal theoretical value for $x \in [-1, 1]$ is $\pi/4$. Intervals or Affine Zonotopes compute a value 144% bigger while Quadratic Zonotopes obtain a 20% imprecision.

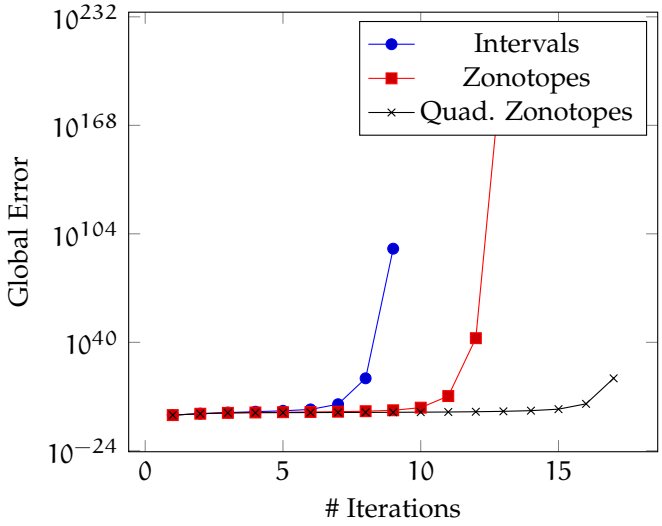
	$x \in [-1, 1]$	$x \in [-10, 10]$
Domain	Bounds	Bounds
Interval	[-1.919149, 1.919149]	[-1.919149, 1.919149]
Aff. Z.	[-1.919149, 1.919149]	[-2.364846, 2.364846]
Quad. Z.	[-1.002866, 1.002866]	[-1.597501, 1.591769]

Table 9.1: Arctan program analysis results

Figure 9.8 Relative precision obtained with different analysis in the experiments (log scale for errors)



(a) Stolfi [CS93] example evaluated on partitioned input range



(b) Householder precision wrt. number of unrolling.

7 Tool and experiments available at https://cavale.enseeiht.fr/QuadZonotopes/

In [CS93], Stolfi *et al* considered the function $\sqrt{(x^2 + x - 1/2)}/\sqrt{(x^2 + 1/2)}$ and the precision obtained using affine arithmetics while evaluating the function on a partition of the input range as sub-intervals. This is the classical bisection or branch-and-bound approach to improve precision. Figure 9.8a compares the obtained results for subdivision from 1 to 14 partitions. The global error represents the width of the interval obtained and is represented in a log scale. Higher partition divisions (eg. 500) converge in terms of precision and are not shown on the picture. The table 9.2 presents the computed values.

Quadratic zonotopes shows here to be a good alternative to interval or affine zonotopes abstractions. Both in terms of precision and runtime. Interestingly for this example the expected additional cost due to the quadratic error terms is not exhibited. This may be explained by a more direct expression of quadratic terms within our quadratic zonotopes.

Another example analyzed is the Householder function; this dynamical system converges towards $1/\sqrt{A}$:

$$x_0 = 2^{-4}$$
$$x_{n+1} = x_n(1 + \tfrac{1}{2}(1 - Ax_n^2) + \tfrac{3}{8}(1 - Ax_n^2)^2)$$

In our experiments the algorithm was computed with a while loop and a finite bound N on the number of iterations. We analyzed it using loop unrolling with $A \in [16, 20]$ and compared the global errors obtained at the i-th iterate, that is, the difference between the max and min values. Analyzing such system with interval diverges quickly while Affine and Quadratic Zonotopes are more stable. The figure 9.8b presents the precision obtained with a variant of the algorithm were $A \in [16, 20]$ is randomly chosen at each loop iteration. While this program is meaningless, its analysis is interesting in terms of precision: intervals diverges from the 7th iteration, affine zonotopes from the 11th and quadratic ones from the 17th. Quadratic zonotopes here provides again better bounds than affine or interval analysis and shows to scale better than all other analyses. The table 9.2 presents a selection of iterates computed values.

Most computation are performed within 30ms. Only the STOLFI example with a large number of partitions shows a much longer time for Affine and Quadratic Zonotopes (about 1s) than intervals (91ms).

9.4 RELATED WORKS

Analysis of accuracy of finite-precision arithmetic is not a new topic. Multiple numerical methods have been proposed to address this issue: interval arithmetics, stochastic arithmetics, automatic differentiation or error series. See [Gou01; Mar05] for comparative surveys. In static analysis, most analyses rely on safely rounded intervals or on zonotopes [GGP10; GGP09; GPV12]. These affine arithmetics based domain are the core of the tool Fluctuat [Gou13] able to bound the numerical errors and identify the terms or variables that participate the most to the final error. These domains are also now implemented as an APRON [JM09] domain.

	Intervals		Affine Z.		Quad. Z.	
	val	ms	val	ms	val	ms
stolfi1	$[0., \infty]$	6	$[-\infty, +\infty]$	0	$[-0.85, 3.25]$	7
stolfi2	$[0., 3.60]$	10	$[-\infty, +\infty]$	7	$[-\infty, +\infty]$	4
stolfi3	$[0., 5.38]$	4	$[-2.98, 10.81]$	5	$[-0.62, 3.24]$	5
stolfi4	$[0., 2.23]$	10	$[-\infty, +\infty]$	6	$[-0.33, 3.26]$	11
stolfi5	$[0., 2.18]$	9	$[-0.42, 3.03]$	3	$[0.08, 2.38]$	11
stolfi6	$[0., 2.18]$	7	$[-0.71, 2.91]$	5	$[0.11, 2.30]$	6
stolfi7	$[0., 1.89]$	7	$[0.19, 2.23]$	3	$[0.29, 1.97]$	10
stolfi30	$[0.35, 1.43]$	15	$[0.48, 1.44]$	20	$[0.48, 1.43]$	18
stolfi40	$[0.40, 1.40]$	15	$[0.49, 1.40]$	20	$[0.50, 1.40]$	18
stolfi50	$[0.43, 1.38]$	19	$[0.50, 1.38]$	34	$[0.50, 1.38]$	21
stolfi55	$[0.44, 1.37]$	24	$[0.51, 1.37]$	33	$[0.51, 1.37]$	35
stolfi100	$[0.48, 1.34]$	29	$[0.52, 1.34]$	80	$[0.52, 1.34]$	66
stolfi200	$[0.51, 1.32]$	48	$[0.53, 1.32]$	337	$[0.53, 1.32]$	269
stolfi300	$[0.52, 1.31]$	73	$[0.53, 1.31]$	916	$[0.53, 1.31]$	554
stolfi400	$[0.52, 1.31]$	91	$[0.53, 1.31]$	1746	$[0.53, 1.31]$	1086
householder #3	$[0.21, 0.24]$	3	$[0.21, 0.24]$	9	$[0.21, 0.24]$	4
householder #4	$[0.17, 0.29]$	0	$[0.22, 0.25]$	7	$[0.22, 0.24]$	8
householder #5	$[0.03, 0.42]$	3	$[0.22, 0.25]$	8	$[0.22, 0.24]$	10
householder #6	$[-0.90, 1.66]$	3	$[0.22, 0.25]$	19	$[0.22, 0.24]$	14
householder #7	$[-1117.82, 1899.48]$	4	$[0.22, 0.25]$	27	$[0.22, 0.24]$	11
householder #8	$[-2.18e^{+18}, 3.70e^{+18}]$	5	$[0.22, 0.25]$	29	$[0.22, 0.24]$	11

Best method is highlighted. Results are shown with two decimal digit precision.

Table 9.2: STOLFI example [CS93] and Householder numerical results

10

CONVEX OPTIMIZATION AND NUMERICAL ISSUES

In this chapter we aim at providing the intuition behind convex optimization algorithms and address their effective use with floating point implementation. A first section presents briefly the algorithms, assuming a real semantics, while Section 10.2 presents our approaches to obtain sound results. Last Section presents our implementation, OSDP, the Ocaml Semi-definite programming library.

10.1 CONVEX OPTIMIZATION ALGORITHMS

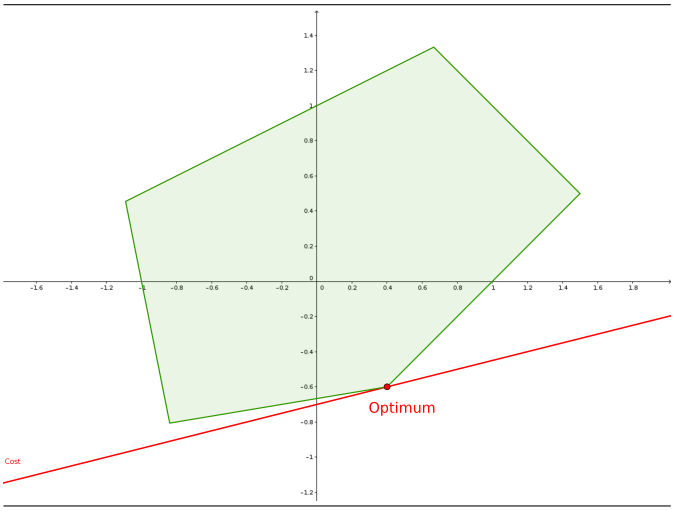
As outlined in Section 4.2.1 convex conic programming is supported by different methods depending on the cone considered.

The most known approach for linear constraints is the simplex method by DANTZIG. While having an exponential-time complexity with respect to the number of constraints, the simplex method performs well in general. Intuitively, starting from a vertex of the convex polytope, it follows the hyperplane minimizing the cost of the objective function to reach the best neighbor vertex. It can be seen as both a discrete and combinatorial methods: it enumerates the finite number of faces of the polytope but each neighbor vertex computation involves numerical values. These steps could be solved exactly, for example with rational arithmetics. Theoretically the optimal solution is always on a vertex. It is either unique if a single vertex is optimal or has an infinite set of solutions, when all a face of the polytope is optimal.

If the simplex method behaves properly, one of the advantages of its use is the obtention of a strictly feasible optimal solution.

On the negative aspects, its complexity could diverge in some ill-shaped cases and it is not extensible to the larger set of linear problems over convex cones.

Figure 10.1 Linear optimization over a polytope



Another method, the set of Interior Point methods were initially proposed by KARMARKAR [Kar84] and made popular by NESTEROV and NEMIROVSKI [NN88; NN94; NN89]. They can be characterized as path-following methods in which a sequence of local linear problems are solved, typically by NEWTON's method.

10.1.1 *Convex optimization with interior point method algorithms*

An interior point method optimization is performed in two steps: A first one computes the analytical center of the convex set of constraints. This element is characterized by a logarithmic barrier function[1].

Definition 10.1 (Analytical center) *Let* $\bigwedge_i(f_i(x) \leqslant 0)$ *be a set of convex inequalities, the analytical center is defined as the optimal solution of the convex problem*

$$\text{minimize } \phi(x) \text{ where } \phi(x) = -\sum_i \log(-f_i(x)) \quad (163)$$

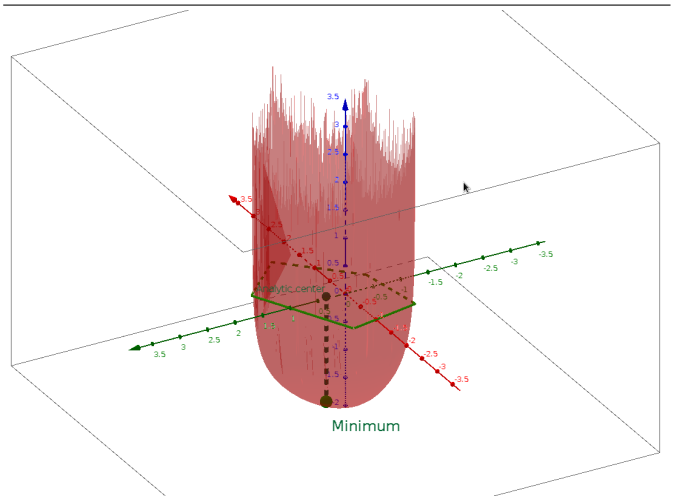
1 In order to keep the presentation simple we provide definition and illustration here on linear constraints instead of more general linear matrix inequalities.

The computation of such value by solving the gradient equal to zero:

$$\nabla\phi(x) = \sum_i \frac{1}{-f_i(x)} \nabla f_i(x) \tag{164}$$

Intuitively this function tends to $+\infty$ when the x is near one of the linear constraints $f_i(x) = 0$. In case of linear constraints, $f_i(x) = a_i(x) - b_i$ and the expression of $\nabla\phi$ becomes $\sum_i \frac{1}{-f_i(x)} a_i$. The following figure illustrates such barrier function on the preceding polytope.

Figure 10.2 Barrier function characterizing the analytical center.



Path-following algorithms encode the search of the solution of a linear optimizing problem as a sequence of local NEWTON problems. A Phase I steps compute the starting point, the analytical center of the set of constraints. Then Phase II performs a sequence of local NEWTON steps. It characterizes a notion of central path, a function $\widetilde{f}(t;x)$ that integrates both the constraint $\phi(x)$ to be in the interior of the set of linear constraints, and the (linear) objective function $f(x)$:

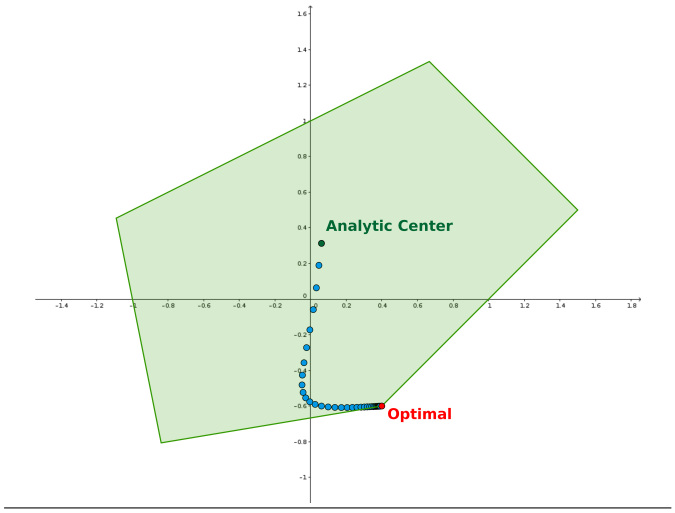
Definition 10.2 (Central path)

$$\widetilde{f}(t;x) = t \times f(x) + \phi(x)$$

Note that when $t = 0$ we have the analytical center $\widetilde{f}(0;x) = \phi(x)$, and when $t \to +\infty$, this constraint vanishes while the objective function becomes stronger in the expression.

Without arguing too much about the details of the computation, let us illustrate on this simple example the computed step towards the optimal solution:

Figure 10.3 Interior point method algorithm in LP.



Interior point methods compute a sequence of intermediate feasible solutions. Each step is performed by computing a linear direction and a step length, such that the next value remains in the neighborhood of the central path while improving the objective function cost. This sequence stops when a required precision is reached. On a typical implementation the stopping criterion is around 10^{-7}.

In contrast to the simplex method, we see clearly that this method will never compute the exact solution but is able to approximate it as precisely as required. On the good sides, interior points methods can be defined for more general settings than linear constraints, for example on the large set of convex conic sets (quadratic programming, second order conic programming, semidefinite programming, etc). On the complexity side, the method has a polynomial complexity, in the number of variable. Typical uses in software works on a thousand of variables while efficient uses can scale to hundreds of thousands of variables when smartly implemented, eg. on FPGA [Jer+14].

10.1.2 *Primal-dual feasibility*

Recall the definition of duality in optimization (Section 4.2.3). Among the most efficient implementations of interior point methods, Primal/Dual methods rely on the notion of duality gap to measure the distance to the optimal solution.

When manipulating matrix variables, the inequality (60) can be rephrased on the Hilbert space of positive definite matrices:

$$\begin{array}{lll} d^* = \max\limits_{X} & \langle c, X\rangle \leqslant & p^* = \min\limits_{y} \quad -\langle y, b\rangle \\ \text{s.t.} \quad AX = b & & \text{s.t.} \quad -c - A'y = Z \\ \quad X \succeq 0 & & \quad Z \succeq 0 \end{array} \tag{165}$$

where d^* and p^* denote, respectively, the dual and primal solutions.

The distance between two feasible points of primal and dual problems is called the duality gap:

$$\langle c, X\rangle + \langle y, b\rangle$$

The barrier function of the method is then characterized using the central path expression as the combination of the analytical center of both primal and dual problems, and the duality gap as the objective function to minimize.

However, it may happen that either the primal or the dual problem are ill-defined: they can be unfeasible or unbounded, ie. $d^* = \infty$ or $p^* = -\infty$. According to the duality theorem, if both primal and dual solutions are finite, then both problems are feasible. Otherwise, either the primal solution is unbounded and its dual unfeasible, or the dual solution is unbounded and the primal unfeasible.

10.1.3 *Issues*

These convex optimization methods have now reached sufficient maturity and are now used in a large set of contexts. However, for their specific use in formal verification, one need to cope with the following issues:

1. In contrast to simplex methods, these methods never reach the optimal value. However, they admit an exponential convergence and can approximate the solution with arbitrary precision. On the positive side, the computed suboptimal solution, as well as all intermediate iterates of Phase II are feasible solutions of the set of constraints. Therefore, when the objective function is not crucial – as it is when we synthesize our invariants with these techniques – these solutions can be used safely.

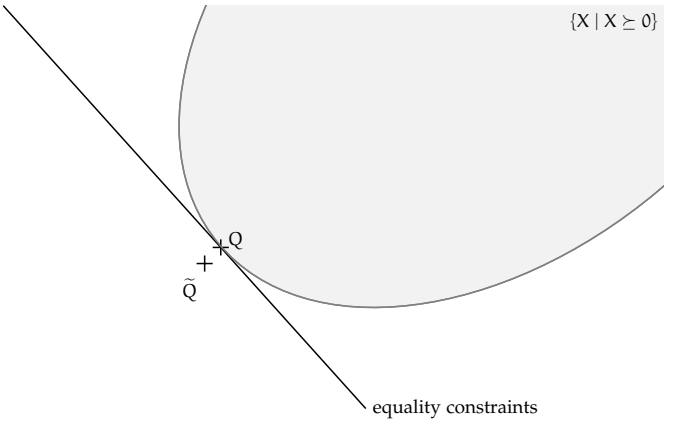
2. Because of floating-point errors, the actual ϵ-optimal solution may be (slightly) an unfeasible solution. This is particularly difficult when manipulating equality constraints.

While often neglected, this second issue has to be addressed in our formal framework. This is a great importance since we rely on computed solutions to bound the behavior of the analyzed systems. In case of SOS programming, the underlying SDP matrix is associated to a set of equality constraints enabling the characterizing of the positive polynomial. The reconstruction of such solution polynomial with floating point arithmetics has also to be addressed in a formal way.

10.2 GUARANTEED FEASIBLE SOLUTIONS WITH FLOATS

As we saw in the preceding figures, the optimal value of a linear objective function with respect to a set of convex constraints is always on the boundary of the convex set. In SDP optimization this could be mapped to the search of the point Q within the ellipsoid characterized by the set of constraints $X \succeq 0$, ie. $\forall x \in \mathbb{R}, x^T X x \geqslant 0$.

Figure 10.4 Floating point errors with interior point methods.



Because of floating point errors, the computed value could be slightly outside of the set of constraints, giving an unfeasible, and therefore unsound, solution $\widetilde{Q}$.

10.2.1 *Computation over a constrained cone*

Since these path-following algorithms never actually reach the optimal value but stay within the interior of the set of constraints, the stopping criterion depends on the achievement of a given precision, typically a duality gap of 10^{-7}. Without any formal result, remark that, for all $|x| < 10^p$, $|x^2| < 10^{2p}$, so when computing with a precision of 16 digits, ie. $\text{ulp}(1) = 10^{-16}$, half the digits are lost by the quadratic form, leading to an imprecision of $\sqrt{(10^{16})} = 10^{-8}$ for Q.

A heuristic to ensure to remain within the interior of the set of constraints is to pad the convex constraint by this precision of 10^{-8}. Figure 10.5 presents such a con-

strained set of conic constraints. One can ensure that the floating point value $\widetilde{Q}$ still remains within the convex set.

This padding may fail when the set of constraints admit an empty interior, which cannot be padded. This happens typically for our invariant search when the dynamic considered admits fixpoints. Let us consider the dynamics f and x_{fp} such as fixpoint, our search for an inductive invariant v will lead to a constraint of the form:

$$v \circ f - v \leqslant 0$$

But since $f(x_{fp}) = x_{fp}$, we have

$$v(f(x_{fp})) - v(x_{fp}) = v(x_{fp}) - v(x_{fp}) = 0$$

and this problem admits an empty interior since v should be exactly equal to zero at x_{fp}. In this case the previous technique over-approximating the floating point semantics of f will impose an even more difficult constraint:

$$\begin{aligned} & v(f(x_{fp})) - v(x_{fp}) - err_{f(x_{fp})} \\ = \;& v(x_{fp}) - v(x_{fp}) - err_{f(x_{fp})} \\ = \;& -err_{f(x_{fp})} = 0 \end{aligned}$$

with $err_{f(x_{fp})} > 0$

In the linear case, when considering stable systems, there is a single fixpoint which is often zero. A first solution is to impose the LYAPUNOV function v to admit a zero coefficient for the monomial 1, allowing the error to vanish in zero. The problem becomes on non empty interior.

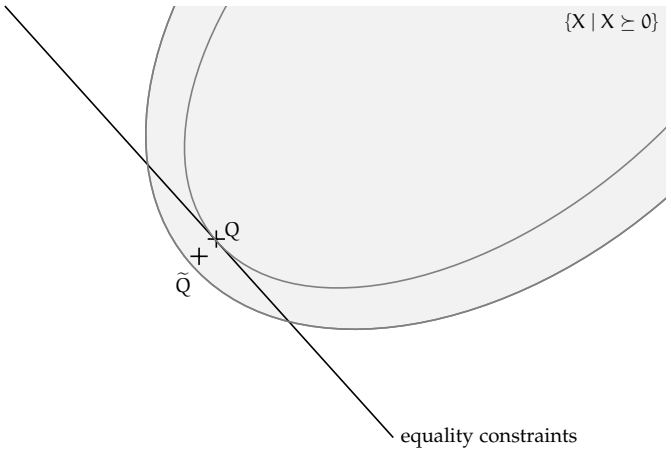
For the more general setting, a solution is to encode inductiveness differently as a convex constraint: instead of $v \circ f - v \leqslant 0$ one can rely on $\alpha v \circ f - v \leqslant 0$ with $\alpha > 1$. This would correspond to a relative padding, applying a growth factor to the semialgebraic set defined by $v \leqslant 0$.

Considering, in addition the floating point semantics of the analyzed program, our inductiveness constraint in the polynomial case becomes:

$$p - \alpha\, p \circ T^i \sigma^i + \sum_{j=1}^{n_i} \mu_j^i r_j^i - err_{p \circ T^i} \geqslant 0$$

where $err_{p \circ T^i}$ is an upper bound on the floating point errors obtained when evaluating $p \circ T^i$.

Figure 10.5 Padding conic convex constraints



10.2.2 *Validate feasibility of the solution*

Despite the additional padding, it may happen that the computed property is not strictly verifying the constraints. It then remains to formally check that the computed values satisfy the set of positive constraints.

When manipulating directly LMIs, this can be done by computing the matrix associated to each positive constraint, for instance with exact rational arithmetic, and then checking that the result is positive definite. This last check can be performed using a CHOLESKY decomposition. For the sake of efficiency, this decomposition can itself be performed using floating-point arithmetic by carefully bounding the rounding errors [Rumo6]. The resulting algorithm being non trivial, it has been proved using the proof assistant Coq [Rou15]. Another approach [MC11] intends to find such positive definite witnesses using rational coefficients: computing rational solutions of general SDP satisfiability problems.

We describe here our conservative approach, based only on floating point computations.

When manipulating SOS polynomials, additional equality constraints relate matrix coefficients to the coefficients of the SOS polynomial. It remains to check that the floating point polynomial solution is such that it verifies all positive constraints.

To show that a polynomial constraint e is positive when all its unknown variables have been valued by the optimization solver, we evaluate in rational arithmetics the polynomial p associated to the expression e. SOS optimization will then exhibit a floating point matrix Q such that $p = x^\intercal Q x$.

Because of numerical imprecision, due both to the method (interior point) and the floating point rounding errors, we rather have $p = x^\intercal Q x + x^\intercal E x = x^\intercal (Q + E) x$ where E denotes the error produced during the SOS programming.

This error term $x^T E x$ can be characterized as $p - x^T Q x$, and we can identify an upper bound ϵ on the error per coefficient:

$$\forall i, j, |E|i, j \leqslant \epsilon$$

In order to prove that there exists a positive semi-definite matrix $Q + E \succeq 0$, we check the stronger condition $Q - n\epsilon Id \succeq 0$ where n is the dimension of Q. The Figure 10.6 presents the different results: Q is not exactly on the constraints but in a neighborhood. In the first case, all the matrices $Q - n\epsilon Id$ are positive definite, so there exists a witness of positivity for the constraint p. In the second case, the constraint intersects the cone and p is positive, but the imprecise computed Q does not enable to prove that $Q - n\epsilon Id \succeq 0$. In the last case, p is not positive, but Q was produced as an SOS witness of positivity for p. Thanks to the invalidity of the check, we are able to reject this value.

10.3 IMPLEMENTATION AS AN OCAML LIBRARY: OSDP

With Pierre Roux, we developed an Ocaml library providing access to existing SDP solvers. This library eases the use of convex optimization in our various Ocaml static analyzer prototypes without rely on Matlab or writing solver specific code.

The library modules can be structured in three parts:

- basic datatypes;
- convex programming modules;
- backend for solvers.

The Figure 10.7 presents a view of such modules.

On the datatype part, OSDP provide means to manipulate linear algebra and polynomials. All modules are defined as functors of scalar values and can be instantiated with either rationals or floats.

The main part consists in convex programming modules. LMI and SOS modules provides types to define expressions over matrices of polynomial variables, respectively. In both cases these high level convex optimization problems are recast as SDP problems. For the SOS module each polynomial has to be instantiated with a given degree to enable the computation of appropriate Gram matrices.

The SDP modules is interfaced with supported solvers, through their provided API in C or C++. For the moment we support Csdp [Bor99], the solver of the COIN-OR project, Sdpa, a solver in C++ with various implementation versions such as GMP based computations or a parallel version, and the very efficient commercial software Mosek.

In addition to these backend, the PreSDP module provides some limited means to preprocess the constraints: mainly the removal of redundant variables to obtain full rank matrices. In case of redundant variables, the set of constraints admits an empty interior because of equality constraint between these variables. In that case, a solution consists in remove these redundant variables while keeping the associated relationship. Once the solution is obtained, these values are recomputed. Sdp modules provides both dense and sparse matrices datatype in which to perform the analyses.

Let us look at the signature of the solve function from the Sdp module:

OCaml
```
val solve :
?options:options ->
?solver:solver ->
?init:matrix block_diag * float array * matrix
     block_diag ->
matrix obj ->
matrix constr list ->
SdpRet.t * (float * float) * (matrix
    block_diag * float array * matrix
    block_diag)
```

The options define the stopping criterion for the optimization process: duality gap target or upper bound on iterations.

The function returns the status of the result (Success, PrimalInfeasible, DualInfeasible, ...), both primal and dual solutions, and the values of primal and dual variables X, y, and Z.

The two main frontends, the modules Lmi and Sos, provides more sophisticated solve functions that integrate the soundness checking:

OCaml
```
val Lmi.solve :
  ?options:options ->
  ?solver:Sdp.solver ->
  obj ->
  matrix_expr list -> SdpRet.t * (float * float)
       * values
```

Figure 10.6 Sound positivity check using SOS

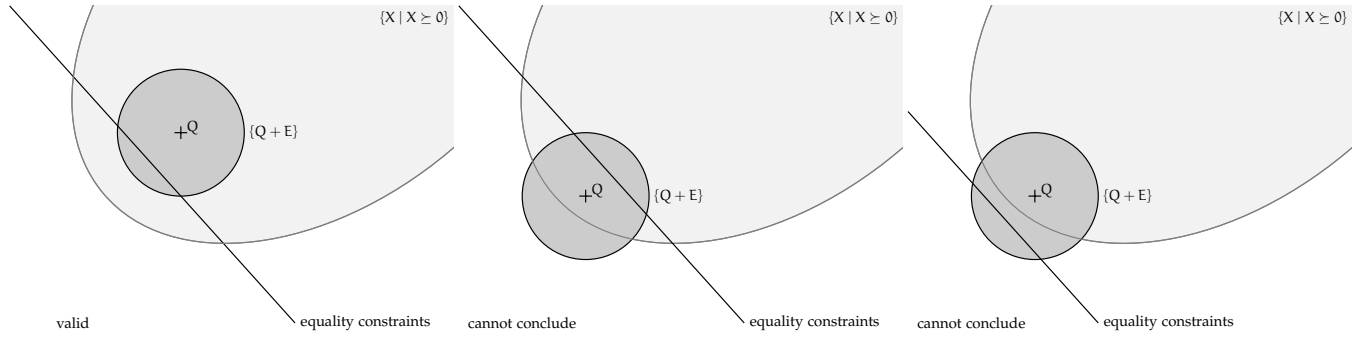
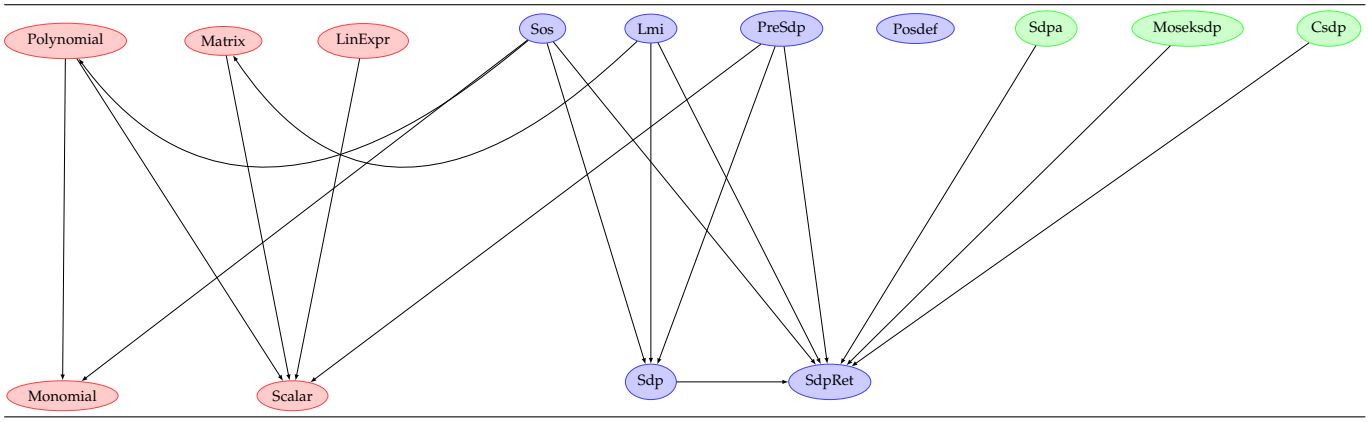


Figure 10.7 OSDP modules



OCaml

```
val Sos.solve :
?options:options ->
?solver:Sdp.solver ->
obj ->
polynomial_expr list ->
SdpRet.t * (float * float) * values * witness
    list
```

The options contain the padding parameter. In both cases, the LMI or the set of positive polynomial constraints is solved thanks to the Sdp frontend, adding the provided padding on the generated Sdp constraints. Once the result is obtained, if the return status is Success, the variables of the original problem are valued using the primal variable values returned by the SDP solver. This generates a context environment, a map from variable name to floating point value[2]. Then, using the rational arithmetics scalar module, the positive constraints, without padding, are evaluated in this context. This gives a list of matrices or polynomial expressions without free variables (except the monomials of the polynomial expression) which have to be proved positive.

2 All our SDP backend run floating point computations.

The Posdef modules provides access to our conservative CHOLESKY decomposition in floats, whose algorithm has been proved in Coq [Rou15]. At last, the return status is updated with the feedback from this sequence of positive checks and the set of valued primal variables (witnesses) is return.

In case of a first Success return status but at least an invalid positive check with our CHOLESKY decomposition, the PartialSuccess return is produced.

Let us know conclude with a run of the tool on a simple example inspired by Yalmip documentation:

$$
\begin{array}{rll}
 & \min & (1+xy)^2 - xy + (1-y)^2 \\
 & & \text{s.t. } |x| <= 1, |y| <= 1 \\
= & \max & t \\
 & & \text{s.t. } (1+xy)^2 - xy + (1-y)^2 - t \\
 & & -q1(1-x) - q2(1+X) \\
 & & -ssq3(1-y) - q4(1+y) \text{ is SOS} \\
 & & \text{and } q1, q2, q3 \text{ and } q4 \text{ are SOS.}
\end{array}
$$

OCaml

```
let solver = Osdp.Sdp.Mosek
open Osdp.Sos.Float
let options = { default with verbose = 3 }

let _ =
  let lower = var "lower" in
  let q1, _ = var_poly "q1" 2 2 in
  let q2, _ = var_poly "q2" 2 2 in
  let q3, _ = var_poly "q3" 2 2 in
  let q4, _ = var_poly "q4" 2 2 in
  let p = (!1. + ??0 * ??1)**2 - ??0 * ??1 +
      (!1. - ??1)**2 in
  let e = p - lower - q1 * (!1. - ??0) - q2 *
      (!1. + ??0) - q3 * (!1. - ??1) - q4 *
      (!1. + ??1) in
  let ret, (pobj, dobj), vars, _ =
    solve ~solver ~options (Maximize lower)
    [e; q1; q2; q3; q4] in
  value lower vars
```

The prefix ! and ?? enable us to declare scalar constant and variables, respectively. Once the expression e is defined, we can print it:

```
e = (1 + x0 * x1)^2 - x0 * x1 + (1 - x1)^2
   - lower
   - q1 * (1 - x0) - q2 * (1 + x0)
   - q3 * (1 - x1) - q4 * (1 + x1)
```

In this case, we have 5 SOS constraints that will become 5 SDP constraints. Each of them is associated to a scalar b_i and the solver will compute the positive matrix X such that $\mathrm{tr}(A_i X) = b_i + \mathrm{perr}$. A first computed value corresponds to this perr constant. It depends on the precision of the representation of this vector b in floats. Typically, depending on the solver it is either the norm-2 or norm-∞ of the vector b multiplied by $\sqrt{(}\,\mathrm{ulp}(1))$. Here we have

$$\mathrm{perr} = 3.43722e-08$$

Heuristically, from that perr and depending on the complexity of each constraint, for example the dimension of the associated matrix, a padding is produced. In this case, we have a padding of $1.37489e-07$ for the first complex constraint, and $1.03116e-07$ for the four SOS multipliers.

Mosek is then called and perform it primal-dual interior point methods, converging in 12 iterations (and 0.01 second). Its verbose output returns the summary

```
Interior-point solution summary
  Problem status  : PRIMAL_AND_DUAL_FEASIBLE
  Solution status : OPTIMAL
  Primal.  obj: -7.4999775936e-01
    Viol.  con: 1e-08
    var: 0e+00    barvar: 0e+00
  Dual.    obj: -7.4999775223e-01
    Viol.  con: 0e+00
    var: 1e-08    barvar: 0e+00
```

This return is propagated back to the Ocaml library and the primal variables are used to rebuild the constraints variables: here the floating expressions of computed polynomials lower, q1, q2, q3 and q4, without the additional padding.

The checking part, evaluate the rational expression of each five constraints: e, q1, q2, q3 and q4 from that environment. Let us consider the expression e: we have its expression as a polynomial p_e, combination of floating point polynomials, and we have its expression as a positive definite matrix $x^t Q_e x$. We compute the maximum difference r between the corresponding coefficients, and obtain

$$r = 3.51434e-08$$

We then check, using a conservative Cholesky decomposition that

$$Q - rI \succ 0$$

In this case, all five checks are valid, guaranteeing that, despite floating point imprecision, the lower bound 0.749998 is sound.

Part V

PERSPECTIVES

11

INTEGRATION OF THE ANALYSES IN THE SYSTEM/SOFTWARE DEVELOPMENT PROCESS

A first perspective is the integration our work in a unified toolchain. Most of the presented works are available as prototypes applicable in their own representation of the system to be analyzed. For example chapters of part II are each of them defined on a different representation of a dynamical system: purely linear, piecewise linear, and piecewise polynomial. Similarly, the floating point precision analysis with quadratic zonotopes is applied on a different representation of imperative code.

While these different settings can be explained by the current objectives when the work was performed, it is a real limitation in the integration of various techniques in a simple unified fronted.

We present here our related perspectives. They are presented over existing or future toolsets we have been developing the last lustre [1].

11.1 COCOSIM AND LUSTREC TOOLCHAIN

In Part III we showed that model-based languages, especially dataflow languages, were widely used to design and autocoded controllers. The second chapter, Chapter 8, was developed around the tool Geneauto.

Geneauto is complex software, developed through first a European ITEA2 project, and then through a French FUI project. It is written in Java as a sequence of compilation steps. While the approach is nice from a engineering perspective, it was difficult for us to adapt it and address, with it, all the formal methods concerns we have been mentioning in this manuscript.

Therefore, in order to showcase the relevancy of applying formal methods for controller systems, we designed our own compilation toolchain from Matlab Simulink to C code. This toolchain is split in two parts:

- Backend: LustreC
- Frontend: CocoSim

11.1.1 *LustreC*

Lustre is an academic dataflow language[Hal+91] with a precise semantics. It was proposed in the 90s and lead to the definition of the industrial Esterel Scade language [2] which is used in major industries including Airbus. In Lustre, synchronous flows of data are computed at each (symbolic) time and complex system can be build by composing these signals. This kind of description is particularly adapted to the development of control system, similarly to the famous industrial tool Matlab Simulink. In 2008, modular compilation of Lustre models was proposed [Bie+08]. The Esterel Scade compiler KCG was able to receive a qualification DAL-A in the A3xx program, enabling its use for the code generation of the most critical parts of aircraft controllers.

In collaboration with Xavier Thirioux, we developed a similar tool, LustreC [GTK12] an open source Lustre compiler with modular compilation to C code.

We used this tool to develop new formal methods at the frontier between model level analysis with Lustre, and C code analysis. This covers compiler validation with MC/DC test generation and mutation testing [Gar+14], to validation of synchronous observers at model and code level [Die+15]. With Temesghen Kahsai we also developed another backend of the LustreC compiler targeting modular Horn encoding. This enabled the use of the compiler as a Lustre interpreter for SMT-based model-checking [GGK14; KTG16]. A lustre model is compiled into an equivalent Horn encoding representation, which can then be used by SMT model-checker such as Spacer [KGC14] or z3 [MB08].

11.1.2 *CocoSim*

CocoSim has been developed as a way to address, with our academic tools, the analysis of realistic industry-size models, written in Matlab Simulink. This is translator from a reasonable subset of the discrete blocks of Matlab Simulink into the Lustre dataflow language. While

1 Lustre is not only a dataflow language, a lustre is also a period of five years.
2 now ANSYS Scade

Simulink can be used to express complex systems mixing discrete and continuous signals or solving dynamical algebraic loops in the model definitions, our choice of a small subset solves most of these issues. In other words, for the tiny subset considered, one have a one-to-one correspondence between Simulink constructs and Lustre expressions. More advanced, but still reasonable, constructs, such as z-expressions, gain interpolation, or more sophisticated datatypes such as buses, are simplified into simpler constructs, with Simulink before their translation to Lustre.

This work was mainly done at NASA/CMU and led by Temesghen Kahsai. A first experiment with Arnaud Dieumegard, one of the researcher of Geneauto team, consisted in a fork of Geneauto, with similar early translation stages, followed by a model-transformation from the intermediate representation into a meta-model of Lustre. This first tool suffered from Geneauto inheritance: it was parsing the XML of the Simulink and was recovering all the information from that file, without any serious information on the structure of that file. Any change in the version of Matlab would change the structure of the file.

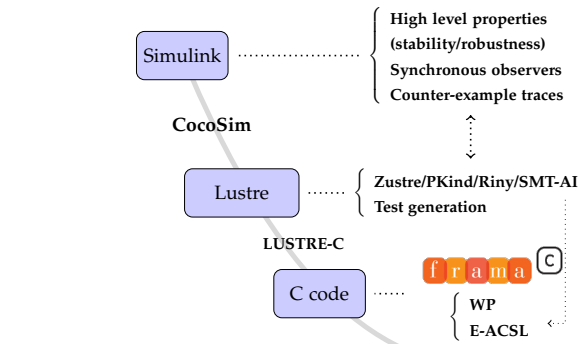
In the next experiment, inspired by Caspi *et al.* approach [Cas+03] to the compilation of Simulink into Lustre, we build upon Claire Pagetti and Thomas Loquen's open-source prototype of the idea, to build the current CocoSim platform.

In addition to basic translation into Lustre, CocoSim also provides generation of traceability information and the integration with model-checker. One possible use is the following: a Simulink model is annotated with an observer block, acting as a synchronous observer – in other words, this block has a boolean output which should be valid for any possible simulation or run; thanks to CocoSim and the Horn encoding backend of LustreC, the model and the property are expressed in a suitable format for Spacer to analyze it. In case of success the property is tagged as valid in Simulink, in case of failure the counter example is propagated back at Simulink level and expressed through a simulation trace.

11.1.3 *Integration*

The following figure illustrates our perspectives: the integration of presented formal verification into that toolchain.

Figure 11.1 Process cycle with autocoders



The toolchain should be compatible with most of the properties that have to be addressed in terms of functional verification.

Control-level properties

At the Simulink level the system-level properties should be expressed as synchronous observers at in Chapter 8. In addition, the plant part should be appropriately tagged to be considered as such by the compilation. Then, through the toolchain, the following Lustre nodes should be produced:

Figure 11.2 Lustre model including discrete plant description, for the example of Section 7.2.5.

Lustre

```
node spring (u:real) returns (y:real);
var xp0, xp1: real ;
let
 xp0 = 0. -> pre xp0 + 0.01 * pre xp1 +
     0.00005 * u;
 xp1 = 0. -> -0.01 * pre xp0 + pre xp1 +
     0.01 * u;
 y = xp0;
tel

--@ plant: spring
node ctl (in,y:real) returns (u:real);
var e, xc0, xc1: real ;
let
 e = in - y;
 xc0 = 0. -> 0.4990 * pre xc0 - 0.05 * pre
     xc1 - e;
 xc1 = 0. -> 0.01 * pre xc0 + pre xc1;
 u = 564.48 * xc0 + 1280. * e;
tel
```

The `plant` annotation enables the margin analysis which extracts the linear representation of the controller and the plant to build the sensitivity system and analyze it. The Lustre model is enriched with analysis results (*cf.* Fig 11.3).

Figure 11.3 Enriched model with computed properties.

Lustre

```
--@ plant: spring
node ctl (in, y: real) returns (u: real);
--@ robustness/gamma: 1.4914;
--@ robustness/P: (computed P value);
```

When generating the final C code, the code is annotated with ACSL predicates [Bau+08] relying on our linear algebra library [Wan+16a]. An additional predicate encodes the dissipativity property (142). The `struct` definitions represent the internal state of each node in our modular compilation scheme. The plant internal state is declared as a ghost `struct` field within the controller own memory. This is presented in Fig. 11.4.

Figure 11.4 Header of the generated C code including node state description.

C+ACSL

```
#include "acsl_matrices.h"

/*@logic matrix P = mat_of_4x4_scalar(
111.8330,  88.4842, -48.4990,  8.8432,
 88.4842, 278.5963, -20.2482,  6.9605,
-48.4990, -20.2482,  28.7964, -3.7961,
 8.8432,   6.9605,  -3.7961,  0.7013);
   logic real gamma = 1.4914; */

/*@ logic vector state(struct ctl_mem self)
    =
 vector_of_4_scalar (
   self->_reg.__ctl_3,
   self->_reg.__ctl_2,
   self->spec.plant._reg.__plant_3,
   self->spec.plant._reg.__plant_2);*/

/*@ predicate dissip(vector snxt, vector s,
    real in, real e, matrix P, real gamma) =
    normP(snxt, P) - normP(s, P) <= gamma**2
         * in**2 - e**2; */

struct plant_mem {
  struct plant_reg {double __plant_2;
           double __plant_3; } _reg;
  struct _arrow_mem *ni_1; };
struct ctl_mem {
  struct ctl_reg {double __ctl_2;
          double __ctl_3; } _reg;
  struct _arrow_mem *ni_0;
  /*@ghost struct spec {
  struct plant_mem plant; } spec; */};
```

Finally, the controller function is associated to an assert statement ensuring the dissipativity property after each iteration of the system dynamics, as presented in Fig. 11.5. It is worth noting that the plant code was introduced as ghost C code within the controller code, following the approach we developed in [Wan+16a].

Figure 11.5 Transfer function of the controller, including the plant description as annotation and the dissipativity property as function contract.

C+ACSL

```
void ctl_step (double in, double y,
               double (*u),
               struct ctl_mem *self) {
_Bool __ctl_1;
double e; double xc0; double xc1;
_arrow_step(1,0,&__ctl_1,self->ni_0);
if (__ctl_1) { xc1 = 0.; } else {
  xc1 = ((0.01 * self->_reg.__ctl_3) +
      self->_reg.__ctl_2); }
e = (in - y);
if (__ctl_1) { xc0 = 0.; } else {
  xc0 = (((0.499 * self->_reg.__ctl_3) -
      (0.05 * self->_reg.__ctl_2)) - e); }
*u = ((564.48 * xc0) + (1280. * e));
self->_reg.__ctl_3 = xc0;
self->_reg.__ctl_2 = xc1;
/*@ghost
_Bool __plant_1;
double xp0; double xp1;
//plant - restricted to state update
_arrow_step (1, 0, &__plant_1, self->spec.
    plant.ni_1);
if (__plant_1) { xp0 = 0.; } else {
  xp0 = ((self->spec.plant._reg.__plant_3 +
      (0.01 * self->spec.plant._reg.
      __plant_2)) + (5e-05 * *u)); }
if (__plant_1) { xp1 = 0.; } else {
  xp1 = ((((- 0.01) * self->spec.plant._reg.
      __plant_3) + self->spec.plant._reg.
      __plant_2) + (0.01 * *u)); }
self->spec.plant._reg.__plant_3 = xp0;
self->spec.plant._reg.__plant_2 = xp1;
*/
//@assert dissip(state(self), \old(state(
    self)), in, e, P, gamma);
return;
}
```

We only sketched the global approach; concerning the proof of this property at the code level, we already analyzed similar properties in PVS [Her+12]. Furthermore, the current property is simpler since it only involves the positivity of a quadratic form. State of the art SMT solvers such as Z3 [MB08] should be able to discharge the generated proof objectives without requiring the use of proof assistants.

Safety properties

Other properties can be expressed on this figure: the correct implementation of a triplex voter, of a safety redundancy construct, etc. Simulink can be used with the same observers to formalize the intended properties. Then,

thanks to the LustreC Horn encoding backend, the proof of those properties could be made. In [Die+15] and current unpublished works, we are studying means to express this induction proof performed at model level, at the code level, in an automatic fashion. This would support the verification and the preservation of the validity of properties all along the toolchain development.

Numerical accuracy

The toolchain should also be extended to address specifically floating point precision. Recent works such as [DMC15; IM13] enable the transformation of numerical expressions to minimize their floating point imprecision. Integrating these approach in the toolchain chain could improve the quality of the generated code but also provide information to analyzers about the floating point errors associated to each numerical expression.

11.2 OSDP: OCAML SEMI-DEFINITE PROGRAMMING

We presented OSDP in Section 10.3.

While already fully usable we see a lot of possible extensions for this library:

- extending the input language to enable or ease the expression of dual problems;
- provide means to pre-process the problem before submitting it to the solvers, eg. extract a full rank matrix on which to solve the problem;
- extend the list of supported solvers.

11.3 SEAL: SYSTEM ANALYSIS LIBRARY

One of the perspective is to build another Ocaml library, on top of OSDP, providing a unified frontend to perform all analyses presented in Part II, invariant computation and SOS policy iterations. As mentioned earlier these analyses were performed through different means: some were done with OSDP, others directly in Matlab with Yalmip.

As OSDP for convex optimization problems, SEAL will provide means to define a dynamical system and compute its properties or invariants as semi-algebraic constraints. While independent from OSDP, since users may want to use convex optimization without discrete dynamical system analysis, SEAL will rely on OSDP to perform all interactions with SDP solvers, including the a-posteriori soundness check of positive definiteness.

This analysis library will also be integrated in the toolchain presented in Section 11.1. We started to develop another backend for LustreC generating a simple while true loop, easing the extraction of the discrete dynamical system. When integrate the tool will be able to compute bounds on those systems and feed the toolchain with additional invariants. An interesting integration could consider both the discrete dynamical system backend to perform the analysis but also the classical C code backend on which floating-point precision analysis will have to be performed to compute bounds on the error. SEAL should also be able to consider float point errors when provided as an input. Last, as mentioned in the first section the numerical invariants will be later revalidated all along the development cycle.

12

EXTENSIONS

Once the SEAL library is available, providing easy access to all developed analyses, we would like to extend the range of systems and properties covered. This chapter enumerates without details the identified systems or properties we would like to target.

12.1 MORE SYSTEMS

Our goal is to enable the formal analysis of realistic systems, such as the ones used in civil aircraft nowadays. We are not specifically interested in the analysis of state-of-the-art non linear controllers as the ones found in academic laboratories, but rather provide applicable methods to old-school linear controller-based systems.

However, while most systems are linear, their composition makes the global system highly non linear. We believe that the capabilities to analyze polynomial systems may enable us to analyze these composed systems.

We are interested in the following constructs or systems:

LINEAR PARAMETER VARYING CONTROLLERS (LPV)

One the approach in control theory to address the control of non linear plant is to linearize the plant equations around specific points. Once the plant is linear, a linear time-invariant controller is synthesized. In case of a large domain of applicability, the controller has to be designed for multiple such linearization points.

$$\begin{aligned} x_{k+1} &= A_1 x_k && \text{when } x_k \approx p_1 \\ &\dots \\ x_{k+1} &= A_n x_k && \text{when } x_k \approx p_n \end{aligned}$$

This large domain captures, for example, the flight envelope for aircraft. A way to integrate this set of linear controllers is to build a unique linear parameter varying controller using these linear controllers. A common construct is gain scheduling; for example a linear gain interpolation in which coefficients of each individual linear controller are composed linearly[1]:

$$\begin{aligned} x_{k+1} &= (\lambda A_i + (1-\lambda) A_{i+1}) x_k \\ &\text{when } x_k = \lambda p_i + (1 - \lambda p_{i+1}) \end{aligned}$$

With current state of art, this kind of systems are hard to analyze at code level. One can consider first their restriction to switched systems, for example:

$$x_{k+1} = A_i x_k \quad \text{when } x_i \in \left[\frac{p_i + p_{i-1}}{2}, \frac{p_{i+1} + p_i}{2} \right]$$

These switched systems fit exactly into our analyzes dedicated to piecewise linear systems. But the introduction of the interpolation parameter λ generates an infinity of intermediate controllers when varying from point p_i to point p_{i+1}. While the naive idea of interpolating directly the quadratic LYAPUNOV functions P_i associated to each linear controller may seems feasible, it generates a considerably large number of term and is not directly expressible as an LMI since we have non linear terms in λ, P_1 and P_2

For example, we have $P_1 \succeq 0$ and $P_2 \succeq 0$ associated respectively to linear controllers A_1 and A_2. For each λ, we can consider the associated quadratic LYAPUNOV function

$$\lambda P_1 + (1-\lambda) P_2 \succeq 0$$

This term can be easily proved positive semi-definite, since SDP is a cone: it is closed by addition and positive external scalar multiplication. Since $\lambda \in [0, 1]$, both terms λP_1 and $(1-\lambda) P_2$ are SDP. And so does their addition.

However, concerning the inductiveness of the interpolation version, we have

$$\begin{aligned} A_1^t P_1 A_1 - P_1 &\preceq 0 \\ A_2^t P_2 A_2 - P_2 &\preceq 0 \end{aligned}$$

1 In practice these linear gain interpolation are interpolated by a single scalar since considered system are SISO.

The inductiveness property can be developed:

$$(\lambda A_1 + (1-\lambda)A_2)^\intercal(\lambda P_1 + (1-\lambda)P_2)(\lambda A_1 + (1-\lambda)A_2) - (\lambda P_1 + (1-\lambda)P_2)$$

$$= \begin{cases} \lambda^3 A_1^\intercal P_1 A_1 + \lambda^2(1-\lambda)A_1^\intercal P_1 A_2 + \\ \lambda^2(1-\lambda)A_1^\intercal P_2 A_1 + \lambda(1-\lambda)^2 A_1^\intercal P_2 A_2 + \\ (1-\lambda)\lambda^2 A_2^\intercal P_1 A_1 + \lambda(1-\lambda)^2 A_2^\intercal P_1 A_2 + \\ \lambda(1-\lambda)^2 A_2^\intercal P_2 A_1 + (1-\lambda)^3 A_2^\intercal P_2 A_2 \\ -(\lambda P_1 + (1-\lambda)P_2) \end{cases}$$

Existing works addressed this issue on the control theory side, more than often performing controller synthesis: an interpolated LYAPUNOV function is exhibited and a stabilizing controller synthesized from it, e.g. see [NLS14] in a discrete setting.

One of the direction we could consider is to search directly for a quadratic or polynomial LYAPUNOV function using our SOS framework, c.f. Sec. 5.5. Another related issue, is the analysis of such LPV controller with a floating point semantics. In practice intermediate values for the linear update A will not evolve on a line between A_i and A_{i+1} but rather on a discontinuous line because of floating-point computation. The interpolation is not strictly linear.

ANTI-WINDUP/SATURATION Controllers are often composed with safety features to avoid submitting large values to the actuators. For example one can limit the absolute value of an output signal with a bound; this is called a saturation; or impose a bound on the rate increase of the signal. These constructs are implemented with simple if-then-else's and could, at least theoretically, be addressed with our policy iteration based analyses

Another issue happens with linear controllers such as PID that contain an integrative term. In case of saturation the output value is bounded but the internal integrative term diverges. When the controller desaturates, the behavior of the controller is badly impacted by the divergent integrative term. Anti-windup patterns observe the activation of the saturation and compensate the integrative term in case of active saturation.

A classical PID controller implemented over a discretization period of T, would be implemented as:

$$\begin{aligned} x_{k+1} &= k_I * T * u_k + x_k \\ y_k &= k_P * u_k + x_{k-1} + k_D * (u_k - u_{k-1})/T \end{aligned}$$

with u_k the input signal, x_k the integrative term, and y_k the output signal. Constants k_P, k_I and k_D denote proportional, integral and derivative coefficients.

A version with saturation would give a saturated output signal s_k:

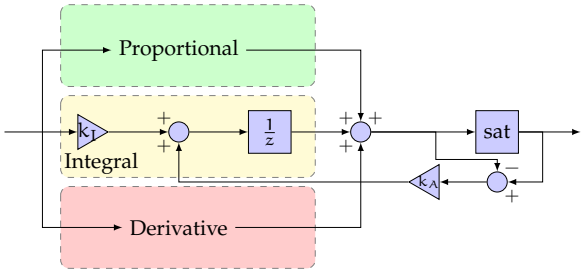
$$s_k = sat(y_k)$$

When introducing an anti-windup with gain k_A, we obtain:

$$\begin{aligned} x_{k+1} &= (s_k - y_k) * k_A + k_I * T * u_k + x_k \\ y_k &= k_P * u_k + x_{k-1} + k_D * (u_k - u_{k-1})/T \\ s_k &= sat(y_k) \end{aligned}$$

When the signal is unsaturated, the term $(s_k - y_k) * k_A$ is null and the behavior is the one of the classical PID. When it saturates, the term is negative and compensates proportionally the integrative term, limiting its divergence.

Figure 12.1 Anti-windup

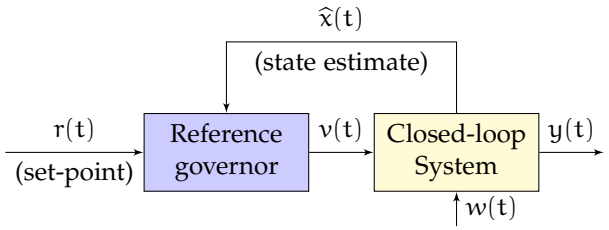


EXTENDED REFERENCE GOVERNOR (E-RG) One of the approach to optimize the behavior of an existing controller is to tune its input. For example if the reference is too large, it can filtered to a smaller value, until the initial reference becomes more accessible. This impact the behavior, providing a slower feedback without large overshoots. These techniques known as input shaping have been largely extended through reference governors patterns and extended reference governors. They enable more advanced behaviors either more aggressive or smoother than what can be achieved with purely linear controllers.

Reference governors and their extended version act as another complete controller with internal memory, that, depending on the input, the observed output and the internal state of the reference governor, control the input of the initial controller.

Existing results [GK99; KGD14] relate LYAPUNOV functions to reference governors. By construction, an approach to the design of reference governors relies on the expression of local LYAPUNOV functions. Furthermore, it could be feasible, when provided a LYAPUNOV function bounding the reachable states of the initial controller, to characterize another one specific to controller with its reference governor.

Figure 12.2 Reference governor



DISCRETE CONTROLLERS WITH CONTINUOUS PLANT Recent works [Cim12; Kon+15] addressed the issue of analyzing the satisfiability of properties including a mix of discrete and continuous equations. One of these approaches is called δ-satisfiability and combines branch-and-bound algorithms and a satisfiability core to build satisfiable models or show their absence.

On the language side, Simulink provides capabilities to express complex hybrid systems but recent works [BP13] studied more formally the semantics of such hybrid systems and proposed means to restrict the language to reasonable constructs.

One of the research directions is to extend SMT-based model checking to address more finely this combination of branch-and-bound and satisfiability targeting aircraft controllers. Another open direction is the study of stability or invariant synthesis for a pair of systems in which the controller is discrete and the plant defined by ODEs.

12.2 MORE PROPERTIES

The presented works was motivated first on bounding reachable states, then it was extended to arbitrary properties expressed as simple semi-algebraic constraints in Section 5.5.2. We were then able to focus on system-level properties such as stability and robustness analysis through vector margins.

Assuming the plant semantics is given in an analyzable form, for example a discrete piecewise polynomial system, we should be able to analyze a large set of system-level properties both at model level, and later on the implementation artifact.

A future research direction will therefore consider the large corpus of robust control literature: e.g. S-procedure for non linear systems [Yak71], matrix inequalities in control [Yak62], the use of LMI to address a variety of properties [Boy+94], or the method of integral quadratic constraints (IQC) [MR97].

Most of these works are defined in the both continuous setting and the frequency domain and their expression in a discrete setting is not a research contribution on its own. However, it may enable the analysis at code and could result on a larger impact on the study of systems.

KYP: KALMAN-YAKUBOVICH-POPOV LEMMA A fundamental theorem in control is due known as KYP, the KALMAN-YAKUBOVICH-POPOV lemma. It relates the frequency domain, time domain (with is the state space representation for computer scientist) and dissipativity of a linear system. In discrete time, one can formulate it in the following way.

Theorem 12.1 (discrete time KYP lemma) *Let us consider the following discrete linear system:*

$$\begin{cases} x_{k+1} = Ax_k + bu_k \\ y_k = Cx_k \\ x_0 = 0 \end{cases}$$

We define its transfer function $G(s)$ *as*

$$G(i\omega) = C(e^{i\omega}I - A)^{-1}B$$

The following properties are equivalent:

DISSIPATIVITY (NON-EXPANSIVENESS)

$$\sum_{k=0}^{T} \|y_k\|_2^2 \leqslant \left(\sum_{k=0}^{T} \|u_k\|_2^2\right)$$

FREQUENCY DOMAIN

$$\exists \gamma \in \mathbb{R}, \|G(i\omega)\|_\infty < \gamma$$

where $\|G(i\omega)\|_\infty$ *is the* H_∞ *norm of* G*, defined as*

$$\|G(i\omega)\|_\infty = \sup_{W(i\omega)\in H_2} \frac{\|Y(i\omega)\|_2}{\|U(i\omega)\|_2}$$

where Y *and* U *denotes the transfer functions associated to* (y_k) *and* (u_k)*, respectively.*

TIME DOMAIN

$$\exists P = P^T, \begin{bmatrix} A^TPA - P + C^TC & A^TPB + C^TD \\ B^TPA + D^TC & B^TPB + D^TD - \gamma^2 I \end{bmatrix} < 0$$

Interestingly, as recalled in [Boy+94], the result initiated the frequency domain analysis as it is known in control theory: a graphical interpretation of a frequency domain constraint implies the existence of the LYAPUNOV function and proves good behavior: POPOV criterion, circle criterion (NYQUIST), TSYPKIN criterion, and many variations. All these techniques amount to solving "by hand" an LMI.

However, since most properties are nowadays expressed on the frequency domain, for example the μ-analysis for non linear systems, it may be interesting to revisit this lemma to drive the computation of frequency domain properties through their LMI (or comparable) representation.

The work of Simon DUVERGER is to focus on this research direction, revisiting performance properties such

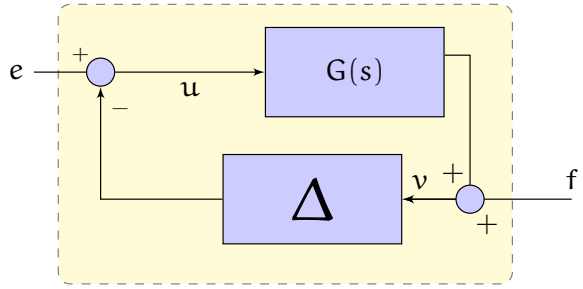
as bounded overshoot as H_∞ properties which we intend to prove through their LMI equivalent. This is still too early work to be presented here.

We refer the reader to [Ran96] for a step-by-step proof of the theorem in a continuous setting, and to [Ran16] for a more recent proof specific to positive systems.

IQC: INTEGRAL QUADRATIC CONSTRAINTS An interesting framework to analyze robustness of non linear system is the Integral Quadratic Constraints (IQC) approach [MR97; MR95].

It enables the study of stability for non linear systems. In that framework a non linear system is split into a linear $G(s)$ transfer function part, and a non linear one Δ.

Figure 12.3 Non linear system Δ



As a cut in a proof, or a loop semantics abstracted by its loop invariant, the non linear part Δ has to be accurately described by integral quadratic constraints (IQC's):

$$\int_{-\infty}^{\infty} \begin{bmatrix} \widehat{v} \\ \widehat{\Delta(v)}(j\omega) \end{bmatrix}^* \Pi(j\omega) \begin{bmatrix} \widehat{v} \\ \widehat{\Delta(v)}(j\omega) \end{bmatrix} d\omega \leqslant 0$$

where $\widehat{\cdot}$ denotes the Fourier transform of a signal. The set of such Π is convex and the associated IQC's can be easily combined. When Δ can be structured in smaller components, one can combine local IQC's.

Stability is proved by characterizing a matrix Π such that

$$\begin{bmatrix} G(j\omega) \\ I \end{bmatrix}^* \Pi(j\omega) \begin{bmatrix} G(j\omega) \\ I \end{bmatrix} < 0, \forall \omega \in \mathbb{R} \cup \{\infty\}$$

As in the previous section, this frequency domain expression can be mapped, thanks to the KYP lemma, to an equivalent LMI.

IQC's have been used to study robustness or convergence under noise of non linear systems. An interesting future direction would recast this technique in the discrete setting and perform the synthesis and the validation of this inequality on the code semantics.

CONTRACTION ANALYSIS Another interesting framework is the theory of contraction. Citing [APS08], we have the following definition: "Contraction analysis is a stability theory for non linear systems where stability is defined incrementally between two arbitrary trajectories. The existence of a contraction metric [...] ensures that a suitably defined distance between nearby trajectories is always decreasing. [...] Contraction analysis is closely related to Krasovskii's Theorem, since one can interpret the search for a contraction metric as the search for a LYAPUNOV function with a certain structure."

Other works[BS15; LS98] mentioned this relationship between contraction analysis and LYAPUNOV stability. As most concepts in control theory, it is mainly developed in the continuous setting.

To summarize roughly the idea, a system is contracting when one can exhibit this contraction metric. For a nonlinear time invariant system defined as

$$x_{k+1} = f(x_k)$$

finding a contracting region of the state space amounts to find a positive definite metric M such that

$$\frac{\partial f}{\partial x_i} M \frac{\partial f}{\partial x_i} - M < 0$$

where $\frac{\partial f}{\partial x_i}$ denotes the (discrete generalized) Jacobian of f. In case of linear systems $f(x) = Ax$ and $\frac{\partial f}{\partial x} = A$; we obtain exactly the LYAPUNOV equation.

This approach has been applied to numerous contexts but was never considered to support code level analysis. Some interesting contraction analysis papers focused on non linear dynamical systems[LS98], advanced observers proof of convergence on classes of extended Kalman filters[BS15], and the use of Sums-of-square programming to address the synthesis of the contraction metric[APS08].

Contracting systems also share nice composition properties: positive parallel composition, negative feedback, series and cascade compositions, translation and scaling, etc.

13

INVARIANTS OF DYNAMICAL SYSTEMS

Inspired by the works by Henrion, Lasserre, Magron and Korda [HK14; KHJ13a; KHJ13b; KHJ12; MHL15], we studied recently with Didier Henrion, Victor Magron and Xavier Thirioux another approach to bound precisely the reachable states, $\mathfrak{C}$ of a dynamical system. Assuming this set $\mathfrak{C}$ lives in a given compact set $\mathbf{X}$, let use denote, in the following, by $\mathbf{X}^{(\infty)}$ the set of reachable states restricted to $\mathbf{X}$:

$$\mathbf{X}^{(\infty)} := \{(\mathbf{x}_t)_{t\in\mathbb{N}} \subseteq \mathbf{X} : \mathbf{x}_{t+1} = f(\mathbf{x}_t), \forall t \in \mathbb{N}, \mathbf{x}_0 \in \mathbf{X}_0\}.$$

The idea is to express $\mathbf{X}^{(\infty)}$ as a minimization optimization problem in which we search for an inductive sub-level set, containing the initial set, and which volume, with respect to the Lebesgue measure, is minimal. Thanks to the compactness of $\mathbf{X}$, the Lebesgue measure is defined. Furthermore, when choosing an appropriate compact set X, for example an hypercube or a ball, the computation of the volume of a semialgebraic set is expressible in a linear fashion, over the moments associated to monomials.

This method is inspired by a long line of works manipulating polynomial systems properties and compact sets: In [HLS09], the authors addressed the problem of computing over-approximations of the volume of a general basic compact semialgebraic set, described by the intersection of a finite number of polynomial superlevel sets, whose coefficients are known in advance. Further work focused on over-approximating semialgebraic sets where such a description is not explicitly known: in [Las15], the author derives converging outer (resp. inner) approximations of sets defined with existential (resp. universal) quantifiers; in [MHL15], the authors approximate the image set of a compact semialgebraic set § under a polynomial map f. The current study can be seen as an extension of [MHL15], when § stands for the set of initial conditions, f represents the dynamics of a discrete-time system and only one iteration is performed from §.

We propose a hierarchy of converging convex approximations derived from an infinite-dimensional linear programming (LP) reformulation of the problem. Through moment relaxations of this LP, and characterize on the dual problem, one can compute tight over-approximations of the reachable set.

13.1 PRIMAL: MAXIMIZING MEASURE SUPPORT

The initial expression of the problem relies on the characterization of the indicator function $\mathbf{1}_{\mathfrak{C}}$ of the set $\mathfrak{C}$.

$$\mathbf{1}_{\mathbf{A}}(\mathbf{x}) := \begin{cases} 1 & \text{if } \mathbf{x} \in \mathbf{A}, \\ 0 & \text{otherwise}. \end{cases}$$

Let us introduce some definitions.

Definition 13.1 (Borel measures vector space) *Given a compact set $\mathbf{A} \subset \mathbb{R}^n$, we denote by $\mathcal{M}(\mathbf{A})$ the vector space of finite signed* Borel *measures supported on $\mathbf{A}$, namely real-valued functions from the* Borel *sigma algebra $\mathcal{B}(\mathbf{A})$.*

Definition 13.2 (measure support) *The support of a measure $\mu \in \mathcal{M}(\mathbf{A})$ is defined as the set of all points $\mathbf{x}$ such that for each open neighborhood $\mathbb{B}$ of $\mathbf{x}$, one has $\mu(\mathbb{B}) > 0$. Note that this set is closed by construction.*

Definition 13.3 (Lebesgue measure on a subset) *The restriction of the* Lebesgue *measure on a subset $\mathbf{A} \subseteq \mathbf{X}$ is $\lambda_{\mathbf{A}}(d\mathbf{x}) := \mathbf{1}_{\mathbf{A}}(\mathbf{x})\, d\mathbf{x}$., where $\mathbf{1}_{\mathbf{A}} : \mathbf{X} \to \{0, 1\}$ stands for the indicator function on $\mathbf{A}$.*

Definition 13.4 (Moments of Lebesgue measure) *The moments of the* Lebesgue *measure on $\mathbf{X}$ are denoted by*

$$y_\beta^{\mathbf{X}} := \int_{\mathbf{X}} \mathbf{x}^\beta \lambda_{\mathbf{X}}(d\mathbf{x}) \in \mathbb{R}, \quad \beta \in \mathbb{N}^n. \tag{166}$$

Definition 13.5 (Lebesgue volume) *The* Lebesgue *volume of $\mathbf{X}$ is defined by $\operatorname{vol}\mathbf{X} := y_0^{\mathbf{X}} = \int_{\mathbf{X}} \lambda_{\mathbf{X}}(d\mathbf{x})$.*

Definition 13.6 (Image measure) *Given a positive measure $\mu \in \mathcal{M}_+(\mathbf{X})$, the so-called* pushforward measure *(or* image measure*, see e.g. [AFP00, Section 1.5]) of μ under f is defined as follows:*

$$f_\# \mu(\mathbf{A}) := \mu(f^{-1}(\mathbf{A})) = \mu(\{\mathbf{x} \in \mathbf{X} : f(\mathbf{x}) \in \mathbf{A}\}),$$

for every set $\mathbf{A} \in \mathcal{B}(\mathbf{X})$.

Definition 13.7 (invariant measures) *A measure μ is invariant w.r.t. f when μ satisfies $\mu = f_\#\mu$.*

The set $\mathbf{X}^{\text{inv}}$ is defined as the union of supports of all invariant measures w.r.t. f being dominated w.r.t. $\lambda_{\mathbf{X}}$ (the restriction of the Lebesgue *measure over $\mathbf{X}$).*

Lemma 13.8 *For any given* $T \in \mathbb{N}^+$, $\alpha > 1$ *and a measure* $\mu_0 \in \mathcal{M}(\mathbf{X}_0)$, *there exist measures* $\mu_T, \nu \in \mathcal{M}(\mathbf{X})$ *which satisfy the discrete* Liouville *Equation:*

$$\mu_T + \nu = \alpha f_\# \nu + \mu_0 \,. \tag{167}$$

Here, T denotes a number of transition steps which is left free. mu_T denotes the occupation measure restricted over of the states reachable after T transitions.

Using Liouville equation, that encodes a sort of certain conservation law for measure supports, we can derive the following primal formulation. To approximate the set $\mathbf{X}^* := \mathbf{X}^{\text{inv}} \cup \mathbf{X}^{(\infty)}$, one considers the infinite-dimensional linear programming (LP) problem, for a given $\alpha > 1$:

$$\begin{aligned} p^* := \sup_{\mu_0,\mu,\hat{\mu},\nu} \quad & \int_{\mathbf{X}} \mu \\ \text{s.t.} \quad & \mu + \hat{\mu} = \lambda_{\mathbf{X}} \,, \\ & \mu + \nu = \alpha f_\# \nu + \mu_0 \,, \\ & \mu_0 \in \mathcal{M}_+(\mathbf{X}_0) \,, \quad \mu, \hat{\mu}, \nu \in \mathcal{M}_+(\mathbf{X}) \,. \end{aligned} \tag{168}$$

Intuitively, μ denotes the measure of terminal reachable states. However, since the trace is not bounded by the equation, it can denote any reachable state. The second constraint is the so-called Liouville equation: it encodes the system semantics within the constraints.

13.2 DUAL: MINIMIZING POSITIVE FUNCTIONS

Positive measures are not fitted with a scalar product and are then not Hilbert spaces. The (pre-)dual of positive measures is the set of positive continuous functions $\mathcal{C}(\mathbf{X})$. Using the elements of duality introduced in Sect. 4.2.3, we can construct the dual problem of our maximization of measure support:

$$\begin{aligned} d^* := \inf_{\nu,w} \quad & \int w(\mathbf{x})\, \lambda_{\mathbf{X}}(d\mathbf{x}) \\ \text{s.t.} \quad & \nu(\mathbf{x}) \geqslant 0, \quad \forall \mathbf{x} \in \mathbf{X}_0, \\ & w(\mathbf{x}) \geqslant 1 + \nu(\mathbf{x}), \quad \forall \mathbf{x} \in \mathbf{X}, \\ & w(\mathbf{x}) \geqslant 0, \quad \forall \mathbf{x} \in \mathbf{X}, \\ & \alpha\, \nu(f(\mathbf{x})) \geqslant \nu(\mathbf{x}), \quad \forall \mathbf{x} \in \mathbf{X}, \\ & \nu, w \in \mathcal{C}(\mathbf{X}). \end{aligned} \tag{169}$$

Intuitively, we are interested in the superlevel set of the function $\nu(x)$. $\nu(x) \geqslant 0$ on initial states, and this positive is preserved along system trajectories: this is encoded by the constraint $\alpha\, \nu(f(\mathbf{x})) \geqslant \nu(\mathbf{x})$. If a state x is reachable, then $\nu(x) \geqslant 0$ and s does its successor $\nu(f(\mathbf{x})) \geqslant \nu(\mathbf{x}) \geqslant 0$. However, ν can be anything outside reachable states. The positive function w is such that, when $\nu(x) > 0$ then $w(x)$ is above a specific threshold, here 1. And since $w(x)$ is positive over X, one can minimize its volume.

13.3 HIERARCHY OF ABSTRACTIONS

Using Henrion and Lasserre's approach [HK14; KHJ13a; KHJ12; Laso1], we abstract positive functions by SOS polynomials. Thanks to theoretical results, the method converges in volume towards X^*.

Positivity of polynomial expressions under certain semialgebraic constraints, is ensured by imposing them to be an SOS polynomial of a given degree $2m$, as we did in Section 5.5.3, for example in Eq.(116). The problem to solve becomes:

$$\begin{aligned} d_r^* := \inf_{\nu,w} \quad & \sum_{\beta \in \mathbb{N}^n_{2r}} w_\beta z^{\mathbf{X}}_\beta \\ \text{s.t.} \quad & \nu = \sigma_0 - \sum_{j=1}^{n_{in}} \sigma_j^0 r_j^{in} \,, \\ & w - 1 - \nu = \sigma_1 - \sum_{j=1}^{n_X} \sigma_j^1 r_j^X \,, \\ & \alpha\, \nu \circ f - \nu = \sigma_2 - \sum_{j=1}^{n_X} \sigma_j^2 r_j^X \,, \\ & w = \sigma_3 - \sum_{j=1}^{n_X} \sigma_j^3 r_j^X \,, \\ & \nu, w \in \mathbb{R}_{2m}[\mathbf{x}]. \\ & \forall i \in \{0,1,2,3\}, \sigma_i \in \Sigma[x] \ , \ \deg(\sigma_i) \leqslant 2m \ , \\ & \forall j = 1, \ldots, n_{in} \ , \ \sigma_j \in \Sigma[x] \ , \ \deg(\sigma_j r_j^{in}) \leqslant 2m \ , \\ & \forall j = 1, \ldots, , n_X, i \in \{1,2,3\} \ , \ \sigma_j^i \in \Sigma[x] \ , \\ & \deg(\sigma_j^i r_j^{in}) \leqslant 2m \ , \end{aligned} \tag{170}$$

where, as in 5.5.3, initial states belong to the semialgebraic set $X^0 = \left\{ x \,\middle|\, \bigwedge_{j \in [1, n_{in}]} r_j^{in}(x) \leqslant 0 \right\}$, and the compact set X is defined as $\left\{ x \,\middle|\, \bigwedge_{j \in [1, n_X]} r_j^X(x) \leqslant 0 \right\}$.

One of the key aspects is the capabilities to express the volume of the zero superlevelset $w(x) \geqslant 0$ within the compact $\mathbf{X}$: $\int w(\mathbf{x})\, \lambda_{\mathbf{X}}(d\mathbf{x})$. Thanks to [Laso1], when w is of fixed degree and $\mathbf{X}$ has a simple shape such as a unit ball, we can pre-compute the moment $z^{\mathbf{X}}_\beta = \int_{\mathbf{X}} \beta(x) dx$ associated to each monomial β and compute the simpler $\sum_{\beta \in \mathbb{N}^n_{2r}} w_\beta z^{\mathbf{X}}_\beta$ where w_β denotes the coefficient associated to the monomial β in w.

13.4 EXPERIMENTS

13.4.1 *Toy Example*

First, let us consider the made-up discrete-time polynomial system defined by

$$x_1^+ := \frac{1}{2}(x_1 + 2x_1x_2)\,,$$
$$x_2^+ := \frac{1}{2}(x_2 - 2x_1^3)\,,$$

with initial states constraints $\mathbf{X}_0 := \{\mathbf{x} \in \mathbb{R}^2 : (x_1 - \frac{1}{2})^2 + (x_2 - \frac{1}{2})^2 \leqslant \frac{1}{4^2}\}$ and general state constraints within the unit ball $\mathbf{X} := \{\mathbf{x} \in \mathbb{R}^2 : \|x\|_2^2 \leqslant 1\}$. On Figure 13.1, we represent in light gray the outer approximations $\mathbf{X}^r$ of $\mathbf{X}^*$ obtained by our method, for increasing values of the relaxation order r (from $r = 4$ to $r = 14$). On each figure, the colored sets of points are obtained by simulation for the first 7 iterates. More precisely, each colored set correspond to (under approximations of) the successive image sets $\mathbf{X}_1, \ldots, \mathbf{X}_7$ of the points obtained by uniform sampling of $\mathbf{X}_0$ under $f, \ldots, f^7$ respectively. The set $\mathbf{X}_0$ is colored in blue and the set $\mathbf{X}_7$ is colored in red. The dotted circle represents the boundary of the unit ball $\mathbf{X}$. Figure 13.1 shows that the over approximations are already quite tight for low degrees.

13.4.2 FitzHugh-Nagumo *Neuron Model*

Here we consider the discretized version (taken from [Ben+12, Section 5]) of the FitzHugh-Nagumo model [Fit61], which is originally a continuous-time polynomial system modeling the electrical activity of a neuron:

$$x_1^+ := x_1 + 0.2(x_1 - x_1^3/3 - x_2 + 0.875)\,,$$
$$x_2^+ := x_2 + 0.2(0.08(x_1 + 0.7 - 0.8x_2))\,,$$

with initial states constraints $\mathbf{X}_0 := [1, 1.25] \times [2.25, 2.5]$ and general state constraints $\mathbf{X} := \{\mathbf{x} \in \mathbb{R}^2 : (\frac{x_1 - 0.1}{3.6})^2 + (\frac{x_2 - 1.25}{1.75})^2 \leqslant 1\}$. Figure 13.2 illustrates that the over approximations provide useful indications on the system behavior, in particular for higher values of r. Indeed, $\mathbf{X}^{10}$ and $\mathbf{X}^{12}$ capture the presence of the central "hole" made by periodic trajectories and $\mathbf{X}^{14}$ shows that there is a gap between the first discrete-time steps and the iterations corresponding to these periodic trajectories.

13.5 ISSUES/FUTURE DIRECTIONS

From an invariant synthesis perspective, in computer science, this approach is interesting. While all other methods, with Lyapunov function, were searching for a positive function decreasing over trajectories, no theoretical convergence properties were available. Here, the search of the Lyapunov-like function is constraint by minimizing the volume of the inductive invariant set. In the quadratic case, minimizing the integral (w.r.t. Lebesgue measure) of w over $\mathbf{X}$ is equivalent to maximizing the integral of the trace of the matrix $\mathbf{x}\,\mathbf{x}^T\mathbb{V}$ on $\mathbf{X}$.

This optimization problem can also be expressed when considering piecewise polynomial systems, as shown in the examples. As for the work presented in Sec. 5.5 it scales linearly in the number of piecewise components.

However, while the theory is attractive, we faced multiples issues.

CERTIFICATION OF THE COMPUTATION RESULTS We faced lot of issues to prove that the generated SDP solution, which reconciles into an SOS polynomial, was actually positive. Since our early padding validation and conservative Cholesky certificate of positiveness were not validating most solutions, we evaluated other means to show soundness, and therefore positiveness.

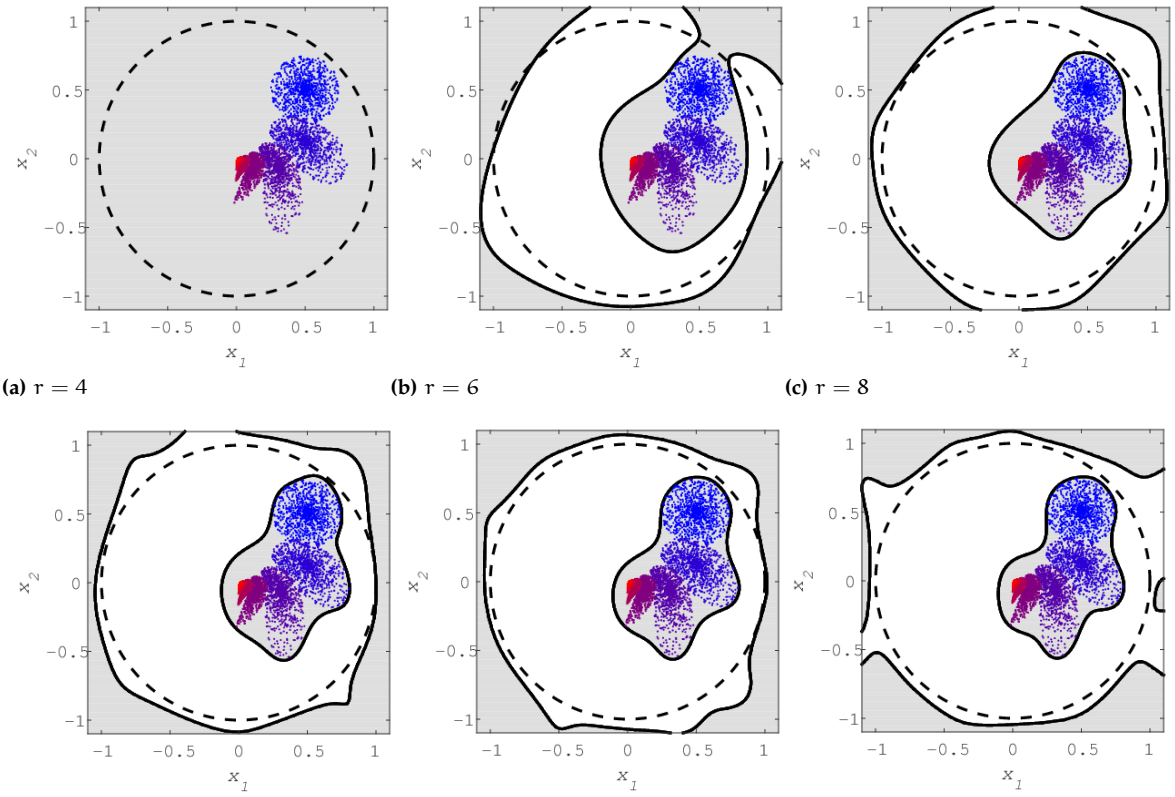
Thanks to the existence of the bigger compact $\mathbf{X}$, we relied on Bernstein polynomial to compute the minimum of the value on the set $\mathbf{X}$. These computations shown to be extremely expensive in terms of computer resources and proved the invalidity of most results. Due to numerical issues, without padding, most solutions were invalidated: they were globally positive, except slightly negative near some violation point, e.g. in case of empty interior problems.

Recent changes in the choice of the parameters, and the scaling applied to the system, seem to provide better results.

TUNING THE ANALYSIS To avoid unsound results we had to precondition the system before the analysis. This amounts to applying linear transformations on each variable in order to adapt the reachable region to the unit circle. While this is a reasonable condition to avoid large differences between coefficients of the same resulting function, no clear theoretical motivations appear to justify this scale. Similarly, the choice of a parameter $\alpha > 1$ impacted a lot the analyzes.

THEORETICAL CONSTRAINT: INVARIANT MEASURES. A more serious concern is the definition of X^* while we are interested only in X_∞. Initially the idea was to capture X_∞, but because of issues in the proof, we discovered that the encoding was adding as reachable points unfeasible ones. This can range from non reachable fixpoints, co-limit cycles or strange attractors. If one consider the simple linear system that rotate its input without contraction, the reachable state space is the closure by rotation of the initial one. But, with the presented method, if $X^0 = \|x\|_2^2 < .5$ while $X = \|x\|_2^2 < 1$ then, by rotation, we have $X^\infty = X^0$ while the method select all X.

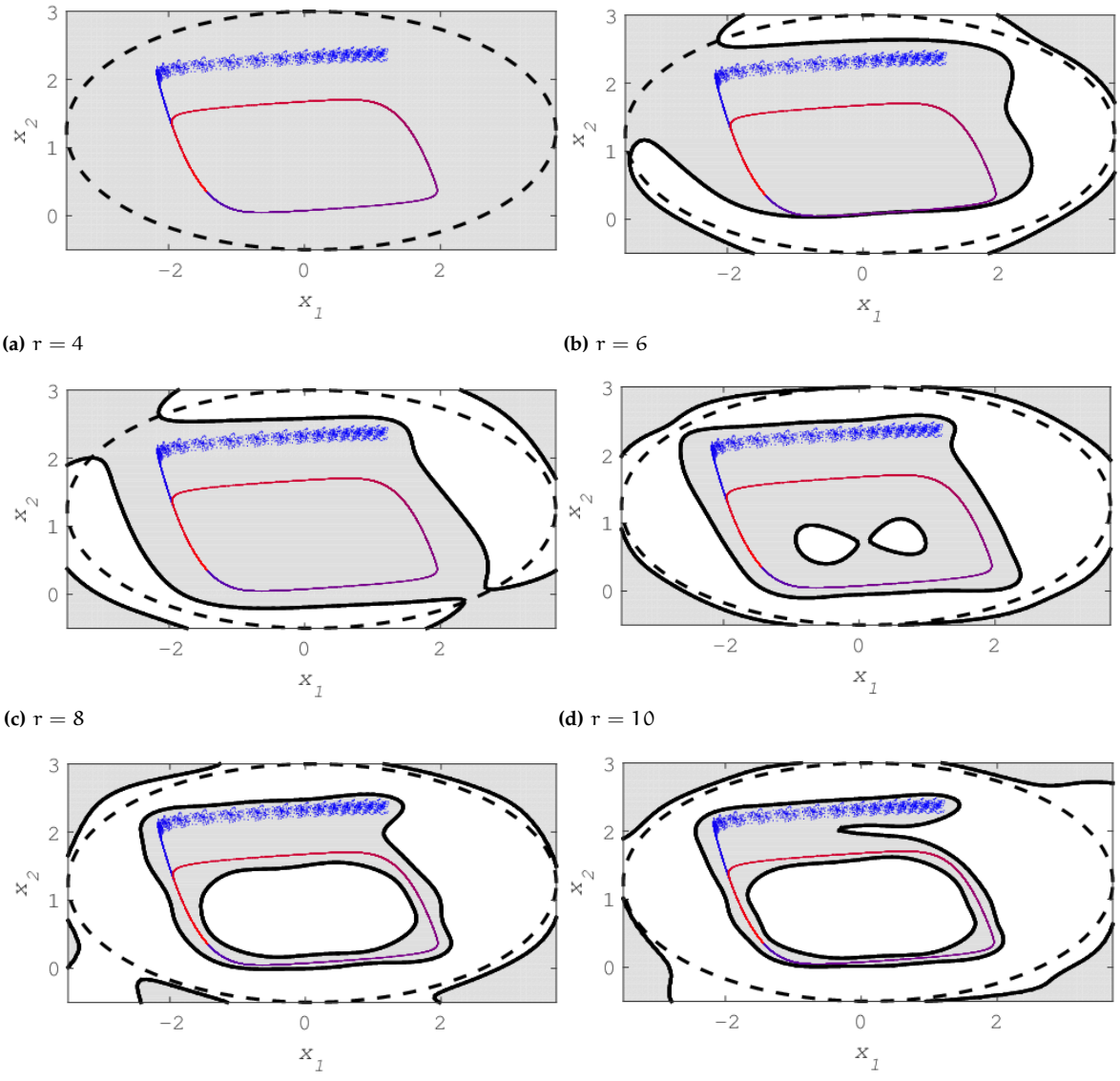
Figure 13.1 Outer approximations $\mathbf{X}^r$ (light gray) of $\mathbf{X}^*$ (color dot samples) for Example 13.4.1, from $r = 4$ to $r = 14$.



(a) $r = 4$ **(b)** $r = 6$ **(c)** $r = 8$

(d) $r = 10$ **(e)** $r = 12$ **(f)** $r = 14$

Figure 13.2 Outer approximations $\mathbf{X}^r$ (light gray) of $\mathbf{X}^*$ (color dot samples) for Example 13.4.2, from $r = 4$ to $r = 14$.



(a) $r = 4$ **(b)** $r = 6$

(c) $r = 8$ **(d)** $r = 10$

(e) $r = 12$ **(f)** $r = 14$

14

PROVING THE IMPLEMENTATION OF CONVEX OPTIMIZATION ALGORITHMS

The applications of optimization algorithms are not limited to large scale, off-line problems on the desktop. They also can perform in a real-time setting as a part of safety-critical systems in guidance, navigation and control. For example, modern aircraft often have redundant control surface actuations, which allows for reconfiguration and recovery in case of emergency. The precise re-allocation of the actuation resources can be posed, in the simplest case, as a linear optimization problem that needs to be solved in real-time [Bodo2]. More recent famous uses include the pinpoint planetary landing of rockets [AIB13; BAI12] as performed by SpaceX Falcon9 and BlueOrigin NewShepard.

In contrast to off-line desktop optimization applications, real-time embedded optimization code needs to satisfy a higher standard of quality, if it is to be used within a safety-critical system. Some important criteria for judging the quality of an embedded code include the predictability of its behaviors and whether or not its worst case computational time can be bounded. Several authors including Richter [RJM13], Feron and McGovern [McGoo; MF98] have worked on the certification problem for on-line optimization algorithms used in control, in particular on worst-case execution time issues. In those cases, the authors have chosen to tackle the problem at a high level of abstraction. For example, McGovern reexamined the proofs of computational bounds on interior-point methods for semi-definite programming; however he stopped short of using the proofs to analyze the implementations of interior-point methods.

While being usable in practice in time-critical settings, eg. [Jer+14], the adoption in civil aircraft of optimization-based control such as model predictive control (MPC) or emergency trajectory planning, requires a deeper understanding of these algorithms and, more specifically, the proof of their implementation.

Together with a team of people including Timothy Wang, Romain Jobredeaux, Marc Pantel, Éric Féron and Didier Henrion, we proposed in [Wan+16b] a Lyapunov-based annotation of an SDP algorithm showing the convergence of the method and feasibility of the computed solution. This work was mainly theoretical and not supported by actual proofs at code level.

The current work of Guillaume Davy is to address specifically the proof, at code level, of these properties, for interior point method, in the LP setting.

14.1 FORMAL PROPERTIES

For a classical interior point method, a sound algorithm shall:

- preserve the intermediate values within a given neighborhood of the central path;
- provide sufficient improvement at each iteration to ensure convergence.

Recall that an interior point method amounts to solving a sequence of Newton problems that corresponds to the linearization of the cost function along the central path (see Chapter 10). Nesterov and Nemirovski provided in [NN94] precise bounds and constructive proofs of convergence.

Let us see the theoretical arguments in the SDP setting.

14.1.1 *SDP Problem*

Let $n, m \in \mathbb{N}$, $F_0 \in \mathbb{S}^n$, $F_1, F_2, \ldots, F_m \in \mathbb{S}^n$, and $b = \begin{bmatrix} b_1 & b_2 & \ldots & b_m \end{bmatrix}^T \in \mathbb{R}^m$. Consider an SDP problem of the form in equation (171). The linear objective function $\langle b, p \rangle$ is to be minimized over all vectors $p = \begin{bmatrix} p_1 & \ldots & p_m \end{bmatrix}^T \in \mathbb{R}^m$ under the semi-definite constraint $F_0 + \sum_{i=1}^m p_i F_i \preceq 0$. The variable $X = -F_0 - \sum_{i=1}^m p_i F_i$ is introduced for convenience of notation.

$$\begin{aligned} \inf_{p,X} \quad & \langle b, p \rangle, \\ \text{subject to} \quad & F_0 + \sum_{i=1}^m p_i F_i + X = 0 \\ & X \succeq 0. \end{aligned} \tag{171}$$

We denote the SDP problem in (171) as the *primal* form. Another SDP problem (172) is called the *dual* form, and it is closely related to the primal form. In the dual formulation, the linear objective function $\langle F_0, Z \rangle$ is to be maximized over the intersection of positive semi-definite cone

$\{Z \in \mathbb{S}^n | Z \succeq 0\}$ and a convex region defined by m affine equality constraints.

$$\begin{aligned} \sup_{Z} \quad & \langle F_0, Z\rangle, \\ \text{subject to} \quad & \langle F_i, Z\rangle + b_i = 0, \quad i = 1, \ldots, m \\ & Z \succeq 0. \end{aligned} \tag{172}$$

The primal and dual feasible sets are defined as

$$\begin{aligned} \mathcal{F}^p &= \left\{ X | \exists p \in \mathbb{R}^m \text{ such that } X = -F_0 - \sum_{i=1}^{m} p_i F_i \succeq 0 \right\}, \\ \mathcal{F}^d &= \{Z | \langle F_i, Z\rangle + b_i = 0, Z \succeq 0\}. \end{aligned} \tag{173}$$

For any primal-dual pair (X, Z) in the feasible sets in (173), the primal cost $\langle b, p\rangle$ is always greater than or equal to the dual cost $\langle F_0, Z\rangle$. The difference between the primal and dual costs for a feasible pair (X, Z) is called the *duality gap*. It is a measure of the optimality of a primal-dual pair. The smaller the duality gap is, the closer to optimal the solution pair (X, Z) is. For (171) and (172), the duality gap is the function

$$G(X, Z) = \operatorname{Tr}(ZX). \tag{174}$$

Assuming both primal and dual feasible sets are not empty, there exists an optimal primal-dual pair (X^*, Z^*) such that

$$\operatorname{Tr}(Z^* X^*) = 0. \tag{175}$$

14.1.2 *Interior point method: short-step path-following primal-dual algorithm in* Kojima, Shindoh, *and* Hara *[KSH97]*

Interior point methods represent a family of algorithms to solve a convex problem, SDP interior points implement the approach by providing a way to compute the *direction*, and 1. a *step* length while remaining in the feasible set – semi-definite matrices – near the central path, and reducing the duality gap of the next iterate.

In [Wan+16b], we choose a short-step path-following primal-dual algorithm in Kojima, Shindoh, and Hara [KSH97]. At each loop iteration the search directions $(\Delta X, \Delta Z, \delta p)$ are computed by solving the Newton equations:

$$\begin{aligned} \langle F_i, \Delta Z\rangle &= 0, \\ \sum_{i}^{m} \delta p_i F_i + \Delta X &= 0, \\ H_T(Z\Delta X + \Delta Z X) &= \sigma\mu I - H_T(ZX), \end{aligned} \tag{176}$$

where H_T is the symmetrizing linear operator

$$H_T : M \to \frac{1}{2}\left(TMT^{-1} + \left(TMT^{-1}\right)^{\intercal}\right), \tag{177}$$

for some invertible scaling matrix T. The iterates are then updated using a fixed step-length of $\alpha = 1$, which guarantees that X and Z remains positive-definite.

The algorithm is described in Figure 14.1.

Table 14.1: Primal-Dual Short Path Interior-Point Algorithm

Input: $F_0 \succ 0$, $F_i \in \mathbb{S}^n, i = 1, \ldots, m$, $b \in \mathbb{R}^m$
ϵ: required optimality

1. Initialize:
   Compute Z such that $\langle F_i, Z\rangle = -b_i, i = 1, \ldots, m$;
   Let $X \leftarrow \hat{X}$; // $\hat{X}$ *is some positive-definite matrix*
   Compute p such that $\sum_i^m p_i F_i = -X_0 - F_0$;
   Let $\mu \leftarrow \frac{\langle Z, X\rangle}{n}$;

   Let $\sigma \leftarrow \hat{\sigma}$ where $0 < \hat{\sigma} < 1$;
   Let $\alpha \leftarrow 1$;
   Let $n \leftarrow \operatorname{sz} F_i$;
   Let $m \leftarrow \operatorname{sz} b_i$;
2. **while** $n\mu > \epsilon$ {
3. Let $\psi_- \leftarrow \langle Z, X\rangle$;
4. Let $T \leftarrow \mathcal{T}$ where $\mathcal{T}$ is an invertible matrix;
5. Compute $(\Delta Z, \Delta X, \delta p)$ that satisfies (176);
6. Let $Z \leftarrow Z + \alpha\Delta Z$, $X \leftarrow X + \alpha\Delta X$, $p \leftarrow p + \alpha\delta p$;
7. Let $\psi \leftarrow \langle Z, X\rangle$;
8. Let $\mu \leftarrow \frac{\langle Z, X\rangle}{n}$;
}

An instantiation for a generic optimization problem in control is presented in Figure 14.1. This is Matlab code.

14.1.3 *Program properties*

An important property of the program in Figure 14.1 is the upper bound on the number of loop iterations required for the program to converge to the required optimality. We want to express this high-level property at the level of the code. The rate of convergence is determined by the amount of reduction in the duality gap ψ during each Newton step. We use the following result from [Mon97].

Theorem 14.1 *If the centrality parameter $\sigma \in (0, 1)$ in Figure 14.1 satisfies the inequality*

$$\frac{\gamma^2 + n + n\sigma^2 - 2n\sigma}{2(1-\gamma)^2 \sigma} \leqslant \gamma \tag{178}$$

for some $\gamma \in (0, 0.5]$, *then*

1.
$$\mathrm{Tr}\,(ZX) - \sigma\,\mathrm{Tr}\,(Z_{-}X_{-}) = 0 \tag{179}$$

2.
$$\left\| Z^{0.5} X Z^{0.5} - \frac{\mathrm{Tr}\,(ZX)}{n} I \right\|_F \leqslant \gamma \frac{\mathrm{Tr}\,(ZX)}{n} \tag{180}$$

holds throughout the execution of the program loop.

Figure 14.1 Optimization program based on the short-step primal-dual interior-point algorithm in Matlab

Matlab
```
F0=[1, 0; 0, 0.1];
F1=[-0.750999 0.00499; 0.00499 0.0001];
F2=[0.03992 -0.999101; -0.999101 0.00002];
F3=[0.0016 0.00004; 0.00004 -0.999999];
b=[0.4; -0.2; 0.2];
n=length(F0);
m=length(b);
Ft=[vecs(F1), vecs(F2), vecs(F3)];
F=Ft';
Z=mats(lsqr(F,-b),n);
X=[0.3409 0.2407; 0.2407 0.9021];
epsilon=1e-8;
sigma=0.75;
psi=trace(Z*X);
P=mats(lsqr(Ft,vecs(-X-F0)),n);
p=vecs(P);
mu=psi/n;
while (n*mu>epsilon)
    Xm=X;
    Zm=Z;
    pm=p;
    mu=trace(Zm*Xm)/n;
    Zh=Zm^(0.5);
    Zhi=Zh^(-1);
    G=krons(Zhi,transpose(Zh)*Xm,n,m);
    H=krons(Zhi*Zm,transpose(Zh),n,m);
    Ginv=G^(-1);
    r=vecs(sigma*mu*eye(n,n)-Zh*Xm*Zh);
    g=-F*Ginv*r;
    B=F*Ginv*H*Ft;
    dpm=B^(-1)*g;
    dxm=-Ft*dpm;
    dzm=Ginv*r-Ginv*H*dxm;
    p=pm+dpm;
    X=Xm+mats(dxm,n);
    Z=Zm+mats(dzm,n);
    psi=trace(Z*X);
end
```

The condition $\left\| Z^{0.5} X Z^{0.5} - \frac{\mathrm{Tr}\,(ZX)}{n} I \right\|_F \leqslant \gamma \frac{\mathrm{Tr}\,(ZX)}{n}$ implies that the iterates (X, Z) remains within the γ-neighborhood of the *central path* i.e. the set of $X \succ 0$, $Z \succ 0$ such that $ZX = \frac{\mathrm{Tr}(ZX)}{n} I, X \in \mathcal{F}^p, Z \in \mathcal{F}^d$. The central path condition in (180) guarantees $X \succ 0$ and $Z \succ 0$ throughout the execution of the loop.

As an consequence of (179), we have the following result for the example optimization program.

Lemma 14.2 *Given an optimality requirement* ϵ *and an initial duality gap* $\epsilon_0 = \mathrm{Tr}\,(Z_0 X_0)$. *If* σ *satisfies the assumption of Theorem 14.1, then the example optimization program in Figure 14.1 is guaranteed to terminate with* $\mathrm{Tr}\,(Z_k X_k) \leqslant \epsilon$ *in* $k \geqslant \frac{\log \epsilon^{-1} \epsilon_0}{\log (\sigma^{-1})}$ *number of iterations.*

To use Theorem 14.1, there must exist a $0 < \gamma \leqslant 0.5$ and a $0 < \sigma < 1$ such that the inequality in (178) holds. One possible choice is $\gamma = 0.3105$ and $\sigma = 0.75$, which is the case in our Matlab implementation. With $\sigma = 0.75$ we get the loop invariant

$$\mathrm{Tr}\,(ZX) - 0.75\,\mathrm{Tr}\,(Z_{-}X_{-}) = 0, \tag{181}$$

which will be used in the annotation of the example implementation. Using (14.2), and the fact that $\epsilon_0 = 0.3419$, the Matlab program in Figure 14.1 is guaranteed to converge to the required optimality of 1×10^{-8} within 62 loop iterations.

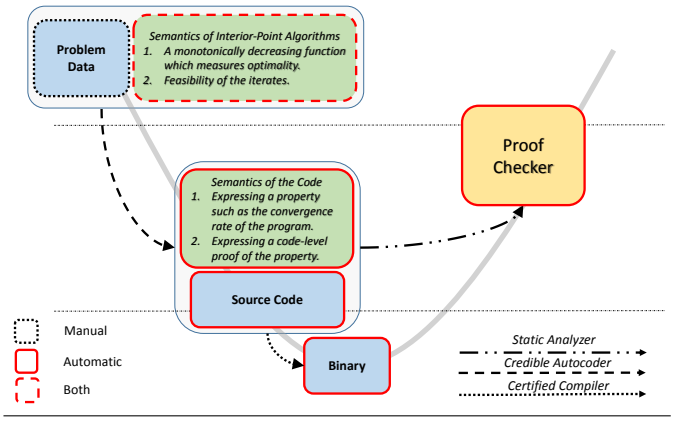
14.2 IMPLEMENTATION

These first results are preliminary in the sense that they are not validated at code level. One interesting aspect of the example of Fig. 14.1 is the specialization of the optimization solver for a specific instance. This is typically the case when embedding optimization to perform online computation of trajectories, for example.

This is compatible with our view of autocoded controllers, here optimizers, that are produced along with their proof arguments.

The general process is sketched in Figure 14.2.

Figure 14.2 Visualization of autocoding and verification process For Optimization Algorithms



In the LP setting, Guillaume Davy implemented in Python a code generator that, considering the description of a linear problem, produces the C code and its ACSL annotations. Figure 14.3 presents the framework.

Figure 14.3 Python-based autocoding framework

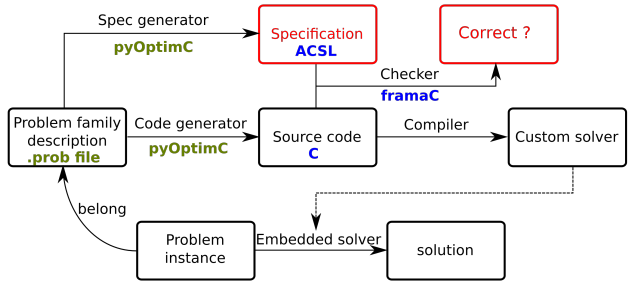


Figure 14.4 C code pure primal interior point for LP with ACSL annotations.

C+ACSL

```
double A[M*N], b[M], c[N];
double t, dt;
double x[N], dx[N];

/*@ requires acc(0, x);
  @ ensures dot(c, x) - dot(c, sol(A, b, c))
       < EPSILON;
  @ ensures A * x > b; */
void pathfollowing() {
    t = 0;
    /*@ loop invariant acc(t, x);
        loop invariant A * x > b;
        loop invariant t > LOWER(l); */
    for (unsigned int l = 0;l<NBR;l++)
    {
        compute_pre();
        compute_dt();
        compute_dx();
        t = t + dt;
        for(unsigned int i = 0;i<N;i++)
            x[i] = x[i] + dx[i];
    }
}
```

The current algorithm considered is a pure primal interior method for LP, presented in [NN94]. The generated code follows the general format of interior point method. The code is generated with ACSL, as presented in Figure 14.4.

Additional ACSL predicates provide the definition of `acc` central path condition, convergence with respect to theoretical solution `sol`, and feasibility of iterates `A * x > b`. The loop invariant $t > \mathrm{LOWER}(l)$ ensures that each step has a step length sufficient to ensure convergence in a given number of iterations.

Current ongoing work addresses the proof in Coq of these contracts, and the automatization on each optimization instance of these proofs. The perspectives are to extend the approach to more complex convex sets such as QP, SOCP or SDP.

BIBLIOGRAPHY

[AIB13] Behçet Açikmese, John M. Carson III, and Lars Blackmore. "Lossless Convexification of Nonconvex Control Bound and Pointing Constraints of the Soft Landing Optimal Control Problem." In: *IEEE Trans. Contr. Sys. Techn.* 21.6 (2013), pp. 2104–2113. DOI: 10.1109/TCST.2012.2237346. URL: http://dx.doi.org/10.1109/TCST.2012.2237346 (page 127).

[Bau+08] Patrick Baudin, Jean-Christophe Filliâtre, Claude Marché, Benjamin Monate, Yannick Moy, and Virgile Prevosto. *ACSL: ANSI/ISO C Specification Language*. frama-c.cea.fr/acsl.html. 2008. URL: frama-c.cea.fr/acsl.html (pages 13, 79, 115).

[AGW15] Assalé Adjé, Pierre-Loïc Garoche, and Alexis Werey. "Quadratic Zonotopes - An Extension of Zonotopes to Quadratic Arithmetics." In: *Programming Languages and Systems - 13th Asian Symposium, APLAS 2015, Pohang, South Korea, November 30 - December 2, 2015, Proceedings*. 2015, pp. 127–145. DOI: 10.1007/978-3-319-26529-2_8 (page 95).

[AGG10] Assalé Adjé, Stéphane Gaubert, and Eric Goubault. "Coupling Policy Iteration with Semi-definite Relaxation to Compute Accurate Numerical Invariants in Static Analysis." In: *ESOP*. Ed. by A. D. Gordon. Vol. 6012. Lecture Notes in Computer Science. Springer, 2010, pp. 23–42. ISBN: 978-3-642-11956-9 (pages 60, 63).

[AJ13] Amir Ali Ahmadi and Raphael M. Jungers. "Switched stability of nonlinear systems via SOS-convex Lyapunov functions and semidefinite programming." In: *CDC'13*. 2013, pp. 727–732 (page 55).

[All+15] Xavier Allamigeon, Stéphane Gaubert, Eric Goubault, Sylvie Putot, and Nikolas Stott. "A scalable algebraic method to infer quadratic invariants of switched systems." In: *2015 International Conference on Embedded Software, EMSOFT 2015, Amsterdam, Netherlands, October 4-9, 2015*. Ed. by Alain Girault and Nan Guan. IEEE, 2015, pp. 75–84. ISBN: 978-1-4673-8079-9. DOI: 10.1109/EMSOFT.2015.7318262. URL: http://dx.doi.org/10.1109/EMSOFT.2015.7318262 (page 56).

[AFP00] Luigi Ambrosio, Nicola Fusco, and Diego Pallara. *Functions of bounded variation and free discontinuity problems*. Oxford mathematical monographs. Autres tirages : 2006. Oxford, New York: Clarendon Press, 2000. ISBN: 0-19-850245-1. URL: opac.inria.fr/record=b1096464 (page 121).

[AA00] Erling D. Andersen and Knud D. Andersen. "The Mosek Interior Point Optimizer for Linear Programming: An Implementation of the Homogeneous Algorithm." English. In: *High Performance Optimization*. Ed. by Hans Frenk, Kees Roos, Tamás Terlaky, and Shuzhong Zhang. Vol. 33. Applied Optimization. Springer US, 2000, pp. 197–232. ISBN: 978-1-4419-4819-9. DOI: 10.1007/978-1-4757-3216-0_8. URL: dx.doi.org/10.1007/978-1-4757-3216-0_8 (page 33).

[App11] Andrew W. Appel. "Verified Software Toolchain - (Invited Talk)." In: *Programming Languages and Systems - 20th European Symposium on Programming, ESOP 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*. Vol. 6602. LNCS. 2011, pp. 1–17. ISBN: 978-3-642-19717-8. DOI: 10.1007/978-3-642-19718-5_1. URL: dx.doi.org/10.1007/978-3-642-19718-5_1 (pages 10, 13).

[AM08] Karl Johan Astrom and Richard M. Murray. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton, NJ, USA: Princeton University Press, 2008. ISBN: 0691135762, 9780691135762 (page 19).

[APS08] Erin M. Aylward, Pablo A. Parrilo, and Jean-Jacques E. Slotine. "Stability and robustness analysis of nonlinear systems via contraction metrics and SOS programming." In: *Automatica* 44.8 (2008), pp. 2163–2170. DOI: 10.1016/j.automatica.2007.12.012. URL: dx.doi.org/10.1016/j.automatica.2007.12.012 (page 120).

[BT07] Clark Barrett and Cesare Tinelli. "CVC3." In: *Proceedings of the 19th International Conference on Computer Aided Verification (CAV '07)*. Ed. by Werner Damm and Holger Hermanns. Vol. 4590. Lecture Notes in Computer Science. Berlin, Germany. Springer-Verlag, July 2007, pp. 298–302 (page 11).

[Bau+02] Patrick Baudin, Anne Pacalet, Jacques Raguideau, Dominique Schoen, and Nicky Williams. "CAVEAT: A Tool for Software Validation." In: *2002 International Conference on Dependable Systems and Networks (DSN 2002), 23-26 June 2002, Bethesda, MD, USA, Proceedings*. IEEE Computer Society, 2002, p. 537. ISBN: 0-7695-1597-5. DOI: 10.1109/DSN.2002.1028953. URL: dx.doi.org/10.1109/DSN.2002.1028953 (page 10).

[Ben+12] Mohamed Amin Ben Sassi, Romain Testylier, Thao Dang, and Antoine Girard. "Reachability Analysis of Polynomial Systems Using Linear Programming Relaxations." In: *ATVA 2012* (2012). Ed. by Supratik Chakraborty and Madhavan Mukund, pp. 137–151. DOI: 10.1007/978-3-642-33386-6_12. URL: dx.doi.org/10.1007/978-3-642-33386-6_12 (page 123).

[Bie+08] Dariusz Biernacki, Jean-Louis Colaço, Grégoire Hamon, and Marc Pouzet. "Clock-directed modular code generation for synchronous data-flow languages." In: *LCTES*. 2008 (pages 77, 113).

[Bis+05] Pratik Biswas, Pascal Grieder, Johan Löfberg, and Manfred Morari. "A Survey on Stability Analysis of Discrete-Time Piecewise Affine Systems." In: *IFAC World Congress*. Prague, Czech Republic, July 2005. URL: control.ee.ethz.ch/index.cgi?page=publications;action=details;id=2030 (page 44).

[BAI12] Lars Blackmore, Behçet Açikmese, and John M. Carson III. "Lossless convexification of control constraints for a class of nonlinear optimal control problems." In: *Systems & Control Letters* 61.8 (2012), pp. 863–870. DOI: 10.1016/j.sysconle.2012.04.010. URL: http://dx.doi.org/10.1016/j.sysconle.2012.04.010 (page 127).

[Blo+01] Vincent D. Blondel, Olivier Bournez, Pascal Koiran, Christos H. Papadimitriou, and John N. Tsitsiklis. "Deciding Stability and Mortality of Piecewise Affine Dynamical Systems." In: *Theoretical Computer Science A* 1–2.255 (2001), pp. 687–696. URL: dx.doi.org/10.1016/S0304-39750000399-6 (page 43).

[Bod02] Marc Bodson. "Evaluation of optimization methods for control allocation." In: *Journal of Guidance, Control, and Dynamics* 25.4 (2002), pp. 703–711 (page 127).

[BS15] Silvere Bonnabel and Jean-Jacques E. Slotine. "A Contraction Theory-Based Analysis of the Stability of the Deterministic Extended Kalman Filter." In: *IEEE Trans. Automat. Contr.* 60.2 (2015), pp. 565–569. DOI: 10.1109/TAC.2014.2336991. URL: http://dx.doi.org/10.1109/TAC.2014.2336991 (page 120).

[Bor99] Brian Borchers. "CSDP, A C library for semidefinite programming." In: *Optimization Methods and Software* 11.1-4 (1999), pp. 613–623. DOI: 10.1080/10556789908805765. eprint: www.tandfonline.com/doi/pdf/10.1080/10556789908805765. URL: www.tandfonline.com/doi/abs/10.1080/10556789908805765 (pages 33, 107).

[Bou+09] Olivier Bouissou, Eric Goubault, Sylvie Putot, Karim Tekkal, and Franck Védrine. "HybridFluctuat: A Static Analyzer of Numerical Programs within a Continuous Environment." In: *Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings*. Vol. 5643. LNCS. 2009, pp. 620–626. ISBN: 978-3-642-02657-7. DOI: 10.1007/978-3-642-02658-4_46. URL: dx.doi.org/10.1007/978-3-642-02658-4_46 (page 75).

[BP13] Timothy Bourke and Marc Pouzet. "Zélus: a synchronous language with ODEs." In: *Proceedings of the 16th international conference on Hybrid systems: computation and control, HSCC 2013, April 8-11, 2013, Philadelphia, PA, USA*. Ed. by Calin Belta and Franjo Ivancic. ACM, 2013, pp. 113–118. ISBN: 978-1-4503-1567-8. DOI: 10.1145/2461328.2461348. URL: doi.acm.org/10.1145/2461328.2461348 (page 119).

[Boy+94] Stephen Boyd, Laurent El Ghaoui, Éric Féron, and Venkataramanan Balakrishnan. *Linear Matrix Inequalities in System and Control Theory*. Vol. 15. SIAM. Philadelphia, PA: SIAM, June 1994. ISBN: 0-89871-334-X (pages 39, 74, 119).

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. New York, NY, USA: Cambridge University Press, 2004 (pages 32, 35, 36, 39).

[Bra12] Aaron Bradley. "Understanding IC3." In: *SAT 2012*. 2012, pp. 1–14 (page 14).

[Cac+14] David Cachera, Thomas P. Jensen, Arnaud Jobin, and Florent Kirchner. "Inference of polynomial invariants for imperative programs: A farewell to Gröbner bases." In: *Sci. Comput. Program.* 93 (2014), pp. 89–109. DOI: 10.1016/j.scico.2014.02.028. URL: http://dx.doi.org/10.1016/j.scico.2014.02.028 (page 56).

[CCN06] Stephen L. Campbell, Jean-Philippe Chancelier, and Ramine Nikoukhah. *Modeling and Simulation in Scilab, Scicos*. Springer, 2006 (page 78).

[Cas+03] Paul Caspi, Adrian Curic, Aude Maignan, Christos Sofronis, and Stavros Tripakis. "Translating Discrete-Time Simulink to Lustre." In: *Embedded Software*. Ed. by Rajeev Alur and Insup Lee. Vol. 2855. Lecture Notes in Computer Science. 10.1007/978-3-540-45212-67. Springer Berlin / Heidelberg, 2003, pp. 84–99 (page 114).

[Cas+87] Paul Caspi, Daniel Pilaud, Nicolas Halbwachs, and John Plaice. "Lustre: A Declarative Language for Programming Synchronous Systems." In: *POPL*. ACM Press, 1987, pp. 178–188. ISBN: 0-89791-215-2 (page 77).

[CDD15] Adrien Champion, Rémi Delmas, and Michael Dierkes. "Generating property-directed potential invariants by quantifier elimination in a k-induction-based framework." In: *Sci. Comput. Program.* 103 (2015), pp. 71–87. DOI: 10.1016/j.scico.2014.10.004. URL: dx.doi.org/10.1016/j.scico.2014.10.004 (page 14).

[Cim12] Alessandro Cimatti. "Application of SMT solvers to hybrid system verification." In: *Formal Methods in Computer-Aided Design, FMCAD 2012, Cambridge, UK, October 22-25, 2012*. Ed. by Gianpiero Cabodi and Satnam Singh. IEEE, 2012, p. 4. ISBN: 978-1-4673-4832-4. URL: ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6462548 (page 119).

[CS11] Michael Colón and Sriram Sankaranarayanan. "Generalizing the Template Polyhedral Domain." In: *ESOP*. Vol. 6602. LNCS. 2011, pp. 176–195. ISBN: 978-3-642-19717-8 (page 38).

[CS93] Joao L. D. Comba and Jorge Stolfi. *Affine Arithmetic and its Applications to Computer Graphics*. 1993 (pages 93, 100–102).

[Con+08] Sylvain Conchon, Evelyne Contejean, Johannes Kanig, and Stéphane Lescuyer. "CC(X): Semantic Combination of Congruence Closure with Solvable Theories." In: *Electr. Notes Theor. Comput. Sci.* 198.2 (2008), pp. 51–69 (page 11).

[Cos+05] Alexandru Costan, Stéphane Gaubert, Eric Goubault, Matthieu Martel, and Sylvie Putot. "A policy iteration algorithm for computing fixed points in static analysis of programs." In: *Computer aided verification*. Ed. by Kousha Etessami and Sriram K. Rajamani. Vol. 3576. LNCS. Springer. Springer, 2005, pp. 462–475. ISBN: 3-540-27231-3 (pages 60, 63).

[Cou05] Patrick Cousot. "Proving Program Invariance and Termination by Parametric Abstraction, Lagrangian Relaxation and Semidefinite Programming." English. In: *Verification, Model Checking, and Abstract Interpretation*. Ed. by Radhia Cousot. Vol. 3385. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2005, pp. 1–24. ISBN: 978-3-540-24297-0. DOI: 10.1007/978-3-540-30579-8_1. URL: dx.doi.org/10.1007/978-3-540-30579-8_1 (page 56).

[CC77] Patrick Cousot and Radhia Cousot. "Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints." In: *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*. POPL '77. New York, NY, USA: ACM, 1977, pp. 238–252 (page 14).

[CC92] Patrick Cousot and Radhia Cousot. "Abstract Interpretation Frameworks." In: *Journal of Logic and Computation* 2.4 (Aug. 1992), pp. 511–547 (page 17).

[CC79] Patrick Cousot and Radhia Cousot. "Systematic design of program analysis frameworks." In: *Conference Record of the Sixth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. San Antonio, Texas: ACM Press, New York, NY, 1979, pp. 269–282 (page 17).

[Cou+07] Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, and Xavier Rival. "Combination of Abstractions in the Astrée Static Analyzer." In: *Eleventh Annual Asian Computing Science Conference (ASIAN'06)*. Ed. by M. Okada and I. Satoh. Tokyo, Japan, LNCS 4435: Springer, Berlin, Dec. 2007, pp. 1–24 (page 18).

[Cou+05] Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, and Xavier Rival. "The ASTRÉE Analyser." In: *ESOP*. Vol. 3444. LNCS. 2005, pp. 21–30 (pages 4, 92).

[Cuo+12] Pascal Cuoq, Florent Kirchner, Nikolai Kosmatov, Virgile Prevosto, Julien Signoles, and Boris Yakobowski. "Frama-C: A Software Analysis Perspective." In: SEFM. Springer, 2012, pp. 233–247 (pages 10, 78, 79).

[DMC15] Nasrine Damouche, Matthieu Martel, and Alexandre Chapoutot. "Transformation of a PID Controller for Numerical Accuracy." In: *Electr. Notes Theor. Comput. Sci.* 317 (2015), pp. 47–54. DOI: 10.1016/j.entcs.2015.10.006. URL: http://dx.doi.org/10.1016/j.entcs.2015.10.006 (page 116).

[Det+14] Morgan Deters, Andrew Reynolds, Tim King, Clark W. Barrett, and Cesare Tinelli. "A tour of CVC4: How it works, and how to use it." In: *Formal Methods in Computer-Aided Design, FMCAD 2014, Lausanne, Switzerland, October 21-24, 2014*. IEEE, 2014, p. 7. ISBN: 978-0-9835678-4-4. DOI: 10.1109/FMCAD.2014.6987586. URL: dx.doi.org/10.1109/FMCAD.2014.6987586 (page 11).

[Die+15] Arnaud Dieumegard, Pierre-Loïc Garoche, Temesghen Kahsai, Alice Tailliar, and Xavier Thirioux. "Compilation Of Synchronous Observers As Code Contracts." In: *30th ACM/SIGAPP Symposium on Applied Computing, SAC 2015, Salamanca, Spain - April 13 - 17, 2015*. Ed. by Roger L. Wainwright, Juan Manuel Corchado, Alessio Bechini, and Jiman Hong. Short paper. ACM, 2015, pp. 1933–1939. ISBN: 978-1-4503-3196-8. DOI: 10.1145/2695664.2695819. URL: doi.acm.org/10.1145/2695664.2695819 (pages 113, 116).

[Dij76] Edsger Wybe Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976 (page 13).

[Doy82] John C. Doyle. "Analysis of feedback systems with structured uncertainties." In: *IEE Proceedings D - Control Theory and Applications* 129.6 (Nov. 1982), pp. 242–250. ISSN: 0143-7054. DOI: 10.1049/ip-d.1982.0053 (page 72).

[Dut14] Bruno Dutertre. "Yices 2.2." In: *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*. Ed. by Armin Biere and Roderick Bloem. Vol. 8559. Lecture Notes in Computer Science. Springer, 2014, pp. 737–744. ISBN: 978-3-319-08866-2. DOI: 10.1007/978-3-319-08867-9_49. URL: dx.doi.org/10.1007/978-3-319-08867-9_49 (page 11).

[DM06] Bruno Dutertre and Leonardo de Moura. *The YICES SMT Solver*. Tech. rep. SRI International, 2006. URL: yices.csl.sri.com/ (page 11).

[Fer05a] Jérôme Feret. "Analysis of Mobile Systems by Abstract Interpretation." PhD thesis. École polytechnique, Paris, France, 2005 (page 17).

[Fer05b] Jérôme Feret. "Numerical Abstract Domains for Digital Filters." In: *International workshop on Numerical and Symbolic Abstract Domains (NSAD)*. 2005 (page 56).

[Fer04] Jérôme Feret. "Static Analysis of Digital Filters." In: *Programming Languages and Systems, 13th European Symposium on Programming, ESOP 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29 - April 2, 2004, Proceedings*. Ed. by David A. Schmidt. Vol. 2986. Lecture Notes in Computer Science. Springer, 2004, pp. 33–48. ISBN: 3-540-21313-9. DOI: 10.1007/978-3-540-24725-8_4. URL: dx.doi.org/10.1007/978-3-540-24725-8_4 (pages 4, 18, 40, 56).

[Fér10] Éric Féron. "From Control Systems to Control Software." In: *Control Systems, IEEE* 30.6 (Dec. 2010), pp. 50–71. ISSN: 1066-033X. DOI: 10.1109/MCS.2010.938196 (pages 21, 69, 80, 82).

[FM07] Jean-Christophe Filliâtre and Claude Marché. "The Why/Krakatoa/Caduceus Platform for Deductive Program Verification." In: *Computer Aided Verification, 19th International Conference, CAV 2007, Berlin, Germany, July 3-7, 2007, Proceedings*. Ed. by Werner Damm and Holger Hermanns. Vol. 4590. Lecture Notes in Computer Science. Springer, 2007, pp. 173–177. ISBN: 978-3-540-73367-6. DOI: 10.1007/978-3-540-73368-3_21. URL: dx.doi.org/10.1007/978-3-540-73368-3_21 (page 10).

[Fit61] Richard FitzHugh. "Impulses and physiological states in theoretical models of nerve membrane." In: *Biophys J.* 1 (1961), pp. 445–466. ISSN: 0006-3495 (page 123).

[Flo67] Robert W. Floyd. "Assigning Meanings to Programs." In: *Proceedings of Symposium on Applied Mathematics* 19 (1967), pp. 19–32 (page 8).

[FWP90] Gene F. Franklin, Michael L. Workman, and Dave Powell. *Digital Control of Dynamic Systems*. 2nd. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1990. ISBN: 0201820544 (page 21).

[Gar08] Pierre-Loïc Garoche. "Static Analysis of Actors by Abstract Interpretation." PhD thesis. University of Toulouse, INPT, 2008 (page 17).

[GGK14] Pierre-Loïc Garoche, Arie Gurfinkel, and Temesghen Kahsai. "Synthesizing Modular Invariants for Synchronous Code." In: *Proceedings of the First Workshop on Horn Clauses for Verification and Synthesis, HCVS 2014, Vienna, Austria, 17 July 2014*. Ed. by Nikolaj Bjørner, Fabio Fioravanti, Andrey Rybalchenko, and Valerio Senni. Vol. 169. EPTCS. 2014, pp. 19–30. DOI: 10.4204/EPTCS.169.4 (page 113).

[Gar+14] Pierre-Loïc Garoche, Falk Howar, Temesghen Kahsai, and Xavier Thirioux. "Testing-Based Compiler Validation for Synchronous Languages." In: *NASA Formal Methods - 6th International Symposium, NFM 2014, Houston, TX, USA, April 29 - May 1, 2014. Proceedings*. Ed. by Julia M. Badger and Kristin Yvonne Rozier. Vol. 8430. Lecture Notes in Computer Science. Short paper. Springer, 2014, pp. 246–251. ISBN: 978-3-319-06199-3. DOI: 10.1007/978-3-319-06200-6_19 (page 113).

[GKT12] Pierre-Loïc Garoche, Temesghen Kahsai, and Cesare Tinelli. *Invariant stream generators using automatic abstract transformers based on a decidable logic*. 2012. URL: arxiv.org/abs/1205.3758 (page 14).

[GTK12] Pierre-Loïc Garoche, Xavier Thirioux, and Temesghen Kahsai. *LustreC: a modular Lustre compiler*. 2012–. URL: https://github.com/coco-team/lustrec (pages 77, 113).

[Gau+07] Stéphane Gaubert, Eric Goubault, Ankur Taly, and Sarah Zennou. "Static Analysis by Policy Iteration on Relational Domains." In: *ESOP*. Ed. by Rocco De Nicola. Vol. 4421. LNCS. Springer, 2007, pp. 237–252. ISBN: 978-3-540-71314-2 (pages 60, 63).

[GS10] Thomas Martin Gawlitza and Helmut Seidl. "Computing Relaxed Abstract Semantics w.r.t. Quadratic Zones Precisely." In: *SAS*. Ed. by Radhia Cousot and Matthieu Martel. Vol. 6337. LNCS. Springer, 2010, pp. 271–286. ISBN: 978-3-642-15768-4 (pages 63, 64).

[Gaw+12] Thomas Martin Gawlitza, Helmut Seidl, Assalé Adjé, Stéphane Gaubert, and Eric Goubault. "Abstract interpretation meets convex optimization." In: *J. Symb. Comput.* 47.12 (2012), pp. 1416–1446 (page 63).

[GS07a] Thomas Gawlitza and Helmut Seidl. "Precise Fixpoint Computation Through Strategy Iteration." In: *ESOP*. Ed. by Rocco De Nicola. Vol. 4421. LNCS. Springer, 2007, pp. 300–315. ISBN: 978-3-540-71314-2 (page 63).

[GS07b] Thomas Gawlitza and Helmut Seidl. "Precise Relational Invariants Through Strategy Iteration." In: *CSL*. Ed. by Jacques Duparc and Thomas A. Henzinger. Vol. 4646. LNCS. Springer, 2007, pp. 23–40. ISBN: 978-3-540-74914-1 (page 63).

[GGP10] Khalil Ghorbal, Eric Goubault, and Sylvie Putot. "A Logical Product Approach to Zonotope Intersection." In: *CAV*. 2010, pp. 212–226 (pages 97, 101).

[GGP09] Khalil Ghorbal, Eric Goubault, and Sylvie Putot. "The Zonotope Abstract Domain Taylor1+." In: *CAV*. Vol. 5643. LNCS. 2009, pp. 627–633. ISBN: 978-3-642-02657-7 (pages 93, 97, 100, 101).

[Gho+12] Khalil Ghorbal, Franjo Ivančić, Gogul Balakrishnan, Naoto Maeda, and Aarti Gupta. "Donut Domains: Efficient Non-convex Domains for Abstract Interpretation." In: *Verification, Model Checking, and Abstract Interpretation - 13th International Conference, VMCAI 2012, Philadelphia, PA, USA, January 22-24, 2012. Proceedings*. Ed. by Viktor Kuncak and Andrey Rybalchenko. Vol. 7148. Lecture Notes in Computer Science. Springer, 2012, pp. 235–250. ISBN: 978-3-642-27939-3. DOI: 10.1007/978-3-642-27940-9_16. URL: http://dx.doi.org/10.1007/978-3-642-27940-9_16 (page 56).

[GK99] Elmer G. Gilbert and Ilya V. Kolmanovsky. "Set-point control of nonlinear systems with state and control constraints: a Lyapunov-function, reference-governor approach." In: *Decision and Control, 1999. Proceedings of the 38th IEEE Conference on*. Vol. 3. 1999, 2507–2512 vol.3. DOI: 10.1109/CDC.1999.831304 (page 118).

[Gil13] Jean-Charles Gilbert. *Éléments d'optimisation différentiable*. 2013 (page 32).

[GVP00] Keith Glover, Glenn Vinnicombe, and George Papageorgiou. "Guaranteed multi-loop stability margins and the gap metric." In: *CDC*. Vol. 4. IEEE. 2000, pp. 4084–4085 (page 74).

[Gou01] Eric Goubault. "Static Analyses of the Precision of Floating-Point Operations." In: *Proceedings of the 8th International Symposium on Static Analysis*. SAS '01. London, UK, UK: Springer-Verlag, 2001, pp. 234–259 (pages 93, 101).

[Gou13] Eric Goubault. "Static Analysis by Abstract Interpretation of Numerical Programs and Systems, and FLUCTUAT." In: *SAS*. 2013, pp. 1–3 (page 101).

[GLP12] Eric Goubault, Tristan Le Gall, and Sylvie Putot. "An Accurate Join for Zonotopes, Preserving Affine Input/Output Relations." In: *ENTCS* 287 (2012), pp. 65–76 (page 97).

[GP09] Eric Goubault and Sylvie Putot. "A zonotopic framework for functional abstractions." In: *CoRR* abs/0910.1763 (2009) (pages 96, 97).

[GP11] Eric Goubault and Sylvie Putot. "Static Analysis of Finite Precision Computations." In: *VMCAI*. Ed. by Ranjit Jhala and David A. Schmidt. Vol. 6538. LNCS. Springer, 2011, pp. 232–247. ISBN: 978-3-642-18274-7 (page 93).

[GPV12] Eric Goubault, Sylvie Putot, and Franck Védrine. "Modular Static Analysis with Zonotopes." In: *SAS*. Vol. 7460. LNCS. Springer, 2012, pp. 24–40. ISBN: 978-3-642-33124-4 (pages 97, 101).

[HC08] Wassim M. Haddad and VijaySekhar Chellaboina. *Nonlinear Dynamical Systems and Control: A Lyapunov-based Appr*. Princeton University Press, 2008 (pages 38, 74).

[Hal+91] Nicolas Halbwachs, Paul Caspi, Pascal Raymond, and Daniel Pilaud. "The synchronous data-flow programming language LUSTRE." In: *Proceedings of the IEEE* 79.9 (Sept. 1991), pp. 1305–1320 (pages 27, 113).

[HLR93] Nicolas Halbwachs, Fabienne Lagnier, and Pascal Raymond. "Synchronous Observers and the Verification of Reactive Systems." In: *AMAST*. Ed. by Maurice Nivat, Charles Rattray, Teodor Rus, and Giuseppe Scollo. Workshops in Computing. Springer, 1993, pp. 83–96. ISBN: 3-540-19852-0 (page 78).

[HK14] Didier Henrion and Milan Korda. "Convex Computation of the Region of Attraction of Polynomial Control Systems." In: *Automatic Control, IEEE Transactions on* 59.2 (2014), pp. 297–312. ISSN: 0018-9286. DOI: 10.1109/TAC.2013.2283095 (pages 121, 122).

[HLS09] Didier Henrion, Jean-Bernard Lasserre, and Carlo Savorgnan. "Approximate Volume and Integration for Basic Semialgebraic Sets." In: *SIAM Review* 51.4 (2009), pp. 722–743. DOI: 10.1137/080730287. eprint: dx.doi.org/10.1137/080730287. URL: dx.doi.org/10.1137/080730287 (page 121).

[Her+12] Heber Herencia-Zapana, Romain Jobredeaux, Sam Owre, Pierre-Loïc Garoche, Éric Féron, Gilberto Perez, and Pablo Ascariz. "PVS Linear Algebra Libraries for Verification of Control Software Algorithms in C/ACSL." In: *NASA Formal Methods - Forth International Symposium, NFM 2012, Norfolk, VA USA, April 3-5, 2012. Proceedings*. Ed. by Alwyn Goodloe and Suzette Person. Vol. 7226. Lecture Notes in Computer Science. Springer, 2012, pp. 147–161. DOI: 10.1007/978-3-642-28891-3_15 (pages 78, 81, 115).

[Hig96] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1996. ISBN: 0898713552 (page 90).

[Hoa69] Charles. A. R. Hoare. "An axiomatic basis for computer programming." In: *Commun. ACM* 12 (Oct. 1969), pp. 576–580 (pages 8, 10).

[ISo00] Saidhakim Dododzhanovich Ikramov and N.V. Savel'eva. "Conditionally definite matrices." In: *Journal of Mathematical Sciences* 98.1 (2000), pp. 1–50. ISSN: 1072-3374. DOI: 10.1007/BF02355379. URL: dx.doi.org/10.1007/BF02355379 (page 43).

[IM13] Arnault Ioualalen and Matthieu Martel. "Synthesizing accurate floating-point formulas." In: *24th International Conference on Application-Specific Systems, Architectures and Processors, ASAP 2013, Washington, DC, USA, June 5-7, 2013*. IEEE Computer Society, 2013, pp. 113–116. ISBN: 978-1-4799-0494-5. DOI: 10.1109/ASAP.2013.6567563. URL: http://dx.doi.org/10.1109/ASAP.2013.6567563 (page 116).

[JM09] Bertrand Jeannet and Antoine Miné. "Apron: A Library of Numerical Abstract Domains for Static Analysis." In: *CAV'09*. 2009, pp. 661–667. DOI: 10.1007/978-3-642-02658-4_52. URL: dx.doi.org/10.1007/978-3-642-02658-4_52 (page 101).

[Jer+14] Juan Luis Jerez, Paul J. Goulart, Stefan Richter, George A. Constantinides, Eric C. Kerrigan, and Manfred Morari. "Embedded Online Optimization for Model Predictive Control at Megahertz Rates." In: *IEEE Trans. Automat. Contr.* 59.12 (2014), pp. 3238–3251. DOI: 10.1109/TAC.2014.2351991. URL: http://dx.doi.org/10.1109/TAC.2014.2351991 (pages 104, 127).

[KGT11] Temesghen Kahsai, Yeting Ge, and Cesare Tinelli. "Instantiation-Based Invariant Discovery." In: *NASA Formal Methods - Third International Symposium, NFM 2011, Pasadena, CA, USA, April 18-20, 2011. Proceedings*. 2011, pp. 192–206 (page 14).

[KTG16] Temesghen Kahsai, Xavier Thirioux, and Pierre-Loïc Garoche. "Hierarchical state machines as modular Horn clauses." In: *Proceedings of the Second Workshop on Horn Clauses for Verification and Synthesis, HCVS 2016, Eindhoven, The Netherlands, April 3rd 2016*. 2016 (page 113).

[KT11] Temesghen Kahsai and Cesare Tinelli. "PKind: A parallel k-induction based model checker." In: *Proceedings 10th International Workshop on Parallel and Distributed Methods in verifiCation, PDMC 2011, Snowbird, Utah, USA, July 14, 2011*. Vol. 72. EPTCS. 2011, pp. 55–62. DOI: 10.4204/EPTCS.72.6. URL: dx.doi.org/10.4204/EPTCS.72.6 (pages 14, 49).

[Kar84] Narendra Karmarkar. "A new polynomial-time algorithm for linear programming." In: *Combinatorica* 4.4 (Dec. 1984), pp. 373–395 (page 103).

[KMW16] Egor George Karpenkov, David Monniaux, and Philipp Wendler. "Program Analysis with Local Policy Iteration." In: *Verification, Model Checking, and Abstract Interpretation - 17th International Conference, VMCAI 2016, St. Petersburg, FL, USA, January 17-19, 2016. Proceedings*. Ed. by Barbara Jobstmann and K. Rustan M. Leino. Vol. 9583. Lecture Notes in Computer Science. Springer, 2016, pp. 127–146. ISBN: 978-3-662-49121-8. DOI: 10.1007/978-3-662-49122-5_6. URL: http://dx.doi.org/10.1007/978-3-662-49122-5_6 (page 64).

[Kar76] Michael Karr. "Affine Relationships Among Variables of a Program." In: *Acta Inf.* 6 (1976), pp. 133–151. DOI: 10.1007/BF00268497. URL: http://dx.doi.org/10.1007/BF00268497 (page 56).

[KSH97] Masakazu Kojima, Susumu Shindoh, and Shinji Hara. "Interior-Point Methods for the Monotone Semidefinite Linear Complementarity Problem in Symmetric Matrices." In: *SIAM Journal on Optimization* 7.1 (Feb. 1997), pp. 86–125 (page 128).

[KGD14] Ilya V. Kolmanovsky, Emmanuele Garone, and Stefano Di Cairano. "Reference and command governors: A tutorial on their theory and automotive applications." In: *2014 American Control Conference*. June 2014, pp. 226–241. DOI: 10.1109/ACC.2014.6859176 (page 118).

[KGC14] Anvesh Komuravelli, Arie Gurfinkel, and Sagar Chaki. "SMT-Based Model Checking for Recursive Programs." In: *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*. Ed. by Armin Biere and Roderick Bloem. Vol. 8559. Lecture Notes in Computer Science. Springer, 2014, pp. 17–34. ISBN: 978-3-319-08866-2. DOI: 10.1007/978-3-319-08867-9_2. URL: dx.doi.org/10.1007/978-3-319-08867-9_2 (pages 13, 113).

[Kom+13] Anvesh Komuravelli, Arie Gurfinkel, Sagar Chaki, and Edmund M. Clarke. "Automatic Abstraction in SMT-Based Unbounded Software Model Checking." In: *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*. Ed. by Natasha Sharygina and Helmut Veith. Vol. 8044. Lecture Notes in Computer Science. Springer, 2013, pp. 846–862. ISBN: 978-3-642-39798-1. DOI: 10.1007/978-3-642-39799-8_59. URL: dx.doi.org/10.1007/978-3-642-39799-8_59 (page 13).

[Kon+15] Soonho Kong, Sicun Gao, Wei Chen, and Edmund M. Clarke. "dReach: δ-Reachability Analysis for Hybrid Systems." In: *Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*. Ed. by Christel Baier and Cesare Tinelli. Vol. 9035. Lecture Notes in Computer Science. Springer, 2015, pp. 200–205. ISBN: 978-3-662-46680-3. DOI: 10.1007/978-3-662-46681-0_15. URL: dx.doi.org/10.1007/978-3-662-46681-0_15 (page 119).

[KHJ13a] Milan Korda, Didier Henrion, and Colin Neil Jones. "Convex computation of the maximum controlled invariant set for discrete-time polynomial control systems." In: *Decision and Control (CDC), 2013 IEEE 52nd Annual Conference on*. Dec. 2013, pp. 7107–7112. DOI: 10.1109/CDC.2013.6761016 (pages 121, 122).

[KHJ13b] Milan Korda, Didier Henrion, and Colin Neil Jones. "Convex computation of the maximum controlled invariant set for polynomial control systems." In: *arXiv preprint arXiv:1303.6469* (2013) (page 121).

[KHJ12] Milan Korda, Didier Henrion, and Colin Neil Jones. "Inner approximations of the region of attraction for polynomial dynamical systems." In: *ArXiv e-prints* (Oct. 2012). arXiv: 1210.3184 [math.OC] (pages 121, 122).

[Las01] Jean-Bernard Lasserre. "Global Optimization with Polynomials and the Problem of Moments." In: *SIAM Journal on Optimization* 11.3 (2001), pp. 796–817 (pages 59, 122).

[Las09] Jean-Bernard Lasserre. *Moments, Positive Polynomials and Their Applications*. Imperial College Press optimization series. Imperial College Press, 2009. ISBN: 9781848164468. URL: books.google.nl/books?id=VY6imTsdIrEC (page 34).

[Las15] Jean-Bernard Lasserre. "Tractable approximations of sets defined with quantifiers." English. In: *Mathematical Programming* 151.2 (2015), pp. 507–527. ISSN: 0025-5610. DOI: 10.1007/s10107-014-0838-1. URL: dx.doi.org/10.1007/s10107-014-0838-1 (page 121).

[LD11] Ji-Woong Lee and Geir E. Dullerud. "Joint synthesis of switching and feedback for linear systems in discrete time." In: *Proceedings of the 14th ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2011, Chicago, IL, USA, April 12-14, 2011*. Ed. by Marco Caccamo, Emilio Frazzoli, and Radu Grosu. ACM, 2011, pp. 201–210. ISBN: 978-1-4503-0629-4. DOI: 10.1145/1967701.1967731. URL: doi.acm.org/10.1145/1967701.1967731 (page 49).

[LD07] Ji-Woong Lee and Geir E. Dullerud. "Uniformly Stabilizing Sets of Switching Sequences for Switched Linear Systems." In: *IEEE Trans. Automat. Contr.* 52.5 (2007), pp. 868–874. DOI: 10.1109/TAC.2007.895924. URL: dx.doi.org/10.1109/TAC.2007.895924 (page 49).

[LDK07] Ji-Woong Lee, Geir E Dullerud, and Pramod P Khargonekar. "An output regulation problem for switched linear systems in discrete time." In: *Proceedings of the 46th IEEE Conference on Decision and Control*. 2007, pp. 4993–4998 (page 49).

[Lev96] William S. Levine. *The control handbook*. The electrical engineering handbook series. Boca Raton (Fl.): CRC Press New York, 1996. ISBN: 0-8493-8570-9. URL: opac.inria.fr/record=b1079196 (pages 19, 75).

[Löf04] Johan Löfberg. "YALMIP : A Toolbox for Modeling and Optimization in MATLAB." In: *Proceedings of the CACSD Conference*. Taipei, Taiwan, 2004. URL: users.isy.liu.se/johanl/yalmip (page 34).

[LS98] Winfried Lohmiller and Jean-Jacques E. Slotine. "On Contraction Analysis for Non-linear Systems." In: *Automatica* 34.6 (1998), pp. 683–696. DOI: 10.1016/S0005-1098(98)00019-3. URL: dx.doi.org/10.1016/S0005-1098(98)00019-3 (page 120).

[MHL15] Victor Magron, Didier Henrion, and Jean-Bernard Lasserre. "Semidefinite Approximations of Projections and Polynomial Images of SemiAlgebraic Sets." In: *SIAM Journal on Optimization* 25.4 (2015), pp. 2143–2164. DOI: 10.1137/140992047. eprint: dx.doi.org/10.1137/140992047. URL: dx.doi.org/10.1137/140992047 (page 121).

[Mar05] Matthieu Martel. "An Overview of Semantics for the Validation of Numerical Programs." In: *Verification, Model Checking, and Abstract Interpretation, 6th International Conference, VMCAI 2005, Paris, France, January 17-19, 2005, Proceedings*. Ed. by Radhia Cousot. Vol. 3385. Lecture Notes in Computer Science. Springer, 2005, pp. 59–77. ISBN: 3-540-24297-X. DOI: 10.1007/978-3-540-30579-8_4. URL: http://dx.doi.org/10.1007/978-3-540-30579-8_4 (pages 93, 101).

[MJ81] D.H. Martin and D.H. Jacobson. "Copositive matrices and definiteness of quadratic forms subject to homogeneous linear inequality constraints." In: *Linear Algebra and its Applications* 35 (1981), pp. 227–258. ISSN: 0024-3795. DOI: dx.doi.org/10.1016/0024-3795(81)90276-7. URL: www.sciencedirect.com/science/article/pii/0024379581902767 (page 43).

[McG00] Lawrence Kent McGovern. "Computational Analysis of Real-Time Convex Optimization for Control Systems." PhD thesis. Boston, USA: Massachussetts Institute of Technology, May 2000 (page 127).

[MF98] Lawrence Kent McGovern and Éric Féron. "Requirements and hard computational bounds for real-time optimization in safety-critical control systems." In: *Decision and Control, 1998. Proceedings of the 37th IEEE Conference on*. Vol. 3. 1998, 3366–3371 vol.3 (page 127).

[MR97] Alexandre Megretski and Anders Rantzer. "System analysis via integral quadratic constraints." In: *Automatic Control, IEEE Transactions on* 42.6 (1997), pp. 819–830 (pages 119, 120).

[MR95] Alexandre Megretski and Anders Rantzer. *System Analysis via Integral Quadratic Constraints – Part I*. 1995 (pages 72, 120).

[MT06] Frédéric Messine and Ahmed Touhami. "A General Reliable Quadratic Form: An Extension of Affine Arithmetic." In: *Reliable Computing* 12.3 (2006), pp. 171–192 (pages 93, 95, 96, 99).

[MFM00] Domenico Mignone, Giancarlo Ferrari-Trecate, and Manfred Morari. "Stability and stabilization of piecewise affine and hybrid systems: an LMI approach." In: *Decision and Control, 2000. Proc. of the 39th IEEE Conference on*. Vol. 1. 2000, 504–509 vol.1. DOI: 10.1109/CDC.2000.912814 (page 44).

[Min06] Antoine Miné. "The octagon abstract domain." In: *Higher-Order and Symbolic Computation* 19.1 (2006), pp. 31–100 (page 27).

[Min04] Antoine Miné. "Weakly relational numerical abstract domains." PhD thesis. École Polytechnique, Dec. 2004, p. 322 (pages 14, 17, 97, 99).

[MT13] Eike Möhlmann and Oliver E. Theel. "Stabhyli: a tool for automatic stability verification of non-linear hybrid systems." In: *Proceedings of the 16th international conference on Hybrid systems: computation and control, HSCC 2013, April 8-11, 2013, Philadelphia, PA, USA*. Ed. by Calin Belta and Franjo Ivancic. ACM, 2013, pp. 107–112. ISBN: 978-1-4503-1567-8. DOI: 10.1145/2461328.2461347. URL: doi.acm.org/10.1145/2461328.2461347 (page 75).

[Mon07] David Monniaux. "Applying the Z-transform for the static analysis of floating-point numerical filters." In: *CoRR* abs/0706.0252 (2007). URL: http://arxiv.org/abs/0706.0252 (page 56).

[Mon08] David Monniaux. "The pitfalls of verifying floating-point computations." In: *ACM Trans. Program. Lang. Syst.* 30.3 (2008) (page 89).

[MC11] David Monniaux and Pierre Corbineau. "On the Generation of Positivstellensatz Witnesses in Degenerate Cases." In: *Interactive Theorem Proving: Second International Conference, ITP 2011, Berg en Dal, The Netherlands, August 22-25, 2011. Proceedings*. Ed. by Marko van Eekelen, Herman Geuvers, Julien Schmaltz, and Freek Wiedijk. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 249–264. ISBN: 978-3-642-22863-6. DOI: 10.1007/978-3-642-22863-6_19. URL: http://dx.doi.org/10.1007/978-3-642-22863-6_19 (page 106).

[MS14] David Monniaux and Peter Schrammel. "Speeding Up Logico-Numerical Strategy Iteration." In: *Static Analysis - 21st International Symposium, SAS 2014, Munich, Germany, September 11-13, 2014. Proceedings*. Ed. by Markus Müller-Olm and Helmut Seidl. Vol. 8723. Lecture Notes in Computer Science. Springer, 2014, pp. 253–267. ISBN: 978-3-319-10935-0. DOI: 10.1007/978-3-319-10936-7_16. URL: http://dx.doi.org/10.1007/978-3-319-10936-7_16 (page 64).

[Mon97] Renato D. C. Monteiro. "Primal–Dual Path-Following Algorithms for Semidefinite Programming." In: *SIAM J. on Optimization* 7.3 (Mar. 1997), pp. 663–678 (page 128).

[Mot51] Théodore Samuel Motzkin. "Two consequences of the transposition theorem on linear inequalities." In: *Econometrica* 19.2 (1951), pp. 184–185 (page 45).

[MB08] Leonardo Mendonça de Moura and Nikolaj Bjørner. "Z3: An Efficient SMT Solver." In: *TACAS*. Ed. by C. R. Ramakrishnan and Jakob Rehof. Vol. 4963. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 337–340. ISBN: 978-3-540-78799-0 (pages 11, 113, 115).

[Moy08] Yannick Moy. "Sufficient Preconditions for Modular Assertion Checking." In: *Verification, Model Checking, and Abstract Interpretation, 9th International Conference, VMCAI 2008, San Francisco, USA, January 7-9, 2008, Proceedings*. Ed. by Francesco Logozzo, Doron A. Peled, and Lenore D. Zuck. Vol. 4905. Lecture Notes in Computer Science. Springer, 2008, pp. 188–202. ISBN: 978-3-540-78162-2. DOI: 10.1007/978-3-540-78163-9_18. URL: dx.doi.org/10.1007/978-3-540-78163-9_18 (page 17).

[MS02] Markus Müller-Olm and Helmut Seidl. "Polynomial Constants Are Decidable." In: *Static Analysis, 9th International Symposium, SAS 2002, Madrid, Spain, September 17-20, 2002, Proceedings*. Ed. by Manuel V. Hermenegildo and Germán Puebla. Vol. 2477. Lecture Notes in Computer Science. Springer, 2002, pp. 4–19. ISBN: 3-540-44235-9. DOI: 10.1007/3-540-45789-5_4. URL: http://dx.doi.org/10.1007/3-540-45789-5_4 (page 56).

[NN88] Yurii Nesterov and Arkadi Nemirovski. "A general approach to the design of optimal methods for smooth convex functions minimization." In: *Ekonomika i Matem. Metody* 24 (1988), pp. 509–517 (page 103).

[NN94] Yurii Nesterov and Arkadi Nemirovski. *Interior-point Polynomial Algorithms in Convex Programming*. Vol. 13. Studies in Applied Mathematics. Society for Industrial and Applied Mathematics, 1994 (pages 39, 103, 127, 130).

[NN89] Yurii Nesterov and Arkadi Nemirovski. *Self-Concordant functions and polynomial time methods in convex programming*. Materialy po matematicheskomu obespecheniiu EVM. USSR Academy of Sciences, Central Economic & Mathematic Institute, 1989 (page 103).

[NLS14] Tuan T. Nguyen, Mircea Lazar, and Veaceslav Spinu. "Interpolation of polytopic control Lyapunov functions for discrete-time linear systems." In: *{IFAC} Proceedings Volumes* 47.3 (2014). 19th {IFAC} World Congress, pp. 2297–2302. ISSN: 1474-6670. DOI: dx.doi.org/10.3182/20140824-6-ZA-1003.01513. URL: www.sciencedirect.com/science/article/pii/S1474667016419543 (page 118).

[NW06] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. 2nd. New York: Springer, 2006 (page 32).

[NW05] Guy Norris and Mark Wagner. *Airbus A380: Superjumbo of the 21st Century*. Zenith Press, 2005. ISBN: 9780760322185. URL: https://books.google.fr/books?id=KcaYjPhRnWUC (page 3).

[ORY01] Peter W. O'Hearn, John C. Reynolds, and Hongseok Yang. "Local Reasoning about Programs that Alter Data Structures." In: *Computer Science Logic, 15th International Workshop, CSL 2001. 10th Annual Conference of the EACSL, Paris, France, September 10-13, 2001, Proceedings*. Ed. by Laurent Fribourg. Vol. 2142. Lecture Notes in Computer Science. Springer, 2001, pp. 1–19. ISBN: 3-540-42554-3. DOI: 10.1007/3-540-44802-0_1. URL: dx.doi.org/10.1007/3-540-44802-0_1 (page 13).

[OV15] Mendes Oulamara and Arnaud J. Venet. "Abstract Interpretation with Higher-Dimensional Ellipsoids and Conic Extrapolation." In: *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I*. Ed. by Daniel Kroening and Corina S. Pasareanu. Vol. 9206. Lecture Notes in Computer Science. Springer, 2015, pp. 415–430. ISBN: 978-3-319-21689-8. DOI: 10.1007/978-3-319-21690-4_24. URL: http://dx.doi.org/10.1007/978-3-319-21690-4_24 (page 56).

[OS01] Sam Owre and Natarajan Shankar. *Theory Interpretation in PVS*. Tech. rep. SRI International, 2001 (page 83).

[Pak+13] Mehrdad Pakmehr, Timothy Wang, Romain Jobredeaux, Martin Vivies, and Éric Féron. "Verifiable Control System Development for Gas Turbine Engines." In: *CoRR* abs/1311.1885 (2013) (page 78).

[Par03] Pablo A. Parrilo. "Semidefinite programming relaxations for semialgebraic problems." English. In: *Mathematical Programming* 96.2 (2003), pp. 293–320. ISSN: 0025-5610. DOI: 10.1007/s10107-003-0387-5. URL: dx.doi.org/10.1007/s10107-003-0387-5 (page 34).

[PH07] Mathias Péron and Nicolas Halbwachs. "An Abstract Domain Extending Difference-Bound Matrices with Disequality Constraints." In: *Verification, Model Checking, and Abstract Interpretation, 8th International Conference, VMCAI 2007, Nice, France, January 14-16, 2007, Proceedings*. Ed. by Byron Cook and Andreas Podelski. Vol. 4349. Lecture Notes in Computer Science. Springer, 2007, pp. 268–282. ISBN: 978-3-540-69735-0. DOI: 10.1007/978-3-540-69738-1_20. URL: http://dx.doi.org/10.1007/978-3-540-69738-1_20 (page 56).

[PW07] Andreas Podelski and Silke Wagner. "Region Stability Proofs for Hybrid Systems." In: *Formal Modeling and Analysis of Timed Systems, 5th International Conference, FORMATS 2007, Salzburg, Austria, October 3-5, 2007, Proceedings*. Ed. by Jean-François Raskin and P. S. Thiagarajan. Vol. 4763. Lecture Notes in Computer Science. Springer, 2007, pp. 320–335. ISBN: 978-3-540-75453-4. DOI: 10.1007/978-3-540-75454-1_23. URL: dx.doi.org/10.1007/978-3-540-75454-1_23 (page 75).

[PS13] Pavithra Prabhakar and Miriam Garcia Soto. "Abstraction Based Model-Checking of Stability of Hybrid Systems." In: *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*. Ed. by Natasha Sharygina and Helmut Veith. Vol. 8044. Lecture Notes in Computer Science. Springer, 2013, pp. 280–295. ISBN: 978-3-642-39798-1. DOI: 10.1007/978-3-642-39799-8_20. URL: dx.doi.org/10.1007/978-3-642-39799-8_20 (page 75).

[Ran96] Anders Rantzer. "On the Kalman-Yakubovich-Popov Lemma." In: *Syst. Control Lett.* 28.1 (June 1996), pp. 7–10. ISSN: 0167-6911. DOI: 10.1016/0167-6911(95)00063-1. URL: dx.doi.org/10.1016/0167-6911(95)00063-1 (page 120).

[Ran16] Anders Rantzer. "On the Kalman-Yakubovich-Popov Lemma for Positive Systems." In: *IEEE Trans. Automat. Contr.* 61.5 (2016), pp. 1346–1349. DOI: 10.1109/TAC.2015.2465571. URL: dx.doi.org/10.1109/TAC.2015.2465571 (page 120).

[RS10] Stefan Ratschan and Zhikun She. "Providing a Basin of Attraction to a Target Region of Polynomial Systems by Computation of Lyapunov-Like Functions." In: *SIAM J. Control and Optimization* 48.7 (2010), pp. 4377–4394. DOI: 10.1137/090749955. URL: dx.doi.org/10.1137/090749955 (page 75).

[RJM13] Stefan Richter, Colin Neil Jones, and Manfred Morari. "Certification aspects of the fast gradient method for solving the dual of parametric convex programs." In: *Mathematical Methods of Operations Research* 77.3 (2013), pp. 305–321 (page 127).

[Rou15] Pierre Roux. "Formal Proofs of Rounding Error Bounds." English. In: *Journal of Automated Reasoning* (2015), pp. 1–22. ISSN: 0168-7433. DOI: 10.1007/s10817-015-9339-z. URL: dx.doi.org/10.1007/s10817-015-9339-z (pages 92, 106, 108).

[Rou13] Pierre Roux. "Static Analysis of Control Command Systems: Synthetizing Non-Linear Invariants." PhD thesis. Institut Supérieur de l'Aéronautique et de l'Espace, 2013 (page 92).

[RG13] Pierre Roux and Pierre-Loïc Garoche. "Integrating Policy Iterations in Abstract Interpreters." In: *Automated Technology for Verification and Analysis - 11th International Symposium, ATVA 2013, Hanoi, Vietnam, October 15-18, 2013. Proceedings*. Ed. by Dang Van Hung and Mizuhito Ogawa. Vol. 8172. Lecture Notes in Computer Science. Springer, 2013, pp. 240–254. ISBN: 978-3-319-02443-1. DOI: 10.1007/978-3-319-02444-8_18 (pages 22, 28).

[Rou+12] Pierre Roux, Romain Jobredeaux, Pierre-Loïc Garoche, and Éric Féron. "A Generic Ellipsoid Abstract Domain for Linear Time Invariant Systems." In: *Proceedings of the 15th ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2012, Beijing, China, April 17-19, 2012*, ed. by Thao Dang and Ian Mitchell. ACM, 2012, pp. 105–114. ISBN: 978-1-4503-1220-2. DOI: 10.1145/2185632.2185651. URL: doi.acm.org/10.1145/2185632.2185651 (page 85).

[RTC11] Special C. RTCA. *DO-178C, Software Considerations in Airborne Systems and Equipment Certification*. 2011 (page 3).

[Rum10] Siegfried M. Rump. "Verification methods: Rigorous results using floating-point arithmetic." In: *Acta Numerica* 19 (May 2010), pp. 287–449. ISSN: 1474-0508. DOI: 10.1017/S096249291000005X. URL: journals.cambridge.org/article_S096249291000005X (page 90).

[Rum06] Siegfried M. Rump. "Verification of positive definiteness." In: *BIT Numerical Mathematics* 46 (2006), pp. 433–452 (page 106).

[Rus12] John Rushby. "The versatile synchronous observer." In: *Proceedings of the 15th Brazilian conference on Formal Methods: foundations and applications*. SBMF'12. Natal, Brazil: Springer-Verlag, 2012, pp. 1–1. ISBN: 978-3-642-33295-1. DOI: 10.1007/978-3-642-33296-8_1. URL: dx.doi.org/10.1007/978-3-642-33296-8_1 (page 78).

[SSM04] Sriram Sankaranarayanan, Henny Sipma, and Zohar Manna. "Non-linear loop invariant generation using Gröbner bases." In: *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2004, Venice, Italy, January 14-16, 2004*. Ed. by Neil D. Jones and Xavier Leroy. ACM, 2004, pp. 318–329. ISBN: 1-58113-729-X. DOI: 10.1145/964001.964028. URL: http://doi.acm.org/10.1145/964001.964028 (page 56).

[ST11] Sriram Sankaranarayanan and Ashish Tiwari. "Relational Abstractions for Continuous and Hybrid Systems." In: *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*. Ed. by Ganesh Gopalakrishnan and Shaz Qadeer. Vol. 6806. Lecture Notes in Computer Science. Springer, 2011, pp. 686–702. ISBN: 978-3-642-22109-5. DOI: 10.1007/978-3-642-22110-1_56. URL: dx.doi.org/10.1007/978-3-642-22110-1_56 (page 75).

[SB13] Yassamine Seladji and Olivier Bouissou. "Numerical Abstract Domain using Support Functions." In: *NFM*. 2013 (page 56).

[SSS00] Mary Sheeran, Satnam Singh, and Gunnar Stålmarck. "Checking Safety Properties Using Induction and a SAT-Solver." In: *FMCAD*. Ed. by Warren A. Hunt Jr. and Steven D. Johnson. Vol. 1954. LNCS. Springer, 2000, pp. 108–125. ISBN: 3-540-41219-0. DOI: 10.1007/3-540-40922-X_8. URL: dx.doi.org/10.1007/3-540-40922-X_8 (pages 14, 49).

[SIAM96] "Inquiry Board Traces Ariane 5 Failure to Overflow Error." In: *SIAM News* 29.8 (1996). available on internet archive, pp. 12–13. URL: web.archive.org/web/19970605165252/www.siam.org/siamnews/general/ariane.htm (page 21).

[SG09] Saurabh Srivastava and Sumit Gulwani. "Program Verification Using Templates over Predicate Abstraction." In: *SIGPLAN Not.* 44.6 (June 2009), pp. 223–234. ISSN: 0362-1340. DOI: 10.1145/1543135.1542501. URL: doi.acm.org/10.1145/1543135.1542501 (page 38).

[SF97] Jorge Stolfi and Luiz Henrique de Figueiredo. *Self-Validated Numerical Methods and Applications*. Brazilian Mathematics Colloquium monograph. IMPA, Rio de Janeiro, Brazil, July 1997 (page 99).

[Tur49] Alan M. Turing. "Checking a large routine." In: *Report of a Conference on High Speed Automatic Calculating Machines*. 1949. URL: http://www.turingarchive.org/browse.php/B/8 (page 8).

[VB96] Lieven Vandenberghe and Stephen Boyd. "Semidefinite Programming." In: *SIAM Review* 38.1 (1996), pp. 49–95. DOI: 10.1137/1038003. eprint: epubs.siam.org/doi/pdf/10.1137/1038003. URL: epubs.siam.org/doi/abs/10.1137/1038003 (page 33).

[Vav90] Stephen A. Vavasis. "Quadratic programming is in NP." In: *Information Processing Letters* 36.2 (1990), pp. 73–77. ISSN: 0020-0190. DOI: dx.doi.org/10.1016/0020-0190(90)90100-C. URL: www.sciencedirect.com/science/article/pii/002001909090100C (page 95).

[Vin01] Glenn Vinnicombe. *Uncertainty and Feedback: H [infinity] Loop-shaping and the [nu]-gap Metric*. World Scientific, 2001 (page 74).

[Wan15] Timothy Wang. "Credible Autocoding of Control Software." PhD thesis. School of Engeering – Georgia Tech, 2015 (page 82).

[Wan+16a] Timothy Wang, Romain Jobredeaux, Heber Herencia-Zapana, Pierre-Loïc Garoche, Arnaud Dieumegard, Éric Féron, and Marc Pantel. "From Design to Implementation: An Automated, Credible Autocoding Chain for Control Systems." In: *Advances in Control System Technology for Aerospace Applications*. Ed. by Éric Féron. Vol. 460. Lecture Notes in Control and Information Sciences. Springer Berlin Heidelberg, 2016, pp. 137–180. ISBN: 978-3-662-47693-2. DOI: 10.1007/978-3-662-47694-9_5 (pages 17, 82, 115).

[Wan+16b] Timothy Wang, Romain Jobredeaux, Marc Pantel, Pierre-Loïc Garoche, Éric Féron, and Didier Henrion. "Credible Autocoding of Convex Optimization Algorithms." In: *Optimization and Engineering* (2016). Ed. by Springer. to appear. URL: arxiv.org/abs/1403.1861 (pages 127, 128).

[Wil72] Jan C Willems. "Dissipative dynamical systems part I: General theory." In: *Archive for rational mechanics and analysis* 45.5 (1972), pp. 321–351 (page 74).

[Yak71] Vladimir A. Yakubovich. "S-procedure in nonlinear control theory." In: *Vestnik Leningrad University* 1 (1971), pp. 62–77 (page 119).

[Yak62] Vladimir A. Yakubovich. "The solution of certain matrix inequalities in automatic control theory." In: *Soviet Math. Dokl.* Vol. 3. 1962, pp. 620–623 (page 119).

[Yam+10] Makoto Yamashita, Katsuki Fujisawa, Kazuhide Nakata, Maho Nakata, Mituhiro Fukuda, Kazuhiro Kobayashi, and Kazushige Goto. *A high-performance software package for semidefinite programs : SDPA7*. Tech. rep. Tokyo Institute of Technology, Tokyo, Japan: Dept. of Information Sciences, 2010. URL: www.optimization-online.org/DB_FILE/2010/01/2531.pdf (page 33).

[Yan92] Qinping Yang. "Minimum Decay Rate of a Family of Dynamical Systems." PhD thesis. Stanford University, 1992 (pages 40, 71).