



HAL
open science

Synthesizing invariants : a constraint programming approach based on zonotopic abstraction

Bibek Kabi

► **To cite this version:**

Bibek Kabi. Synthesizing invariants : a constraint programming approach based on zonotopic abstraction. Computer science. Institut Polytechnique de Paris, 2020. English. NNT : 2020IPPAX017 . tel-02925914

HAL Id: tel-02925914

<https://theses.hal.science/tel-02925914v1>

Submitted on 31 Aug 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



INSTITUT
POLYTECHNIQUE
DE PARIS

NNT : 2020IPPAX017

Thèse de doctorat



Synthesizing invariants: a constraint programming approach based on zonotopic abstraction

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à l'École Polytechnique

École doctorale n°626 École doctorale de l'Institut Polytechnique de Paris (ED IP Paris)

Spécialité de doctorat : Informatique

Thèse présentée et soutenue à Paris, France, le 24 Juin 2020, par

BIBEK KABI

Composition du Jury :

Laurent Fribourg Directeur de Recherche, ENS Paris-Saclay (LSV)	Président
Charlotte Truchet Maître de Conférences, Université de Nantes (LINA)	Rapportrice
Michel Rueher Professeur Émérite, Université Cote d'Azur	Rapporteur
Khalil Ghorbal Chargé de Recherche, INRIA Rennes	Examineur
Antoine Miné Professeur, Sorbone Université (LIP6)	Examineur
Eric Goubault Professeur, École polytechnique (LIX)	Examineur
Sylvie Putot Professeur, École polytechnique (LIX)	Directrice de thèse
Eric Goubault Professeur, École polytechnique (LIX)	Co-directeur de thèse



RÉSUMÉ

Les systèmes dynamiques sont des modèles mathématiques pour décrire l'évolution temporelle de l'état d'un système. Il y a deux classes de systèmes dynamiques pertinentes à cette thèse : les systèmes discrets et les systèmes continus. Dans *les systèmes dynamiques discrets* (ou les programmes informatiques classiques), l'état évolue avec un pas de temps discrets. Dans *les systèmes dynamiques continus*, l'état du système est fonction du temps continu, et son évolution caractérisée par des équations différentielles. Étant donné que ces systèmes peuvent prendre des décisions critiques, il est important de pouvoir vérifier des propriétés garantissant leur sûreté. Par exemple, sur un programme, l'absence de débordement arithmétique.

Dans cette thèse, nous développons un cadre pour la vérification automatique des propriétés de sûreté des programmes. Un élément clé de cette vérification est la preuve de propriétés invariantes. Nous développons ici un algorithme pour synthétiser des *invariants inductifs* (des propriétés vraies pour l'état initial, qui sont stables dans l'évolution des états du programme, donc sont toujours vraies par récurrence) pour des programmes numériques. *L'interprétation abstraite* (IA) est une approche traditionnelle pour la recherche d'invariants inductifs des programmes numériques. L'IA interprète les instructions du programme dans un *domaine abstrait* (par exemple intervalles, octogones, polyèdres, zonotopes), domaine qui est choisi en fonction des propriétés à prouver. Un invariant inductif peut être calculé comme limite possiblement infinie des itérées d'une fonctionnelle croissante. L'analyse peut recourir aux *opérateurs d'élargissement* pour forcer la convergence, au détriment de la précision. Si l'invariant n'est pas prouvé, une solution standard est de remplacer le domaine par un nouveau domaine abstrait davantage susceptible de représenter précisément l'invariant.

La *programmation par contraintes* (PPC) est une approche alternative pour synthétiser des invariants, traduisant un programme en contraintes, et les résolvant en utilisant des solveurs de contraintes. Les contraintes peuvent opérer sur des domaines soit discrets, soit continus. La programmation classique par contraintes continues est basée sur un domaine d'intervalle, mais peut approximer une forme invariante complexe par une collection d'éléments abstraits. Une approche existante combine IA et PPC, raffinant de façon itérative, par *découpage* et contraction, une collection d'éléments abstraits, jusqu'à obtenir un invariant inductif. Celle-ci a été initialement présentée en combinaison avec intervalles et octogones. La nouveauté de notre travail est d'étendre ce cadre au domaine abstrait des zonotopes, un domaine sous-polyédrique qui présente un bon compromis en terme de précision et de coût. Cette extension demande de définir de nouveaux opérateurs sur les zonotopes, pour permettre

le découpage et la contraction, ainsi que d'adapter l'algorithme générique.

Nous introduisons notamment un nouvel algorithme de découpage de zonotopes basé sur un *pavage* par sous-zonotopes et parallélotopes. Nous proposons également des alternatives à certains opérateurs existants sur les zonotopes, mieux adaptés que les existants à la méthode. Nous avons implémenté ces opérations dans la bibliothèque APRON et avons testé l'approche sur des programmes présentant des invariants complexes, éventuellement non convexes. Les résultats démontrent un bon compromis par rapport à l'utilisation de domaines simples, comme les intervalles et les octogones, ou d'un domaine plus coûteux comme les polyèdres. Enfin, nous discutons de l'extension de l'approche pour trouver des ensembles d'invariants positifs pour des systèmes dynamiques continus.



ABSTRACT

Dynamical systems are mathematical models for describing temporal evolution of the state of a system. There are two classes of dynamical systems relevant to this thesis : discrete and continuous. In *discrete dynamical systems* (or classical computer programs), the state evolves in discrete time steps, as described by difference equations. In *continuous dynamical systems*, the state of the system is a function of continuous time, characterized by differential equations. When we analyse the behaviour of a dynamical system, we usually want to make sure that it satisfies a *safety property* expressing that nothing bad happens. An example of a safety property of programs is the *absence of arithmetic overflows*. In this thesis, we design a framework related to the automatic verification of the *safety properties* of programs. Proving that a program satisfies a safety property of interest involves an invariance argument.

We develop an algorithm for inferring invariants more precisely *inductive invariants* (properties which hold during the initial state, remains stable under the program evolution, and hence hold always due to induction) for numerical programs. A traditional approach for finding inductive invariants in programs is *abstract interpretation* (AI) that interprets the states of a program in an *abstract domain* (intervals, polyhedra, octagon, zonotopes) of choice. This choice is made based on the property of interest to be inferred. Using the AI framework, inductive invariant can be computed as limits of iterations of functions. However, for abstract domains which feature infinite increasing chain, for instance, interval, these computations may fail to converge. Then, the classical solution would be to withdraw that particular domain and in its place redesign a new abstract domain which can represent the shape of the invariant. One may also use convergence techniques like *widening* to enforce convergence, but this may come at the cost of precision. Another approach called *constraint programming* (CP), can be used to find invariants by translating a program into constraints and solving them by using constraint solvers. Constraints in CP primarily operate on domains that are either discrete or continuous.

Classical *continuous constraint programming* corresponds to interval domain and can approximate a complex shape invariant by a set of boxes, for instance, upto a precision criterion. An existing framework combines AI and continuous CP inspired by iterative refinement, *splitting* and tightening a collection of abstract elements. This was initially presented in combination with simple underlying abstract elements, boxes and octagons. The novelty of our work is to extend this framework by using zonotopes, a sub-polyhedric domain that shows a good compromise between cost and precision. However, zonotopes are not closed under intersection, and we had to extend the existing framework, in addition to designing new operations on zonotopes.

We introduce a novel splitting algorithm based on *tiling* zonotopes by sub-zonotopes and parallelotopes. We also propose few alternative operators to the existing ones for a better efficiency of the method. We implemented these operations on top of the APRON library, and tested it on programs with non-linear loops that present complex, possibly non-convex, invariants. We present some results demonstrating the interest of this splitting-based algorithm to synthesize invariants on such programs. This algorithm also shows a good compromise by its use in combination with zonotopes as regards to its use with both simpler domains such as boxes and octagons, and more expressive domains like polyhedra. Finally, we discuss the extension of the approach to infer positive invariant sets for dynamical systems.



CONTENTS

RÉSUMÉ	1
ABSTRACT	3
LIST OF FIGURES	7
LIST OF TABLES	10
LIST OF ALGORITHMS	10
I INTRODUCTION AND STATE OF THE ART	13
1 INTRODUCTION	14
1.1 Motivation	14
1.1.1 Safety properties of programs	14
1.2 Our contribution	20
1.3 Thesis outline	21
2 ABSTRACT INTERPRETATION	22
2.1 Abstract interpretation	22
2.2 Notations and Definitions	23
2.3 Numerical abstract domains	31
2.3.1 Non-Relational Abstract Domain	32
2.3.2 Relational Abstract Domains	33
Polyhedras	33
Ellipsoids	35
2.3.3 Weakly-relational abstract domains	35
Octagons	37
Template polyhedra	38
Affine sets or zonotopes	38
Parallelotope abstract domains	45
2.3.4 Combining abstract domains	46
2.3.5 Support libraries	47
2.3.6 Abstract interpretation tools	47
3 CONSTRAINT PROGRAMMING	49
3.1 From AI to CP	49
3.2 Constraint programming	50
3.2.1 Propagation	51

3.2.2	Splitting	52
3.2.3	A continuous solver	52
4	INTERACTIONS BETWEEN ABSTRACT INTERPRETATION AND CONSTRAINT PROGRAMMING	54
4.1	Are we introducing AI ideas into CP or CP into AI?	54
4.2	Refinement-based inductive invariant inference	55
4.2.1	Concrete semantics.	55
4.2.2	Target invariant.	56
4.2.3	Abstract semantics.	56
4.3	Search algorithm.	58
4.3.1	Coverage	60
4.3.2	Tightening	62
4.3.3	Splitting	62
4.3.4	Size	62
4.3.5	Failure	62
4.3.6	Data structure.	63
4.4	Related work	64
4.4.1	CP using SAT/SMT solvers	64
4.4.2	SAT-based model checking	65
4.4.3	Combined AI and CP approaches	65
4.4.4	Learning loop invariants	66
4.4.5	Eigen vectors as invariants	67
II	INVARIANTS OF DISCRETE SYSTEMS	68
5	ZONOTOPES AND CONSTRAINT SOLVING	69
5.1	Constraint solving algorithm on zonotopes	69
5.2	Inclusion test	71
5.3	Intersection test	74
5.4	Meet	74
5.5	Size	81
5.6	Volume of a zonotope.	82
5.7	Coverage metric	83
5.7.1	Test for benign.	85
5.8	Splitting	86
5.8.1	Splitting with overlap	86
5.8.2	Effect of partitioning on splitting	88
5.8.3	Splitting zonotopes by tiling	91
	Concepts and Definitions	91
	A survey on zonotopal tilings	94
	De Bruijn grids school	94
	Hyperplane arrangement-matroid theory school	97
	Is zonotopal tiling a vertex enumeration problem?	98
	Our tiling algorithm	101
	Hyperplane arrangements and zonotopes	101
	Enumerating sign vectors	102
	Notions whose sequel is the tiling algorithm	102
	Examples	105

6	IMPLEMENTATION AND EXPERIMENTS ON PROGRAMS	118
6.1	Implementation	118
6.1.1	Apron	118
6.1.2	Taylor1+	118
6.1.3	Our contribution with respect to implementation	119
6.2	Experiments	119
6.3	Conclusion	130
III INVARIANTS OF CONTINUOUS SYSTEMS		132
7	INVARIANTS OF DYNAMICAL SYSTEMS	133
7.1	Preliminaries	133
7.2	The CP algorithm revisited	136
7.3	Taylor model approximation of flow map	148
7.3.1	Picard iteration	152
	Operations on Taylor models	157
7.4	Examples: illustrating evaluation of remainder interval by Picard operator	158
7.5	Conclusion	171
CONCLUSION AND FUTURE SCOPE		172
BIBLIOGRAPHY		174



LIST OF FIGURES

1.1	Example program.	16
1.2	Non-inductive invariants for the program in Figure 1.1.	16
1.3	Inductive invariant found for the program in Figure 1.1	18
1.3	Inductive invariant found for the program in Figure 1.1	19
2.1	(a) Example program; (b) The reachable states (s_0, s_1) form an ellipsoid	26
2.2	Inductive invariant for the program in Figure 2.1	27
2.3	The image of abstract elements in Figure 2.2 by a loop iteration in the abstract domain used for computing the inductive invariant	27
2.4	Superposition of the two figures 2.2 and 2.3 showing that 2.3 is included in 2.2, i.e., 2.2 is inductive	28
2.5	Kleene iterations in the interval domain	29

2.6	Example program	33
2.7	(a) The interval abstract values for the target invariant of the program in Figure 2.7 and its image after one iteration of the body loop; (b) In blue: the target invariant (X), in pink: $F^\sharp(X)$.	33
2.8	(a) Halfspace representation for the target invariant of the program in Figure 2.7 and its image after one iteration of the body loop using polyhedra abstract domain; (b) H-representation, in blue: the target invariant (X), in pink: $F^\sharp(X)$; (c) Vertex or V-representation, In blue: convex hull of the vertices corresponding to the target invariant (X), in pink: convex hull of the vertices corresponding to the image ($F^\sharp(X)$) of the target invariant.	36
2.9	(a) Octagonal constraints for the target invariant of the program in Figure 2.7 and its image after one iteration of the body loop; (b) In blue: target invariant (X) abstracted using octagon abstract domain, in pink: its image $F^\sharp(X)$	39
2.10	Zonotope concretization $\gamma(A)$	41
2.11	Linear concretization $\gamma_{\text{lin}}(A_+)$ of affine set (\hat{x}, \hat{y}) without its center	43
2.12	(a) Affine forms corresponding to the target invariant of the program in Figure 2.7 and its image after one iteration of the body loop; (b) In blue: target invariant (X) abstracted using zonotopic abstract domain, in pink: its image $F^\sharp(X)$	45
4.1	Inductive invariant for the program in Figure 2.6	57
4.2	The image of abstract elements in Figure 2.2 by a loop iteration in the abstract domain used for computing the inductive invariant	57
4.3	Superposition of the two figures 4.1 and 4.2 showing that 4.2 is included in 4.1, i.e., 4.1 is inductive	58
4.4	A collection of abstract elements manipulated by the algorithm	59
4.5	Classification of abstract elements	59
4.6	<i>Useful</i> abstract elements	60
4.7	Computation of coverage information	61
5.1	Intersecting case	74
5.2	Point inclusion for intersecting case	74
5.3	Non-intersecting case	75
5.4	Point inclusion for non-intersecting case	75
5.5	The zonotope concretization of S_0 and $F^\sharp(S_0)$	78
5.6	The hyperplanes with which the intersection of \mathfrak{Z}_1 will be computed	79
5.7	The over-approximation of the intersection of \mathfrak{Z}_1 and the half-space in the direction u_1	80
5.8	The over-approximation of the intersection of \mathfrak{Z}_3 and the half-space in the direction u_2	80
5.9	The zonotopic over-approximation of the intersection $\mathfrak{Z}_1 \cap \mathfrak{Z}_2$	81
5.10	An overview of the data structure based on partitioning used for Algorithm 5.1	84
5.11	Sub-zonotopes obtained after splitting	87
5.12	Computing the coverage measure in case of overlapping zonotopes	88
5.13	Split zonotopes for example 1 illustrating the issue with conventional coverage measure	89
5.14	Inductive invariant for the program in Figure 2.6 with the target invariant being the box $[-2, 2]$ abstracted using zonotopes	89
5.15	In red, the image of the zonotopes in Figure 5.14 by a loop iteration in the zonotope abstract domain and superposition of both showing that Figure 5.14 is inductive	90

5.16	Partitioning and its effect on splitting by overlap	91
5.17	Figures illustrating the ideas of fixing and freeing the signs of generators	93
5.18	De Bruijn lines of a two-dimensional tiling.	95
5.19	Examples of tilings	96
5.20	A hyperplane arrangement in \mathbb{R}^2 with four lines.	98
5.21	Polar dual of the hyperplane arrangement in Figure 5.20, i.e., a zonotope.	99
5.22	A tiling of the zonotope in Figure 5.21 and the sign vectors of the corresponding tiles.	100
5.23	2^5 vertices of the 5-dimensional hypercube projected with the generator matrix.	100
5.24	Arrangement of hyperplanes.	103
5.25	Ray shooting and sign enumeration.	103
5.26	The primitive zonotope and its tiling	104
5.27	Illustrating, how fixing the sign of a zonotope defined by 3 generators in 2-dimension implicitly enumerates all the tiles	105
5.28	Illustrating, how fixing the sign of a zonotope defined by 4 generators in 4-dimension implicitly enumerates all the tiles	106
5.29	Illustrating one-by-one all sub-zonotopes obtained after fixing the sign of generators	108
5.30	Illustrating one-by-one all parallelotopic tiles being enumerated	110
5.31	Illustrating one-by-one all sub-zonotopes obtained after fixing the sign of generators	114
5.32	Illustrating one-by-one all parallelotopic tiles being enumerated	115
5.33	Illustrating one-by-one all parallelotopic tiles being enumerated	116
5.33	Illustrating one-by-one all parallelotopic tiles being enumerated	117
5.34	3-dimensional parallelotopic tiles delineating the zonotope in Figure 5.31a.	117
6.1	Inductive invariant for <i>Filter</i> example	122
6.1	Inductive invariant for <i>Filter</i> example	123
6.2	Inductive invariant for <i>Sine</i> example	124
6.2	Inductive invariant for <i>Sine</i> example	125
6.3	Inductive invariant for <i>Newton</i> example	126
6.3	Inductive invariant for <i>Newton</i> example	127
6.4	Inductive invariant for <i>Newton2</i> example	128
6.4	Inductive invariant for <i>Newton2</i> example	129
6.5	Structure of a program for the analyzer	131
7.1	Hénon attractor	142
7.2	An outer-approximation of the positive invariant set of Hénon map (in blue are the abstract elements which are benign, in pink are the ones whose state cannot be decided by the algorithm, and in red is the image of the abstract elements by a loop iteration in the abstract domain used for computing the positive invariant set)	143
7.3	The set obtained using the CP Algorithm 7.1 upto a size criterion for an iterated function sequence F, F^2	144
7.4	The set obtained using the CP Algorithm 7.1 upto a size criterion for an iterated function sequence F, F^2, F^3	145

7.5	The set obtained using the CP Algorithm 7.1 upto a size criterion for an iterated function sequence F, F^2, F^3, F^4	146
7.6	An outer-approximation of the positive invariant for the Van-der-Pol oscillator described by the map shown in Equation 7.31	149
7.7	The image of abstract elements in Figure 7.6 by a loop iteration in the abstract domain used for computing the positive invariant set	150
7.8	Superposition of the two figures 7.6 and 7.7 showing the abstract elements which belong to the invariant set	151
7.9	The set obtained using the CP Algorithm 7.1 upto a size criterion for an iterated function sequence F, F^2	152
7.10	The set obtained using the CP Algorithm 7.1 upto a size criterion for an iterated function sequence F, F^2, F^3	153
7.11	The set obtained using the CP Algorithm 7.1 upto a size criterion for an iterated function sequence F, F^2, F^3, F^4	154
7.12	The set obtained using the CP Algorithm 7.1 upto a size criterion for an iterated function sequence F, F^2, F^3, F^4, F^5	155
7.13	Taylor model over-approximation for the function $\exp(x)$	157
7.14	The set obtained after using our CP Algorithm on every new Taylor model (6 order) evaluated at each time step over an interval $[0, 5.20]$.168	
7.15	The set obtained after using our CP Algorithm on every new Taylor model (6 order) evaluated at each time step over an interval $[0, 6.70]$.169	
7.16	The set obtained after using our CP Algorithm on every new Taylor model (7 order) evaluated at each time step over an interval $[0, 6.80]$.170	



LIST OF TABLES

6.1	Experimental results with tightening applied only during first iteration.	120
6.2	Experimental results with tightening (tightening is applied after each split).	130



LIST OF ALGORITHMS

3.1	A simple continuous solver	53
-----	--------------------------------------	----

4.1	A CP based AI algorithm for inferring inductive invariants [MBR16]	58
5.1	The zonotopic variant of the CP based AI algorithm 4.1 for inferring inductive invariants	71
5.2	Tiling Algorithm	107
7.1	The zonotopic variant of the CP based AI algorithm 4.1 for inferring inductive invariants while considering an iterated map sequence F, F^2, \dots, F^n	140

PART I

INTRODUCTION AND STATE OF
THE ART



1.1 MOTIVATION

Cyber-physical systems (CPS) are systems which combine the cyber world (computation/communication/data storage) with physical entities. A few examples of CPS are robots, cars, air-crafts, power plants, etc. With their ubiquitous presence in this society, verifying the correctness of programs and systems is becoming a major challenge.

In order to rely on them so as not to put our lives at stake, it is crucial to verify if they satisfy safety properties. For instance, an unmanned aerial vehicle (UAV) includes a mechanical body, remote ground control system, sensors (camera, inertial measurement unit, GPS, etc.), actuators, software and additional hardware like battery, electronic speed controller (ESC) and motors. All together it makes it a cyber-physical system. For such an UAV control system, it is important to verify if the UAV could potentially be involved in a collision within the next short period of time. This is where *dynamical systems* play a major role in approaches for studying whether a CPS satisfies crucial safety properties. They are mathematical models for describing temporal evolution of the state of a system.

Dynamical systems can contain discrete and continuous components. In discrete dynamical systems (or classical computer programs), the state evolves in discrete time steps, as described by difference equations. In continuous dynamical systems, the state of the system is a function of continuous time, characterized by differential equations. Determining the safety of a dynamical system requires to prove that the system is continuously safe.

A considerable part of this thesis is focused on developing method for the verification of safety properties of numerical programs. Additionally, we discuss the extension of this method to prove the safety properties for continuous time dynamical systems. In order to motivate the readers, we will present below a case study of verification of safety property of a computer program. However, prior to that we will recall the concepts from program verification that will constitute the background for all the chapters that follow.

1.1.1 *Safety properties of programs*

Ensuring whether or not a program satisfies a safety property is one of the widely studied fields in program verification. Various mathematical foundations aid the problem of program verification by providing proofs which help users to ascertain that a program is free from errors or behaves as

intended. However, for any Turing complete programming language [Tur37] this problem is undecidable i.e., it is impossible to produce any sound and complete non-trivial assertion about the computational result of any program. That is Rice’s theorem [Ric53] which is a generalization of the well-known Halting problem [Tur37]. Partly the reason for this undecidability issue is loops. This is one of the reasons why analyzing loops is very crucial in program verification.

An informal definition of safety property is, “nothing bad happens” or “something bad never happens”, that is, a program never reaches an unacceptable state. Safety properties on programs can be for instance, the fact that program variables stay within their expected bounds or some region is not reachable at some set of program locations. A safety property of interest express conditions that should be continuously maintained by the program. Hence, proving that a program satisfies a safety property of interest involves an invariance argument which is why loop invariants is a key ingredient in the verification of safety property on programs. An invariant is a property that holds on every iteration of the loop. Reasoning on invariants frees us from proving the safety of each loop iteration separately, which is costly for large loops and impossible for programs exhibiting unbounded loops and an infinite state space.

The classic method to prove that a set is indeed an invariant is to look for an inductive invariant which implies it, i.e., a state property that is stable by an iteration of the loop. An inductive invariant is an invariant (G) such that $F(G) \subseteq G$, as is used in, e.g., Floyd Hoare logic [Hoa69] [Flo67]. Furthermore, Tarski’s theorem [Tar55] states that all inductive invariants are indeed invariants.

Inductive invariants play a special role in program verification because they can be checked by running a single loop iteration and checking its stability, even for unbounded loops. It is often necessary, given a target invariant property to prove, to first strengthen it into an inductive invariant.

We will motivate the importance of this thesis work further below with the illustration of a piece of code.

EXAMPLE 1.1 We illustrate the concept of inductive invariants on a program in Figure 1.1 taken from the online additional material of [MBR16] (similar benchmarks are considered in [Mar14]) having two variables, x and y whose initial values lie in the box $I \stackrel{\text{def}}{=} [0.9, 1.1]^2$ and the effect of a loop iteration on a set X of possible variable values $(x, y) \in X$ given by the function $F: \mathcal{P}(\mathbb{R}^2) \rightarrow \mathcal{P}(\mathbb{R}^2)$ defined as the loop body of Figure 1.1.

We choose an axis aligned bounding box such as $G = [-2.1, 2.1] \times [-2.1, 2.1]$. Notice that the program state, considered as a point (x, y) , is guaranteed to lie inside G every time that execution reaches the head of the loop. In other words, G includes notably all the states reachable at the loop head, i.e., $\bigcup_{n \in \mathbb{N}} F^n(I) \subseteq G$. Then, the box $G = [-2.1, 2.1] \times [-2.1, 2.1]$, shown in blue in Figure 1.2, is a valid invariant. However, notice that $F(G) \not\subseteq G$: indeed, the transformation induced by F on G maps the box G to a circle, that goes a bit outside the box G , as illustrated in Figure 1.2. Consider the four-petals-flower shape towards the center shown in Figure 1.2 : its interior is not reachable from the initial box I , and it contains the four small circles

the effect of a loop iteration of the program is also abstracted using the same property. Now the inductive invariant is computed as limits of iterations of functions. However, for abstract domains which feature infinite increasing chain (for example, interval), these computations may fail to converge. In such a case, the classical solution would be to withdraw that particular domain and in its place redesign a new abstract domain which can represent the shape of the invariant. One may also use techniques like widening to enforce convergence, but this may come at the cost of precision.

In this thesis, we will focus on a particular abstract domain: zonotope.

Zonotope abstract domain. Zonotope is an implicitly relational abstract domain. It is based on affine forms. It is a cost-effective, versatile, and precise abstract domain that can represent restricted forms of polyhedra as Minkowski sums of line segments. It features more lightweight algorithms than general polyhedra, while being more expressive than other sub-polyhedra domains (e.g., octagons). They are particularly well suited to approximate non-linear functions. Zonotopes do not form a lattice and do not enjoy an exact intersection. Thus, the major subject of this thesis is the introduction of new operators for this domain.

Another popular technique called constraint programming (CP), is used to find invariants by translating a program into constraints and solving them by using constraint solvers.

Constraint Programming. *Constraint programming* (CP) is a method for solving combinatorial problems, by expressing them as conjunctions of first-order logic formulas. It is a paradigm which formalizes invariant synthesis problem using constraints and solves them using efficient algorithms. These algorithms inherently know how to approximate a complex shape by a set of boxes, for instance, up to a precision criterion. Constraints in CP primarily operate on domains that are either discrete or continuous. Classical continuous constraint solving, over real-valued variables, works by refining the domain of the variables, i.e., a box representing candidate solutions: the box is tightened as much as possible by removing variable values that cannot participate in a solution. Whenever the box cannot be tightened anymore, it is split into two or more boxes, that are tightened and split themselves iteratively, until every box either contains only solutions, or no solution, or has a size below a user-defined threshold. When the algorithm terminates, it returns a set of definitive and candidate solutions as a collection of boxes.

Synthesizing invariants of programs has been an active research from early days of computer science, and recently many techniques which combine AI and CP have sprung up. In this thesis, we combine the zonotope based abstraction with an existing CP algorithm [MBR16] for inferring inductive invariants. The idea of [MBR16], inspired from constraint programming approaches, is to synthesize an inductive invariant as a collection of abstract elements, that are iteratively split and refined. In set-based constraint programming, these elements are generally boxes. Previous work [MBR16] was limited to abstract domains that are closed by intersection and required non-standard operations: split and size, such as octagons. In this thesis, we extend this work

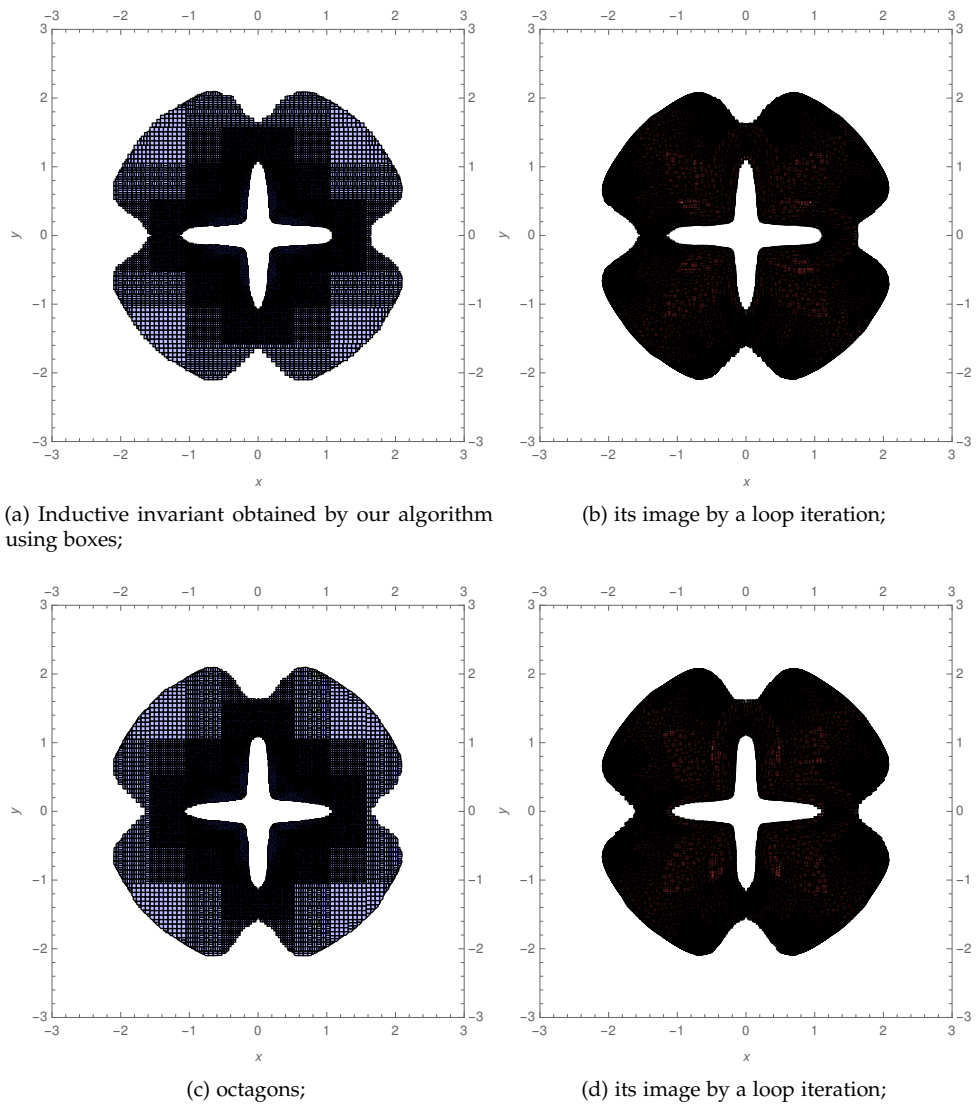


Figure 1.3 – Inductive invariant found for the program in Figure 1.1

to zonotopes, which we show they provide an interesting trade-off between expressiveness and efficiency for such a use, by comparing their use with that of boxes, octagons, and polyhedra.

EXAMPLE 1.2 Recall that no box is an inductive invariant for the program in Figure 1.1. The possible shapes¹ are as illustrated in Figures 1.3a, 1.3c, 1.3e and 1.3g. Instead of trying to guess the shape of the invariant, we will look for a set of abstract elements as shown in Figures 1.3a, 1.3c, 1.3e and 1.3g such that one iteration of the loop that is its image (shown in Figures 1.3b, 1.3d, 1.3f and 1.3h) obtained by a computable abstract function modeling the effect

¹Method for synthesizing this inductive invariant set is discussed in the following chapter

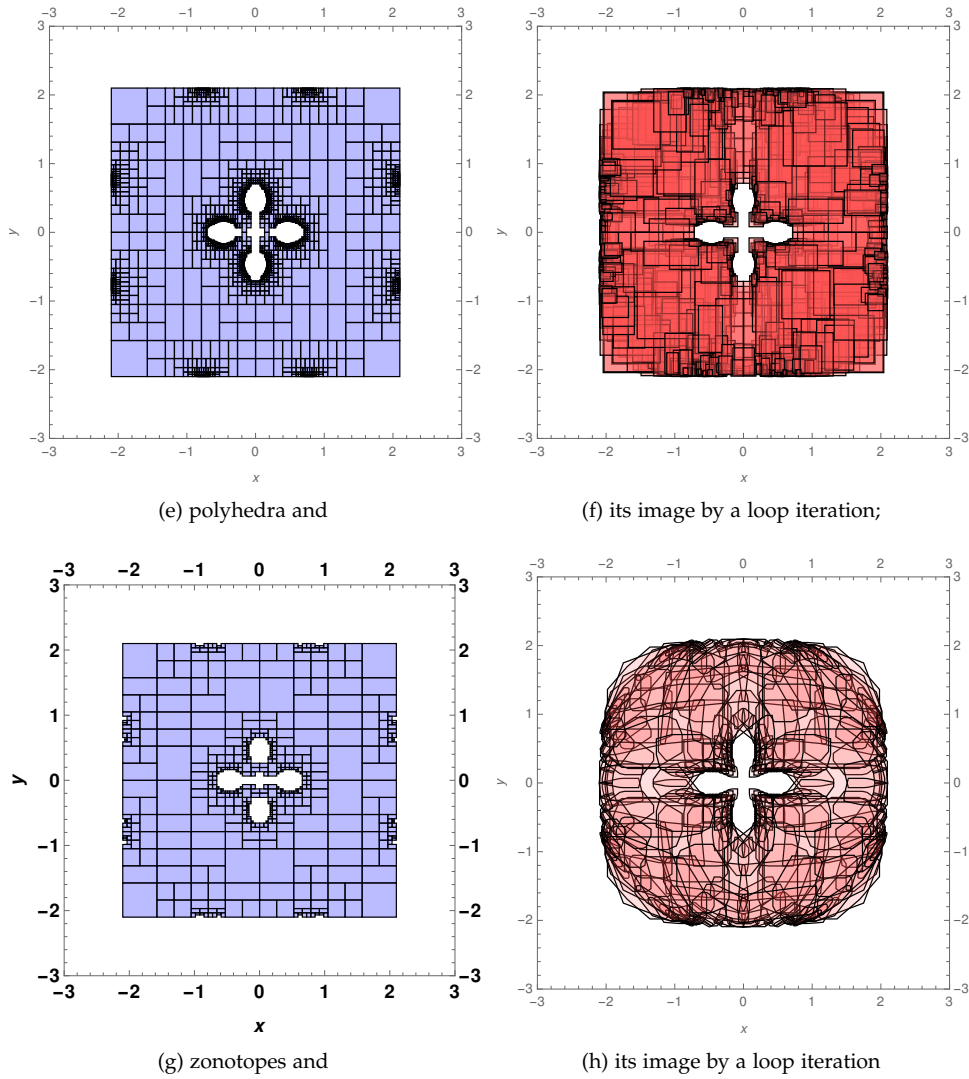


Figure 1.3 – Inductive invariant found for the program in Figure 1.1

of a loop iteration on the abstract world) maps these set of abstract elements into a subset of itself as shown in Figure 2.4. Even though no single abstract element is an inductive invariant, a set of them can be inferred as an inductive invariant.

Figures 1.3a-1.3g show the inductive invariant within G as found by the CP algorithm in combination with box, octagon, polyhedron and zonotope abstract domains. Inference with intervals (resp. octagons, polyhedra, and zonotopes) takes 646.8 (resp. 8850.8, 126.8 and 35.6) seconds and produces an inductive invariant composed of 129781 (resp. 129767, 2368 and 488) parts. Less expressive domains such as boxes and octagons rely heavily on splitting, hence output a large set of elements and are slower in total, in spite of a smaller cost of manipulating a single abstract element. In particular, in polyhedra and zonotopes, the image by the loop body in the abstract domain, of a box on the left corner in Figure 1.3e-1.3g is within the collection of elements, thus proven invariant, whereas it won't be the case in the box or octagon abstract domains, ultimately leading by splitting to the refinement of Figures 1.3a-1.3c.

Remark 1.3. An invariant associated with a safety property of interest can be relatively simple. For example, the value of a variable is bounded by so-and-so quantity. However, inductive invariants can have much more complex shapes as shown in Figures 1.3a-1.3g which makes them difficult to compute.

1.2 OUR CONTRIBUTION

Our main goal during this thesis research was to extend an existing continuous constraint programming approach to domains which are not closed under intersection for synthesizing numerical invariants. More precisely, our contributions are:

- Zonotopes are not closed under intersection. So, we had to extend the existing framework, in addition to designing new operations on zonotopes, such as a novel splitting algorithm based on *paving* zonotopes by parallelotopes. We improved the complexity of the inclusion test on zonotopes by an exponential bound compared to the previous work. We also propose a new meet operation on zonotopes which is geometrical in nature. We implemented these operations in APRON library.
- We present a prototype that strengthens numerical invariants into inductive ones on a small language, extending the Taylor1+ zonotope abstract domain in the Apron library, and adapting the CP algorithm.
- We illustrate that the method is better than the previous one with an experimental proof on a small set of benchmarks.
- Finally, we show that our constraint programming framework can be used to find an over-approximation of positive invariants in continuous systems.

Some results described in Chapters 3 to 6 have been the subject of publications in workshops [KGP16, KGP17, KGMP20]. We also co-wrote a paper discussing the new splitting operator for zonotopes and also the implementation and experimentation with Apron.

1.3 THESIS OUTLINE

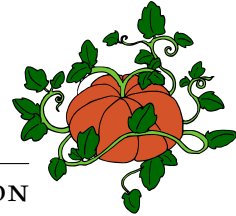
This thesis is organised as follows. Chapter 1 to 3 provide the requisite notions to understand this thesis research. Chapters 1 and 2 recall the concept of program invariants and gives an illustrative explanation as to why they must be inferred.

Chapter 2 recalls the formal framework of abstract interpretation and its application to infer inductive invariants of programs. It discusses the different abstract domains and the support libraries which implement their basic operations. This chapter also provides a brief survey on abstract interpretation based static analyzers. Chapter 3 gives the concepts of constraint programming. Chapter 4 discusses the recent continuous constraint programming approaches. It explains in detail an existing framework [MBR16] which combines AI and continuous CP inspired by an iterative refinement, splitting and tightening a collection of abstract elements. It will be extended throughout the thesis to be combined with the zonotope abstraction.

Chapter 5 defines the zonotope domain. It discusses how the constraint programming algorithm introduced in chapter 3 can be extended to domains that are not intersection-closed. It introduces new operators, such as a novel splitting algorithm based on tiling zonotopes by sub-zonotopes and parallelogons. It provides an extensive literature survey on tiling and the prerequisites to understand our tiling algorithm. It discusses the other zonotopic operators like meet, test for inclusion and intersection, and size that we needed to redesign in this context. Chapter 6 illustrates the experimental evaluation of our zonotopic abstraction based constraint programming method on programs with non-linear loops that present complex, possibly non-convex, invariants.

Chapter 7 demonstrates how the CP framework can be extended to find invariants for continuous systems.

2 CHAPTER



ABSTRACT INTERPRETATION

In the previous chapter, we exemplified why inferring invariant sets is important. During this part of the thesis, we will be discussing the state-of-the-art methods for computing these sets. This chapter talks about a method which is a general theory for approximating the semantics of programs. The prominent static analysis approaches¹ are based on this method, otherwise known as *abstract interpretation* (AI).

AI is an approach which computes properties of programs using mathematical structures (lattices), transfer functions and fixed-points. Here, we discuss the key facts about abstract interpretation by introducing these mathematical structures and different fixed-point theorems. We also introduce the different existing abstractions (which are relevant to the present work) used by the AI framework for expressing numerical properties of programs.

2.1 ABSTRACT INTERPRETATION

Recall that the main principle for proving that a property is an invariant for a loop of a program is to look for a stronger property that is an inductive invariant. The least (i.e. most precise) inductive invariant is the set of program states reachable from the initial states and can be expressed mathematically as a least fixpoint². Computing this fixpoint using the real behavior of the program, or otherwise known as concrete semantics³ (more precise) is in general undecidable. Therefore, we have to rely on an interpretation which is based on less precise or abstract semantics but computable, and hence the name *abstract interpretation* [CC77]. It is a framework which expresses program semantics as fixpoints of functions over some ordered mathematical structures.

The relationship between the concrete semantics and abstract semantics is formally known as abstraction. For example, the abstraction of the concrete semantics could be the sign, or the range of the variables instead of their precise values. Different properties of a program can be represented by different abstract semantics. The different forms of abstraction over the

¹These approaches analyze the program source code directly and without user intervention at some level of abstraction.

²A least fixpoint (a fixpoint is any α such that $F(\alpha) = \alpha$ where F is a function $F : A \rightarrow A$ and $\alpha \in A$) of a function which is a mapping from a mathematical structure to itself is a fixpoint smaller than one another fixpoint based on the structure's order

³Semantics is the set of all possible executions in all possible environments. Informally, it is the pseudo-code of one's code, or what the code means.

semantics are known as abstract domains⁴. Denoted by \mathcal{D}^\sharp such that $\mathcal{D}^\sharp \subseteq \mathcal{P}(\mathbb{R}^n)$, an abstract domain is a subset of properties of interest with a computer representation, where $\mathcal{P}(\mathbb{R}^n)$ or \mathcal{D} is the concrete domain. The transfer function F^\sharp in the abstract domain i.e., $F^\sharp: \mathcal{D}^\sharp \rightarrow \mathcal{D}^\sharp$, over-approximates the effect of $F: \mathcal{P}(\mathbb{R}^n) \rightarrow \mathcal{P}(\mathbb{R}^n)$. Formally, this alternation between the concrete and abstract world can be defined by functions: an abstraction function from \mathcal{D} to \mathcal{D}^\sharp and a concretization function from \mathcal{D}^\sharp to \mathcal{D} .

One of the widely used abstract domains is interval which abstracts set of points as a pair of bounds $[a, b]$ where $a \leq b$ [CC76]. It is based on interval arithmetic [Moo69], quite simple and inexpensive, but non-relational. There are several other relational abstract domains like affine inequalities domain, or polyhedra domain [CH78] which integrates the ability of interval abstract domain in addition to the ability to infer relation among variables. This makes the polyhedra domain very expressive. There are other restrictions of polyhedra (weakly relational) like octagons [Min06], templates [SSM05] and zonotopes [GP06, GGP09] which rely on different algorithmic axioms. We will detail about the different abstract domains in the later part of this chapter.

Henceforth, we introduce the basic concepts of abstract interpretation theory featuring the definitions of mathematical structures (partial order) and their relation to programs, characterizations about concept of fixpoints and a rich collection of fixpoint theorems. The readers can refer to [BCC⁺15, Min04, M⁺17, Gho11, Urb15] for more information.

2.2 NOTATIONS AND DEFINITIONS

We use standard notations from set theory: the empty set \emptyset , set union \cup and intersection \cap , set inclusion \subseteq , set membership \in , set difference \setminus . Consider a set X , we denote as $\mathcal{P}(A)$ the set of all the sets included in X , otherwise known as powerset. Provided with two sets X and Y , the set of functions from X to Y is denoted by $X \rightarrow Y$. We use standard notation for introducing definitions $\stackrel{\text{def}}{=}$ or $:=$, logical operators: \wedge for conjunction, \vee for disjunction, \implies for implication, \iff for equivalence, \models for entailment and quantifiers: \forall for universal quantification, \exists for existential quantification. The set of real numbers, integers are denoted by \mathbb{R} and \mathbb{Z} respectively. Consider the sets $S_1, \dots, S_k \subseteq \mathbb{R}^n$, then the Minkowski sum (denoted by \oplus) of S_1, \dots, S_k is the set $S_1 \oplus \dots \oplus S_k = \{s_1 + \dots + s_k \mid s_i \in S_i\}$. Consider two vectors x and y in \mathbb{R}^n . The inner product denoted by $\langle (x_1, x_2, \dots, x_n), (y_1, y_2, \dots, y_n) \rangle$ is given by

$$x_1y_1 + x_2y_2 + \dots + x_ny_n.$$

The ℓ_1 norm of x is defined as

$$\|x\|_1 \stackrel{\text{def}}{=} \sum_{i=1}^n |x_i|.$$

⁴It is a computer-representable abstract version of the concrete domain to compute semantic over-approximations.

Consider a matrix $M = \begin{pmatrix} a & c \\ b & d \end{pmatrix}$ with column vectors. Its determinant is denoted by $\det(M)$ and is computed as

$$\det \begin{pmatrix} a & c \\ b & d \end{pmatrix} = ad - bc.$$

A Cartesian product of n intervals is a box given by $B = I_1 \times \dots \times I_n$. Consider a program with initial values or entry sets as $I \subseteq \mathbb{R}^n$ and a transfer function as $F: \mathcal{D} \rightarrow \mathcal{D}$ where $\mathcal{D}: = \mathcal{P}(\mathbb{R}^n)$ is the concrete domain.

DEFINITION 2.1 (Concrete domain.) We are interested in inferring program invariants, i.e., properties of the state (mapping of each variable to its value) a program can be in at each program location. Thus, denoted by \mathcal{D} , where $\mathcal{D}: = \mathcal{P}(\mathbb{R}^n)$ a concrete domain corresponds to the values that can be taken by the variables throughout the program.

DEFINITION 2.2 (Transfer function.) Provided with a precondition or an initial set of states, a transfer function $F: \mathcal{P}(\mathbb{R}^n) \rightarrow \mathcal{P}(\mathbb{R}^n)$ maps sets of environments to sets of environments after the execution of the line of code being analyzed. When the transfer function is applied to a set of environments X , we denote it by $F(X)$.

DEFINITION 2.3 (Abstract domain.) Computing in $\mathcal{P}(\mathbb{R}^n)$ can be undecidable. Therefore we need a computer-representable abstract version of the concrete domain to compute semantic over-approximations. Thus denoted by \mathcal{D}^\sharp such that $\mathcal{D}^\sharp \subseteq \mathcal{P}(\mathbb{R}^n)$, an abstract domain is a subset of properties of interest with a computer representation. The computable abstract function $F^\sharp: \mathcal{D}^\sharp \rightarrow \mathcal{D}^\sharp$ over-approximates the effect of $F: \mathcal{D} \rightarrow \mathcal{D}$.

DEFINITION 2.4 (Partially ordered set or poset.) A partial order \sqsubseteq on a set X is a relation which satisfies the following axioms:

- Reflexive: $\forall x \in X: x \sqsubseteq x$
- Anti-symmetric: $\forall x, y \in X: (x \sqsubseteq y) \wedge (y \sqsubseteq x) \rightarrow x = y$
- Transitive: $\forall x, y, z \in X: (x \sqsubseteq y) \wedge (y \sqsubseteq z) \rightarrow x \sqsubseteq z$

The set X armed with such a partial order relation⁵ \sqsubseteq is called a partially ordered set or poset and can be denoted as the pair (X, \sqsubseteq) .

Remark 2.5. Partial orders are very crucial in theoretical computer science, but they also provide a mathematical foundation in programs. Consider a program prog which satisfies a specification spec , i.e., a property of the program. One of these properties could be for instance, values of any variable of prog during execution is always between a particular bound. That means, whether or not the program prog satisfies the specification spec is equivalent to a set inclusion problem, i.e., if $\text{prog} \subseteq \text{spec}$.

DEFINITION 2.6 (Complete partial order.) A poset (X, \sqsubseteq) is called complete partial order or CPO if every totally ordered subset $(M \subseteq X)$ in the poset X has

⁵A binary relation can also be pre-order, if a relation is reflexive, transitive, but not necessarily anti-symmetric.

a least upper bound, where totally ordered means $(x \sqsubseteq y) \vee (y \sqsubseteq x) \forall x, y \in M$. When a poset (X, \sqsubseteq) is a CPO then it is denoted as (X, \sqsubseteq, \sqcup)

Remark 2.7. A totally ordered subset of a poset is otherwise known as a chain.

DEFINITION 2.8 (Lattice.) A lattice is a poset $(\mathcal{L}, \sqsubseteq, \sqcup, \sqcap)$ where every collection of elements has a least upper bound, denoted by \sqcup and a greatest lower bound, denoted by \sqcap . For example, consider any two elements $a, b \in \mathcal{L}$ there is a least upper bound or join, i.e., $a \sqcup b$, and a greatest lower bound, i.e., $a \sqcap b$. We can claim that the lattice \mathcal{L} is complete if every subset \mathcal{M} of \mathcal{L} has a least upper bound $\sqcup \mathcal{M}$ and a greatest lower bound $\sqcap \mathcal{M}$ and \mathcal{L} has a least element \perp ⁶.

DEFINITION 2.9 (Monotonicity & Continuity.) Consider two posets (D_1, \sqsubseteq_1) and (D_2, \sqsubseteq_2) . A function $F : (D_1, \sqsubseteq_1) \rightarrow (D_2, \sqsubseteq_2)$ is called a monotonic function if $x \sqsubseteq_1 y \implies F(x) \sqsubseteq_2 F(y) \forall x, y \in D_1$.

Consider two complete partial orders $(D_1, \sqsubseteq_1, \sqcup_1)$ and $(D_2, \sqsubseteq_2, \sqcup_2)$. A function $F : (D_1, \sqsubseteq_1, \sqcup_1) \rightarrow (D_2, \sqsubseteq_2, \sqcup_2)$ is said to be a continuous function if for every chain $C \subseteq D_1$, $F(C)$ is also a chain, i.e., $F(C) \sqsubseteq_2 D_2$, and $F(\sqcup_1 C) = \sqcup_2 F(C)$.

DEFINITION 2.10 (Fixpoints.) Consider a partially ordered set (Y, \sqsubseteq) and a function F which is a mapping from the poset to itself, i.e., $F : Y \rightarrow Y$. A fixpoint of a transfer function F is any element X satisfying $F(X) = X \mid X \in Y$. A pre-fixpoint X is such that $F(X) \sqsupseteq X$ and a post-fixpoint X such that $F(X) \sqsubseteq X$. We denote the least fixpoint of F as $\text{lfp}_X F$ which is defined as

$$\text{lfp}_X F = \sqcap \{F\text{'s post-fixpoints larger than } X\} \quad (2.1)$$

We will denote the greatest fixpoint of F by $\text{gfp}_X F$ which is defined as

$$\text{gfp}_X F = \sqcup \{F\text{'s pre-fixpoints smaller than } X\} \quad (2.2)$$

The existence of a least fixpoint and the fact that it is the meet of all the post-fixpoints, both follow from Tarski's theorem [Tar55] defined below.

DEFINITION 2.11 (Tarski's theorem.) If $F : \mathcal{L} \rightarrow \mathcal{L}$ is a monotonic function on a complete lattice \mathcal{L} , then the set of fixpoints of F is a non-empty complete lattice and a least fixpoint exists.

Remark 2.12. Among the set of fixpoints, the least fixpoint is the smallest and unique, if exists. It can refer to critical parts of the semantics of a program. Below, we will illustrate the post-fixpoint of a computer program.

EXAMPLE 2.13 Consider a simple program shown in Figure 2.1(a) taken from [MBR16, Fer04] having two variables, s_0 and s_1 , and its loop that implements a second order digital filter. The variables s_0 and s_1 are initially set to values in $[-0.1, 0.1]$. The numbers 1.5, -0.7 are the coefficients of the filter. At each loop iteration, the variable s_0 denotes the value of the current filter output, the variable s_1 denotes the last value of the filter output and the interval $[-0.1, 0.1]$ denotes the value of the current filter input.

⁶ \perp (also called *bottom*) and \top (also called *top*) are the least and greatest elements of a poset, if they exist.

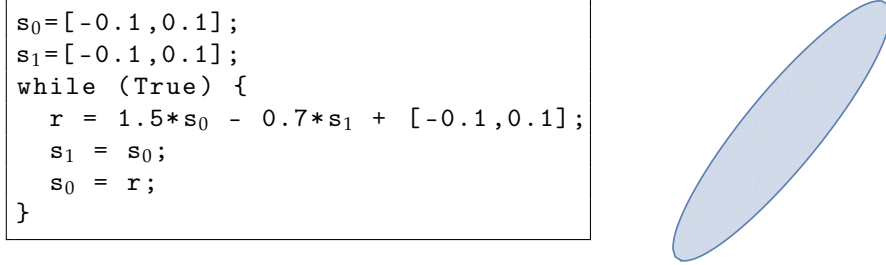


Figure 2.1 – (a) Example program; (b) The reachable states (s_0, s_1) form an ellipsoid

The initial values of variables (s_0, s_1) are in the box $I \stackrel{\text{def}}{=} [-0.1, 0.1] \times [-0.1, 0.1]$ and the effect of a loop iteration on a set X of possible variable values $(s_0, s_1) \in X$ given by the function $F: \mathcal{P}(\mathbb{R}^2) \rightarrow \mathcal{P}(\mathbb{R}^2)$ defined as $F(X) \stackrel{\text{def}}{=} \{(1.5 \times s_0 - 0.7 \times s_1 + [-0.1, 0.1], s_0) \mid (s_0, s_1) \in X\}$. The evaluation of the interval $[-0.1, 0.1]$ can be inferred as picking a different value between -0.1 and 0.1 at each loop iteration. This indeterminacy in the evaluation of the interval makes the program non-deterministic.

For the above program semantics, a set $G \subseteq \mathbb{R}^n$ (here $n = 2$) can be claimed as inductive invariant if $I \subseteq G \wedge F(G) \subseteq G$ with the least fixpoint of F or $\text{lfp}_I F$ being the smallest one. In that case any set G satisfying that $G \supseteq \text{lfp}_I F$ is an invariant which means that all inductive invariants are invariants but not all invariants are always inductive. Notably, the set of program states reachable from the initial states is the least (i.e. most precise) inductive invariant as illustrated in Figure 2.1(b). Generally, this set is difficult to compute, so we settle for an over-approximation, as any such over-approximation is also an invariant. In other words, any post-fixpoint of F is a sound over-approximation of the least fixpoint of F because $\text{lfp}_I F$ is characterized as the meet of all post-fixpoints. Thus, any post-fixpoint is a constructive expression for an inductive invariant. A post-fixpoint for the program in Figure 2.1(a) is shown in Figures 2.2-2.4.

Remark 2.14. Note that in this thesis work, given a target invariant property to prove, we first strengthen it into an inductive invariant.

Abstract interpretation provides tools to infer inductive invariants. For instance, as the limit of an iteration sequence. We discuss this below in detail.

Remark 2.15. Although Tarski's theorem ensures the existence of least fixpoint, it does not provide any ground rule on how to compute them effectively. In other words, it does not say how a post-fixpoint can be computed in abstract, where the least fixpoint being the meet of all the post-fixpoints. Thus, one of the variants of fixpoint approximation theorem is a classical method based on the work of Kleene *et al.* [KdBdGZ52] was provided by Cousot and Cousot [CC77] to infer inductive invariants, which states that least fixpoints can be

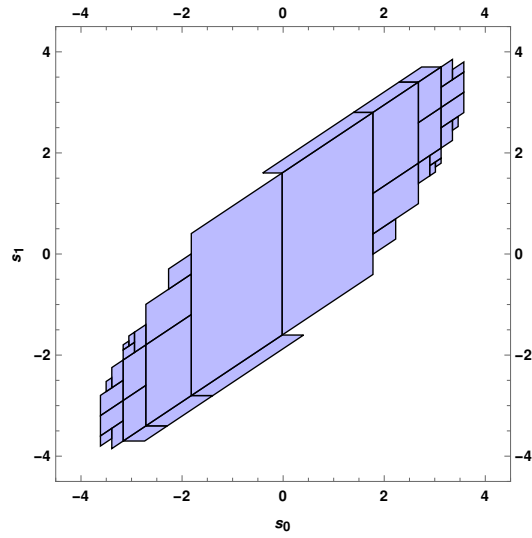


Figure 2.2 – Inductive invariant for the program in Figure 2.1

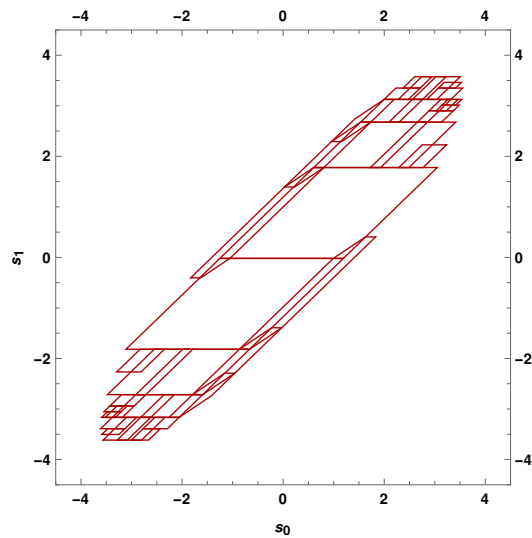


Figure 2.3 – The image of abstract elements in Figure 2.2 by a loop iteration in the abstract domain used for computing the inductive invariant

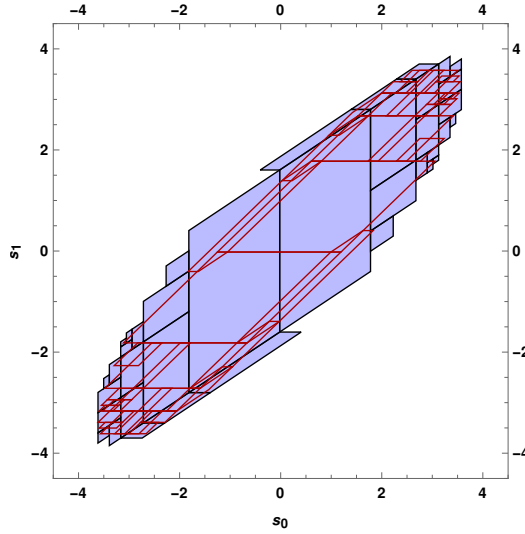


Figure 2.4 – Superposition of the two figures 2.2 and 2.3 showing that 2.3 is included in 2.2, i.e., 2.2 is inductive

computed as limits of iterations, or otherwise known as Kleene’s theorem [Def. 2.16].

DEFINITION 2.16 (Kleene’s Theorem.) If $F: \mathcal{L} \rightarrow \mathcal{L}$ is a continuous function in a complete partial order \mathcal{L} , then least fixpoint exists and it can be expressed as:

$$\text{lfp}_{\perp} F = \sqcup \{F^i(\perp) \mid i \in \mathcal{N}\} \quad (2.3)$$

where \mathcal{N} is the set of natural integers.

Remark 2.17. Unlike Tarski’s fixpoint theorem, Kleene’s fixpoint theorem [KdBdGZ52, CC77, CC79] requires only a complete partial order and more intriguing is the fact that it characterizes the least fixpoint as a limit of value iteration. This iteration is guaranteed to converge provided the complete partial order has no infinite strictly increasing chain. Consider computing these iterates $(\perp, F(\perp), F(F(\perp)), \dots, F^n(\perp), \dots)$ of \perp which may converge but to a useless one, i.e., \top . To address this convergence issue, Cousot and Cousot [CC77] introduced a widening operator ∇ .

Kleene iteration. An abstract domain is chosen to represent effectively specific sets of program states, such as boxes or convex polyhedra for environments over numeric variables. It is adapted to a class of invariants we want to express (such as variable bounds) while abstracting away irrelevant information to improve efficiency. Then, a computable abstract function $F^{\#}$ modeling the effect of a loop iteration on the abstract world is defined (e.g., a function from boxes to boxes). The inductive invariant is computed as the limit of a Kleene iteration sequence, iterating $F^{\#}$ from an abstract representation I of the set of states before entering the loop of a program: $X^0 = I$, $\forall k. X^{k+1} = X^k \cup^{\#} F^{\#}(X^k)$. In general, this sequence does not terminate. A widening operator is employed to force termination so that, after a finite num-

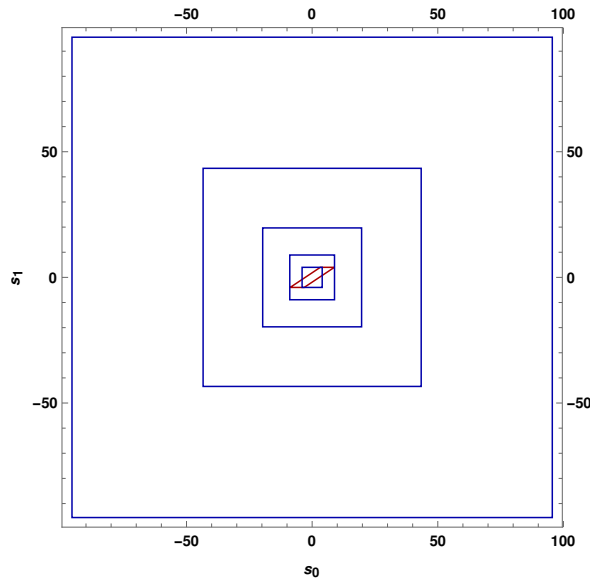


Figure 2.5 – Kleene iterations in the interval domain

ber of iterations, we find an abstract inductive invariant $X^{k+1} \subseteq X^k$. However, the widening operator can lead to loss of precision.

Below, we will consider again the program in Figure 2.1(a) and illustrate the Kleene iterations for inferring inductive invariant.

EXAMPLE 2.18 Consider a box $G = [-4, 4] \times [-4, 4]$. G is a valid invariant for the program in Figure 2.1(a) because G includes notably all the program states reachable at the loop head. On the domain of boxes, the kleene iteration will continue with larger oxes, as shown in Figure 2.5, until it is widened to $[-\infty, +\infty]^2$, which does not imply our invariant G : the method fails as no box is an inductive invariant (the transformation induced by F on G makes its corners overflow G).

Remark 2.19. As of now, we have discussed Kleene iteration based fixpoint computation. Below we will review one of its type, which is rather based on policy iteration⁷ and not value iteration⁸.

Policy or strategy iterations. The inductive invariants (of course not the strongest one) in programs can be expressed as a post-fixpoint. In the static analysis community, Kleene iterations with widening is a well-known approach for computing these fixpoint overapproximations [CC77]. Another approach which solves similar problems, is *policy iteration* [AGG10]. For instance, the policy iterations with ellipsoid abstract domain has been more

⁷In general, policy iteration algorithms start with a random policy, then finds the value function of that policy, and then finds a new or improved policy based on the previous value function.

⁸In value iteration, one starts with a random value function and then improves the value function in an iterative way, until reaching the optimal value function. In value iteration, once the value function reaches its optimal value, the policy out of it is optimal.

effective in inferring quadratic invariants compared to Kleene iterations with ellipsoids [RG14].

The idea is to use appropriate mathematical solvers to solve the fixpoint equation for a given abstract domain using policy iterations instead of optimal value iteration (like Kleene). For instance, if the abstract domain and the fixpoint equation use quadratic equations then semi-definite programming is considered [AGG10]. Similarly, if linear equations are used then one can benefit from linear programming [GGTZ07]. Thus, thanks to these solvers that one can compute the solution without using the convergence techniques.

The policy iterations are otherwise known as *strategy iterations* [GS07a, GS07b, GSA⁺12]. Both the iterations aim at fixpoint computation by a symbolic reasoning based on mathematical solvers like semi-definite programming. However, there is a minor difference in between the two iterations with respect to the policies on which they iterate. For instance, in strategy iterations, one iterates on max-policies, starting from bottom and increasing the bounds until the fixpoint is reached, whereas policy iterations iterate on min-policies, starting from an over-approximation and decreasing the bounds until the fixpoint is attained. Thus, these approaches can be seen as an alternative to Kleene iterations with widening.

There is a recent work [KMW16] which is based on max-policies and formulates the policy iteration as traditional Kleene iteration, with a widening operator.

Both the iterations require templates (appropriate shapes) to be given prior to the analysis. Thus, before using any abstract domain with policy iterations, it must be expressed in terms of template domains. This no doubt makes the method less automatic. Also, the quality of the fixpoint reached by either of the iterations depend on the initial policy used [RJGF12, RG14].

DEFINITION 2.20 (Concretization & Abstraction function.) Consider two posets: (\mathcal{D}, \leq) representing the concrete world and $(\mathcal{D}^\sharp, \sqsubseteq)$, the abstract world. A concretization function $\gamma : \mathcal{D}^\sharp \rightarrow \mathcal{D}$ is a monotonic function which converts each abstract element in \mathcal{D}^\sharp to a concrete one.

Consider a reverse function, called an abstraction function denoted by α which converts from a concrete world back to an abstract one, i.e., $\alpha : \mathcal{D} \rightarrow \mathcal{D}^\sharp$.

DEFINITION 2.21 (Widening.) Widening, on an abstract domain $(\mathcal{D}^\sharp, \sqsubseteq)$ is an operator $\nabla : \mathcal{D}^\sharp \times \mathcal{D}^\sharp \rightarrow \mathcal{D}^\sharp$ such that:

- $\forall x, y \in \mathcal{D}^\sharp : x, y \sqsubseteq (x \nabla y)$, and
- for any sequence $x_i \in \mathcal{D}^\sharp$ where $i \in \mathbb{N}$, the increasing sequence y_i calculated as

$$y_0 := x_0, y_{i+1} := y_i \nabla x_{i+1}$$

stabilizes after a finite number of iterations, i.e., $\exists k \geq 0 : y_{k+1} = y_k$.

DEFINITION 2.22 (Galois connection.) Consider two posets (X, \leq) (concrete) and (Y, \sqsubseteq) (abstract), an abstraction function $\alpha : X \rightarrow Y$, a concretization function $\gamma : Y \rightarrow X$ and $\forall x \in X, y \in Y : \alpha(x) \sqsubseteq y \iff x \leq \gamma(y)$ then the pair $\langle \alpha, \gamma \rangle$ is a Galois connection denoted by:

$$(X, \leq) \xleftrightarrow[\alpha]{\gamma} (Y, \sqsubseteq)$$

Remark 2.23. An essential property established by Galois connection is the strong connection between the concrete and the abstract world. However, not all abstract domains enjoy a Galois connection because they may not have an explicit α which is why the minimum requirement to interact between the two worlds is to at least have a concretization function γ . This is what is called the soundness property through concretization function.

DEFINITION 2.24 (Soundness property.) Consider a concretization function $\gamma: (\mathcal{D}^\sharp, \sqsubseteq) \rightarrow (\mathcal{D}, \leq)$, a concrete transfer function $F: \mathcal{D} \rightarrow \mathcal{D}$, and an abstract transfer function $F^\sharp: \mathcal{D}^\sharp \rightarrow \mathcal{D}^\sharp$. The function F^\sharp is called a sound abstraction of F if $\forall x \in \mathcal{D}^\sharp: F(\gamma(x)) \leq \gamma(F^\sharp(x))$.

2.3 NUMERICAL ABSTRACT DOMAINS

In general, computing in concrete domain $\mathcal{D}: = \mathcal{P}(\mathbb{R}^n)$ can be undecidable because the set of environments may need infinite memory to be represented exactly, and computation of the set of locations infinite time. Therefore, we reason in an approximation (abstract domain) where we forget some of the properties of the concrete semantic domain (or in other words subset of properties of interest) in order to get the domain computable and machine-representable. Let \mathcal{D}^\sharp be the abstract domain such that $\mathcal{D}^\sharp \subseteq \mathcal{P}(\mathbb{R}^n)$. To define an abstract domain \mathcal{D}^\sharp one needs to characterize the following:

- a partial order \sqsubseteq^\sharp on \mathcal{D}^\sharp ,
- a concretization function $\gamma: \mathcal{D}^\sharp \rightarrow \mathcal{D}$,
- a Galois connection, which is optional because there may not exist an abstraction function $\alpha: \mathcal{D} \rightarrow \mathcal{D}^\sharp$ to form a Galois connection $(\mathcal{D}, \leq) \xleftrightarrow[\alpha]{\gamma} (\mathcal{D}^\sharp, \sqsubseteq^\sharp)$. However, what we require at least is the soundness property, i.e., properties of program proved to hold in abstract domain also holds in the concrete one, when we do not have an abstraction function ,
- a join operator \cup^\sharp (an abstraction of set union \cup) and a meet operator \cap^\sharp (an abstraction of set intersection \cap) over the abstract elements,
- a smallest element \perp^\sharp and a largest element \top^\sharp ,
- a widening operator ∇^\sharp .

Finally, we must note that the abstract domain needs only have a poset structure, but not necessarily a CPO nor a lattice.

Many numerical abstract domains are developed because different domains can be used to obtain different properties of programs. The major ones are intervals [CC76] and polyhedra [CH78]. There are many new abstract domains been developed over these years, capturing other properties, such as octagons [Min06], template [SSM05], zonotopes [GP06, GGP09] and ellipsoids [Fer04].

2.3.1 Non-Relational Abstract Domain

The abstract domains of this family are the least expressive. They abstract the set of possible values of each variable independently of the other variables, and hence the name. The well-known in this area is the interval abstract domain. It is based on interval arithmetic, introduced by Moore [Moo69] for numeric analysis, and later adapted to static analysis by Cousot and Cousot [CC76] with the inception of abstract interpretation.

The interval abstract domain, as its name implies, represents each variable as an interval of its possible values, for e.g., $[a, b]$ with $a \leq b$. It represents several variables by using a Cartesian product defined as $\{\prod_{i=1}^n [a_i, b_i] \mid \forall i : a_i, b_i \in \mathbb{R} \cup \{-\infty, \infty\}\}$. Even though it is a non-relational domain, but yet it is simple and inexpensive to implement, and also it can infer useful properties for program verification. However, being an abstract domain with strictly infinite increasing chains, it requires widening to enforce convergence. Cousot *et al.* in [CC76] showed that an interval analysis can over-approximate least fixpoint with widening.

The basic operations available in the interval abstract domain are:

- concretization: $\gamma([a, b]) \stackrel{\text{def}}{=} \{x \in \mathbb{R} \mid a \leq x \leq b\}$
- ordering: $[a, b] \sqsubseteq^\# [c, d] \iff (a \geq c) \wedge (b \leq d)$ (semantically equivalent to set inclusion, i.e., $[a, b] \subseteq [c, d]$)
- join: $[a, b] \cup^\# [c, d] \stackrel{\text{def}}{=} [\min(a, c), \max(b, d)]$
- meet: $[a, b] \cap^\# [c, d] \stackrel{\text{def}}{=} \begin{cases} [\max(a, c), \min(b, d)], & \text{if } \max(a, c) \leq \min(b, d) \\ \perp^\#, & \text{otherwise} \end{cases}$
- addition: $[a, b] +^\# [c, d] \stackrel{\text{def}}{=} [a + c, b + d]$
- subtraction: $[a, b] -^\# [c, d] \stackrel{\text{def}}{=} [a - d, b - c]$
- multiplication: $[a, b] \times^\# [c, d] \stackrel{\text{def}}{=} [\min(ac, ad, bc, bd), \max(ac, ad, bc, bd)]$
- division: $[a, b] /^\# [c, d] \stackrel{\text{def}}{=} \begin{cases} [\min(a/c, a/d), \max(b/c, b/d)], & \text{if } 1 \leq c \\ [\min(b/c, b/d), \max(a/c, a/d)], & \text{if } d \leq -1 \\ (([a, b] /^\# ([c, d] \cap^\# [1, +\infty])) \cup^\# ([a, b] /^\# ([c, d] \cap^\# [-\infty, -1])), & \text{otherwise} \end{cases}$

EXAMPLE 2.25 Consider a program shown in Figure 2.6 having two variables, x and y , and its loop body performs a rotation of the point (x, y) about the origin, with a minor scaling penetrating inward. The initial values of variables (x, y) are in the box $I \stackrel{\text{def}}{=} [-1, 1] \times [-1, 1]$ and the effect of a loop iteration on a set X of possible variable values $(x, y) \in X$ given by the function $F : \mathcal{P}(\mathbb{R}^2) \rightarrow \mathcal{P}(\mathbb{R}^2)$ defined as $F(X) \stackrel{\text{def}}{=} \{(0.7 \times (x - y), 0.7 \times (x + y)) \mid (x, y) \in X\}$.

Provided with an interval $[-2, 2]$, we can define an axis aligned bounding box such as $X = [-2, 2] \times [-2, 2]$ shown in Figure 2.7 in blue. The image of this box by a loop iteration in the interval domain is $F^\#([-2, 2], [-2, 2]) \stackrel{\text{def}}{=} \dots$

```

x=[-1,1];
y=[-1,1];
while (True) {
    xnew=0.7*(x + y);
    ynew=0.7*(x - y);
    x=xnew;
    y=ynew;
}

```

Figure 2.6 – Example program

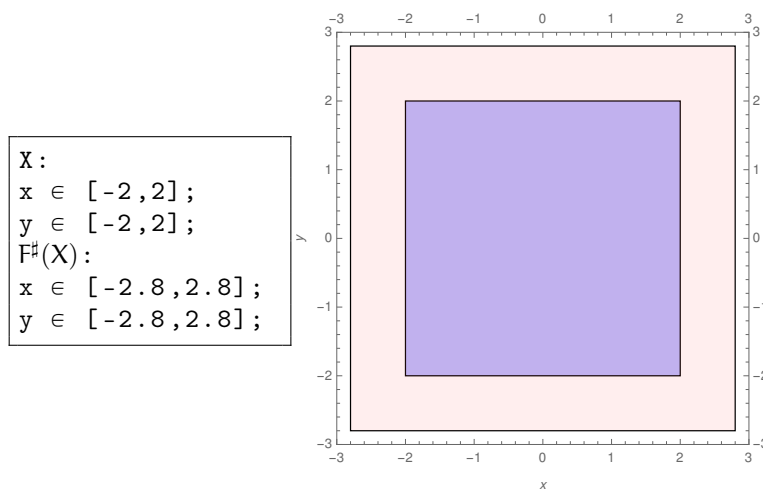


Figure 2.7 – (a) The interval abstract values for the target invariant of the program in Figure 2.7 and its image after one iteration of the body loop; (b) In blue: the target invariant (X), in pink: $F^\#(X)$.

$([0.7(-2-2), 0.7(2+2)], [0.7(-2-2), 0.7(2+2)])$), which is the box in Figure 2.7 in pink.

2.3.2 Relational Abstract Domains

Presumably, very often, for simple programs, the bound information provided by interval abstract domain is sufficient. However, it does not guarantee the tightest possible bounds. This led to the development of relational domains (more expressive) which incorporate the properties inferred by non-relational domains in conjunction with affine relationships among variables of a program. One of the most widespread relational domains is affine inequalities domain or polyhedra.

Polyhedras. The polyhedra abstract domain was introduced by Cousot and Halbwachs [CH78]. As the name suggests, this domain abstracts a set of

points in the form of a convex polyhedron⁹. Recall that the existence of a best abstraction is useful but not necessary. One of the examples is the polyhedra domain which can abstract set of points as an unbounded convex polyhedron.

No Galois connection. There is no Galois connection for a polyhedra because it lacks an abstraction function or has no best abstraction as a polyhedron. The reason for no abstraction function is owing to shapes, such as circles, which do not have a smallest enclosing polyhedron. In other words, we can have an infinity number of tangents to the circle leading to the existence of a polyhedron with infinite number of constraints approximating the circle.

Few important results of polyhedra theory are the Farkas-Lemma and the Weyl-Minkowski Theorem [Wey34, Sch98], which state that polyhedras have dual representations: one using constraints, and one using generators. For a subset \mathcal{P} (polyhedron) of \mathbb{R}^n , the following definitions are equivalent:

DEFINITION 2.26 (H-Representation or exterior representation) A polyhedron is defined as the intersection of finitely many halfspaces, i.e., there exists a matrix A and a vector b with

$$\mathcal{P} = \{x \in \mathbb{R}^n \mid Ax \leq b\} \quad (2.4)$$

where the halfspaces are represented by the inequalities $Ax \leq b$ and n being the number of abstracted numerical variables. In other words, the polyhedron \mathcal{P} is the solution set $x \in \mathbb{R}^n$ to a finite system of linear inequalities.

DEFINITION 2.27 (V-Representation or interior representation) Given a finite a set of extremal points or vertices ($s_i, 1 \leq i \leq k$ or \mathcal{S}) and a finite set of extreme directions or rays ($t_j, 1 \leq j \leq m$ or \mathcal{T}) a polyhedron can be defined as

$$\mathcal{P} = \text{conv}(\mathcal{S}) + \text{cone}(\mathcal{T}) \quad (2.5)$$

where $\text{conv}(Y)$ denotes the convex hull of a set $Y \subseteq \mathbb{R}^n$ defined by

$$\text{conv}(Y) = \left\{ \sum_{y \in Y'} \alpha_y y : Y' \subseteq Y, |Y'| < \infty, \sum_{y \in Y'} \alpha_y = 1, \alpha_y \geq 0 \forall y \in Y' \right\} \quad (2.6)$$

In other words, any point $x \in \mathcal{P}$ can be represented as,

$$x = \sum_{i=1}^k \lambda_i s_i + \sum_{j=1}^m \beta_j t_j \quad (2.7)$$

where $\lambda_i \geq 0, \beta_j \geq 0, \sum_{i=1}^k \lambda_i = 1$.

A bounded polyhedron can be constructed simply by taking convex hull of finite set of vertices. The definition by Equation (2.5) is more general, i.e., by adding rays furthermore, one can obtain an unbounded polyhedron. Switching between the two representations is a well-known result in the polyhedral theory. Changing from H-representation to V-representation is a vertex enumeration problem and the contrariwise is a facet enumeration problem. The best representation varies from one operator to the other. For

⁹Polygon is a two-dimensional polytope. Polyhedra or polyhedron is a three-dimensional polytope

instance, intersection can be modeled by joining constraints, whereas the convex hull can be modeled by joining generators. Hence, it is necessary to have a way to switch between representations. The standard algorithm for the switching between representations was proposed by Chernikova [Che68] and later improved by Le Verge [LV92]. We will be dealing with these kind of problems in the later chapters.

EXAMPLE 2.28 Consider the same example as in Example 2.25. Figure 2.8 illustrates the polyhedral abstraction using constraint and vertex representations. All the constraints of the input and the effect on it after one iteration of the body loop are also shown. Comparing the figures 2.25 and 2.8, it is clear that the polyhedral abstraction is very expressive.

Abstract operators. The inclusion test, i.e., checking if $\gamma(X^\sharp) \subseteq \gamma(Y^\sharp)$, is equivalent to verifying if each generator of X^\sharp satisfies every constraint of Y^\sharp . The intersection of two polyhedra is exactly represented by a polyhedron, by joining the set of constraints, and the join of two polyhedra is their convex hull which is also a polyhedron. Note that, although polyhedras lack an abstraction function α , yet there exists a smallest polyhedron entailing two polyhedras. Thus, the polyhedra abstract domain defines a lattice by using the convex hull as a join and the geometrical inclusion as a partial order. However, the lattice is not complete, as we cannot extract a smallest over-approximation of a circle as a convex polyhedron.

Ellipsoids. In the relational family, there exist domains which are specialized in tracing quadratic invariants in digital filters, which include the ellipsoid abstract domain [Fer04]. An ellipsoid is represented by a pair (P, λ) where $P \in \mathbb{R}^{n \times n}$ is a symmetric positive definite matrix accounting for the shape of the ellipsoid and λ is a scalar hinting to its ratio. Such a pair denotes the following set:

$$\gamma(P, \lambda) \stackrel{\text{def}}{=} \{x \in \mathbb{R}^n \mid x^T P x \leq \lambda\} \quad (2.8)$$

where $\gamma(P, \lambda)$ is the concretization function. The set of ellipsoids provided with the geometric inclusion order \subseteq is not a lattice. Moreover, there is no least upper bound, i.e., there usually does not exist a smallest ellipsoid containing two other ellipsoids.

2.3.3 Weakly-relational abstract domains

The motivation behind the inception of relational abstract domains like polyhedra is the fact that for many programs the interval domain is insufficient to provide precise bounds for the variables. Unfortunately, the polyhedra domain is very expensive: it has worst-case exponential space and time complexity [NQ10, SPV17]. This instigated the introduction of so-called weakly relational abstract domains, which offer a trade-off between expressiveness and cost, and can represent some of the possible expressible properties between variables. They are usually sub-polyhedral domains relying on different algorithmic principles, which include octagons by Miné [Min06], template polyhedra by Sankaranarayanan *et al.* [SSM05], affine forms or zonotopes by Goubault, Putot and Ghorbal [GP06, GGP09, GGP10a], and parallelotopes by Amato and Scozzari [AS12a].

2. ABSTRACT INTERPRETATION

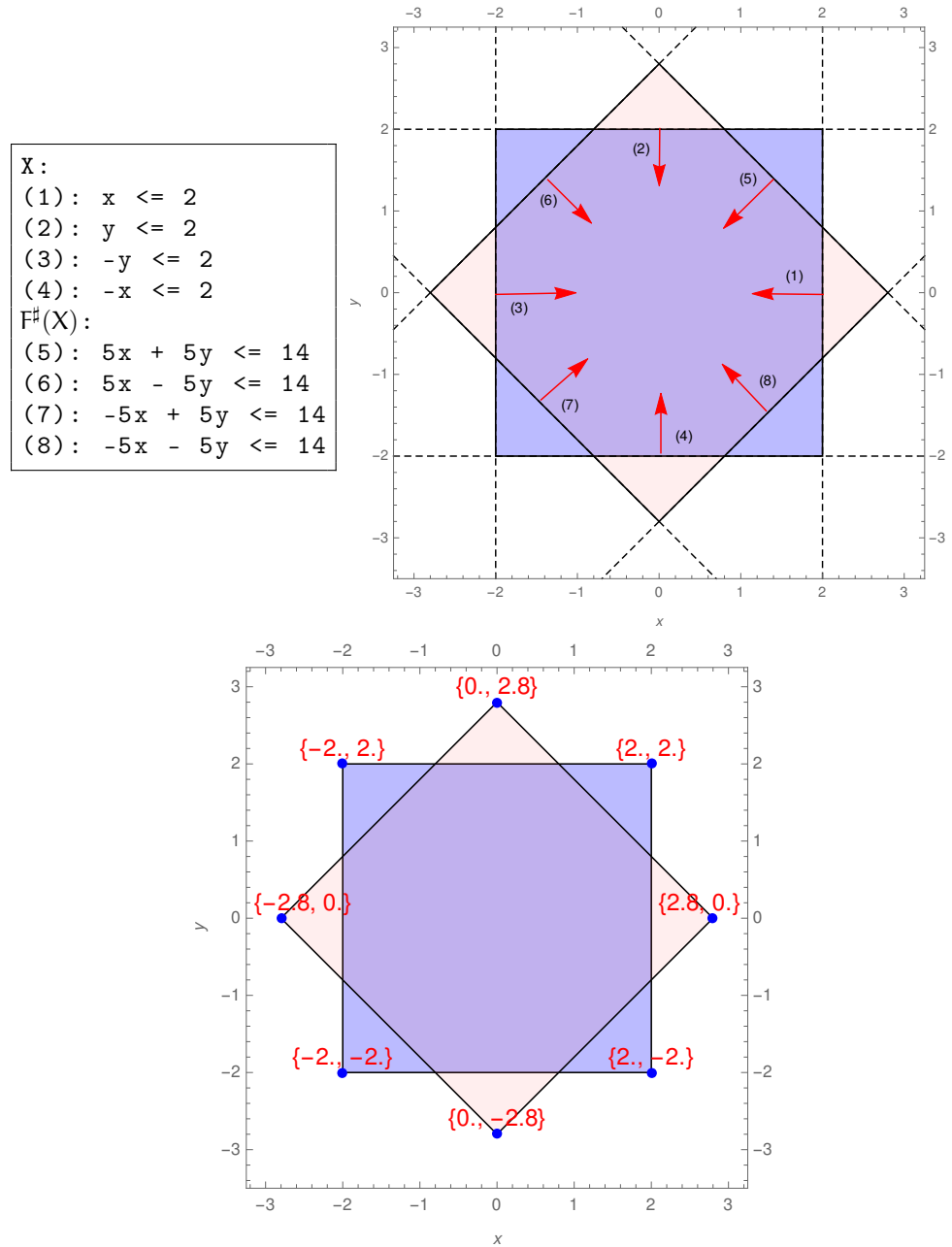


Figure 2.8 – (a) Halfspace representation for the target invariant of the program in Figure 2.7 and its image after one iteration of the body loop using polyhedra abstract domain; (b) H-representation, in blue: the target invariant (X), in pink: $F^\sharp(X)$; (c) Vertex or V-representation, in blue: convex hull of the vertices corresponding to the target invariant (X), in pink: convex hull of the vertices corresponding to the image ($F^\sharp(X)$) of the target invariant.

Octagons. As an extension of the zone abstract domain [Min01], which could only express constraints of the form: $x - y \leq c$, the octagon domain was introduced by Miné [Min06], to precise octagonal constraints¹⁰: $\pm x \pm y \leq c$. With minimal c , the set of octagonal constraints are to be in their tightest form. The name of this domain comes from shape of the concretization, i.e., the set of points that satisfy the congruence of the constraints, which is an octagonal shape with eight corners in 2-dimension. This may not be true in higher dimensions. In dimension p , an octagon has at most $2p^2$ faces.

An equivalent representation for an octagon with a set of octagonal constraints is the difference bound matrix¹¹(DBM). Consider an octagon with the constraints, $\pm v_i \pm v_j \leq c$. There are two variants to which the variable v_i can be mapped, i.e., v'_{2i-1} and v'_{2i} for $+v_i$ and $-v_i$ respectively. Similarly, all the octagonal constraints can be mapped as follows:

- $v_i - v_j \leq c \rightarrow v'_{2i-1} - v'_{2j-1} \leq c$ and $v'_{2j} - v'_{2i} \leq c$
- $v_i + v_j \leq c \rightarrow v'_{2i-1} - v'_{2j} \leq c$ and $v'_{2j-1} - v'_{2i} \leq c$
- $-v_i - v_j \leq c \rightarrow v'_{2i} - v'_{2j-1} \leq c$ and $v'_{2j} - v'_{2i-1} \leq c$
- $-v_i + v_j \leq c \rightarrow v'_{2i} - v'_{2j} \leq c$ and $v'_{2j-1} - v'_{2i-1} \leq c$
- $v_i \leq c \rightarrow v'_{2i-1} - v'_{2i} \leq 2c$
- $-v_i \leq c \rightarrow v'_{2i} - v'_{2i-1} \leq 2c$

Thus, an octagon with n variables can be represented by $2n \times 2n$ size difference bound matrix whose each element corresponding to row i and column j is the constant c from the difference constraint $v'_i - v'_j \leq c$. Consider two octagons represented by their difference bound matrices, U' and V' . The inclusion test, checking if one octagon is contained inside the other, is equivalent to verifying if $U'(i, j) \leq V'(i, j)$. The octagons are closed under intersection (conjunction of constraints maintaining a strict bound on the expression $\pm x \pm y$), but not under union. However, there exists a smallest octagon containing any two given octagons.

EXAMPLE 2.29 Consider once again the same example as in Example 2.25. Figure 2.9 illustrates the octagonal abstraction using constraint representation. All the constraints of the input and the effect on it after one iteration of the body loop are also shown. Even though the octagon abstract domain is more expressive than intervals, the abstract semantics for addition and subtraction is too coarse to obtain an octagonal shape rather than a box.

Let us denote the variables x and y from the example as v_1 and v_2 respectively. Now, applying the above discussed rule for mapping the octagonal constraints into difference constraints, the difference bound matrices obtained for the octagonal constraints in Figure 2.9 are:

¹⁰Constraints with two variables and unit coefficients, i.e., each coefficient can be independently $+1$ or -1 .

¹¹It is based on normalization of the octagonal constraints, by mapping the octagonal constraints to difference constraints.

$$\text{DBM}_X = \begin{pmatrix} v'_1 & v'_2 & v'_3 & v'_4 \\ 0 & 4 & 4 & 4 \\ 4 & 0 & 4 & 4 \\ 4 & 4 & 0 & 4 \\ 4 & 4 & 4 & 0 \end{pmatrix} \begin{matrix} v'_1 \\ v'_2 \\ v'_3 \\ v'_4 \end{matrix} \quad \text{DBM}_{F^\#(X)} = \begin{pmatrix} v'_1 & v'_2 & v'_3 & v'_4 \\ 0 & 5.6 & 5.6 & 5.6 \\ 5.6 & 0 & 5.6 & 5.6 \\ 5.6 & 5.6 & 0 & 5.6 \\ 5.6 & 5.6 & 5.6 & 0 \end{pmatrix} \begin{matrix} v'_1 \\ v'_2 \\ v'_3 \\ v'_4 \end{matrix}$$

Template polyhedra. Like polyhedra domain, the template polyhedra abstract domain introduced by Sankaranarayanan *et al.* [SSM05] deduces affine inequalities of the form $Ax \leq b$ among the program variables. Unlike polyhedra, it is less expensive because the directions of the polyhedra are fixed during the analysis. In other words, the coefficients of the variables in the constraints, are fixed before the analysis is run, and not inferred during the analysis. No doubt, this approach makes it less expressive compared to conventional polyhedra, yet it also reduces the effort of switching between H and V-representations. Thus, the core algorithm for template polyhedras is based on linear programming which improves on the exponential bound bringing the complexity to polynomial order in regard to the program variables. The matrix A is called the template, and hence the name of the domain, template polyhedra.

The interval and octagon domains are special case of template polyhedra domain under specific choices of matrix A or the template. Meaning, in these domains, the matrix A is fixed by the domain and not the user, which is not the case in template polyhedra. Thus, it gives the user more freedom within the template polyhedra domain to freely decide on A before the analysis by analyzing the cost versus precision trade-off.

Affine sets or zonotopes. Recall that interval abstract domain represent program variables with intervals. There exists another type of representation which can express variables in the form of an affine expression or affine combination over a set of noise symbols ε_i

$$\hat{x} = \alpha_0^x + \alpha_1^x \varepsilon_1 + \dots + \alpha_n^x \varepsilon_n, \alpha_i^x \in \mathbb{R} \quad (2.9)$$

and hence the name, affine forms. Transfer function for arithmetic expression on these forms, first introduced by Stolfi and Figueiredo [SDF97, DFS04] rely on affine arithmetic (AA)¹² to handle correlation among variables. $\alpha_0^x \in \mathbb{R}$ is the central value of the affine form. Coefficients $\alpha_i^x \in \mathbb{R}$ are the partial deviations. Given an interval $[a, b]$ representing a variable x , we can build an affine form

$$\hat{x} = \frac{(a + b)}{2} + \frac{(b - a)}{2} \varepsilon_i \quad (2.10)$$

where ε_i is a fresh noise symbol with value in $[-1, 1]$. Dually, given an affine form $\hat{x} = \alpha_0^x + \sum_{i=1}^n \alpha_i^x \varepsilon_i$, the interval concretization is given by

$$x = \alpha_0^x + \left(\sum_{i=1}^n |\alpha_i^x| \right) \times [-1, 1].$$

¹²Affine arithmetic [SDF97] is an extension of interval arithmetic allowing to express dependencies between variables.

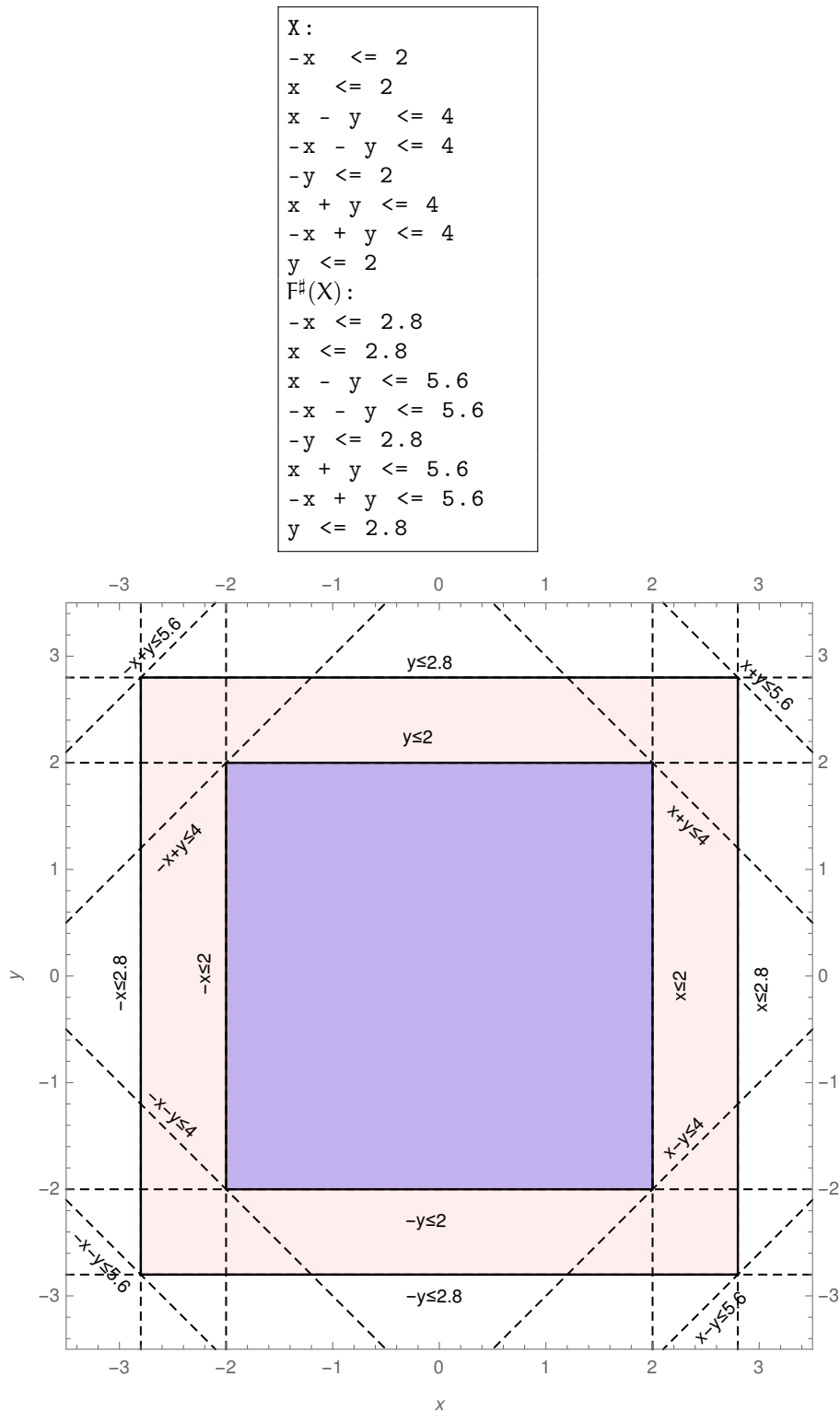


Figure 2.9 – (a) Octagonal constraints for the target invariant of the program in Figure 2.7 and its image after one iteration of the body loop; (b) In blue: target invariant (X) abstracted using octagon abstract domain, in pink: its image $F^{\#}(X)$.

These forms express dependency by sharing the noise symbols ε_i among variables. Meaning, there could be two affine forms representing two different variables, but sharing the same noise symbols, i.e., those affine forms are not independent. The noise symbols can not only model uncertainty due to input data, but also account for uncertainty generated due to computations.

Arithmetic operations over the affine forms. Consider two affine forms \hat{x} and \hat{y} . The linear operations like addition and subtraction operations can be defined as

$$\hat{x} \pm \hat{y} \stackrel{\text{def}}{=} (\alpha_0^x \pm \alpha_0^y) + \sum_{i=1}^n (\alpha_i^x \pm \alpha_i^y) \varepsilon_i.$$

Non-affine operation like multiplication

$$\begin{aligned} \hat{x} \times \hat{y} &= (\alpha_0^x + \sum_{i=1}^n \alpha_i^x \varepsilon_i)(\alpha_0^y + \sum_{j=1}^n \alpha_j^y \varepsilon_j) \\ &= \alpha_0^x \alpha_0^y + \sum_{i=1}^n (\alpha_0^x \alpha_i^y + \alpha_0^y \alpha_i^x) \varepsilon_i + \sum_{i=1}^n \sum_{j=1}^n \alpha_i^x \alpha_j^y \varepsilon_i \varepsilon_j \end{aligned} \quad (2.11)$$

The quadratic term $\alpha_i^x \alpha_j^y \varepsilon_i \varepsilon_j$ in Equation (2.11) can be linearized by adding a new noise symbol ε_{n+1} for $\sum_{i=1}^n |\alpha_i^x| \times \sum_{j=1}^n |\alpha_j^y|$ as shown in the following equation

$$\hat{x} \times \hat{y} = \alpha_0^x \alpha_0^y + \sum_{i=1}^n (\alpha_0^x \alpha_i^y + \alpha_0^y \alpha_i^x) \varepsilon_i + \left(\sum_{i=1}^n |\alpha_i^x| \times \sum_{j=1}^n |\alpha_j^y| \right) \varepsilon_{n+1} \quad (2.12)$$

There are several different ways described in [Gho11] to linearize the non-linear term.

Affine forms to zonotopes. Recall that one of the significant features of the affine forms is their ability to infer the dependency (partial) among the variables they represent by sharing the noise symbols. This dependency can be determined by the corresponding partial deviations. Consider two affine forms \hat{x} and \hat{y} for two variables x and y , given by their affine combinations as

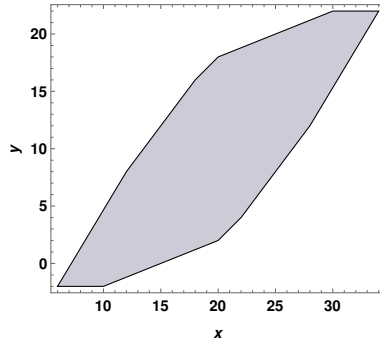
$$\begin{aligned} \hat{x} &= \alpha_0^x + \alpha_1^x \varepsilon_1 + \dots + \alpha_n^x \varepsilon_n, \\ \hat{y} &= \alpha_0^y + \alpha_1^y \varepsilon_1 + \dots + \alpha_n^y \varepsilon_n. \end{aligned}$$

No doubt, the above affine forms imply interval bounds, i.e., $x = [\alpha_0^x - d_x, \alpha_0^x + d_x]$, $y = [\alpha_0^y - d_y, \alpha_0^y + d_y]$ where d_x and d_y are total deviations of \hat{x} and \hat{y} respectively and d_x is given by

$$d_x = |\alpha_0^x| + \dots + |\alpha_n^x|.$$

DEFINITION 2.30 (Geometric concretization.) Consider a tuple of affine forms for p variables over n noise symbols ε_j , $\hat{x}_i = \alpha_0^{x_i} + \sum_{k=1}^n \alpha_k^{x_i} \varepsilon_k$.

$$\hat{X} = \begin{cases} \hat{x}_1 = \alpha_0^{x_1} + \sum_{k=1}^n \alpha_k^{x_1} \varepsilon_k \\ \hat{x}_2 = \alpha_0^{x_2} + \sum_{k=1}^n \alpha_k^{x_2} \varepsilon_k, & (\varepsilon_1, \dots, \varepsilon_n) \in [-1, 1]^n \\ \hat{x}_3 = \dots \end{cases}$$


 Figure 2.10 – Zonotope concretization $\gamma(A)$

This defines a matrix $A \in M(n+1, p)$ whose (j, i) entry, for $i = 1, \dots, p$, $j = 0, \dots, n$ is $A_{j,i} = \alpha_j^{x_i}$.

$$\hat{X} = \underbrace{\begin{pmatrix} \alpha_0^{x_1} & \dots & \alpha_n^{x_1} \\ \vdots & & \vdots \\ \alpha_0^{x_p} & \dots & \alpha_n^{x_p} \end{pmatrix}}_{A^T} \times \underbrace{\begin{pmatrix} \varepsilon_0 \\ \varepsilon_1 \\ \vdots \\ \varepsilon_n \end{pmatrix}}_{\varepsilon}, \varepsilon \in \mathbf{1} \times [-1, 1]^n$$

The partial dependency underlying the shared noise symbols signify that the joint range is the set or the geometric concretization given by

$$\gamma(A) = \left\{ A^T \varepsilon \mid \varepsilon \in \{\mathbf{1}\} \times [-1, 1]^n \right\} \subseteq \mathbb{R}^p, A \in M(n+1, p).$$

This set $\gamma(A)$ is the image of a hypercube¹³ $[-1, 1]^n$ under the affine transformation $\mathbb{R}^n \rightarrow \mathbb{R}^p$. $\gamma(A)$ is a class of polytope which exhibit special symmetries, i.e., a center-symmetric polytope (a bounded polyhedron) with center-symmetric faces. This class of polytopes are known as zonotopes. Principally, the joint range of p affine forms is a zonotope in \mathbb{R}^p whose center is the vector given by the first column of A^T . The other vectors of A^T are the generators of the zonotope whose Minkowski sum of the line segments described by those vectors is the zonotope itself.

EXAMPLE 2.31 Consider the affine forms $X = (\hat{x}, \hat{y})$ with

$$\hat{x} = 20 - 3\varepsilon_1 + 5\varepsilon_2 + 2\varepsilon_3 + 1\varepsilon_4 + 3\varepsilon_5 \quad (2.13)$$

$$\hat{y} = 10 - 4\varepsilon_1 + 2\varepsilon_2 + 1\varepsilon_4 + 5\varepsilon_5 \quad (2.14)$$

For $n = 5$ and $p = 2$, the zonotope in Figure 2.10 is the concretization of the affine forms $X = (\hat{x}, \hat{y})$, that is, $A^T = \begin{pmatrix} 20 & -3 & 5 & 2 & 1 & 3 \\ 10 & 4 & 2 & 0 & 1 & 5 \end{pmatrix}$. In

¹³Hypercube is an n -dimensional correspondent of a square in 2-dimension or a cube in 3-dimension.

what follows, we note $A_+ \in \mathcal{M}(n, p)$ the submatrix of $A \in \mathcal{M}(n + 1, p)$, without its first column, that corresponds to the center of the zonotope. Each column of A_+ defines a generator of the zonotope, and we also represent zonotope A as (c, g_1, \dots, g_n) , i.e., as its center c and its collection of generators g_1, \dots, g_n . For the example above, we would write $A = ((20, 10), (-3, 4), (5, 2), (2, 0), (1, 1), (3, 5))$.

Consider the matrix A_+ associated to affine set (\hat{x}, \hat{y}) without its center. For $A \in \mathcal{M}(n + 1, p)$, $A_+ \in \mathcal{M}(n, p)$ is the submatrix A where we have been removing the first column. Its affine concretization is the same zonotope as $\gamma(A)$ but centered on 0, which we denote by $\gamma_{\text{lin}}(A)$ (linear concretization), as defined below :

DEFINITION 2.32 (Linear concretization.) Consider p affine forms (p variables with n noise symbols) given by

$$\hat{X} = \underbrace{\begin{pmatrix} \alpha_1^{x_1} & \cdots & \alpha_n^{x_1} \\ \vdots & & \vdots \\ \alpha_1^{x_p} & \cdots & \alpha_n^{x_p} \end{pmatrix}}_{A^T} \times \underbrace{\begin{pmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_n \end{pmatrix}}_{\varepsilon}, \varepsilon \in [-1, 1]^n.$$

Its geometric concretization is the zonotope defined as

$$\gamma_{\text{lin}}(A) \stackrel{\text{def}}{=} \left\{ A^T \varepsilon \mid \varepsilon \in [-1, 1]^n \right\} \subseteq \mathbb{R}^p, A \in \mathcal{M}(n, p).$$

which is centered on 0.

Remark 2.33. Thus, so far, we have seen three equivalent characterizations for a zonotope. A zonotope in \mathbb{R}^p is a polytope which can be represented as a Minkowski sum of finitely many line segments. This leads to another equivalent definition, i.e., a zonotope is a polytope in \mathbb{R}^p whose k -dimensional faces are centrally symmetric, $\forall 1 \leq k \leq p$.

A third equivalent characterization for a zonotope is that it is the projection of a n -dimensional cube or n -cube (also known as n -dimensional hypercube), for some n by an affine map from \mathbb{R}^n to \mathbb{R}^p denoted by a real matrix $G = \{g_1, g_2, \dots, g_n\}$ of size $p \times n$ and the columns g_1, g_2, \dots, g_n are called the generators of the zonotope. The rank of the matrix G is the rank of the zonotope defined by this G .

A rank k sub-zonotope formed from k generators (independent columns) is a special kind of zonotope also known as parallelotope. We will detail them in the later chapters.

Dual representation of a zonotope using support functions. Zonotopes, being convex shaped objects, can be dually represented as intersection of half-spaces, also known as support functions.

DEFINITION 2.34 (Support function.) We know that a hyperplane can be characterized by a direction $u \in \mathbb{R}^p$ and a scalar $b \in \mathbb{R}$. We denote one such hyperplane $\mathcal{H} = (u, b)$. The set of points $y \in \mathbb{R}^p$ lying on this hyperplane can be given by

$$\langle y, u \rangle = b$$

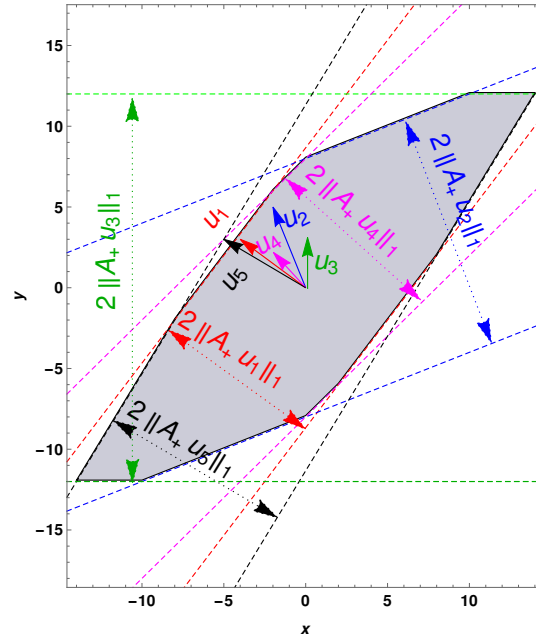


Figure 2.11 – Linear concretization $\gamma_{\text{lin}}(A_+)$ of affine set (\hat{x}, \hat{y}) without its center

where $\langle \cdot, \cdot \rangle$ is the inner product. For the set of points y lying on one side of the hyperplane, we can write $\langle y, u \rangle \leq b$. Thus, for the direction u , finding $b(u)$ such that a convex set \mathfrak{Z} lies on one side of the hyperplane $(u, b(u))$ or in other words, such that for all $y \in \mathfrak{Z}$, $\langle y, u \rangle \leq b(u)$, corresponds to considering the support function formally defined as

$$\sup_{y \in \mathfrak{Z}} \langle y, u \rangle.$$

Remark 2.35. Thus, from the above definition, it is implicit that a zonotope can be understood as an intersection of all the half-spaces. The authors in [GP15] (Lemma 2) have proved that given a matrix $A \in M(n, p)$ (a zonotope centered on zero defined by n affine forms in p dimension) $\forall u \in \mathbb{R}^p$, the support function, $\sup_{y \in \gamma_{\text{lin}}(A)} \langle y, u \rangle = \|Au\|_1$. Below we consider the zonotope in Figure 2.10 and illustrate its characterization by support functions.

EXAMPLE 2.36 For $l \in \mathbb{R}$, $u \in \mathbb{R}^p$, the (l, u) -level set corresponds to points on the hyperplane defined by: for $x \in \mathbb{R}^p$, $p_u(x) = \langle u, x \rangle = l$. This hyperplane is orthogonal to the line L_u with direction u . Given u a direction in \mathbb{R}^2 , the (l, u) -level set that intersects $\gamma_{\text{lin}}(A_+)$ with maximal value for l realizes $l = \sup_{y \in \gamma_{\text{lin}}(A_+)} p_u(y) = \|A_+u\|_1$ by Lemma 2 of [GP15]. Now, we take five vectors u which are normal to the generators of $\gamma(A)$ or face of the zonotope shown in Figure 2.11. It is quite apparent from the figure that $\gamma_{\text{lin}}(A_+) \subseteq H_{u_i}$ where H_{u_i} is the region between the two dashed lines orthogonal to u_i . More generally, given any matrix A , $\gamma_{\text{lin}}(A)$ can be completely characterized by providing the set of values $\|Au\|_1$.

Testing inclusion. By the proof of Lemma 3 in [GP15], we can say that given two matrices $X \in M(n_x, p)$ and $Y \in M(n_y, p)$, $\gamma_{\text{lin}}(X) \subseteq \gamma_{\text{lin}}(Y)$, iff $\|Xu\|_1 \leq \|Yu\|_1$ for all $u \in \mathbb{R}^p$. As a generalization of the Lemma 3 to zonotopes which are no longer centered on zero, the authors in [GP15] also proved that given two matrices $X \in M(n_x + 1, p)$ and $Y \in M(n_y + 1, p)$, $\gamma(X) \subseteq \gamma(Y)$, iff $\forall u \in \mathbb{R}^p$

$$\left| \langle u_i, \alpha_0^{x_i} - \alpha_0^{y_i} \rangle \right| \leq \|Y_+ u_i\|_1 - \|X_+ u_i\|_1, i = 1, \dots, p \quad (2.15)$$

where $\alpha_0^{x_i}, \alpha_0^{y_i}$ for $i = 1, \dots, p$ are the centers of the zonotopes $\gamma(X), \gamma(Y)$ respectively.

Remark 2.37. We will prove in a later chapter that in order to test if $\gamma(X) \subseteq \gamma(Y)$, it is sufficient to verify the inequality $\left| \langle u_i, \alpha_0^{x_i} - \alpha_0^{y_i} \rangle \right| \leq \|Y_+ u_i\|_1 - \|X_+ u_i\|_1, i = 1, \dots, p$ for a fixed number of directions and not for all $u \in \mathbb{R}^p$.

No Galois connection. Recall that for the polyhedra abstract domain, there does not exist a Galois connection owing to the fact that there is no abstraction function α , e.g., no best outer-approximation of a circle. Likewise, zonotopes which are some special class of polytopes, do not offer a smallest, over-approximation of a set of points in \mathbb{R}^n . Thus, the abstract interpretation one can define with zonotopes is concretization based.

Ordering. If one is only interested in using affine forms to abstract current value of the variables in a program, then one can consider, on the affine forms a (partial) order relation which is the subset inclusion (\subseteq) of their geometric concretizations, i.e., the affine set X is less or equal than Y , iff $\gamma(X) \subseteq \gamma(Y)$ (geometric ordering on zonotopes), where $\gamma(X) \subseteq \gamma(Y)$ iff the inequality in Equation (2.15) is satisfied for all $u \in \mathbb{R}^p$. There also exists a stronger order on affine sets, in a sense that it tracks input/output relations, i.e., functional ordering on zonotopes, which expresses the inclusion order on the geometric concretization of the affine set augmented by the inputs of the program [GP09, GPV12]. In this thesis, we will be using the geometric ordering on zonotopes and we explain below why.

Remark 2.38. Recall the Figures 1.3a-1.3g from Chapter 1, where the final inductive invariant clearly illustrates that one has to rely on a framework not only inspired by abstraction but also splitting and discarding abstract elements. A natural way of splitting a zonotope is to use a split operation on the n -dimensional unit cube of which the zonotope is the projection of, where n being the number of noise symbols. Consider a set of affine forms with p variables defined over n noise symbols, whose joint range is a zonotope. Splitting this zonotope is equivalent to bisecting into two the coefficient of the j^{th} noise symbol for all p variables [ASB08, WVL09]. However, we introduce in this work a new splitting operator which can split a zonotope concretization defined by a set of generators into a set of parallelotopes (subsequence of the generators) where each adjacent parallelotope share utmost one noise symbol. This splitting operator relies on geometrical and combinatorial properties of a given a set of generators. Thus, for simplicity reasons, we stick to the geometric ordering of zonotopes in the current work, which is still sound.

Remark 2.39. In this thesis work, because of the splitting in particular, we use a purely geometrical interpretation of zonotopes, and not the functional

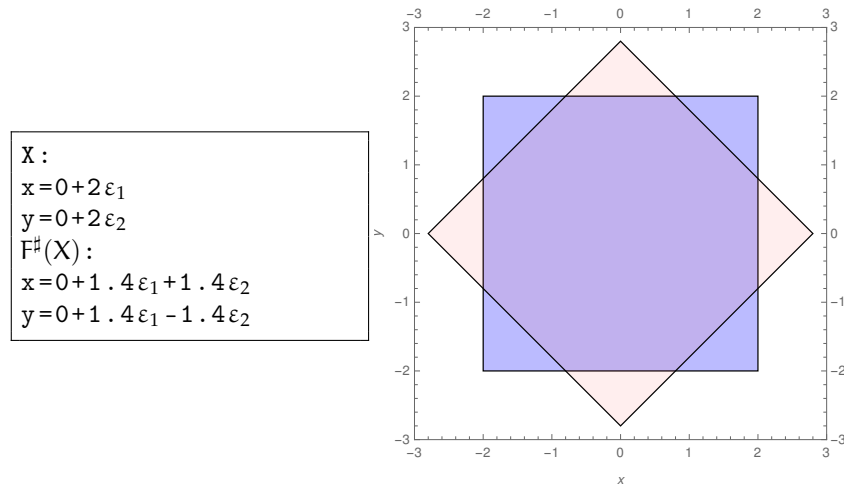


Figure 2.12 – (a) Affine forms corresponding to the target invariant of the program in Figure 2.7 and its image after one iteration of the body loop; (b) In blue: target invariant (X) abstracted using zonotopic abstract domain, in pink: its image $F^\#(X)$.

interpretation. If we had used the functional interpretation then it would refer to the inclusion test of Definition 3 in [GP15]. It is not quite sure whether the simpler test for inclusion based on Lemma 3 in [GP15] which we introduce in Chapter 5 would still be applicable for the functional inclusion test. Moreover, that is the same reason that the set operations like meet defined for functional abstraction in [GGP10b] are not very well suited to the geometrical interpretation, and which is why we will define our own operations in Chapter 5.

EXAMPLE 2.40 Figure 2.12 shows the zonotopic abstraction of the input ($[-2, 2]$) to the program in Example 2.25 and its image after one iteration of the body loop. The shared noise symbols between the affine forms associated with the image ($F^\#$) results in a concretization more precise than the octagons (Figure 2.9).

Remark 2.41. In terms of expressiveness, the zonotope abstract domain fills the gap between weakly relational domains and fully linear relational domain like the linear templates polyhedra domain. They definitely present a good trade-off between cost and precision when it is used to analyze program with non-linear operations. Compared to other restrictions of polyhedra, such as octagons or templates, the directions of zonotopic faces are not fixed *a priori*, providing some flexibility in the analysis. Zonotopes are closed under Minkowski sum and difference. Non-affine arithmetic operations, including float rounding, can easily be interpreted in zonotopes by computing an approximate resulting affine form, and adding a new noise symbol to account for the approximation error. Zonotopes have already proved to be a simple and tractable set representation for program analysis [GP15, GGP09].

Parallelootope abstract domains. Recall that given a set of p independent vectors (generators) in \mathbb{R}^p whose Minkowski sum defines a zonotope, can be viewed

as a skewed, stretched or shrunken image of a cube, or otherwise known as parallelotope. They can also be formally represented as a polyhedron defined by at most n linearly independent constraints, where n being the number of variables [AS12a]. In other words, parallelotope is a polyhedron whose constraint matrix is square and invertible.

Also, recall that there exists a weaker relational variant of polyhedra known as template polyhedra, where for each program the user fixes the constraint matrix a priori unlike intervals or octagons (the constraint matrix is fixed for all programs). Doubtlessly, this domain can handle expressive constraints but the abstract operators are mainly based on linear programming and can be computed in polynomial time. In order to have a better trade-off between expressiveness and cost, Amato *et al.* [AS12a] introduced the parallelotope domain where the number of constraints are limited to n linearly independent ones, rather than restricting the syntactic form of constraints (as in template polyhedra). This helped in designing efficient abstract operators without relying on linear programming tools.

2.3.4 Combining abstract domains

Heretofore, we have seen that abstract interpretation framework uses several different abstract domains. After all, a complex abstract domain can be disintegrated into a combination of simpler ones. In other words, it is also possible to build new abstract domains from the existing ones by generic operators like: reduced product [CC79, CCM11], partitioning [Cou81] and logical product [GT06, GGP10a], which leads to the design of a more expressive domain.

Reduced product combines two or more existing abstract domains, useful to express conjunction of heterogenous properties. Bertrane *et al.* in [BCC⁺] and Cousot *et al.* in [CCF⁺06] describe a reduced product of a large set of abstract domains, including the interval, octagon and polyhedra domains. Also, Amato *et al.* in [Rub18] described a reduced product of parallelotopes [AS12a] and intervals.

In partitioning, instead of a single abstract domain element, one combines several elements of the same abstract domain in order to express disjunctive properties.

Remark 2.42. In this thesis, we will define some sort of partitioning, which is disjunction of abstract elements instead of classical disjunctive-domain techniques, such as disjunctive completion, or trace-partitioning.

A logical product is a method to combine abstract domains based on decision procedures (Nelson-Oppen methodology [NO79]) and not just based on their concretizations, which is the case in reduced product. It is a more expressive (facilitates in better exchange of relations in between the involved domains) combination compared to reduced product [GT06]. For instance, Ghorbal *et al.* [GGP10a] designed a meet operation using a logical product of the affine sets with an abstract domain (preferentially intervals, zones or octagons so that the abstraction eventually is not expensive) over the noise symbols. The constraints produced by the tests in a program are interpreted over the noise symbols of the affine forms, and hence the name, constrained affine sets.

2.3.5 Support libraries

There are several libraries being developed for the abstract domains and their operators. Among the publicly available ones are Apron¹⁴ [JM09], ELINA¹⁵ [SPV17], the Parma Polyhedra library¹⁶ [BHZ08] and the Polka library¹⁷.

Apron is a C library covering a wide set of numerical abstract domains, namely boxes, octagons, polyhedra, zonotopes (named after Taylor1+) and also reduced products. The numerical abstractions provided by the library can be used for static analysis in C, C++, OCaml and Java. It includes the implementation of different abstract operators for e.g., meet, join, test for inclusion and equality, verify if an abstract element is \perp^\sharp or \top^\sharp . It allows to switch from one domain to another. The default constructor type argument for any apron abstract domain is boxes. The constrained affine sets and their abstract operators have been implemented in Apron [GGP09]. All the experiments associated to the current work have been carried out with the help of Apron library.

Recall that the polyhedra abstract domain has dual representations: one using constraints, and one using generators. The standard algorithm for switching between two representations was provided by Chernikova [Che68] and later essentially enhanced by LeVerge [LV92]. This algorithm is implemented in the Parma Polyhedra Library and the Polka Library. The polyhedra abstract domain in APRON Library are based on these two implementations. There are other polyhedral libraries namely Komei Fukuda's cddlib¹⁸, PolyLib¹⁹ and ELINA. ELINA also covers few other domains like zones and octagons. It includes an efficient implementation of polyhedra domain based on decomposition of a large polyhedron into a set of smaller polyhedras, thus reducing the asymptotic complexity of the domain, without losing precision [SPV17].

2.3.6 Abstract interpretation tools

An important application of abstract interpretation is to provide complex information (what are the reachable states?) about the behavior of a program without requiring the users to provide specification or execute their programs, in short known as static analysis. Even though the program being analyzed does not terminate, its static analysis does.

There exist several abstract interpretation-based static analyzers among which few of them are Astrée²⁰ [CCF⁺05, BCC⁺, Mau04], Polyspace²¹, Fluctuat²² [BCC⁺09, DGP⁺09, GPBG07].

Astrée is a static program analyzer used for proving the absence of runtime errors in C programs. It has been successfully used to analyze the flight

¹⁴<http://apron.cri.enscm.fr/library/>

¹⁵ELINA: ETH Library for Numerical Analysis. <http://elina.ethz.ch>

¹⁶PPL Project. The Parma Polyhedra Library. <http://www.cs.unipr.it/ppl/>

¹⁷<http://www.inrialpes.fr/pop-art/people/bjeannet/newpolka>

¹⁸https://www.inf.ethz.ch/personal/fukudak/cdd_home/index.html

¹⁹<http://icps.u-strasbg.fr/polylib/>

²⁰<https://www.absint.com/astree/index.htm>

²¹<http://www.mathworks.fr/products/polyspace>

²²Static analysis for numerical precision. <http://www-list.cea.fr/labos/gb/LSL/fluctuat/index.html>

control code of AIRBUS A340, A380 [SD07, DS07], and the source code of the Monitoring and Safing Unit (MSU) of Astrium's Automated Transfer Vehicle (ATV) .

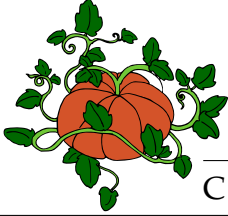
The Polyspace verifier tool of Mathworks was used to analyze the flight control code for the Ariane 502 rocket [LMR⁺98].

Fluctuat is a static analyzer developed at the Laboratory for the Modelling and Analysis of Interacting Systems (LMeASI) at CEA LIST. It is used to study the propagation of uncertainties in C programs due to for instance finite precision (floating-point or fixed-point) computations. It relies on abstract domains like intervals and affine arithmetic (with zonotopic concretization). It also addresses other issues like division by zero, overflows and also checking complex functional specifications. Fluctuat was involved in analyzing the source code of ATV [BCC⁺09]. For instance, it provided the output ranges of some critical functions of the MSU of the ATV space vehicle. It also analyzed the numerical stability of an 8-th order filter (used in the software of MSU) by deriving its tight invariant. It has also been used by nuclear plants and AIRBUS for the A350.

These analyzers have had different industrial applications. However, there are few of them with more educational and scientific motivation. Interproc is an analyzer based on Apron abstract domain library. Frama-C²³ is an open source C program analyzer [KKP⁺15]. Random²⁴ is a R-based analyzer for imperative programs [AS12b]. It covers numerical abstract domains, namely intervals, parallelotopes, reduced product of intervals and parallelotopes.

²³<https://frama-c.com/>

²⁴<https://www.sci.unich.it/~amato/random/>



Besides abstract interpretation, there is another approach called *constraint programming* (CP) which has also been used to discover invariants. Here we discuss the key facts about constraint programming.

3.1 FROM AI TO CP

Up to now, we have witnessed a large number of representations or abstract domains with different shapes (box, ellipse, polyhedra and etc.) provided by abstract interpretation.

Recall that for the example program in Figure 2.6, the analysis with boxes will fail to find an inductive invariant and it is because of the same reason as discussed in Example 2.18. Even if the inductive invariant is octagonal, the abstraction of the body loop in the octagon domain is not precise enough to show directly that it is inductive. Moreover, no box or octagon was stable for such a program, but rather there is a stable polyhedric domain (with octagon shape). In other words, the method would however succeed using the, more costly, polyhedra domain. One drawback is that there is no way, by only looking at the shape of the target invariant G , to guess that a more complex domain is required to infer an adequate inductive invariant. Classic abstract interpretation infers an abstract inductive invariant, i.e., an element or a shape G^\sharp that is stable by a loop iteration: $F^\sharp(G^\sharp) \subseteq G^\sharp$. As this implies by soundness that $F(\gamma(G^\sharp)) \subseteq \gamma(G^\sharp)$ while only requiring a computable abstract version F^\sharp of F .

After having not much luck with boxes, the standard solution inspired by abstract interpretation for the program in Figure 2.6 would be to design a new abstract domain (more expressive) representing directly the octagon shape that is the final inductive invariant in Figure 2.2. Moreover, it must be redesigned for every different shape and there is no standardized way to construct such a domain. Instead, we have to reason in a framework based on *disjunctive completion*, $\mathcal{P}(\mathcal{D}^\sharp)$ rather than \mathcal{D}^\sharp : that will search for an inductive invariant that can be expressed as a finite collection $\mathcal{S} \subseteq \mathcal{D}^\sharp$ of abstract elements, i.e., $\mathcal{S} = \{S_1, \dots, S_n\}$. For example, one can still use boxes, and search for a disjunction of boxes which is inductive, rather than looking for an inductive invariant in a single box. One such framework exists and it is called *continuous constraint programming* (CP) inspired by splitting and tightening abstract elements.

First, we shall recall about constraint programming in general, and then go into details of continuous CP.

3.2 CONSTRAINT PROGRAMMING

Constraint programming is a method for solving combinatorial problems, by expressing them as conjunctions of first-order logic formulas, called constraints and solving them by off-the-shelf constraint solvers [Mon74]. It has been used in a wide spectrum of applications:

- Job-shop scheduling [LP94] and manufacturing scheduling problems [Bap96]
- Vehicle routing problems [Sha98]
- Handling ribonucleic acid (RNA) secondary structures [BKV96]
- Automatic composition of music [HLZ96], constraint programming system for contemporary music [TAC01]
- Cryptoanalysis against block ciphers [GMS16]
- Urban planning [BCT14, BCT12]
- Finding invariants for program verification [GSV08]

Constraint programming is a programming paradigm which a computer uses to solve a user defined problem, i.e., an aggregate of constraints. The formal manner to state such problems to the computer is known as *constraint satisfaction problem* (CSP) where the variables of the problem can be integer or real number.

DEFINITION 3.1 (Constraint satisfaction problem.) A constraint satisfaction problem (CSP) is defined by a set of variables $\mathcal{V} \stackrel{\text{def}}{=} \{v_1, \dots, v_n\}$ taking values from a set of initial domains $\{D_1, \dots, D_n\}$ where $\forall i : D_i \in \mathbb{R}$, D_i is bounded in \mathbb{R} or $\forall i : D_i \in \mathbb{Z}$, D_i is bounded in \mathbb{Z} , and a set of relations or otherwise known as constraints $\mathcal{C} \stackrel{\text{def}}{=} \{C_1, \dots, C_p\}$ on a subset of variables.

Each variable v_i can take values from the domain D_i . A constraint C_i defines a relation $R_{C_i} \subset D_1 \times \dots \times D_n$ where $\mathcal{D} \stackrel{\text{def}}{=} D_1 \times \dots \times D_n$ is the set of all possible values that can be assigned to the variables, also known as search space. The constraint C_i is satisfied if $\{v_1, \dots, v_n\} \in R_{C_i}$. Constraints in CP primarily operate on domains that are either discrete ($\mathcal{D} \subseteq \mathbb{Z}^n$) or continuous ($\mathcal{D} \subseteq \mathbb{R}^n$) where n is the number of variables.

Remark 3.2. Consider the problems are continuous and the domain (D_i) is an interval. Then, each variable $v_i \in V$ can take value in the interval D_i and \mathcal{D} is the Cartesian product $D_1 \times \dots \times D_n$ which is a box. In computers, we cannot represent real numbers. Therefore, the bounds of an interval D_i are defined as floating-point numbers. We will denote the lower and upper bounds as \underline{D}_i and \overline{D}_i respectively.

The solution to a CSP is the set of all points in domain \mathcal{D} satisfying all constraints \mathcal{C} . In other words, a complete assignment of values to variables satisfying all the constraints. In general, the solution set cannot be enumerated exactly. Thus, in continuous constraint programming, we compute a collection

of abstract elements, for instance, boxes with floating-point bounds, that will contain all solutions and tightly fit the solution set.

The class of CP relevant to this thesis is the continuous constraint solving [CDR98, PMTB13], over real-valued variables, that works by refining the domain of the variables, i.e., a box representing candidate solutions: the box is tightened (or reduced) as much as possible by removing variable values that cannot participate in a solution. A continuous constraint solver alters two kinds of steps: *propagation* and *splitting* [Pel15].

3.2.1 Propagation

Consider a CSP which is a set $\mathcal{C} \stackrel{\text{def}}{=} \{C_1, \dots, C_p\}$ of constraints on a set of variables $\{v_1, \dots, v_n\}$ where each variable can have values from the domain $\{D_1, \dots, D_n\}$. Declaratively, a constraint defines a relation on the Cartesian product $D_1 \times \dots \times D_n$ of the corresponding domains. In general, it is computationally expensive to compute all the values that satisfy the constraint. Thus, typically a constraint programming system filters or reduces the domain by removing domain values which do not satisfy the constraints or cannot be a part of the solution. These values are otherwise known as inconsistent. There are several notions of consistencies proposed by the constraint programming community, for example the hull consistency, box consistency (generalized arc consistency for discrete constraints [Mac77a, Mac77b]) for continuous variable domains [SHF96, Ilo99].

DEFINITION 3.3 (Generalized Arc-consistency.) Consider a CSP problem defined over the set $\{v_1, \dots, v_n\}$ of variables taking values from discrete domains $D_1 \dots D_n$, $D_i \subseteq \hat{D}_i$ and a constraint C . Then the domains $D_1 \dots D_n$ are said to be generalized arc-consistent, if and only if $\forall i \in \llbracket 1, n \rrbracket, \forall x \in D_i, x$ has a support ($x_i \in D_i$ has a support if and only if $\forall j \in \llbracket 1, n \rrbracket, j \neq i, \exists x_j \in D_j$ such that $C(x_1, \dots, x_n)$ is true).

DEFINITION 3.4 (Hull-consistency.) Consider a CSP problem defined over the set $\{v_1, \dots, v_n\}$ of variables taking values from continuous domain characterized by intervals $D_1 \dots D_n$ where $\forall i : D_i \in \mathbb{I}$ and a constraint C . Then the domains $D_1 \dots D_n$ are said to be hull consistent, if and only if the smallest box represented by $D_1 \times \dots \times D_n$ contains the solution for C .

Remark 3.5. Hull consistency is a relaxation of the arc-consistency which only requires to check the arc-consistency property for each bound of the intervals [CDR99].

The propagation algorithm used to enforce hull-consistency is known as **HC4** [Ilo99]. It takes as input a set of constraints and a box, and reduces the domains of the variables as much as possible.

Generally speaking, a constraint programming system can have many constraints. While one achieves consistency for one constraint by removing the inconsistent values, other constraints, which were consistent earlier may go inconsistent. Therefore, it is necessary to keep tightening the domain repeatedly until it reaches the *consistency* (no further domain reduction is possible), or in other words, cannot find anymore inconsistent (based on the constraints) values. This process is called *propagation*.

3.2.2 Splitting

Usually, the propagation step is not sufficient to find the solutions. Therefore, we need to cut the remaining search space into smaller domains. Whenever a domain cannot be reduced anymore, it is split into two or more domains, that are tightened and split themselves iteratively, until every domain either contains only solutions, or no solution, or has a size below a user-defined threshold. When the algorithm terminates, it returns a set of candidate solutions as a collection of abstract elements. Below we define the size and splitting operators for a box.

DEFINITION 3.6 (Size of a box.) Consider $\mathcal{D}^\sharp = D_1 \times \dots \times D_n$ a box where each D_i is an interval with its lower and upper bounds as \underline{D}_i and \overline{D}_i respectively. Then, the size of the box \mathcal{D}^\sharp is given as

$$\sum_{i=1}^n (\overline{D}_i - \underline{D}_i) \quad (3.1)$$

A different notion of size can also be defined as the maximum width among all variables, i.e.:

$$\max\{\overline{D}_i - \underline{D}_i \mid i \in [1, n]\} \quad (3.2)$$

DEFINITION 3.7 (Splitting a box.) Consider a box $\mathcal{D}^\sharp = D_1 \times \dots \times D_n$ where each interval domain D_i is assigned to variable v_i . The box \mathcal{D}^\sharp can be bisected into two sub-boxes \mathcal{D}_1^\sharp and \mathcal{D}_2^\sharp by splitting the i^{th} variable (v_i). The two sub-boxes can be defined as

$$\mathcal{D}_1^\sharp \stackrel{\text{def}}{=} D_1 \times \dots \times \left[\underline{D}_i, \frac{\underline{D}_i + \overline{D}_i}{2} \right] \times \dots \times D_n \quad (3.3)$$

$$\mathcal{D}_2^\sharp \stackrel{\text{def}}{=} D_1 \times \dots \times \left[\frac{\underline{D}_i + \overline{D}_i}{2}, \overline{D}_i \right] \times \dots \times D_n \quad (3.4)$$

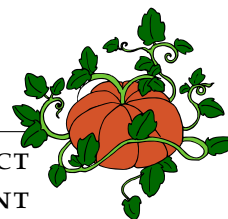
Remark 3.8. Spitting and size operators for other abstract domains like octagon and polyhedra are defined in [PMTB13]. In a later chapter of this thesis, we will be introducing these operators for zonotope abstract domain.

3.2.3 A continuous solver

Based on the propagation and splitting steps, below we describe a continuous solver on boxes from [PMTB13, MBR16]. Recall that we are interested in a real-valued program semantics, and we focus on continuous constraints. Each domain is an interval of reals and so the initial search space G^\sharp is a box. The algorithm iteratively shrinks the boxes using the constraints (consistency), keep them if they contain only solution and splits them until it reaches an instance where all the boxes contain only solutions or shrink below a user specified size limit. Thus, the solution set \mathcal{S} is tightly enclosed by a set of boxes such that $\mathcal{S} \subseteq \cup G^\sharp$.

Algorithm 3.1 – A simple continuous solver

```
solution set  $S \leftarrow \emptyset$   
search space, a set of boxes  $G^\sharp := \{\mathcal{D}\} // \mathcal{D} \stackrel{\text{def}}{=} D_1 \times \dots \times D_n$   
WHILE  $G^\sharp \neq \emptyset$  DO  
   $S^\sharp \leftarrow$  pop a box from  $G^\sharp$   
   $S^\sharp \leftarrow$  hull-consistency( $S^\sharp$ )  
  IF  $S^\sharp = \emptyset$  THEN  
    continue  
  ELSE IF  $S^\sharp$  contain only solution THEN  
     $S \leftarrow S \cup \{S^\sharp\}$   
  ELSE IF  $S^\sharp < \epsilon$  THEN  
     $S \leftarrow S \cup \{S^\sharp\}$   
  ELSE  
    split  $S^\sharp$  in into  $S_1^\sharp$  and  $S_2^\sharp$  along the largest dimension  
    push  $S_1^\sharp$  and  $S_2^\sharp$  into  $G^\sharp$ 
```



INTERACTIONS BETWEEN ABSTRACT
INTERPRETATION AND CONSTRAINT
PROGRAMMING

As of now, we have discussed abstract interpretation and constraint programming, the two important research areas in computer science. This thesis work is at the interface between these two areas. AI aims at analyzing programs by extracting properties through various representations. In CP, the goal is to solve hard combinatorial problems efficiently. No doubt the problems of these two areas are different and also their applications. However, both the areas are linked with a common interest, i.e., to compute an over-approximation of a desired set which is sometimes difficult or impossible.

In this chapter, we explain in detail the existing CP-based AI framework that is further extended to zonotope abstract domain in the later part of the thesis. This chapter also surveys a recent line of work which combines abstract interpretation and constraint programming for inferring invariants.

4.1 ARE WE INTRODUCING AI IDEAS INTO CP OR CP INTO AI?

The constraint programming community covers a huge amount of work on continuous CP [eeBO97, CJ09, Rue05, PMTB13]. The classical continuous constraint programming corresponds to interval domain [PMR12, eeBO97, CJ09]. However, there are few recent works [PTB11, PTB14] in which the authors propose consistency (using octagonal constraints to remove the domains) and propagation algorithms for octagon domain. They also proposed a splitting algorithm and a notion of precision or size adapted to the octagonal case. These works are based on using abstract domains in a constraint programming solver and solving actual constraints (system of equations) corresponding to the domain used. More precisely speaking, they aim at designing a new class of constraint solvers parameterized by abstract domains through introducing abstract interpretation ideas into constraint programming.

A very recent line of work [MBR16] on which we extend our zonotopic abstraction for finding invariants, is in the same spirit as [PTB14]. However, the authors claim that they rather introduce techniques from constraint programming in abstract interpretation since they aim at analyzing programs which has been a typical AI application for years.

The authors in [MBR16] adapt constraint programming algorithms to infer post-fixpoints of semantic functions associated with a program instead of solving constraints. Their goal is to find a set of abstract elements $G^\#$ that satisfy the following properties:

- $I^\sharp \subseteq G^\sharp$
- $G^\sharp \subseteq T^\sharp$
- $F^\sharp(G^\sharp) \subseteq G^\sharp$

Recall that, here the aim is to infer inductive invariants of numerical programs. In the first property, I^\sharp is the abstraction of the initial states of a program. For example, recall the program in Figure 2.6, the initial states of the program with respect to interval domain is the box $[-1, 1] \times [-1, 1]$.

In the second property, T^\sharp (for instance, the box $[-2, 2] \times [-2, 2]$ for the program in Figure 2.6 with interval domain) is the target invariant which is strengthened into an inductive invariant G^\sharp .

If one uses only abstract interpretation to satisfy the above properties then the set is composed of a single abstract element, trying to infer an inductive invariant in it. However, when combined with CP, the set becomes a union or collection of abstract elements. Thus, it wouldn't be wrong to say that this technique is also a CP based AI framework which solves constraints on sets. There is a recent work of Jaulin *et al.* [Jau12] which solves similar problems, and they call it set-based constraint satisfaction problems.

The idea of [MBR16], inspired from constraint programming approaches, is to synthesize an inductive invariant as a collection of abstract elements, that are iteratively split and refined. In set-based constraint programming, these elements are generally boxes. Previous work [MBR16] was limited to abstract domains that are closed by intersection and required non-standard operations: split and size, such as octagons. In this thesis, we extend this work to zonotopes, which we show they provide an interesting trade-off between expressiveness and efficiency for such a use, by comparing their use with that of boxes, octagons, and polyhedra. In the forthcoming chapters, we will introduce the operators required to combine the zonotopic abstraction with the CP algorithm.

4.2 REFINEMENT-BASED INDUCTIVE INVARIANT INFERENCE

Here we recall the algorithm from [MBR16] to tighten an invariant into an inductive invariant, by splitting and tightening a collection of abstract elements. We further extend this algorithm with zonotopic abstraction for finding inductive invariants.

4.2.1 Concrete semantics.

Recall that we analyze a loop in a numeric program. We assume without loss of generality that variables are real-valued (which includes integers and non-special float values). A program environment is a subset of \mathbb{R}^n , where n is the number of variables. The concrete semantics of a program is the collecting semantics¹ of [CC77]: it is given as the least fixpoint of a function $F : \mathcal{P}(\mathbb{R}^n) \rightarrow \mathcal{P}(\mathbb{R}^n)$ over an initial environment $I \subseteq \mathbb{R}^n$; typically, when a program consists of just one loop, F is the transfer function for one iteration of the loop.

¹It is the strongest program property of interest and computing it would answer all safety queries. However, it is impossible (except for finite systems) to compute this collecting semantics.

4.2.2 *Target invariant.*

We also assume that we are given a target invariant $T \subseteq \mathcal{P}(\mathbb{R}^n)$. We wish to infer an inductive invariant $G \subseteq \mathcal{P}(\mathbb{R}^n)$ that proves that T is indeed an invariant. This requires finding G such that: $I \subseteq G$ (includes the initial states), $F(G) \subseteq G$ (induction), and $G \subseteq T$ (invariant entailment). Note that the least fixpoint of F greater than I , $\text{lfp}_I F$, is always a solution as it is the smallest invariant and it is inductive. However, neither F , nor $\text{lfp}_I F$ is computable in general.

4.2.3 *Abstract semantics.*

As a first step, we replace computations in $\mathcal{P}(\mathbb{R}^n)$ with computations in an abstract domain $\mathcal{D}^\sharp \subseteq \mathcal{P}(\mathbb{R}^n)$, that is, a set of well-chosen abstract elements that can be efficiently represented in memory. Moreover, we replace the concrete function F with a computable version $F^\sharp : \mathcal{D}^\sharp \rightarrow \mathcal{D}^\sharp$ within the abstract world. As generally the image by F of an abstract element is not exactly representable in the abstract, F^\sharp computes an approximation that, to be sound, over-approximates the set of environments.

Abstract interpretation provides a library of domains \mathcal{D}^\sharp as well as systematic methods to derive a sound abstract F^\sharp from the program source; for instance, \mathcal{D}^\sharp can be the axis-aligned boxes (i.e., the interval domain) and F^\sharp is derived through interval arithmetic.

Classic abstract interpretation would infer an abstract inductive invariant, i.e., an element G^\sharp such that $F^\sharp(G^\sharp) \subseteq G^\sharp$, as this implies by soundness that $F(G^\sharp) \subseteq G^\sharp$ while only requiring a computable abstract version F^\sharp of F . The algorithm proposed in [MBR16] reasons instead in the *disjunctive completion*, $\mathcal{P}(\mathcal{D}^\sharp)$: it searches for an inductive invariant that can be expressed as a finite collection $G^\sharp \subseteq \mathcal{D}^\sharp$ of abstract elements, i.e., $G^\sharp = \{S_1^\sharp, \dots, S_n^\sharp\}$ that satisfies:

Property 1. $S_1^\sharp \neq S_2^\sharp \in G^\sharp \implies \text{vol}(S_1^\sharp \cap S_2^\sharp) = 0$

Property 2. $I^\sharp \subseteq \bigcup_i S_i^\sharp$

Property 3. $\forall k : F^\sharp(S_k^\sharp) \subseteq \bigcup_i S_i^\sharp$

Property 4. $\forall k : S_k^\sharp \subseteq T^\sharp$

which implies that $\bigcup_i S_i^\sharp$ is an inductive invariant. Property 1 ensures that the boxes in G^\sharp do not overlap. The authors in [MBR16] use the interval abstract domain, which allows non-strict inequality constraints only. They do not enforce that G^\sharp forms a partition. So, if the boxes do intersect, their intersection has a null volume. Property 2 certifies that G^\sharp covers initial set I^\sharp . Property 3 implies $F^\sharp(\bigcup_k S_k^\sharp) \subseteq \bigcup_i S_i^\sharp$, i.e., G^\sharp is inductive and property 4 ensures that G^\sharp implies the candidate invariant.

To simplify, it is assumed that both the initial states I^\sharp and the target invariant T^\sharp are exactly expressible in the abstract domain \mathcal{D}^\sharp (this is often the case, e.g., if both represent variable bounds, expressible with boxes). On the other hand, the disjunctive completion of \mathcal{D}^\sharp , where G^\sharp lives, is strictly more

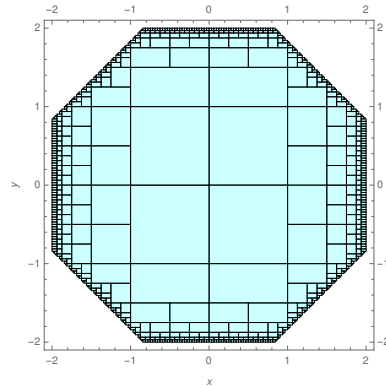


Figure 4.1 – Inductive invariant for the program in Figure 2.6

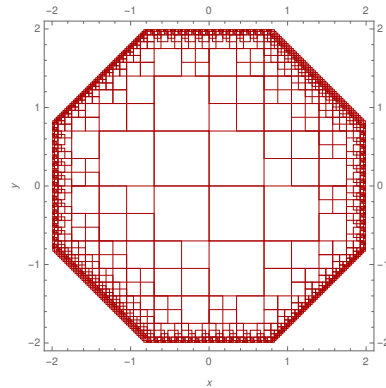


Figure 4.2 – The image of abstract elements in Figure 2.2 by a loop iteration in the abstract domain used for computing the inductive invariant

expressive than \mathcal{D}^\sharp , as abstract domains are seldom closed by set union \cup . For instance, the algorithm can find an inductive invariant that is a collection of boxes that cannot be exactly expressed as a single box. This feature is key to avoid the expressiveness escalation observed in classic abstract interpretation, where one must design more complex abstract domains to handle programs with complex inductive invariants. This feature is illustrated below with the help of an example.

EXAMPLE 4.1 Consider the example in Figure 2.6 again. This time, we will use the CP algorithm of [MBR16] parameterized by the interval abstract domain. It searches for an inductive invariant by strengthening a user provided initial invariant which is here the $[-2, 2] \times [-2, 2]$ box. Indeed, the algorithm finds an inductive invariant inferred by a collection of boxes in Figure 4.1.

Figure 4.3 illustrates that these set of boxes are stable by the body loop iteration (Figure 4.2) of the program: $F^\sharp(G^\sharp) \subseteq G^\sharp$ where G^\sharp is the set of boxes inferring the inductive invariant.

Remark 4.2. Thus, with the ideas from constraint programming, it is possible to approximate complex shapes using non-relational abstract domains. We

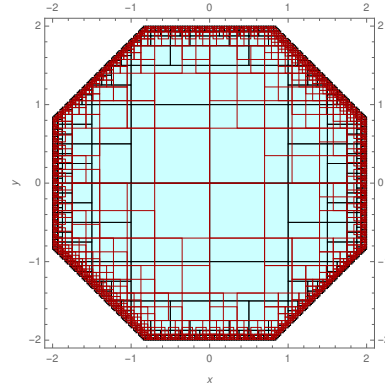


Figure 4.3 – Superposition of the two figures 4.1 and 4.2 showing that 4.2 is included in 4.1, i.e., 4.1 is inductive

Algorithm 4.1 – A CP based AI algorithm for inferring inductive invariants [MBR16]

```

Function{SearchInvariant}{I#, F#, T#}
search space & final solution set, a set of abstract elements G# := {T#}
WHILE G# ≠ ∅ DO
  S# ← pop an abstract element from G# // Based on minimum coverage
  IF coverage(S#, G# ∪ {S#}) = 1 THEN
    return G# ∪ {S#}
  ELSE IF S# is not necessary and (coverage(S#, G# ∪ {S#}) <
    εc or size(S#) < εs or S# is not useful ) THEN
    remove S#
  ELSE IF size(S#) < εs THEN
    return failure
  ELSE
    split S# in into S1# and S2# // every split element is also tightened
    push S1# and S2# into G#
EndFunction

```

are further interested in investigating if CP influences relational domains like zonotopes. This curiosity is due to the fact that zonotopes, being a subpolyhedral domain that shows a good compromise between cost and precision, combined with CP can prove efficient in analyzing programs with non-linear loops that present complex, possibly non-convex, invariants.

4.3 SEARCH ALGORITHM.

The CP based AI framework for finding inductive invariants is shown in Algorithm 4.1. The algorithm maintains a finite collection $G^\# \subseteq \mathcal{D}^\#$ that forms a candidate inductive invariant. This is illustrated in Figure 4.4. It is initialized with the target invariant: $G^\# = \{T^\#\}$. It is ensured at all times that $I^\# \subseteq \bigcup_i S_i^\# \subseteq T^\#$, and iteratively refine $G^\#$ until it becomes inductive, i.e., until $\forall k : F^\#(S_k^\#) \subseteq \bigcup_i S_i^\#$. While $G^\#$ is not inductive, the following steps are iterated:

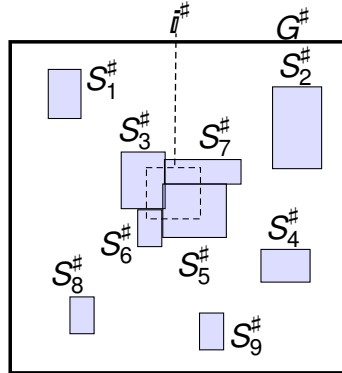


Figure 4.4 – A collection of abstract elements manipulated by the algorithm

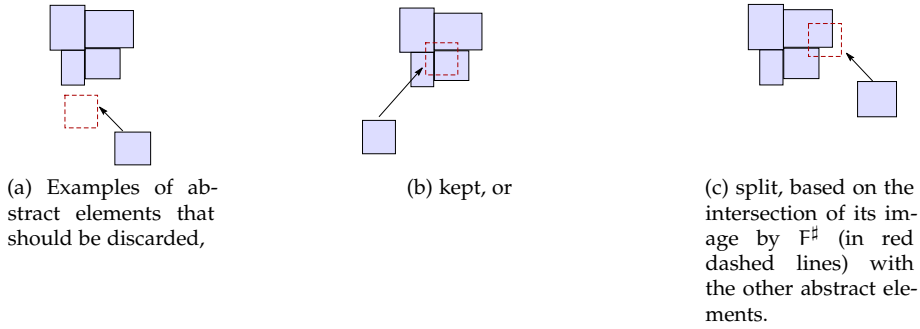


Figure 4.5 – Classification of abstract elements

- pick an element $S_k^\sharp \in G^\sharp$ with $F^\sharp(S_k^\sharp) \not\subseteq \bigcup_i S_i^\sharp$, i.e., preventing inductiveness;
- either discard S_k^\sharp from G^\sharp , split it into two elements that are added back to G^\sharp , or tighten it.

Remark 4.3. The algorithm presented in Figure 8 in [MBR16] splits an abstract element always into two sub-parts. However, in the zonotopic abstracted version of the algorithm, we split a zonotope into a partition of more than two sub-zonotopes. We will discuss this splitting method in the later part of the thesis.

To decide the action to perform, it is important to classify an abstract element S_k^\sharp in relation to the other elements in G^\sharp and their image by F^\sharp . Thus, an abstract element S_k^\sharp can be classified in different ways:

- *doomed* if $F^\sharp(S_k^\sharp) \cap (\bigcup_i S_i^\sharp) = \emptyset$ (Figure 4.5a); such an element will always prevent inductiveness and must be discarded;
- *benign* if $F^\sharp(S_k^\sharp) \subseteq \bigcup_i S_i^\sharp$, i.e., it does not prevent inductiveness and does not need to be changed (Figure 4.5b);

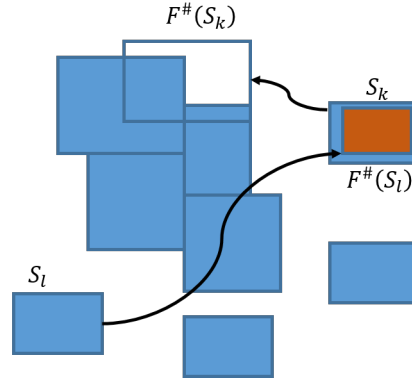


Figure 4.6 – Useful abstract elements

- *necessary* if $S_k^\sharp \cap I^\sharp \neq \emptyset$, i.e., it cannot be discarded, to keep ensuring that $I^\sharp \subseteq \bigcup_i S_i^\sharp$ always holds;
- *useful* if $S_k^\sharp \cap (\bigcup_i F^\sharp(S_i^\sharp)) \neq \emptyset$, i.e., an element of G^\sharp relies on S_k^\sharp to be benign.

EXAMPLE 4.4 In the figure, the box S_k^\sharp intersects the image of a box S_l^\sharp under F^\sharp which implies that $F^\sharp(S_l^\sharp) \subseteq \bigcup_i S_i^\sharp$, i.e., S_k^\sharp helps make S_l^\sharp benign. If S_k^\sharp is discarded then it leaves S_l^\sharp non-benign, i.e., $F^\sharp(S_l^\sharp) \not\subseteq \bigcup_{i \neq k} S_i^\sharp$ and eventually the algorithm fails.

The algorithm first selects a non-benign element S_k^\sharp . It is discarded if it is either doomed or not useful, unless it is necessary. Otherwise, it is generally split, as exemplified in Figure 4.5c. By splitting S_k^\sharp , we can hope to isolate the part that is doomed, and is ultimately discarded, from the part that is benign, and kept in the final inductive invariant.

4.3.1 Coverage

It is also possible to discard a useful, non-benign S_k^\sharp . On the one hand, this is beneficial as it removes an offender preventing inductiveness, as it is not benign. On the other hand, discarding an element that is useful means that other elements that depend on it to be benign (i.e., S_k^\sharp intersects the image by F^\sharp of these elements) will not be benign anymore, triggering a sequence of splits and removals. We exercise this option when we deem that it is not worth splitting S_k^\sharp to expose benign parts. To guide our choice, a useful quantitative measure is that of coverage:

$$\text{coverage}(S_k^\sharp) := \frac{\sum_i \text{vol}(F^\sharp(S_k^\sharp) \cap (\bigcup_i S_i^\sharp))}{\text{vol}(F^\sharp(S_k^\sharp))} \quad (4.1)$$

where $\text{vol}(X^\sharp)$ is the volume of an abstract element X^\sharp . Intuitively, the coverage measure is: how much the image of S_k^\sharp lies in the candidate invariant. Or in

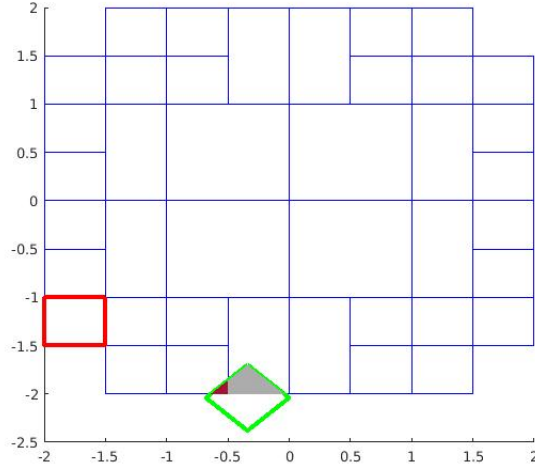


Figure 4.7 – Computation of coverage information

other words how much it is inductive. Note that, for this to make sense, it is important to rely on the fact that the S_k^\sharp do not overlap (up to the common borders that have a null volume).

EXAMPLE 4.5 Consider the box in red color and its image in green shown in Figure 4.7. The coverage information for this box can be computed as $\frac{\text{red} + \text{gray}}{\text{green}}$.

Remark 4.6. Coverage is first used to decide which S_i^\sharp in G^\sharp to consider. Then, it is used to decide whether to split or discard. Computing $\sum_i \text{vol}(F^\sharp(S_k^\sharp) \cap (\cup_i S_i^\sharp))$ can be fairly expensive. So, in the later part of the thesis, we propose a coverage metric which no longer relies on volume computation to decide whether to split an abstract element or not.

Note that benign elements have a coverage of 1, and doomed elements have a coverage below the user specified threshold ϵ_c . The algorithm systematically picks the element $S_k^\sharp \in G^\sharp$ with the least coverage in priority, as these require the most urgent actions. For deciding if an abstract element is benign, the algorithm does not use coverage measure, and we will discuss why in the later part of this chapter. Rather, the benignness property is evaluated by inclusion checking.

Remark 4.7. In [MBR16], elements with a very low coverage are systematically discarded, as unlikely to become benign. However, it is possible that an abstract element may intersect every inductive invariant, in which case discarding this element will result in a failure. So, in the current work, we only discard an abstract element if it has a coverage of 0.

4.3.2 Tightening

As in constraint programming, the abstract elements can also be tightened by removing useless parts. More precisely, any part of S_k^\sharp that does not intersect any $F^\sharp(S_i^\sharp)$ is not useful and can safely be discarded, improving the likelihood that S_k^\sharp becomes benign, without making other benign elements non-benign. More precisely, tightening replaces S_k^\sharp with:

$$\bigcup_i (S_k^\sharp \cap F^\sharp(S_i^\sharp)) \cup (S_k^\sharp \cap I^\sharp) \quad (4.2)$$

suitably over-approximated to an element in the abstract domain (e.g., in the interval domain, Equation (4.2) is a union of boxes, and we compute its box hull).

4.3.3 Splitting

Consider an abstract element S_k^\sharp which is not benign but cannot be discarded since it is either necessary or useful. So, the Algorithm 4.1 performs splitting operation on S_k^\sharp to help make it benign. S_k^\sharp is split into its two sub-parts $S_{k_1}^\sharp$ and $S_{k_2}^\sharp$ such that $S_{k_1}^\sharp \cup S_{k_2}^\sharp = S_k^\sharp$. It is split, unless it is either benign or too small to split it anymore.

While replacing S_k^\sharp with $S_{k_1}^\sharp$ and $S_{k_2}^\sharp$ does not change G^\sharp but it is possible to reduce the size of $F^\sharp(G^\sharp)$, helping G^\sharp to become benign, $F^\sharp(G^\sharp) \subseteq G^\sharp$.

Remark 4.8. It is important to note that, currently the split operation on boxes, octagons and polyhedras, in the algorithm splits an abstract element into its two sub-parts. We introduce in the next chapter a splitting operation on zonotopes adapted with the algorithm which allows a zonotope to be cut into more than two sub-parts.

4.3.4 Size

In constraint programming, a complex shape invariant is approximated by a collection of abstract elements up to a precision or size criterion. Thus, the Algorithm 4.1 requires a threshold ϵ_s for the size-parameter below which it restrains from splitting the abstract elements. An approximate notion of size is sufficient for the algorithm. Thus, it simply uses as size of an abstract element the size of its bounding box, which is very easy to compute.

4.3.5 Failure

Unlike constraint programming but similarly to abstract interpretation, the algorithm can fail to find an inductive invariant. This comes from several reasons. One of them is the bad decision to discard an element containing environments required in all inductive invariants. For instance, a useful abstract element that actually intersects the smallest inductive invariant. Moreover, because the algorithm starts from the target invariant and discards portions of it until it becomes inductive, it often outputs inductive invariants near the weakest invariants able to prove the property. This is in contrast with abstract

interpretation, which goes upwards and over-approximates the strongest invariant.

The algorithm can also terminate with a failure if it cannot find an inductive invariant after finitely many splits with the size of the abstract elements reaching less than ϵ_s .

When there is a failure, values of ϵ_s and ϵ_c are lowered and the algorithm is called again with few additional steps:

- discarding abstract elements from G^\sharp which are not reachable from the entry states
- splitting abstract elements actively or re-splitting them to isolate the doomed parts

This procedure is applied iteratively until an inductive invariant is found. Note that iteratively reapplying the algorithm followed by the failure recovery steps can be very costly.

Remark 4.9. In that way, the algorithm is *relatively-complete*, i.e., if there is an inductive invariant in the abstract domain then the algorithm will find it.

4.3.6 Data structure.

Recall that the Algorithm 4.1 requires maintaining information about abstract elements in G^\sharp , such as their coverage and whether they are benign or useful. It is important to note that modifying a single abstract element in G^\sharp may not only modify the coverage or class of the modified abstract element, but also of other abstract elements in G^\sharp . We discuss here data structures which was already proposed in [MBR16] to perform such updates efficiently, without examining G^\sharp entirely after each operation.

Throughout the execution of the algorithm, a set of abstract elements B^\sharp is maintained, similarly to the set G^\sharp of abstract elements which is ultimately going to partition the inductive invariant, by initializing it as the initial (tightened) invariant, and splitting it the same way G^\sharp is split. Unlike G^\sharp where we also discard some (sub-)abstract elements, B^\sharp always covers the whole (tightened) initial invariant space. Therefore, an abstract element in B^\sharp is matched by at most one abstract element from G^\sharp . This notion can be ensured by maintaining a *contents-of* function $\text{cnt} : B^\sharp \rightarrow (G^\sharp \cup \{\emptyset\})$ indicating which abstract element of G^\sharp , if any, is contained in the partition B^\sharp .

Remark 4.10. Maintaining B^\sharp is an optimization to compute the coverage easily. The idea is that, with B^\sharp , the algorithm does not need to consider all abstract elements in the volume computation, and it is easy to maintain B^\sharp when G^\sharp evolves.

A map $\text{post} : G^\sharp \rightarrow P(B^\sharp)$ is retained to indicate which parts of B^\sharp intersect the image of an abstract element $S_k^\sharp \in G^\sharp$:

$$\text{post}(S_k^\sharp) := \{P^\sharp \in B^\sharp \mid F^\sharp(S_k^\sharp) \cap P^\sharp \neq \emptyset\} \quad (4.3)$$

The map post is further used to compute coverage and also to decide whether a box is benign or not. The coverage of an abstract element S_k^\sharp , defined in

Equation (4.1), can be computed as:

$$\text{coverage}(S_k^\#) := \frac{\sum \{\text{vol}(F^\#(S_k^\#) \cap \text{cnt}(P^\#)) \mid P^\# \in \text{post}(S_k^\#)\}}{\text{vol}(F^\#(S_k^\#))} \quad (4.4)$$

Remark 4.11. The coverage information is first used to decide which S_i in $G^\#$ to consider. Then, it is used to decide whether to split or discard. To guide the split, this is just a heuristic without much requirement. In [MBR16], in order to compute this heuristic, the authors used as the volume of an abstract element the volume of its bounding box, which is very easy to compute. In this thesis work, we introduce a simple heuristic which does not rely anymore on computing volume. We will discuss this in the next chapter.

It will not be safe to use coverage information to determine whether an abstract element is benign or not because the computation in Equation (4.4) is subject to floating-point round-off error. Thus, in order to check for inductiveness or benignness, a sound coverage test is needed, but this is inclusion checking.

An abstract element is inductive or benign, if whenever $F^\#(S_k^\#)$ intersects some partition $P^\# \in B^\#$, is included in $\text{cnt}(P^\#) \in B^\#$. It is also additionally checked if $F^\#(S_k^\#)$ is included within the candidate invariant $T^\#$. Thus, the formal representation of the benign test is,

$$S_k^\# \text{ is benign} \iff \forall P^\# \in \text{post}(S_k^\#) : P^\# \cap F^\#(S_k^\#) \subseteq \text{cnt}(P^\#) \wedge F^\#(S_k^\#) \subseteq T^\# \quad (4.5)$$

4.4 RELATED WORK

By this time, we have discussed how iterative fixpoint computation based technique like abstract interpretation has been combined with continuous constraint programming for discovering inductive invariants. However, recent years have seen many satisfiability based constraint solvers (they encode programs as formulas) at the core of program analysis and verification tools. So, we will be reviewing the state-of-the-art methods which combine techniques based on abstract interpretation and based on satisfiability for inferring inductive invariants.

4.4.1 CP using SAT/SMT solvers

In program verification, the problem of finding inductive invariants can be boiled down to the problem of solving second-order constraints because programs can be translated into constraints [GSV08, CSS03]. One can use fixed-point based techniques like abstract interpretation to solve these constraints. Besides, there also exist constraint-based invariant generation techniques where the constraints are solved by already existing constraint solvers like SAT/SMT² [DMB08]. However, in these techniques the analysis problem have to be converted into constraints that can be solved by the SAT/SMT solvers. For instance, first, the second-order constraints have to be converted into the first-order constraints because SAT/SMT solvers are based on some

²SAT-Boolean satisfiability, SMT-Non-boolean satisfiability

logical theories to determine if a first-order formula is satisfiable. Then, these first-order constraints are converted into a SAT/SMT formula using bit-vector modeling. Thus, these constraint-based invariant generation techniques have to rely on lot of preprocessing before solving the constraints for finding the inductive invariant.

There are also linear and non-linear solvers which are used for inferring inductive invariants [CSS03].

4.4.2 SAT-based model checking

Like abstract interpretation, *model checking* (MC) is also an iterative fixed-point technique. Provided with a property (φ) of a system and a structure (S), model checking verifies if the structure meets the property or in other words if S is a model of φ (i.e., $S \models \varphi$). In particular, it checks if S with an initial state s is a model of φ (i.e., $S, s \models \varphi$). It verifies if all the computations of S meet φ from the initial state s .

A simple example of such a model checking problem is propositional satisfiability, i.e., given a formula φ and an assignment σ (which maps propositional variables to truth table values) whether σ is a model of φ . With the advent and popularity of SAT algorithms, SAT-based model checking became the core of many program analysis tools.

SAT-based model checking algorithms like IC3 (Incremental Construction of Inductive Clauses for Indubitable Correctness) [Bra11], PDR (Property Directed Reachability) [HB12] are used for inferring safe inductive invariants. These algorithms work iteratively, either strengthening a candidate invariant into an inductive invariant or find a counter example.

IC3 is an incremental SAT-based MC algorithm which maintains a collection of candidates. It strengthens and weakens the candidates based on the examples provided by the SAT-solver until one of them becomes a safe inductive invariant. IC3 has been successfully used for hardware model checking [Bra11, een11].

4.4.3 Combined AI and CP approaches

There are various ways in which abstract interpretation and constraint programming frameworks have been combined together for inferring inductive invariant in programs. Below we discuss them in detail.

Decision procedure based abstract interpreters. Abstract interpretation and constraint programming are combined in a way to automate AI framework by designing a sound abstract transfer function or transformer³ using SMT solvers [TLLR15, RT16]. The authors show, how sound abstract transformers can be constructed by using logic: they call this connection (using logic to define program semantics) between abstract interpretation and logic as *symbolic abstraction*.

DEFINITION 4.12 (Symbolic abstraction.) Consider a formula $\varphi \in \mathcal{L}$, where \mathcal{L} is a logic, and an abstract domain \mathcal{A} . Then, symbolic abstraction can be

³Functions on an abstract domain are called abstract transformers and those on a concrete domain are concrete transformers.

defined as the relationship that maps φ to the best or most-precise value (A^\sharp) in the abstract domain \mathcal{A} that over-approximates φ .

Remark 4.13. If such a precise descriptor A^\sharp exists in \mathcal{A} which over-approximates the formula φ (i.e., $\llbracket \varphi \rrbracket \subseteq \gamma(A^\sharp)$) then

$$\hat{\alpha}_{\mathcal{A}}(\varphi) = \alpha_{\mathcal{A}}(\llbracket \varphi \rrbracket) \quad (4.6)$$

where $\hat{\alpha}_{\mathcal{A}}(\varphi)$ denotes the symbolic abstraction of φ with respect to the abstract domain \mathcal{A} .

Similarly, the authors defined *symbolic concretization*, a relationship which maps an abstract value $A^\sharp \in \mathcal{A}$ to a formula. In [TLLR15], the authors discuss, how they use symbolic abstraction ($\hat{\alpha}_{\mathcal{A}}(\varphi)$) for finding the most precise inductive \mathcal{A} -invariant for a given program and an abstract domain \mathcal{A} . There are several algorithms for computing the symbolic abstraction: RSY algorithm [RSY04], bilateral algorithm [TER12]. The key notion of both the algorithms is to use SMT solver for the logic \mathcal{L} to look for models⁴ ($\sigma \models \varphi$) of the formula φ . These models (σ) are used to compute the abstract values A^\sharp .

Characterizing satisfiability algorithms using fixpoints. There are other recent works [DHK12, DHK14], in which AI and CP have been integrated together to help design efficient satisfiability algorithms using abstract interpretation. The authors show that the algorithms for solving satisfiability have abstract interpretation characterizations, i.e., *satisfiability solvers are abstract interpreters*. They do so by deriving fixpoint (least and greatest fixpoints) descriptions of models and countermodels⁵ of a formula.

4.4.4 Learning loop invariants

Besides AI and CP, there are other approaches for inferring invariants that are based on learning paradigms, i.e., they try to learn from the past mistakes as opposed to search based methods like abstract interpretation.

Garg et al. [GLMN14, GNMR16] developed a machine learning-based paradigm for loop invariant synthesis known as ICE, where the learner synthesizes a candidate invariant and the teacher verifies it using a constraint solver. The teacher also provides feedback in the form of a positive or negative examples. The learner uses the example to enhance its inference. This process continues until the teacher concludes that the candidate invariant is strengthened into an inductive invariant.

The ML-based invariant inference techniques use different learning algorithms: decision trees [GNMR16], neural networks [SDR⁺18].

IC3 (the SAT-based model checker) and ICE have few similarities at a high-level abstraction. For example, the SAT solver in IC3 can be considered as a teacher and the rest of the algorithm as a learner with satisfying assignments playing the role of examples [VGSM17].

⁴In propositional logic, σ is an assignment that maps variables to truth table values, i.e., true or false. If an assignment σ satisfies a formula φ then it is a model of φ .

⁵An assignment σ is a counter model if it does not satisfy a formula φ (i.e., $\sigma \not\models \varphi$)

4.4.5 Eigen vectors as invariants

The authors in [dOBP16, dOBP17, DO18] presented a technique based on linear algebra concept of eigenspace for inferring linear invariants of linear loops. They proved that the left eigenvectors of a loop transformation F are the set of invariants of a loop. However, in order to compute the eigenvectors, first they had to reduce the polynomial loop into a linear one using linearization procedures (the problem boils down to searching for linear invariants).

Remark 4.14. Essentially, the recent line of work dedicated to synthesizing invariants combines abstract interpretation and constraint-based techniques. In constraint-based techniques, we reviewed techniques based on continuous constraint programming, working on geometric entities, such as boxes, octagons, polyhedras and techniques on SAT/SMT working on formulas (a model based on Boolean variables) for which the algorithms are dedicated. There are similarities between the solving process of these two techniques. However, both the model and the solving methods differ.

PART II

INVARIANTS OF DISCRETE
SYSTEMS



We have already seen that the algorithm discussed in the previous chapter is parameterized by a choice of abstract domain \mathcal{D}^\sharp . In addition to an abstract version F^\sharp of F , already provided by abstract interpretation, it requires a split operator, a test for deciding if two abstract elements intersect and similarly the test for inclusion. Such operators have been proposed for boxes and octagons in [MBR16]. This chapter recalls the zonotope domain, introduces our new operators (provide the missing operators for zonotopes), and discusses how to handle domains that are not intersection-closed.

The missing operator for zonotopes is mainly the splitting. An obvious way to split a zonotope is with overlap which is simple, close to that on boxes. We introduce a novel splitting based on paving zonotopes by sub-zonotopes and parallelotopes.

We have designed a new inclusion test for zonotopes based on the work of Goubault *et al.* [GP15] improving the complexity of the test by an exponential bound compared to the previous work.

Regarding the meet operator, there exists already the work of Ghorbal *et al.* [GGP10b] by using constrained affine sets: the constraints produced by the tests in a program are interpreted over the noise symbols of the affine forms. Unlike the previous work which is a meet operation over the affine forms, here we introduce a new meet operation on zonotopes which is rather a geometrical meet between two zonotopes keeping in mind the geometrical aspect of our splitting operator applied later. Also, this meet introduces new direction of faces which is useful for our splitting by tiling fashion, paving the zonotope obtained after the meet into sub-zonotopes.

5.1 CONSTRAINT SOLVING ALGORITHM ON ZONOTOPES

The framework illustrated in Algorithm 5.1 is the CP based AI Algorithm 4.1 on zonotope abstract domain. As discussed in Section 4.3.2 in the base algorithm, tightening (Equation (4.2)) is applied at each iteration after the split operation for boxes and octagons. However, in the algorithm on zonotopes tightening is only applied during the first iteration as shown in Algorithm 5.1. Recall that tightening requires set-theoretic operation like meet and the precision and efficiency of the meet operation plays an important role in the effectiveness of tightening. Since tightening removes parts of an abstract element that prevents it from becoming necessary or useful, it is important to have a meet operation which is exact.

It is well-known that zonotope is a sub-polyhedral abstract domain which does not enjoy the property of being closed under intersection. In the later

part of the thesis, we propose a meet operator that over-approximates the intersection of zonotopes. Applying this over-approximated meet operation is crucial for the initial states, but applying it in further iterations proved to gain little. So, for this current work, we restrict the tightening operation only to the first iteration. However, applying tightening at every iteration for abstract domains which are not closed under intersection is a scope for future work.

The algorithm 5.1 maintains, in G^\sharp , a set of zonotopes to explore, initialized with the tightened candidate invariant T^\sharp of interest. Then, while there are zonotopes to explore, the zonotope S^\sharp with coverage (Equation (4.1) or Equation (4.4)) equal to 0 is removed from G^\sharp . Coverage is also used to pick any zonotope from G^\sharp with least coverage value (but not zero) because this zonotope requires urgent action to become benign. As discussed in Section 4.3.1 and 4.3.6 the coverage information is a heuristic and it is not required to compute the exact volume of the abstract element to calculate the coverage measure. Moreover, computing the volume of a zonotope can be reasonably expensive and we will illustrate this with an example afterwards. In the later part of this chapter, we propose a simpler and efficient coverage metric to guide the splitting operation. Recall that however to decide whether a zonotope is benign, the algorithm requires inclusion checking (Equation (4.5) in Section 4.3.6). So, it is important to have a test for inclusion check which must be exact. We will define later a cost-effective test based on an earlier work on inclusion tests for zonotopes.

If a zonotope S^\sharp is benign, and so are the remaining zonotopes in G^\sharp ; hence all of the zonotopes are benign and the algorithm has found an inductive invariant: we add back S^\sharp to G^\sharp and return that set of zonotopes. Otherwise, we split S^\sharp to help make S^\sharp benign. In that way, a sequence of splits will reduce a zonotope's size. The algorithm refrains from splitting zonotopes below a certain size-parameter cut-off ϵ_s and so the algorithm requires a size measure for zonotopes. Recall that an approximate notion of size is sufficient for the algorithm. For instance, the size of a zonotope can be over-approximated as the size of its bounding box.

In the general case, where a zonotope S^\sharp is not necessary (does not intersect I^\sharp), we are free to discard S^\sharp , which we do if it is useless. To decide if a zonotope is necessary or useless, the algorithm requires test for intersection check. Moreover, recall that it is possible that a zonotope may intersect every inductive invariant, in which case discarding this zonotope will result in a failure. Hence, the test for intersection check must be an exact one. We discuss the test for intersection check on zonotopes in the later part of this chapter.

Thus, the list of operations required by the Algorithm 5.1 that must be defined for the zonotope abstract domain are:

- inclusion test,
- intersection test,
- meet,
- volume,
- coverage metric,
- size, and

Algorithm 5.1 – The zonotopic variant of the CP based AI algorithm 4.1 for inferring inductive invariants

```

Function{SearchInvariant}{ $T^\sharp, F^\sharp, T^\sharp$ }
 $T^\sharp \leftarrow T^\sharp \cap F^\sharp(T^\sharp)$  // Tightening the target invariant
search space & final solution set, a set of zonotopes  $G^\sharp := \{T^\sharp\}$ 
WHILE  $G^\sharp \neq \emptyset$  DO
   $S^\sharp \leftarrow$  pop a zonotope from  $G^\sharp$  // Based on minimum coverage
  IF  $S^\sharp$  is benign THEN
    return  $G^\sharp \cup \{S^\sharp\}$ 
  ELSE IF  $S^\sharp$  is not necessary and (coverage( $S^\sharp, G^\sharp \cup \{S^\sharp\}$ ) = 0 or size( $S^\sharp$ ) <  $\epsilon_s$  or  $S^\sharp$  is not useful ) THEN
    remove  $S^\sharp$ 
  ELSE IF size( $S^\sharp$ ) <  $\epsilon_s$  THEN
    return failure
  ELSE
    split  $S^\sharp$  into a set  $\{S_1^\sharp, S_2^\sharp, \dots\}$  such that  $S^\sharp = \cup_i S_i^\sharp$ 
    push  $\{S_1^\sharp, S_2^\sharp, \dots\}$  into  $G^\sharp$ 
EndFunction

```

- splitting.

Remark 5.1. We will be using the same example from Figure 2.10 for the rest part of this chapter. Recall that for $n = 5$ and $p = 2$, the zonotope was the concretization of the affine forms $X = (\hat{x}, \hat{y})$ with $\hat{x} = 20 - 3\epsilon_1 + 5\epsilon_2 + 2\epsilon_3 + 1\epsilon_4 + 3\epsilon_5$, $\hat{y} = 10 - 4\epsilon_1 + 2\epsilon_2 + 1\epsilon_4 + 5\epsilon_5$, that is, $A^T = \begin{pmatrix} 20 & -3 & 5 & 2 & 1 & 3 \\ 10 & -4 & 2 & 0 & 1 & 5 \end{pmatrix}$. We will represent zonotope A as (c, g_1, \dots, g_n) , i.e., as its center c and its collection of generators g_1, \dots, g_n . For the example above, we would write $A = ((20, 10), (-3, 4), (5, 2), (2, 0), (1, 1), (3, 5))$

5.2 INCLUSION TEST

We know from Equation (4.5) in Section 4.3.6 that the algorithm requires inclusion checking to verify if an abstract element is benign (Equation (5.22)). We proceed as described below.

The best known method for inclusion tests are known to have exponential time (in terms of the number of generators) for zonotopes [GP15]. Lemma 5.2 below is an extension of Lemma 4 of [GP15], which transforms the inclusion test into an infinite number of simple inequalities, that in turn translate into an exponential number of linear programs to be solved. Here we can further decrease the number of linear programs to solve to a polynomial number.

LEMMA 5.2 *For two zonotopes given by matrices $X \in \mathcal{M}(n_X + 1, p)$ and $Y \in \mathcal{M}(n_Y + 1, p)$, let $u = \{u_1, \dots, u_k\}$ be vectors in \mathbb{R}^p such that each face in $\gamma(Y)$ has a vector in u that is normal to it. Then $\gamma(X) \subseteq \gamma(Y)$ if and only if*

$$\left| \langle u_i, c_x - c_y \rangle \right| \leq \|Y_+ u_i\|_1 - \|X_+ u_i\|_1, \forall i = 1, \dots, k \quad (5.1)$$

where c_x, c_y are the centers of the zonotopes $\gamma(X), \gamma(Y)$ respectively

Proof. Let us prove first that inequalities in Equation (5.1) are sufficient conditions for inclusion of X into Y .

Based on Lemma 1 of [GP15], let us define $\gamma_{\text{lin}}(Y_+)$ as

$$\gamma_{\text{lin}}(Y_+) = \bigcap_{1 \leq i \leq k} \left\{ x \in \mathbb{R}^p \mid |u_i^T x| \leq \|Y_+ u_i\|_1 \right\}$$

where each u_i is normal to the faces of $\gamma(Y)$ (or equivalently of $\gamma_{\text{lin}}(Y_+)$). Let x be any point such that $x \in \gamma(X)$. Let $x = x' + c_x$ such that $x' \in \gamma_{\text{lin}}(X_+)$ and x'' be any point such that $x'' = x' + (c_x - c_y)$. Let us assume that

$$\left| \langle u_i, c_x - c_y \rangle \right| \leq \|Y_+ u_i\|_1 - \|X_+ u_i\|_1, \forall i = 1, \dots, k.$$

Under this assumption and also by Lemma 2 of [GP15] i.e., $\sup_{x' \in \gamma_{\text{lin}}(X_+)} \langle u, x' \rangle = \|X_+ u\|_1$, we can say that

$$\left| \langle u_i, c_x - c_y \rangle \right| + \langle u_i, x' \rangle \leq \|Y_+ u_i\|_1.$$

This implies $\langle u_i, x'' \rangle \leq \|Y_+ u_i\|_1$ which means $x'' \in \gamma_{\text{lin}}(Y_+)$. Thus, $x \in \gamma(Y)$ where the difference between $\gamma_{\text{lin}}(Y_+)$ and $\gamma(Y)$ is the translation c_y .

Let us prove now that inequalities in Equation (5.1) are necessary conditions for inclusion of X into Y .

By Lemma 4 of [GP15], we know that if $\gamma(X) \subseteq \gamma(Y)$ then $\forall u$, $\left| \langle u_i, c_x - c_y \rangle \right| \leq \|Y_+ u\|_1 - \|X_+ u\|_1$. Thus, we can say that if $\gamma(X) \subseteq \gamma(Y)$ then $\forall i = 1, \dots, k$ $\left| \langle u_i, c_x - c_y \rangle \right| \leq \|Y_+ u_i\|_1 - \|X_+ u_i\|_1$ \square

Remark 5.3. For a zonotope of dimension p with n generators, the upper bound on the number of faces, and thus on vectors u_i , is $2 \binom{n}{p-1}$. Thus, the complexity of the inclusion test is $2 \binom{n}{p-1} \times \mathcal{O}(np)$ which improves on the exponential bound of [GP15]. Indeed, the authors proved in [GP15] that $\gamma(X) \subseteq \gamma(Y)$, if and only if Equation (5.1) is satisfied for all $u \in \mathbb{R}^p$. Lemma 5.2 shows that it is sufficient to check the inequality in Equation (5.1) for only a finite (polynomial in p) number of u . It is not necessary to test the Lemma for u and $-u$ because zonotope is center symmetric. When $p = 2$, the u_i correspond to the normals to each generator.

EXAMPLE 5.4 (Computing u) Consider the set of generators $((-3, 4), (5, 2), (2, 0), (1, 1), (3, 5))$ where each generator is a vector specified by (x, y) and x, y being the components. Computing u_i for this set is straightforward: by computing $(-y, x)$ for each vector. Thus, the normal vectors are: $((-4, -3), (-2, 5), (0, 2), (-1, 1), (-5, 3))$.

When $p = 3$, one can still use the cross product to find the normals u_i . However, in higher dimension, we had to resort to linear algebra methods (singular value decomposition (SVD)) for computing these normals. Singular value decomposition (SVD) is a well-known numerical linear algebra technique for factorizing any given matrix M with real entries and of size $m \times n$ into an

$m \times m$ orthogonal matrix U (left singular vectors), a diagonal $m \times n$ matrix Σ (diagonal entries are singular values) and an $n \times n$ orthogonal matrix V (right singular vectors) such that

$$M = U \cdot \Sigma \cdot V^T \quad (5.2)$$

The geometrical intuition of Equation (5.2) is, the matrices U and V being orthogonal, the columns of U form an orthogonal basis of \mathbb{R}^m and the columns of V produce an orthogonal basis of \mathbb{R}^n .

Consider a linearly independent set of generators $((2, -4, 2), (-1, 2, -4), (0, 0, 1), (0, 1, 0))$ or the generator matrix

$$\begin{pmatrix} 2 & -1 & 0 & 0 \\ -4 & 2 & 0 & 1 \\ 2 & -4 & 1 & 0 \end{pmatrix} \quad (5.3)$$

which defines a zonotope in dimension $p = 3$. We want to compute the normals u_i such that each face (dimension $p - 1$) of the zonotope has a vector in u that is normal to it (Lemma 5.2), where a face in this example is defined by a 2-membered subset of the set of generators. So, in total, there will be $\binom{4}{2}$ or 6 faces. Thus, we have to compute 6 vectors where each of them is normal to the corresponding face.

Consider one of the sub-matrices given by

$$\begin{pmatrix} 2 & -1 \\ -4 & 2 \\ 2 & -4 \end{pmatrix} \quad (5.4)$$

In order to find the vector normal to the face formed by the vectors in Equation (5.4), we first compute its SVD:

$$\begin{pmatrix} 2 & -1 \\ -4 & 2 \\ 2 & -4 \end{pmatrix} = U \Sigma V^T = \begin{pmatrix} -0.3374 & 0.2935 & -0.8944 \\ 0.6748 & -0.5871 & -0.4472 \\ -0.6564 & -0.7545 & 0.0000 \end{pmatrix} \begin{pmatrix} 6.3689 & 0 \\ 0 & 2.1066 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} -0.7359 & 0.6771 \\ 0.6771 & 0.7359 \end{pmatrix} \quad (5.5)$$

$$(5.6)$$

We know that, each singular value is linked to each left/right singular vector. Thus, the matrix Σ in Equation (5.5) has two non-zero singular values and a third one which is equal to zero. We also know from the definition of SVD that the matrix U or the left singular vectors span the space of the columns of the matrix whose SVD is computed. It yields an orthogonal basis of \mathbb{R}^m . So, the third vector of the matrix U associated to the third singular value which is zero, will not be included in that plane, but normal to it. Thus, the vector normal to the face defined by the sub-matrix in Equation (5.4) is

$$\begin{pmatrix} -0.8944 \\ -0.4472 \\ 0.0000 \end{pmatrix}. \quad (5.7)$$

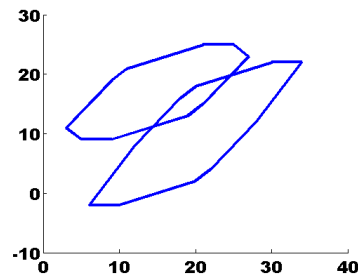


Figure 5.1 – Intersecting case

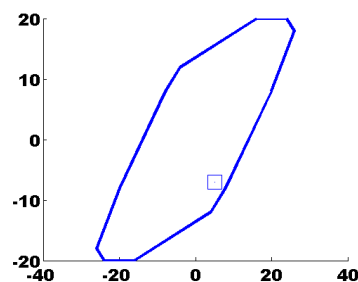


Figure 5.2 – Point inclusion for intersecting case

5.3 INTERSECTION TEST

Recall that in the algorithm it is required to know when two abstract elements intersect to decide if an abstract element is necessary or useful. We proceed as described below. This can in general be done using intersection and a test for non emptiness, but in the case of zonotopes, there is a more direct method, that we develop below. Consider a zonotope \mathfrak{Z}_1 given by its center c_1 and generators g_1, \dots, g_k , and \mathfrak{Z}_2 given by c_2 and h_1, \dots, h_m . As observed in [GNZ03], $\mathfrak{Z}_1 \cap \mathfrak{Z}_2 \neq \emptyset$ if the point $c_1 - c_2$ is included in the zonotope centered at the origin, with generators $g_1, \dots, g_k, h_1, \dots, h_m$. This is solved by finding the values of the noise symbols ε_i , which is a simple linear satisfiability problem.

EXAMPLE 5.5 We present some case studies (Figure 5.1–5.4) where we detect the intersection between the zonotopes, if any, using the above-discussed method. Figure 5.1 shows that two zonotopes intersect and so the point corresponding to the difference between the centers of the zonotopes is entailed inside the new zonotope as illustrated in Figure 5.2. Just opposite is the case for Figure 5.3 as illustrated in Figure 5.4.

5.4 MEET

Recall that zonotopes are closed under linear transformation and under the Minkowski sum, but the set-theoretic intersection of zonotopes is not always a zonotope [GP15]. We know from previous works in [LSA⁺13, TS13] that in order to find an intersection of a zonotope and a polyhedron, we need to

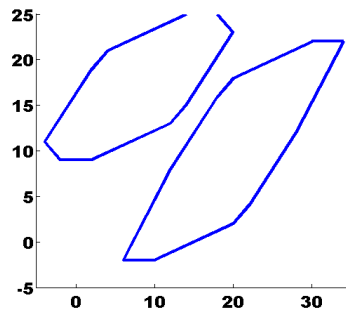


Figure 5.3 – Non-intersecting case

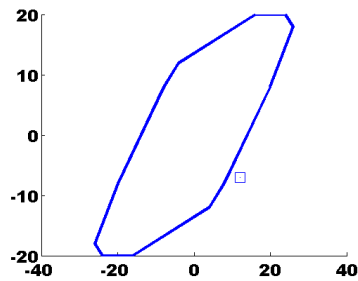


Figure 5.4 – Point inclusion for non-intersecting case

compute the intersection of a zonotope and a half-space. If this intersection is over-approximated by a zonotope then the intersection of a zonotope and a polyhedron can be found by sequential computation of intersection of the zonotope and the half-spaces¹. Similarly, the problem of computing the meet between two zonotopes can be reduced to the problem of computing the meet between a zonotope and a linear space. Several methods have been proposed to compute an over-approximation (e.g., in [GGP10b, GLG08, CZL08]) of the intersection of a zonotope and a linear space.

Girard et al. [GLG08] transformed the problem of computing an over-approximation of the intersection of a zonotope with a hyperplane to the problem of computing several intersections of a 2-dimensional zonotope with a plane by applying projections. However, the over-approximation of the intersection is a polytope for which a tight zonotope over-approximation must be computed. A solution based on zonotope/polytope transformation has been proposed by Althoff et al. in [ASB10] where the polytope is over-approximated using parallelotopes. However, the conversion from half-space to generator representation can be costly. Combastel et al. [Com03, CZL08] introduced an algorithm for computing an approximation of the intersection of a zonotope with a hyperplane based on singular value decomposition. Tabatabaeipour et al. [TS13] proposed that in order to over-approximate the intersection of a zonotope $\mathfrak{Z} = \{x \in \mathbb{R}^n \mid x = c + G\varepsilon, \varepsilon \in [-1, 1]^m\}$ and a half-space $\mathcal{H} = \{x \in \mathbb{R}^n \mid \langle u, x \rangle \leq \gamma\}$ by a zonotope, first the support functions of \mathfrak{Z}

¹A polyhedron is the intersection of a finite number of half-spaces.

in the direction \mathbf{u} and in the direction $-\mathbf{u}$ must be computed. In other words we must compute a zonotope support strip²: $S_3 = \{x \in \mathbb{R}^n \mid q_l \leq \langle \mathbf{u}, x \rangle \leq q_u\}$ where q_l and q_u are given by

$$q_l = \mathbf{u}^T \mathbf{c} - \|\mathbf{G}^T \mathbf{u}\|_1 \quad (5.8)$$

$$q_u = \mathbf{u}^T \mathbf{c} + \|\mathbf{G}^T \mathbf{u}\|_1 \quad (5.9)$$

An intersection between the zonotope \mathfrak{Z} and the half-space \mathcal{H} is possible only under the condition:

$$q_l \leq \gamma \leq q_u.$$

This intersection is bounded in the direction \mathbf{u} by the hyperplane $\{x \in \mathbb{R}^n \mid \langle \mathbf{u}, x \rangle = \gamma\}$ and in the direction $-\mathbf{u}$ by the hyperplane $\{x \in \mathbb{R}^n \mid \langle -\mathbf{u}, x \rangle = q_l\}$. This is true while over-approximating an intersection of a zonotope and a polyhedron. However, between a zonotope-zonotope intersection, zonotope being a center symmetric polyhedron, the value of q_l is $-\gamma$. Thus, the tight zonotope strip bounding the intersection is:

$$S_{\mathfrak{Z} \cap \mathcal{H}} = \{x \in \mathbb{R}^n \mid q_l \leq \langle \mathbf{u}, x \rangle \leq \gamma\} \quad (5.10)$$

where $q_l = -\gamma$. Equation (5.10) is a strip S with $\sigma = \frac{\gamma - q_l}{2}$ and $d = \frac{q_l + \gamma}{2}$ where recall that a strip S is given by $S = \{x \in \mathbb{R}^n \mid |\mathbf{u}^T x - d| \leq \sigma\}$. So, here in a zonotope-zonotope intersection $\sigma = \gamma$ and $d = 0$. The problem finally boils to finding the intersection between a zonotope and a strip. This intersection can be over-approximated by a zonotope $\hat{\mathfrak{Z}} = \{x \in \mathbb{R}^n \mid x = \hat{\mathbf{c}} + \hat{\mathbf{G}}\varepsilon, \varepsilon \in [-1, 1]^{m+1}\}$ which is parameterized by a vector λ that affects the size and bound of intersection and can be computed by minimizing the Frobenius norm of matrix $\hat{\mathbf{G}}$ [ABC05]. Thus, the vector λ , the center $\hat{\mathbf{c}}$ and the generator matrix $\hat{\mathbf{G}}$ can be computed as

$$\lambda = \frac{\mathbf{G}\mathbf{G}^T \mathbf{u}}{\mathbf{u}^T \mathbf{G}\mathbf{G}^T \mathbf{u} + \sigma^2} \quad (5.11)$$

$$\hat{\mathbf{c}} = \mathbf{c} + \lambda(d - \mathbf{u}^T \mathbf{c}) \quad (5.12)$$

$$\hat{\mathbf{G}} = [(\mathbf{I} - \lambda \mathbf{u}^T) \mathbf{G} \quad \sigma \lambda] \quad (5.13)$$

All the above discussed methods focus on computing the intersection of a zonotope and a linear space geometrically. In other words, no information is kept concerning the input/output relationships.

Ghorbal et al. [GGP10b] proposed a method based on functional interpretation of the intersection of a zonotope with a guard. It computes a simple yet sufficiently precise over-approximation by using constrained affine sets: the constraints produced by the tests in a program are interpreted over the noise symbols of the affine forms. Note that, the noise symbols will longer be defined in the range $[-1, 1]$, but rather in the range of the values accounting for the abstraction of the constraints on the noise symbols. Formally, constrained affine sets can be seen as a (logical) product of the abstract domain of

²A strip is denoted as $S = \{x \in \mathbb{R}^n \mid |\mathbf{u}^T x - d| \leq \sigma\}$

affine sets with a lattice structure that abstracts the value of the noise symbols taking into account the constraints induced by tests.

Computing the meet of a zonotope with another one as the sequence of meet of the zonotope with the faces of the other can be imprecise, as the meet with linear space is an over-approximation, and imprecision will accumulate quickly. Moreover, it depends on the number of faces. Hence, the need for a zonotope meet that can take into account all faces of the second argument at once.

There are methods which directly focus on meet between zonotopes by set representations based on collection of sets. For instance, Althoff et al. [AK11] introduced zonotope bundles, defined as the intersection of a set of zonotopes. Consider a finite set of zonotopes \mathfrak{Z} , a zonotope bundle is $\mathfrak{Z}^\cap = \{\bigcap_{i=1}^s \mathfrak{z}_i \mid \mathfrak{z}_i \in \mathfrak{Z}\}$, i.e., the intersection of zonotopes \mathfrak{z}_i where $\mathfrak{z} = (c, g^{(1)}, \dots, g^{(p)})$. A bundle $\{\mathfrak{z}_1, \dots, \mathfrak{z}_s\}$ allows to symbolically represent a polytope $\mathfrak{Z}^\cap = \{\mathfrak{z}_1, \dots, \mathfrak{z}_s\}^\cap$. Note that the intersection is not computed explicitly. Rather the zonotopes are stored in a list and all operations are performed on individual zonotopes. A similar method has been proposed in [DDP16] for parallelotopes. These methods can be accurate, but the related cost increases with the number of sets required, which can be large.

Here, we develop a simple meet operation based on the following observation. Let \mathfrak{Z}_1 (resp. \mathfrak{Z}_2) be a zonotope represented by matrix M_1 (resp. M_2) and let x be a point in the intersection of \mathfrak{Z}_1 and \mathfrak{Z}_2 . Then, there exists $e \in [-1, 1]^p$ (resp. $e' \in [-1, 1]^p$) such that $x = M_1^T \begin{pmatrix} 1 \\ e \end{pmatrix}$ and $x = M_2^T \begin{pmatrix} 1 \\ e' \end{pmatrix}$. For any $\alpha \in [0, 1]$, trivially, $x = \alpha x + (1 - \alpha)x$, therefore:

$$\mathfrak{Z}_1 \cap \mathfrak{Z}_2 \subseteq \left\{ \alpha M_1^T \begin{pmatrix} 1 \\ e \end{pmatrix} + (1 - \alpha) M_2^T \begin{pmatrix} 1 \\ e' \end{pmatrix}, \|e\|_\infty \leq 1, \|e'\|_\infty \leq 1 \right\} \quad (5.14)$$

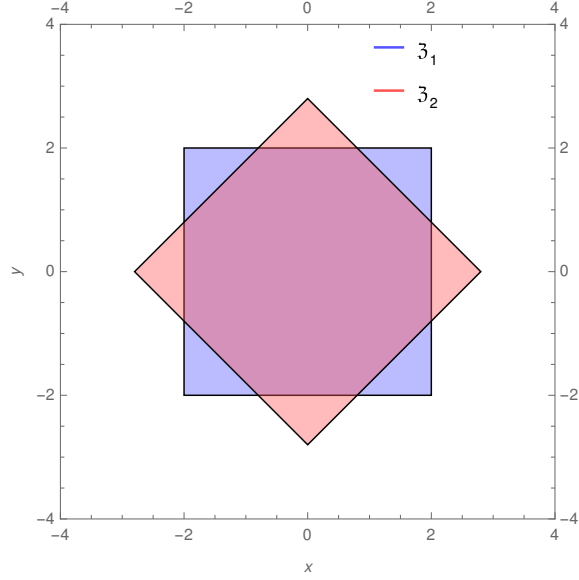
The right hand side of the inclusion above is the zonotope obtained as the Minkowski sum of zonotope \mathfrak{Z}_1 (scaled by coefficient α) with zonotope \mathfrak{Z}_2 (scaled by coefficient $1 - \alpha$), up to some translation: $\alpha c_1 + (1 - \alpha)c_2$ where c_1 and c_2 are the centers of zonotopes \mathfrak{Z}_1 and \mathfrak{Z}_2 . If we substitute in Equation (5.14) $\alpha = 0$, we have \mathfrak{Z}_2 else if we substitute $\alpha = 1$, we obtain \mathfrak{Z}_1 and any value between 0, 1 would result in a zonotope intervening the two intersecting zonotopes and including the actual intersection. Taking the barycenters (with weight α) of zonotopes \mathfrak{Z}_1 and \mathfrak{Z}_2 will always give something containing the intersection. This is because a point in the intersection of \mathfrak{Z}_1 and \mathfrak{Z}_2 (seen as functions of the ε), satisfies $\mathfrak{Z}_1(\varepsilon_i) = \mathfrak{Z}_2(\varepsilon_j)$ only for some values of $\varepsilon_i, \varepsilon_j$. So, for these values, we have

$$\alpha \times \mathfrak{Z}_1(\varepsilon_i) + (1 - \alpha) \times \mathfrak{Z}_2(\varepsilon_j) = \mathfrak{Z}_1(\varepsilon_i) = \mathfrak{Z}_2(\varepsilon_j) \quad (5.15)$$

which is this point in the (set theoretic) intersection. Thus the right side in Equation (5.14) is always going to include the intersection.

EXAMPLE 5.6 Consider the example of Figure 2.6. We set \mathfrak{Z}_1 to be the initial box $(x, y) \in S_0 = [-2, 2]^2$ that can be abstracted using zonotopes as $\mathfrak{Z}_1 = \begin{pmatrix} 2\varepsilon_1 & 2\varepsilon_2 \end{pmatrix}^T$. Consider now \mathfrak{Z}_2 to be the effect of the body loop on $\mathfrak{Z}_2 = \mathfrak{Z}_1$:

$$F^\#(S_0) = \begin{pmatrix} 1.4\varepsilon_1 & 1.4\varepsilon_2 \\ 1.4\varepsilon_1 & -1.4\varepsilon_2 \end{pmatrix}.$$


 Figure 5.5 – The zonotope concretization of S_0 and $F^\sharp(S_0)$

The zonotopes \mathfrak{Z}_1 and \mathfrak{Z}_2 are shown in Figure 5.5. In the parametrization of S_0 and $F^\sharp(S_0)$ by the same noise symbols ε_1 and ε_2 , we consider each point of S_0 and its image F^\sharp . To get a geometric intersection of the two zonotopes, we need to parameterize them with different noise symbols.

First, we will illustrate how the intersection $S_0 \cap F^\sharp(S_0)$ can be over-approximated by a sequential computation of intersection of the zonotope \mathfrak{Z}_1 and the faces of \mathfrak{Z}_2 and later we will discuss how to over-approximate this intersection using our proposed method. As discussed earlier the intersection $\mathfrak{Z}_1 \cap \mathfrak{Z}_2$ can be reduced to the problem to finding the intersection of \mathfrak{Z}_1 and the strips whose meet is the zonotope \mathfrak{Z}_2 .

Initially, we need to compute the half-spaces $(\mathcal{H}_i = \{x \in \mathbb{R}^2 \mid \langle u, x \rangle \leq \gamma\})$ whose intersection is the zonotope \mathfrak{Z}_2 . We already know from Section 5.2 and Definition 2.34 that computing γ requires to find the support function in the direction u . Moreover, recall that each u_i is normal to the faces of \mathfrak{Z}_2 . Thus, for \mathfrak{Z}_2 , the u_i 's are

$$u_1 = \begin{pmatrix} -1.4 \\ 1.4 \end{pmatrix} \quad (5.16)$$

$$u_2 = \begin{pmatrix} 1.4 \\ 1.4 \end{pmatrix} \quad (5.17)$$

As \mathfrak{Z}_2 is centered on zero, the value of γ in the direction u_1 can be simply computed as $\|Au_1\|_1 = 3.92$ where A^T is the generator matrix of \mathfrak{Z}_2 , i.e., $\begin{pmatrix} 1.4 & 1.4 \\ 1.4 & -1.4 \end{pmatrix}$.

The next step is to find the tight supporting strip bounding the intersection. As already discussed in a zonotope-zonotope intersection, this strip is the support function of \mathfrak{Z}_2 in the direction u_1 and $-u_1$. Thus, the values of q_l

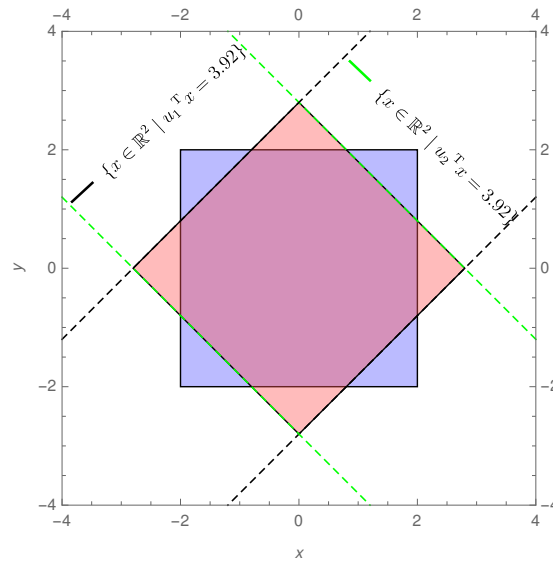


Figure 5.6 – The hyperplanes with which the intersection of \mathfrak{Z}_1 will be computed

and q_u in Equations (5.8) and (5.9) are equal to -3.92 and 3.92 respectively. Subsequently, the values of σ and d are 3.92 and 0 . The hyperplanes in the direction u_1 and u_2 with which we will compute the intersection of \mathfrak{Z}_1 are shown in Figure 5.6.

Now, we compute the value of λ and accordingly the center and the generator matrix of the zonotope over-approximation of the intersection $\mathfrak{Z}_1 \cap \mathcal{H}_1$ using Equations (5.12)-(5.11). This zonotope over-approximation denoted as \mathfrak{Z}_3 and shown in Figure 5.7 is characterized by the generator matrix given by

$$\begin{pmatrix} 1.4949 & 0.5051 & -0.7071 \\ 0.5051 & 1.4949 & 0.7071 \end{pmatrix} \quad (5.18)$$

and is centered on zero.

Following the above steps, now in the direction u_2 (Equation (5.17)), the tight supporting strip and the over-approximation of its intersection with the zonotope \mathfrak{Z}_3 whose generator matrix is in Equation (5.18) is computed. Let's denote this zonotope over-approximation as \mathfrak{Z}_4 which is shown in Figure 5.8. It is centered on zero and its generator matrix is given by

$$\begin{pmatrix} 0.9898 & 0 & -0.7071 & 0.7071 \\ 0 & 0.9898 & 0.7071 & 0.7071 \end{pmatrix}.$$

EXAMPLE 5.7 In the previous example, we illustrated the computation of meet $S_0 \cap F^\sharp(S_0)$ by computing sequentially the intersection of S_0 and the faces of $F^\sharp(S_0)$. Now, we will find the over-approximation of the intersection $S_0 \cap F^\sharp(S_0)$ by taking into account all faces of $F^\sharp(S_0)$ at once. Thus, the intersection will

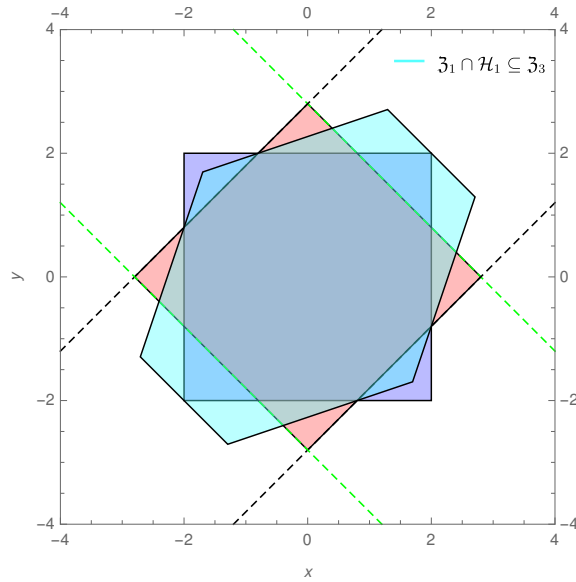


Figure 5.7 – The over-approximation of the intersection of \mathfrak{Z}_1 and the half-space in the direction u_1

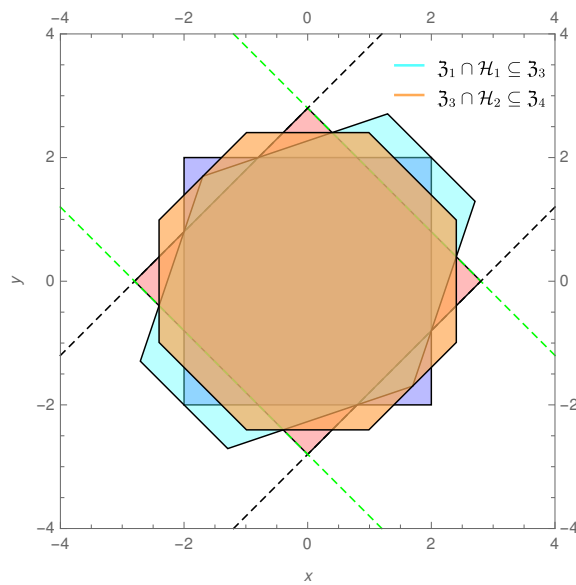


Figure 5.8 – The over-approximation of the intersection of \mathfrak{Z}_3 and the half-space in the direction u_2

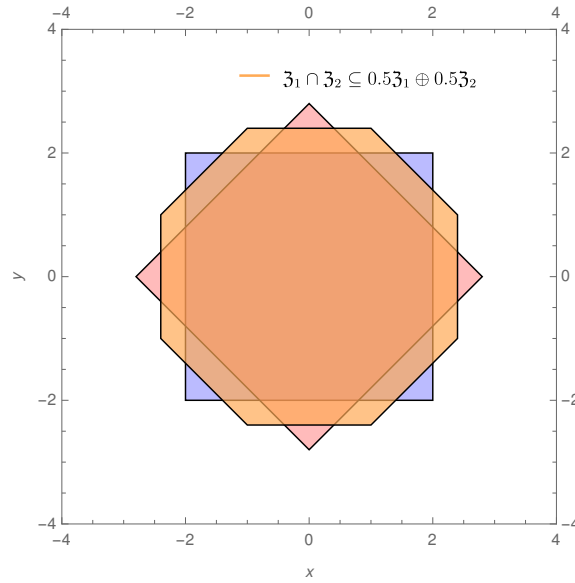


Figure 5.9 – The zotopic over-approximation of the intersection $\mathfrak{Z}_1 \cap \mathfrak{Z}_2$

be over-approximated by

$$\alpha \begin{pmatrix} 2\varepsilon_1 \\ 2\varepsilon_2 \end{pmatrix} + (1 - \alpha) \begin{pmatrix} 1.4 \times (\varepsilon_3 + \varepsilon_4) \\ 1.4 \times (\varepsilon_3 - \varepsilon_4) \end{pmatrix}.$$

We then choose α so as to minimize the distance $\|A_+ u_i\|_1$ (which can be solved by linear programming), for u_i the normals to the faces of zonotopes \mathfrak{Z}_1 and \mathfrak{Z}_2 (in dimension 2, those u_i are directly given by the generators). Here, we obtain α equal to 0.5. Thus, the zonotope over-approximating the intersection is shown in Figure 5.9. This approach only provides an over-approximation of the intersection. Therefore, when using the Algorithm 5.1, we will not apply the tightening in Equation (4.2) at each step, but only at the initial step. We comment on this in the chapter dedicated to experiments. Future work can include using a tighter intersection operator, for instance relying on zonotopes tilings.

Remark 5.8. The volume of the zonotopes shown in Figure 5.8 and 5.9 which over-approximate the intersection $S_0 \cap F^\sharp(S_0)$ is 19.2.

5.5 SIZE

Recall that a key requirement of the Algorithm 4.1 is to provide it with the size information of an abstract element below which it restrains from splitting the abstract elements. Thus, the size of a zonotope can be computed in the following way.

Consider a zonotope given by matrix $X \in \mathcal{M}(n_X + 1, p)$. Let $u = \{u_1, \dots, u_k\}$ be vectors in \mathbb{R}^p such that each face in $\gamma(X)$ has a vector in u that is normal to it. Similar to boxes (Equation (3.2)), the size of a zonotope can be defined as the maximum width among all variables, i.e.:

$$\max\{H_{u_i} \mid i \in \{1, \dots, k\}\} \quad (5.19)$$

where H_{u_i} is the region between the two hyperplanes orthogonal to the lines in the direction u_i (shown in Figure 2.11) such that $\gamma_{\text{tin}}(X_+) \subseteq H_{u_i}$. The width of this region can be computed as $2\|X_+u\|_1$.

Remark 5.9. The algorithm never discards a zonotope if its size-parameter ϵ_s reaches less than the cut-off value. By that moment, if the algorithm has not discovered inductive invariant then it terminates with a failure. Later the algorithm is reapplied with lower cut-off values for ϵ_s until it finds the inductive invariant. So, within the implementation of the algorithm we do not require to compute the size of a zonotope precisely by Equation (5.19). Rather, we choose to approximate the size of a zonotope as the size of its bounding box, which is far easier to compute.

Remark 5.10. The algorithm from Section 4.2 needs to compute some information about abstract elements, such as their coverage and whether the abstract elements are benign or useful.

Recall that the coverage information of an abstract element required computing volume of the abstract elements, which for zonotopes can be fairly expensive. We adapt the data structure to develop a coverage metric to decide whether to split an abstract element or not, instead of computing their volumes.

We also demonstrate with the help of examples, why resorting to such space-partitioning data structure technique constrained to have us design a splitting method based on tiling a zonotope by parallelotopes.

5.6 VOLUME OF A ZONOTOPE.

Recall that the Algorithm 5.1 needs to compute coverage information to find which abstract element to be treated first. Then it is used to decide if an abstract element is to be discarded (i.e., it is doomed) or to be split. The coverage defined in Equation (4.1) or Equation (4.4) requires computing volume of the abstract elements. Computing volume of a zonotope can be fairly expensive, and we see below why.

Consider a zonotope $\mathfrak{Z}(V)$ in p -dimension for any vector configuration $V = \{v_1, \dots, v_n\}$. Recall that tiling of the zonotope $\mathfrak{Z}(V)$ is an arrangement of tiles or sub-zonotopes (a sub-zonotope is a zonotope generated by a subset $\{v_{i_1}, \dots, v_{i_k}\}$ of V) such that intersection of any two such tiles is a face of both the tiles and the union of all the tiles equals $\mathfrak{Z}(V)$. We can have an equivalent interpretation for tiling with respect to the volume of the zonotope.

Tiling of the zonotope $\mathfrak{Z}(V)$ is an arrangement of tiles such that intersection of any two tiles has zero volume and the total volume of the set of zonotopal tiles is equal to the volume of $\mathfrak{Z}(V)$ [She74, RGZ94]. Considering a tiling by parallelotopes, one can compute the volume of the zonotope from the total sum of the volume of all the parallelotopic tiles. Below we define how the volume can be computed.

DEFINITION 5.11 (See [RGZ94, GK10] for the proof) The volume of a zonotope $\mathfrak{Z}(V)$ defined by a set of n vectors $V = \{v_1, \dots, v_n\}$ in p -dimension is given by

$$2^p \cdot \sum |\det(v_{i_1}, \dots, v_{i_p})| \quad (5.20)$$

where the summation is over all p -membered subsets $\{i_1, \dots, i_p\}$ of $\{1, \dots, n\}$ and each p -membered subset is a p -dimensional parallelotope.

EXAMPLE 5.12 Consider a set of vectors $V = ((-3, 4), (5, 2), (2, 0), (1, 1), (3, 5))$ which defines a zonotope in Figure 2.10. Recall that with our tiling algorithm we could enumerate all the $\binom{5}{2}$ parallelotopic tiles whose total volume can be computed as

$$\begin{aligned} \text{vol}(Z(V)) = & \det \begin{pmatrix} -3 & 5 \\ -4 & 2 \end{pmatrix} + \det \begin{pmatrix} -3 & 2 \\ -4 & 0 \end{pmatrix} + \det \begin{pmatrix} -3 & 1 \\ -4 & 1 \end{pmatrix} + \\ & \det \begin{pmatrix} -3 & 3 \\ -4 & 5 \end{pmatrix} + \det \begin{pmatrix} 5 & 2 \\ 2 & 0 \end{pmatrix} + \det \begin{pmatrix} 5 & 1 \\ 2 & 1 \end{pmatrix} + \\ & \det \begin{pmatrix} 5 & 3 \\ 2 & 5 \end{pmatrix} + \det \begin{pmatrix} 2 & 1 \\ 0 & 1 \end{pmatrix} + \det \begin{pmatrix} 2 & 3 \\ 0 & 5 \end{pmatrix} + \det \begin{pmatrix} 1 & 3 \\ 1 & 5 \end{pmatrix} \end{aligned}$$

Remark 5.13. Computing all the determinants of the sub-matrices of a zonotope can be fairly expensive. Also, the coverage calculation is subject to floating-point round-off error, so it will not be safe to use Equation (4.4) to compute the coverage and determine whether a zonotope is benign or not. Therefore, it is critical that we rely on some other measure to compute the coverage.

5.7 COVERAGE METRIC

Note that the implementation of the Algorithm 4.1 presented in Section 4.3 computed volume of the bounding boxes of the abstract elements to calculate the coverage information. Recall that to compute and maintain the coverage information efficiently, the algorithm depends on the data structure introduced in Section 4.3.6 which allows to consider all abstract elements in the volume computation without scanning G^\sharp entirely after each operation.

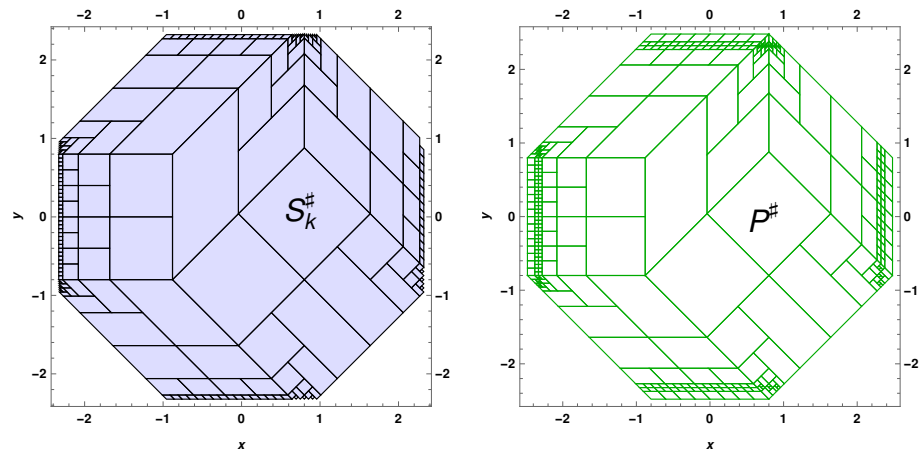
Here we will introduce a cheaper coverage metric which does not rely on any kind of volume computation except the data structure. First, we will recall quickly about the data structure before introducing the coverage metric.

As discussed in Section 4.3.6 the algorithm is enhanced with a data structure based on partitioning which allows maintaining efficiently the coverage information and the state (whether they are benign or useful) of the abstract elements after any test or operation. For instance when the algorithm discards an abstract element based on any test or splits, it may not only modify the coverage of that element, but also of other abstract elements. This data structure allows maintaining two sets of zonotopes: B^\sharp and G^\sharp , throughout the operation of the algorithm. We know that G^\sharp is the set of zonotopes which will divide the inductive invariant. B^\sharp is the set of zonotopes split in the similar manner as G^\sharp . The only difference between the two is that unlike G^\sharp where (sub-)zonotopes can be discarded following the outcome of any test, B^\sharp always covers the whole (tightened) initial invariant space. Therefore, a zonotope in B^\sharp is matched by at most one zonotope from G^\sharp .

A *contents-of* function $\text{cnt} : B^\sharp \rightarrow (G^\sharp \cup \{\emptyset\})$ is maintained indicating which zonotope of G^\sharp , if any, is contained in the partition B^\sharp .

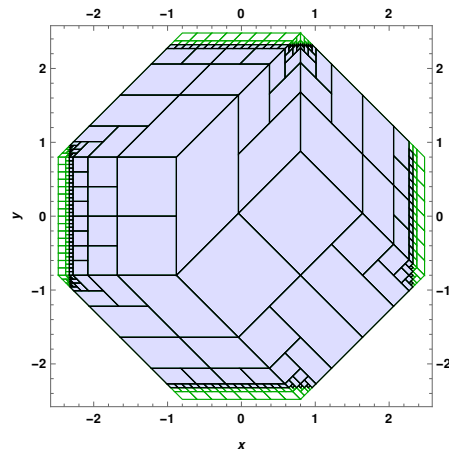
A map $\text{post} : G^\sharp \rightarrow \mathcal{P}(B^\sharp)$ as presented in Equation (4.3) is maintained to indicate which parts of B^\sharp intersect the image of a zonotope $S_k^\sharp \in G^\sharp$. The map

5. ZONOTOPES AND CONSTRAINT SOLVING

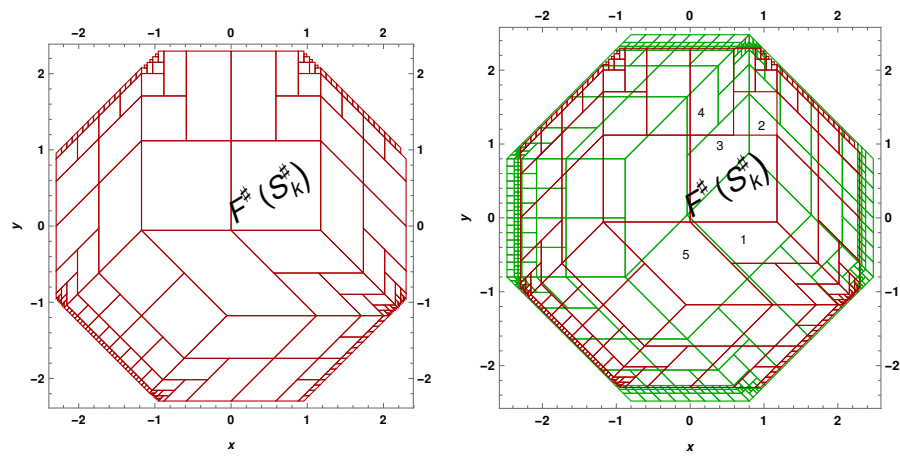


(a) Set of zonotopes $G^\#$ partitioning the inductive invariant

(b) Partitions $B^\#$



(c) Contents-of function



(d) The image of the set of zonotopes $G^\#$

(e) The map post

post is further used to compute coverage and also to decide whether a box is benign or not.

EXAMPLE 5.14 Consider the program in Figure 2.6 for which the set of zonotopes G^\sharp partitioning the inductive invariant is shown in Figure 5.10a. Figure 5.10b illustrates the set of partitions B^\sharp which covers the whole invariant space. We superimpose the Figures 5.10a and 5.10b in Figure 5.10c to demonstrate the notion of *contents-of* function. Consider the zonotope partition noted P^\sharp in Figure 5.10b. The *contents-of* function corresponding to P^\sharp (i.e., $\text{cnt}(P^\sharp)$) will return the parallelotope S_k^\sharp marked in Figure 5.10a. The image of the zonotopes in Figure 5.10a by a loop iteration is shown in Figure 5.10d. Let's superimpose the Figures 5.10b and 5.10d in Figure 5.10e that shows which parts of B^\sharp intersect the image of a zonotope $S_k^\sharp \in G^\sharp$. That is the map *post*.

Thanks to the partitioning data structure that we will not be needing the coverage measure to decide whether a zonotope is benign. Rather, it can be a simple test that we will discuss later in this chapter. Still, the algorithm requires some form of coverage measure to decide which zonotope to further split. We actually only need some approximation (with bounded ratio) of the coverage for deciding to split elements. We use here a heuristic measure instead of computing volume.

We count the number of zonotopes $P^\sharp \in B^\sharp$ which intersect $F^\sharp(S_k^\sharp)$, i.e.,

$$\#\{P^\sharp \mid P^\sharp \in \text{post}(S_k^\sharp)\}.$$

Then, among these zonotopes, we count the ones for which $\text{cnt}(P^\sharp) \neq \emptyset$, i.e., we compute

$$\#\{P^\sharp \mid \text{cnt}(P^\sharp) \neq \emptyset, P^\sharp \in \text{post}(S_k^\sharp)\}.$$

Our heuristic measure is thus:

$$\text{coverage}(S_k^\sharp) := \frac{\#\{P^\sharp \mid \text{cnt}(P^\sharp) \neq \emptyset, P^\sharp \in \text{post}(S_k^\sharp)\}}{\#\{P^\sharp \mid P^\sharp \in \text{post}(S_k^\sharp)\}} \quad (5.21)$$

EXAMPLE 5.15 Consider the zonotope S_k^\sharp shown in Figure 5.10a (the parallelotope labeled as S_k^\sharp), its image under F^\sharp is the zonotope labeled as $F^\sharp(S_k^\sharp)$ in Figure 5.10d and the partition $P^\sharp \in B^\sharp$ containing S_k^\sharp . The sub-parallelotopes numbers 1, 2, up to 5 in Figure 5.10e are thus in $\text{post}(S_k^\sharp)$. All the partitions (1, 2, up to 5) in B^\sharp contain one zonotope each from G^\sharp .

Thus, we compute the coverage of S_k^\sharp by Equation (5.21) and $\text{coverage}(S_k^\sharp) = 1$. Thus, S_k^\sharp need not be split further.

We extend further the coverage metric to design a test for deciding if a zonotope is benign or not.

5.7.1 Test for benign.

In the above example, although $\text{coverage}(S_k^\sharp) = 1$, we cannot insist that S_k^\sharp is benign because the coverage metric does not guarantee if the image of S_k^\sharp under F^\sharp is entailed inside the candidate invariant. For instance the image $F^\sharp(S_k^\sharp)$ may intersect with partitions whose *contents-of* function (*cnt*)

is not empty but a part of the image can be outside the candidate invariant. Therefore, additionally we check if $F^\sharp(S_k^\sharp)$ is included in the initial target invariant T^\sharp . Thus, formally we can write,

$$S_k^\sharp \text{ is benign} \iff \forall P^\sharp \in \text{post}(S_k^\sharp) : \text{cnt}(P^\sharp) \neq \emptyset \wedge F^\sharp(S_k^\sharp) \subseteq T^\sharp \quad (5.22)$$

Remark 5.16. In practice, in Equation (5.22) we first verify, if $\forall P^\sharp \in \text{post}(S_k^\sharp) : \text{cnt}(P^\sharp) \neq \emptyset$ satisfies because it is computationally less expensive (due to the partitioning data structure) compared to the inclusion test, $F^\sharp(S_k^\sharp) \subseteq T^\sharp$.

5.8 SPLITTING

5.8.1 Splitting with overlap

A key requirement for the Algorithm 5.1 is the ability to split a zonotope into smaller zonotopes. For this, it is useful to view a zonotope as the affine projection of an n -dimensional unit cube (n being the number of noise symbols) onto a p -dimensional space (p being the number of program variable), and perform the split operation on the unit cube to define a splitting operation on the resulting zonotope. Thus, a zonotope $\mathfrak{Z} = (c, g_1, \dots, g_n)$ can be bisected into two sub-zonotopes \mathfrak{Z}_1 and \mathfrak{Z}_2 by splitting the j^{th} generator of \mathfrak{Z} such that $\mathfrak{Z}_1 \cup \mathfrak{Z}_2 = \mathfrak{Z}$. The two sub-zonotopes can be determined as:

$$\mathfrak{Z}_1 = (c - \frac{1}{2}g_j, g_1, \dots, g_{j-1}, \frac{1}{2}g_j, g_{j+1}, \dots, g_n) \quad (5.23)$$

and

$$\mathfrak{Z}_2 = (c + \frac{1}{2}g_j, g_1, \dots, g_{j-1}, \frac{1}{2}g_j, g_{j+1}, \dots, g_n) \quad (5.24)$$

where the index j is denoted to be:

$$j = \{i \mid \max(\|g_i\|_1), i = 1, \dots, n\} \quad (5.25)$$

EXAMPLE 5.17 Consider the following affine forms:

$$\hat{x} = 20 - 3\varepsilon_1 + 5\varepsilon_2 + 2\varepsilon_3 + 1\varepsilon_4 + 3\varepsilon_5 \quad (5.26)$$

$$\hat{y} = 10 - 4\varepsilon_1 + 2\varepsilon_2 + 1\varepsilon_4 + 5\varepsilon_5 \quad (5.27)$$

as earlier whose concretization is a zonotope in Figure 2.10. Taking this zonotope as an example, the above-discussed bisection method returns two sub-zonotopes, which are shown in Figure 5.11 and their affine forms are

$$\hat{x} = 18.5 - 3\varepsilon_1 + 5\varepsilon_2 + 2\varepsilon_3 + 1\varepsilon_4 + 1.5\varepsilon_5 \quad (5.28)$$

$$\hat{y} = 7.5 - 4\varepsilon_1 + 2\varepsilon_2 + 1\varepsilon_4 + 2.5\varepsilon_5 \quad (5.29)$$

and

$$\hat{x} = 21.5 - 3\varepsilon_1 + 5\varepsilon_2 + 2\varepsilon_3 + 1\varepsilon_4 + 1.5\varepsilon_5 \quad (5.30)$$

$$\hat{y} = 12.5 - 4\varepsilon_1 + 2\varepsilon_2 + 1\varepsilon_4 + 2.5\varepsilon_5 \quad (5.31)$$

respectively.

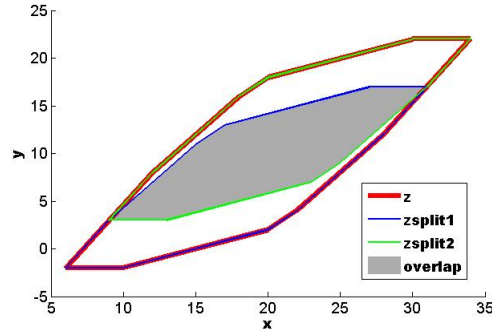


Figure 5.11 – Sub-zonotopes obtained after splitting

Splitting with overlap is simple, close to that on boxes (as we split the box which the zonotope is a projection of). It also helps in keeping the same kind of shape and the direction of the faces fixed. However, we had to resort to a different splitting technique (we will see later) because of the choices we made about the data structure of the search algorithm. These choices were meant for an efficient implementation of the algorithm. The data structure relied on the fact that the zonotopes must not overlap. Maintaining a similar data structure for the resulting algorithm with the overlapping zonotopes was not feasible. We will provide more details about this in the following.

In order to give a short intuition on the resulting algorithm while splitting a zonotope along one of its line segments as in Equation (5.23) and (5.24), we provide below an example.

EXAMPLE 5.18 We developed a prototype of the Algorithm 5.1 in MATLAB using the CORA Toolbox [Alt15]. As data structures, we used a simple binary search tree. Note that for this prototype, the zonotopes were bisected along the line segment generator with maximum length, and hence always producing two sub-zonotopes. We used the coverage information to split, discard and also to decide whether a zonotope is benign. So, for computing $\text{vol}(F^\sharp(S_k^\sharp))$ and eventually the coverage by Equation (4.1) we used the exact volume calculation of zonotopes provided in Equation (5.20). For computing the volume: $\text{vol}(F^\sharp(S_k^\sharp) \cap (\cup_i S_i^\sharp))$, first we computed the intersection: $F^\sharp(S_k^\sharp) \cap S_i^\sharp$. Since we had to compute an exact coverage measure, we calculated the intersection by set representations based on collection of sets or zonotope bundles [AK11]. After constructing the zonotope bundle, the volume is computed by converting it to a polytope and using a volume computation of polytopes.

Consider the example in Figure 2.6 for which we will use the Algorithm 5.1 with the above setting for finding inductive invariant. The zonotope in red shown in Figure 5.12 is the target invariant about to be strengthened into an inductive invariant. Its image under F^\sharp is the zonotope in black. Recall that the splitting approach based on bisecting the j^{th} noise symbol produces overlapping zonotopes. Thus, splitting the zonotope in red produces two sub-zonotopes, one in blue and the other in green. Lets compute the coverage information of the sub-zonotope in blue. Recall that for computing coverage

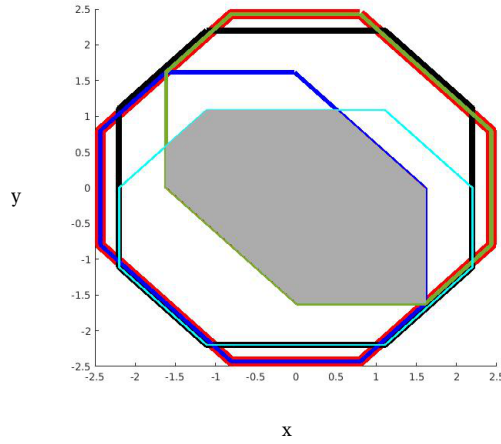


Figure 5.12 – Computing the coverage measure in case of overlapping zonotopes

we must find how much the image (the zonotope in cyan color) of the zonotope in blue lies in the candidate invariant. While doing so, the volume of the region of overlap between the image (the zonotope in cyan) and the polytope representing the overlay between the split zonotopes (blue and green) must be removed. This region of overlap is the gray color area shown in Figure 5.12. To do so, first a zonotope bundle is constructed for the overlap between the two split zonotopes and then form another zonotope bundle accounting for the overlap between the image and the old zonotope bundle. The volume of this new zonotope bundle is subtracted from the total volume.

This is just a simple case study with two split zonotopes, but as the algorithm iterates further this overlap increases and leads to inefficiency. For instance, we ran the algorithm for **14354.702** secs producing 12 zonotopes in G^\sharp shown in Figure 5.13 and yet could not find the inductive invariant. On the contrary, if the same algorithm is run but this time with the target invariant abstracted using zonotope abstract domain as a box: $(2\varepsilon_1 \ 2\varepsilon_2)^T$, we managed to infer an inductive invariant in **45.274** secs with the zonotopes shown in Figure 5.14-5.15. Although we used the same splitting strategy, since the target invariant was a box this time, the splitting did not produce any overlap. Accordingly, we did a time profiling for the algorithm while it produced overlapping zonotopes and it turns out that from **14354.702** secs the algorithm spent **14204.930** secs on calculating: $\sum_i \text{vol}(F^\sharp(S_k^\sharp) \cap (\cup_i S_i^\sharp))$ for coverage measure. Thus, a simple data structure like a binary search tree is not sufficient for an efficient implementation of the algorithm and maintain overlapping zonotopes.

5.8.2 Effect of partitioning on splitting

The whole notion of partitioning is based on the fact that we maintain a set of zonotopes (partitions) B^\sharp that, similarly to G^\sharp , do not overlap. Thanks to tiling a zonotope by parallelotopes that we could maintain this data structure.

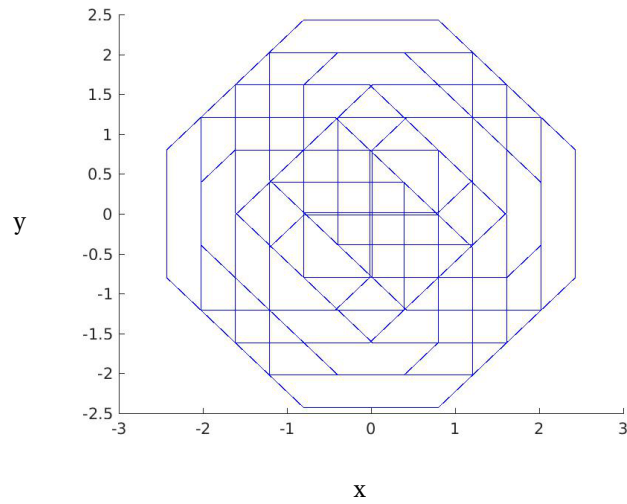


Figure 5.13 – Split zonotopes for example 1 illustrating the issue with conventional coverage measure

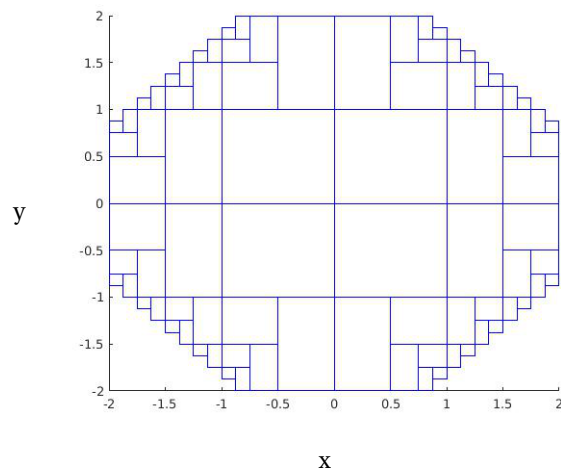


Figure 5.14 – Inductive invariant for the program in Figure 2.6 with the target invariant being the box $[-2, 2]$ abstracted using zonotopes

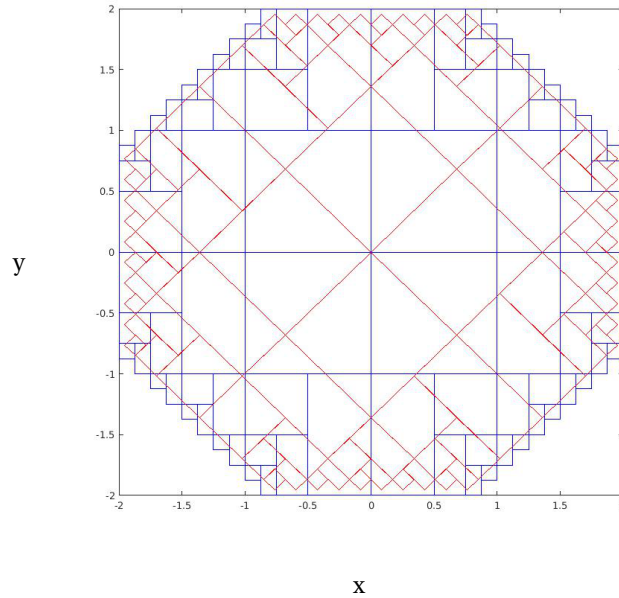


Figure 5.15 – In red, the image of the zonotopes in Figure 5.14 by a loop iteration in the zonotope abstract domain and superposition of both showing that Figure 5.14 is inductive

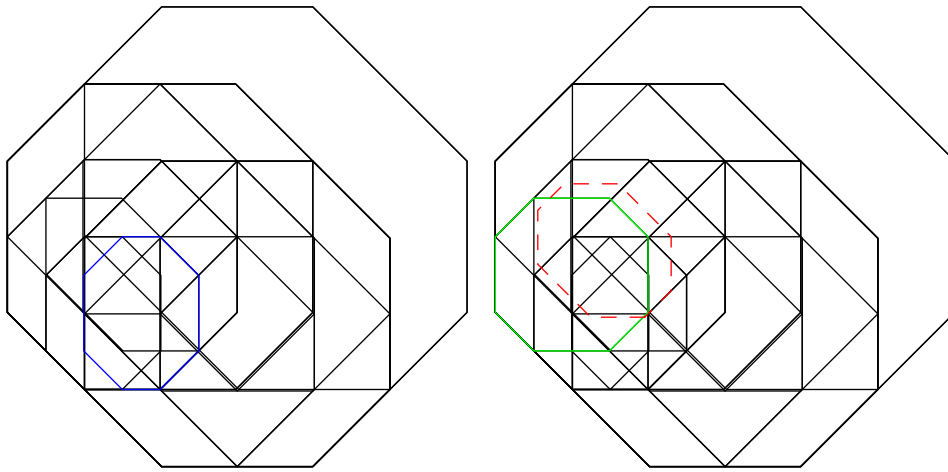
Splitting a zonotope by overlap and maintaining the partitioning is possible but complicated. We exemplify this below.

EXAMPLE 5.19 Consider the Figure 5.16a that illustrates a collection of zonotopes that are being split by overlap. The partitions of these zonotopes are shown in Figure 5.16b that are split in the same way as the set in Figure 5.16a, but never discarded even though its *contents-of* function (*cnt*) is discarded as a result of any test.

Consider the zonotope in Figure 5.16a with the edgeform color in blue. We will denote this zonotope as S_k^\sharp . Its image by F^\sharp is the zonotope with a dashed edgeform in red color, shown among the partitions in Figure 5.16b. Now, checking if S_k^\sharp is benign by the test in Equation (5.22) will return “no” because among the partitions $P^\sharp \in \text{post}(S_k^\sharp)$, there is one partition (the one with edgeform color in green) whose *contents-of* function is equal to zero. However, S_k^\sharp here is actually benign because its image $F^\sharp(S_k^\sharp)$ is covered by the set of zonotopes shown in Figure 5.16a.

Remark 5.20. From the above experiments (Examples 5.18 and 5.19, we can infer that fitting the splitting via overlap approach within the framework of Algorithm 5.1 is feasible, but would require some careful re-design of the overall algorithm, with more complex data-structures. It is realizable but what is gained by a simpler splitting algorithm is lost by a more complicated fixed-point solving algorithm.

Remark 5.21. Another, natural way to split zonotopes, without overlapping



(a) Set of zonotopes, that are being split by overlap (b) Partitions P^\sharp for the set of zonotopes shown in Figure 5.16a

Figure 5.16 – Partitioning and its effect on splitting by overlap

this time, is to use the property that zonotopes can be tiled, using generally more than 2 sub-zonotopes. These tiles are more precisely parallelotopes, as we describe below.

5.8.3 Splitting zonotopes by tiling

Zonotopes are Minkowski sums of segments, and tiling of a zonotope \mathfrak{Z} are polytopal subdivisions of \mathfrak{Z} into smaller zonotopes. These smaller zonotopes can be obtained by the solution of a tiling problem, that we define below.

DEFINITION 5.22 (Tiling problem.) A tiling problem is defined by a finite set of tiles T , called the prototiles, and a polygon P . A solution to this problem is a tiling: an arrangement of translated copies of prototiles which covers exactly P with no gap and no overlap.

The study of these problems is an important topic in mathematics which had its inception from more recreational point of view and now it has connections with combinatorics, topology. We are concerned here with tilings of p -dimensional zonotopes with p -dimensional or p -parallelotopic tiles. Trivially, a zonotope decomposes into zonotopes and hence into parallelotopes (by a theorem of Shephard and McMullen [She74, McM76]). In the literature associated with tiling, it is said, if a zonotopal tiling is only composed of parallelotopes then the tiling is *tight*.

CONCEPTS AND DEFINITIONS Consider a zonotope $\mathfrak{Z}(V)$ on a set of generators $V = (v_1, \dots, v_n) \in \mathbb{R}^{p,n}$. A zonotopal tiling of $\mathfrak{Z}(V)$ is a set of tiles $\{Z_1, Z_2, \dots, Z_M\}$ constructed from the vectors in V such that $\bigcup_{i=1}^M Z_i = \mathfrak{Z}(V)$.

Provided with a sign vector $\sigma \in \{+, -, 0\}^n$ we can define a zonotope:

$$\mathfrak{Z}_\sigma := \sum_{i \in \sigma^0} [-v_i, +v_i] + \sum_{i \in \sigma^+} v_i - \sum_{i \in \sigma^-} v_i \quad (5.32)$$

with the vector v_i associating the line segment $[-v_i, +v_i]$, where $\mathfrak{Z}_{(0,0,\dots,0)}$ is the largest zonotope obtainable, i.e., $\mathfrak{Z}(V)$ and for all other sign vectors we obtain zonotopes which are contained in $\mathfrak{Z}(V)$. The zero entries of σ characterize the shape of \mathfrak{Z}_σ and the non-zero entries describe how \mathfrak{Z}_σ will be translated with respect to the origin. Thus, given a set of vectors $V \in \mathbb{R}^{p,n}$, which generates $\mathfrak{Z} := \mathfrak{Z}(V)$, we can associate a zonotopal tiling $\mathfrak{Z}_\sigma \subseteq \mathfrak{Z}(V)$ with every sign vector $\sigma \in \{+, -, 0\}^n$. One such special kind of tiling is known as a parallelotope tiling, i.e., a tiling formed from all linearly independent subsets of $\{v_1, v_2, \dots, v_n\}$. Thus, a zonotope can be decomposed into $\binom{n}{p}$ parallelotopes whose total volume equals the volume of \mathfrak{Z} (see the work of Shephard [She74] and Richter-Gebert *et al.* [RGZ94] for the proof on volume). This set of parallelotopic tiles is unique, i.e., for every set of zones³ of the zonotope we obtain an exactly one full-dimensional tile.

A sub-zonotope is obtained by removing any one of the generators of a zonotope. This operation can be defined as follows.

DEFINITION 5.23 Let $\{+, -, 0\}^n$ be a collection of sign vectors. A (single-element) fixing defines a sub-zonotope

$$\mathfrak{Z}(V \setminus j^{\{+,-\}}) := \sum_{i \in \{0\}^{(n-1)}} [-v_i, +v_i] + \sum_{i \in \{+,-\}} v_i - \sum_{i \in \{+,-\}} v_i \quad (5.33)$$

where $j \in 1, \dots, n$.

DEFINITION 5.24 Let $\{+, -, 0\}^{(n-1)}$ be a collection of sign vectors with an element already fixed. A (single-element) freeing defines a zonotope

$$\mathfrak{Z}(\hat{V}/j^{\{+,-\}}) := \sum_{i \in \{0\}^n} [-v_i, +v_i] \quad (5.34)$$

where $j \in 1, \dots, n$.

Notation and Terminology. We shall be using certain notations and nomenclatures throughout the tiling section. The original zonotope to be split into parallelotopic tiles is denoted by $\mathfrak{Z}(V)$ and we will use the term *primitive zonotope* to describe it. Any zonotope (which is not a p -parallelotope) constructed after fixing a sign of one of the generators is denoted by $\mathfrak{Z}(V \setminus j^{\{+,-\}})$ where $j \in 1, \dots, n$ and phrased as *sub-zonotope*. We may use different terminologies to describe a p -parallelotope: *p-dimensional parallelotopic tile* or *p-parallelotopic tile* or *parallelotopic tile* or simply *tile*.

EXAMPLE 5.25 Consider the zonotope in Figure 5.17a with its center and its collection of generators as $c = (20, 10)$, $V = ((-3, 4), (5, 2), (2, 0), (1, 1), (3, 5))$. If we fix the sign of the first generator to '-' we obtain a sub-zonotope. We shall denote it by $\mathfrak{Z}(V \setminus 1^-)$ according to Definition 5.23. This sub-zonotope is

³A zonotope \mathfrak{Z} with n zones in \mathbb{R}^p is the Minkowski sum of n line segments.

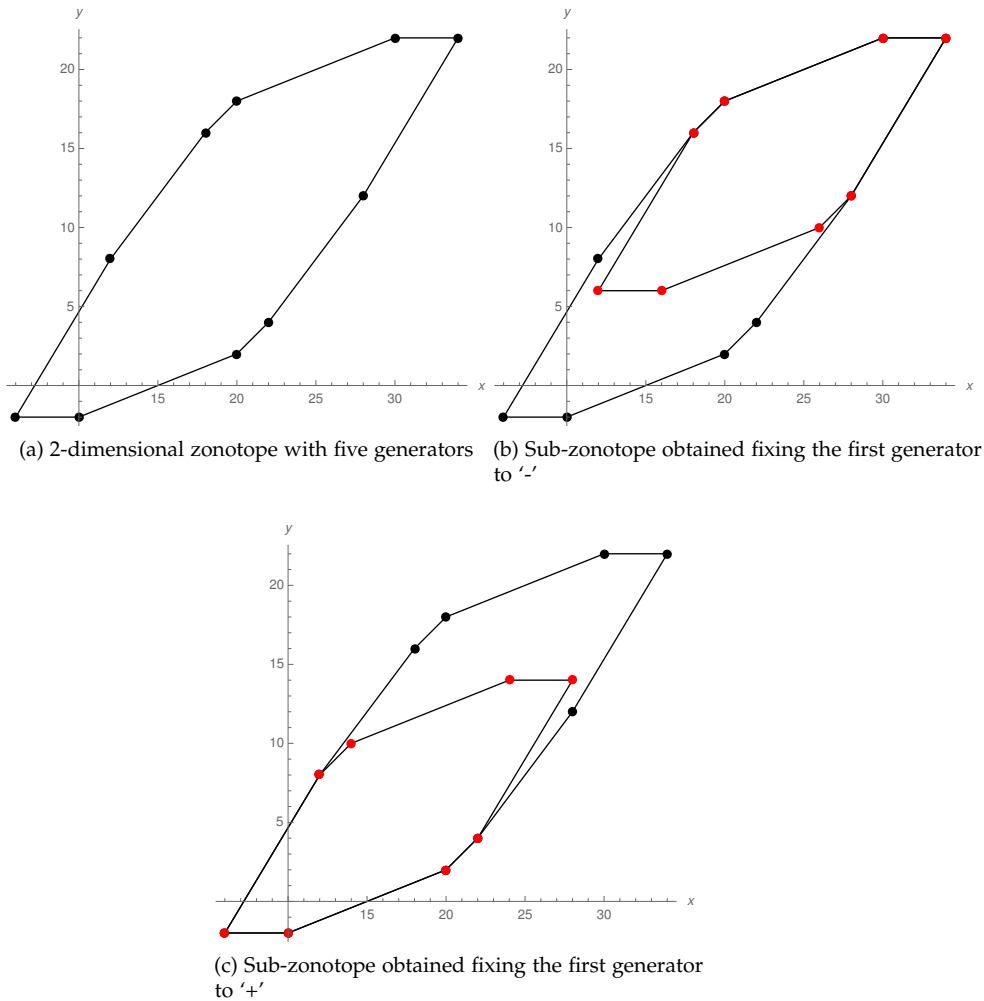


Figure 5.17 – Figures illustrating the ideas of fixing and freeing the signs of generators

shown in Figure 5.17b whose extremal points are marked in red. Its center can be determined as

$$(20, 10) + (-1)(-3, -4) + 0(5, 2) + 0(2, 0) + 0(1, 1) + 0(3, 5)$$

where $(20, 10)$ is the center of the primitive zonotope $\mathfrak{Z}(V)$. In the similar manner, if you fix the first generator to '+' we obtain a sub-zonotope $\mathfrak{Z}(V \setminus 1^+)$ shown in Figure 5.17c) different from the one in Figure 5.17b. The center of this new sub-zonotope can be determined as

$$(20, 10) + (+1)(-3, -4) + 0(5, 2) + 0(2, 0) + 0(1, 1) + 0(3, 5).$$

Remark 5.26. By definition 5.23 and 5.24, we know that a sub-zonotope is obtained by fixing the sign of any one of the generators of a zonotope. Thus, a parallelotopic tile is also a zonotope with p linearly independent generators free and $n - p$ generators fixed. The zero entries of the sign vectors corresponding to the p generators characterize the shape of the tile and the non-zero entries of the sign vectors corresponding to the $n - p$ generators describe how the tile will be translated with respect to the center of the primitive zonotope. Using (5.32) a parallelotopic tile can be defined as:

$$\mathfrak{Z}_\sigma := \sum_{i \in \{0\}^p} [-v_i, +v_i] + \sum_{i \in \{+, -\}^{(n-p)}} v_i - \sum_{i \in \{+, -\}^{(n-p)}} v_i \quad (5.35)$$

A SURVEY ON ZONOTOPAL TILINGS About tilings, mainly two different “schools” exist in the zonotopal tiling community. One of these schools focussed on enumerating the set of tilings possible for a given zonotope where two tilings are connected if one can reach from one tiling to the other by a local transformation. We will call this school, the “*de Bruijn grids school*”. A parallel study has been done by the other school, “*hyperplane arrangement-matroid theory*”, which claims that tilings of zonotopes can also be interpreted as extensions of matroids, i.e., collection of sign vectors.

De Bruijn grids school. Recall that tiling of a zonotope means a set of tiles (translated copies of prototiles) which covers the zonotope exactly with no gap and overlap. Whereas set of tilings means all different ways of tiling the same zonotope with a given set of prototiles. One can reach from one tiling to the other by endowing a local rearrangement of tiles.

This school offers to enumerate all the p -dimensional tiles of a zonotope by deriving the relationship between a tiling and de Bruijn surfaces. De Bruijn grids are $(p - 1)$ -dimensional surfaces which join together the middles of the two opposite sides of each tile [dB81, DB86]. De Bruijn proved that these grids are dual representations of the rhombus tilings⁴ of a zonotope.

De Bruijn surfaces and its connection with tiles. The set of tiles traversed by a de Bruijn surface are always adjacent. In $p = 2$, de Bruijn surfaces are lines which join together the opposite edges of the rhombic tiles. Each tile is crossed by exactly p de Bruijn surfaces, i.e., 2 de Bruijn lines in dimension 2 or 3 de Bruijn surfaces in dimension 3. All the adjacent tiles that are joined by a de Bruijn line, share an edge of the same orientation. Likewise, in dimension 3, there is a de Bruijn surface that joins the adjacent tiles sharing a face of the same orientation. Thus, we can define a de Bruijn family of tiles characterized by a vector among the ones which generate the tiled zonotope and this family contains all the tiles which have this vector as an edge. The de Bruijn surfaces of the same family never intersect. Below, we see these characteristics of de Bruijn lines with the help of an example.

EXAMPLE 5.27 An example of de Bruijn lines and a zonotopal tiling is shown in Figure 5.18. There are four de Bruijn lines (a,b,c,d) corresponding to four de Bruijn family of tiles where each family contain the set of adjacent tiles that

⁴Rhombic tiling of a zonotope defined by n vectors in p -dimension is a set of rhombus tiles obtained as the Minkowski sum of p vectors among the ones which generate the zonotope we want to tile. They are also known as parallelotopic tiles.

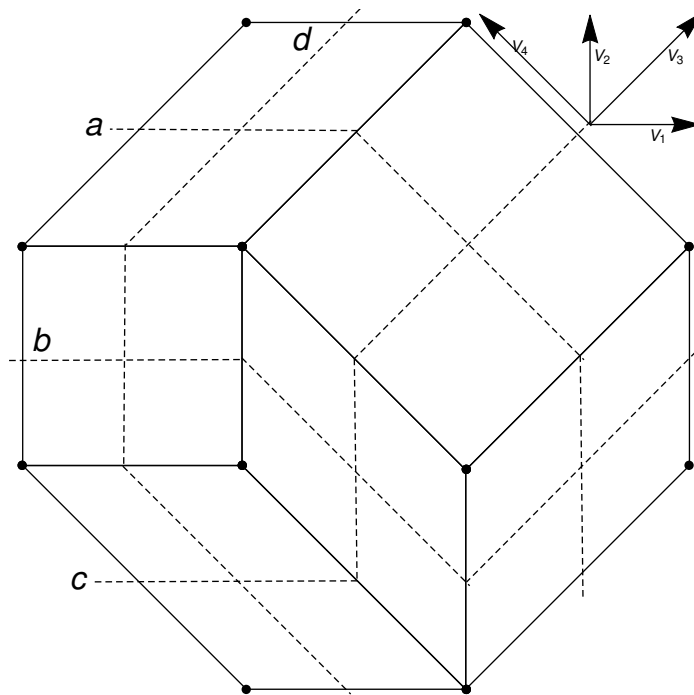


Figure 5.18 – De Bruijn lines of a two-dimensional tiling.

share the same vector as an edge orientation. The de Bruijn lines join together the middle of the edges of the rhombic or parallelotopic tiles. Notice that, the intersection of a set of 2 de Bruijn lines that are pairwise not parallel is a tile.

From de Bruijn lines to enumerating tiles Consider a zonotope defined by the Minkowski sum of n vectors in p -dimension. We will denote by $n \rightarrow p$ a rhombic tiling of this zonotope. Provided with such a $n \rightarrow p$ tiling which is composed of set of distinguished tiles, if one deletes a de Bruijn family of tiles sharing the same vector as an edge orientation then it gives a $n - 1 \rightarrow p$ tiling. Correspondingly, one can construct $n + 1 \rightarrow p$ tiling from a $n \rightarrow p$ tiling for any n [DMB97, Lat00, DD05].

Destainville *et al.* in [DMB97] and Latapy in [Lat00] defined a dual graph for a tiling whose set of vertices is the set of intersection points of de Bruijn lines, and there is an edge (i, j) if and only if i and j belong to the adjacent tiles. From the dual graph of a $n \rightarrow p$ tiling, the authors define a partition problem, the solutions of which are equivalent to the $n \rightarrow p$ tiling. They claim that the tiling associated to a partition is a bijection from the set of partitions solutions to the problems for all dual graph of a $n \rightarrow p$ tiling to the set of $n + 1 \rightarrow p$ tilings.

EXAMPLE 5.28 Figure 5.19 shows an example of $4 \rightarrow 2$ tiling. One possible vector v_4 is indicated in the figure. All the shaded tiles in Figure 5.19a share this vector and belong to the same de Bruijn family. Notice that if we delete the shaded tiles in the $4 \rightarrow 2$ tiling then we obtain a $3 \rightarrow 2$ tiling in Figure 5.19b. One can refer to Figure 3 in [DMB97], Figures 5 and 7 in [Lat00] and Figures

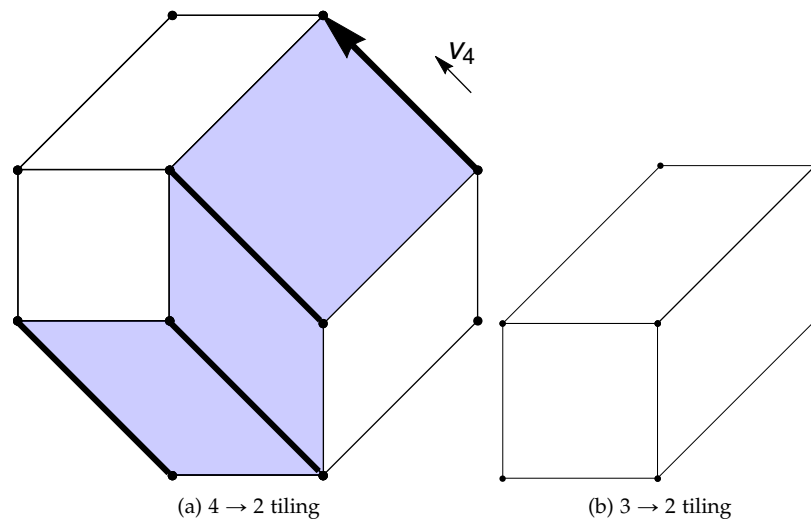


Figure 5.19 – Examples of tilings

2 and 3 in [DD05] for more such examples.

De Bruijn lines and flip transformation. This duality between the de Bruijn grids and tiling was useful in constructing all the p -dimensional tilings of a zonotope (i.e., all the different ways one can tile a given zonotope using the same tiles but locally rearranged).

Consider a zonotope with two tilings T and T' which have the same set of tiles but locally rearranged. T' can be obtained from T through a transformation called *flip*, if and only if it involves at least one tile of the de Bruijn family of tiles. Also known as *phason-flip* or *phason*, in dimension p , a flip is an operation which rearranges $p + 1$ tiles in a zonotope inside the tiling, e.g., in dimension 2, it is a rearrangement of three rhombus tiles inside a hexagon. The name flip comes from the transformations which occur in the field of quasicrystals⁵ [SBGC84, LCCG96].

One can enumerate all the tilings of a zonotope from a given one by iterating the following rule, i.e., a dynamics can be defined over the tiling T , $T \rightarrow T'$ such that the tiling T' can be obtained from T by a flip. However, is it true for all dimensions? This is known as the famous *generalized Baues problem* [Rei99], i.e., *are all the cubical tilings of a zonotope and their cube-flips connected?* It is a challenging question involving issues like Markov chain ergodicity. The fact that all the tilings are flip-connected was proved for 2-dimension by Kenyon [Ken93], Elnitsky [Eln97], Latapy (see Figure 5 in [Lat00]), Chavanon (see Figure 10 in [CR06]) and in 3-dimension by Desoutter (see Figure 3 in [DD05]). The flip-connectivity problem is still an open problem in higher dimensions.

⁵The study of tilings evolved a special interest with the uncovering of quasicrystals because tiling could extract some significant property like non-periodicity by analyzing the manner in which the neighboring tiles match

Hyperplane arrangement-matroid theory school. Recall that a zonotope can be defined in various ways: for instance, Minkowski sum of line segments, linear projection of a cube, a convex polytope with centrally symmetric faces in all dimension. However, there exists another characterization which a zonotope is a polar dual of hyperplane arrangement [McM71, She74, Sta98, S⁺04, Bai97].

The second school has been indispensable towards the tiling problem of zonotopes. By using the dual relationship between a zonotope and its corresponding hyperplane arrangement it proved that given a set of tiles which defines a tiling of a zonotope, each of the tiles can be uniquely represented by a collection of sign vectors or matroid⁶ [HR11, Arb16, Zie12, Fel12, ZRG17]. In order to continue with the relationship between tiling and sign vectors, we need to recall the classical notions of hyperplane arrangement.

An arrangement of hyperplane is a collection $\mathcal{A} := \{h_1, \dots, h_n\}$ of finitely many hyperplanes in \mathbb{R}^p , where each h_i is of the form $h_i = \{x \in \mathbb{R}^p \mid v_i^\top x = 0\}$ for some $v_i \in \mathbb{R}^p$. This kind of representation of hyperplanes is called the central hyperplane arrangement (i.e., all the hyperplanes are passing through the origin). The hyperplane arrangement \mathcal{A} decomposes \mathbb{R}^p into a fan like structure where the cones of the fan are referred to as cells of the hyperplane arrangement. One can extract information about the configuration of the vectors $\{v_1, \dots, v_n\}$ from the combinatorics of the cones.

Characterizing tiles by sign vectors. Dually, provided with a vector configuration $V = \{v_1, \dots, v_n\}$ which defines a zonotope $\mathfrak{Z}(V)$ by their Minkowski sum, one can define the associated central arrangement $\mathcal{A} = \mathcal{A}(V)$ of n hyperplanes in \mathbb{R}^p , each having v_i as its normal vector: $\mathcal{A}(V) = \{h_i \mid i = 1, 2, \dots, n\}$ where $h_i = \{x \in \mathbb{R}^p \mid v_i^\top x = 0\}$ for $i = 1, 2, \dots, n$. For each of these hyperplanes h_i , there is a positive halfspace given by $h_i^+ = \{x \mid v_i^\top x > 0\}$ and the negative one $h_i^- = \{x \mid v_i^\top x < 0\}$. The position of x with respect to the set $\{h_i, h_i^+, h_i^-\}$ is determined by the sign of $v_i^\top x$. Indeed, if $\text{sign}(v_i^\top x) = 0$, then x lies in h_i ; if the sign is $+$, then x lies inward of h_i^+ ; and if it is $-$, then x lies inward of h_i^- . Thus, for specific cells of the hyperplane arrangement there are specific sign vectors. This means that the combinatorial structure of a zonotope can be defined by a collection of sign vectors otherwise known as matroid. In fact, there is a natural bijection between the sign vectors of the vertices of the zonotope $\mathfrak{Z}(V)$ and the cells of the hyperplane arrangement $\mathcal{A}(V)$. This bijection could be further used to solve tiling problem by the notion that tilings of zonotopes can be interpreted as extensions of matroids, which is well-known as *Bohne-Dress Theorem* [Boh92, RGZ94].

We show an example below illustrating that given a tiling of a zonotope, how all the tiles can be characterized by their associated sign vectors uniquely.

EXAMPLE 5.29 Figure 5.20 gives an example of a set of vectors $V = \{v_1, \dots, v_3\}$ ($V = ((0.8, 0), (0, 0.8), (0.816, 0.816), (-0.816, 0.816))$) and its associated hyperplane arrangement $\mathcal{A} := \{h_1, \dots, h_4\}$ in \mathbb{R}^2 . In red, are illustrated the sign of $v_i^\top x$, where the position of x is given by the sign vector: $\text{sign}(xV) \in \{+, -, 0\}^2$, whose first coordinate traces the corresponding position in regard to h_1 , the second coordinate associates to h_2 , and so forth. For instance, the sign vectors

⁶Given a vector configuration $V \in \mathbb{R}^{p \times n}$, which generates $\mathfrak{Z} := \mathfrak{Z}(V)$ a polytope in \mathbb{R}^p then the matroid is a collection of sign vectors

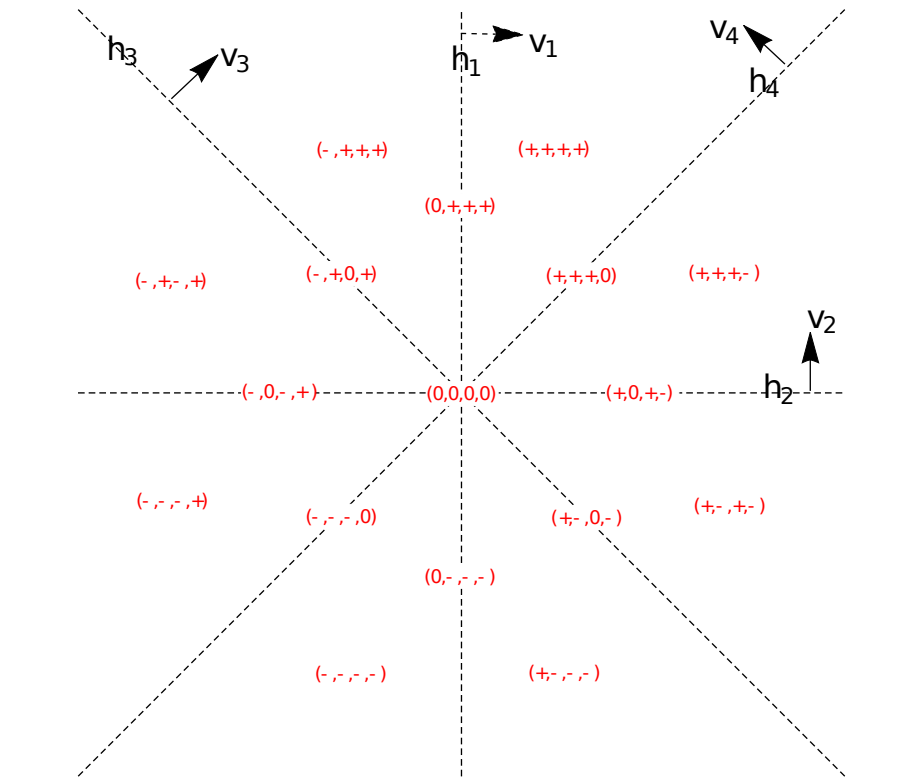


Figure 5.20 – A hyperplane arrangement in \mathbb{R}^2 with four lines.

$(+, +, +, -)$ marked in Figure 5.20 refers to a point which lies interior of h_1^+, h_2^+, h_3^+ but belongs to the negative half-space of h_4 .

EXAMPLE 5.30 Consider the Minkowski sum of the above set of vectors which is a convex polytope, a zonotope polar to the hyperplane arrangement \mathcal{A} , as shown in Figure 5.21. Notice that the vertices of this zonotope are in bijection with the cells of the hyperplane arrangement, i.e., they have the same sign vectors.

Figure 5.22 illustrates a tiling of the zonotope composing of four tiles. Each of these tiles are formed from the subset of the vectors $\{v_1, \dots, v_3\}$ and translated based on the dual representation of hyperplane arrangement. Observe, the parallelotopic or rhombic tile marked with the sign vector $(0, +, 0, +)$. It points to the position of this zonotopic tile formed by the vectors v_1, v_3 , and hence '0' on their respective coordinates. It has a '+' sign on second and fourth coordinates because the tile is located on the positive half-space with respect to h_2, h_4 hyperplanes.

Later in this chapter, we will be using the duality principle (between zonotopic tiles and matroids) for developing our tiling algorithm.

IS ZONOTOPAL TILING A VERTEX ENUMERATION PROBLEM? Vertex enumeration is a well-known problem in polyhedras while switching from H (hyperplane)-

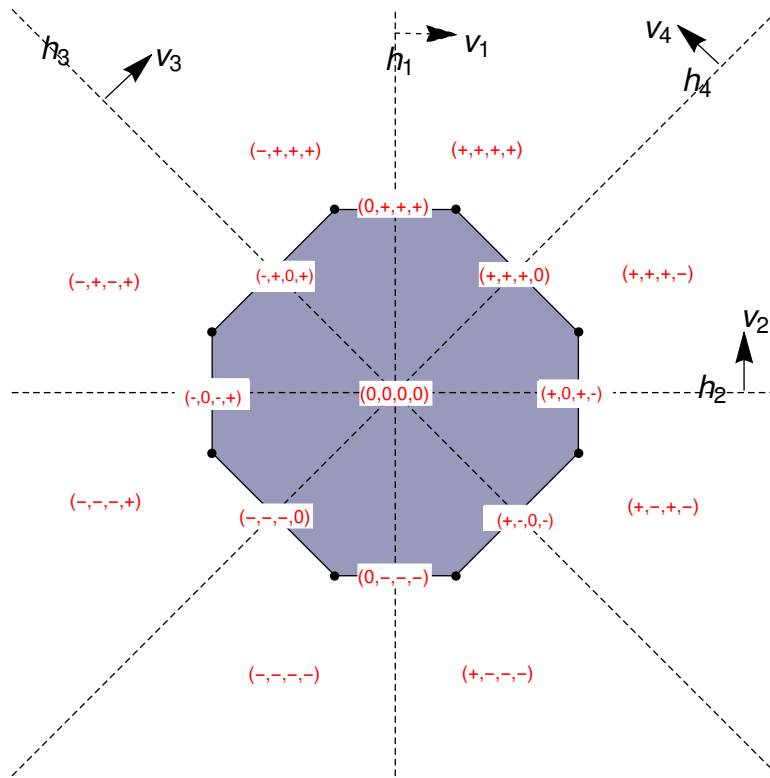


Figure 5.21 – Polar dual of the hyperplane arrangement in Figure 5.20, i.e., a zonotope.

representation to V (vertex)-representation [AF92],[Zie12]. Now, we shall see this connection with respect to a zonotopal tiling.

Recall that a zonotope defining p variables over n noise symbols is a projection of the n -dimensional hypercube by an affine map in a p -dimensional space. The vertices of a zonotope are a projection of a subset of the corners of the hypercube—i.e., n -vectors with elements in $\{-1, 1\}$ —projected with the generator matrix. Some of the 2^n vertices are extremal points of the zonotope (the red ones of Figure 5.23). In other words, a subset of projected vertices that define the zonotope. However, there are some projected vertices of the hypercube which do not map to the vertices of the zonotope. In total, there are 2^n corners of the n -dimensional hypercube which can be projected with the generator matrix.

EXAMPLE 5.31 Consider the zonotope shown in Figure 2.10 which is a projection of 5-dimensional hypercube by an affine map in a 2-dimensional space. Figure 5.23 illustrates all the 2^5 vertices (black circles) of the 5-dimensional hypercube projected with the generator matrix. The red markers describe the subset of projected vertices that define this zonotope.

Remark 5.32. Thus, one can say that the solution to tiling problem is finding sufficiently many of these not existant vertices that are not part of the extremal points of the zonotope to be tiled. In other words, the non-extremal projections

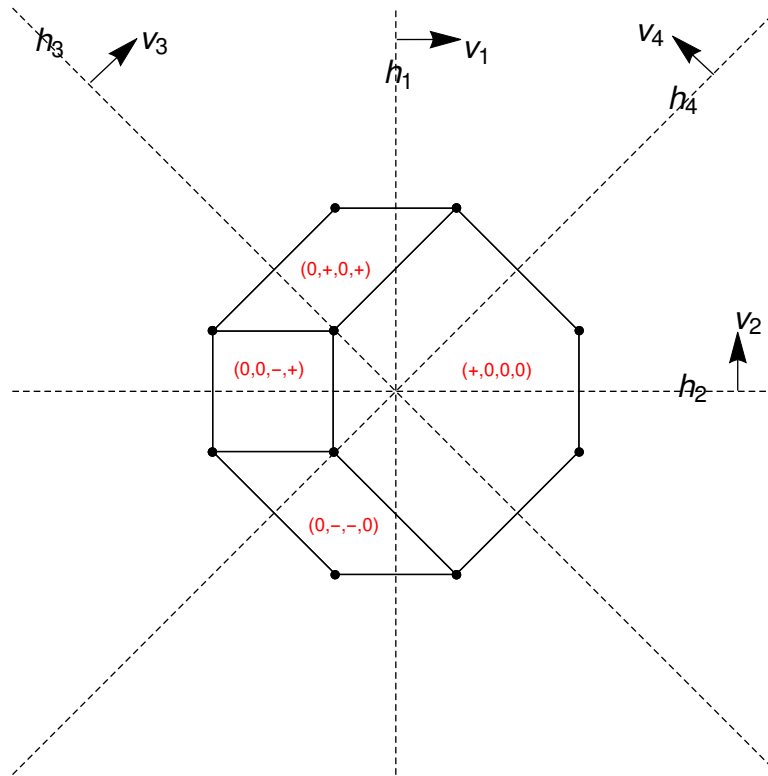


Figure 5.22 – A tiling of the zonotope in Figure 5.21 and the sign vectors of the corresponding tiles.

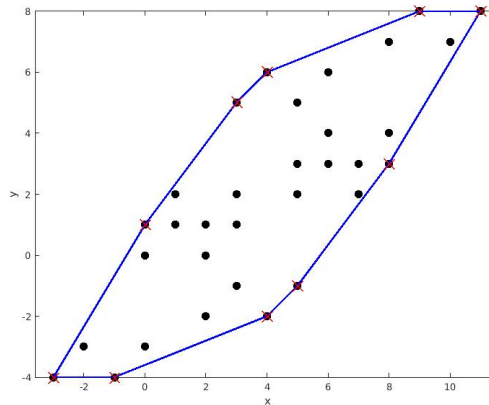


Figure 5.23 – 2^5 vertices of the 5-dimensional hypercube projected with the generator matrix.

become the extremal points of the parallelotopes in the tiling. Thus, tiling indeed is a vertex enumeration problem.

OUR TILING ALGORITHM We develop below an algorithm for tiling, illustrated in Algorithm 5.2 which instead of enumerating all the vertices, enumerate only the vertices characterizing the sub-zonotopes tiling a given zonotope. We use ideas issued from matroid theory established by Bohné-Dress theorem [RGZ94], i.e., there is a close connection between a zonotope and the signs of its vectors since they abstract combinatorial facts about the structure of the zonotope. Thus, the key objective of the algorithm is to enumerate the vertices of the tiles as sign vectors of the so-called hyperplane arrangement [FFL05] corresponding to a zonotope, that we are going to define now.

Hyperplane arrangements and zonotopes. A finite family $\mathcal{A} = \{h_j : j = 1, \dots, m\}$ of hyperplanes in \mathbb{R}^p is called an arrangement of hyperplanes. Any hyperplane partitions the space \mathbb{R}^p into three sets: $h_j^+ = \{x \mid v_j^\top x > b_j\}$, $h_j^0 = \{x \mid v_j^\top x = b_j\}$ and $h_j^- = \{x \mid v_j^\top x < b_j\}$. For each point x in \mathbb{R}^p , there is a sign vector $\sigma(x) \in \{+, -, 0\}^{[n]}$ giving its relative location with respect to the hyperplane arrangement, defined as follows:

$$\sigma(x)_j = \begin{cases} +, & \text{if } x \in h_j^+ \\ -, & \text{if } x \in h_j^- \\ 0, & \text{if } x \in h_j^0 \end{cases} \quad (5.36)$$

The set of points with a given sign vector is an open polyhedron, whose faces of every dimension (including full dimensional p -dimensional cells partitioning the polyhedron) are determined by the intersection of some sets of the form h_j^0 , h_j^- and h_j^+ , hence are in one-to-one correspondence with sign vectors. Such a set is a cell if the corresponding sign vectors do not have zero entries.

For a zonotope $\mathfrak{Z} = \mathfrak{Z}(V)$ generated by the columns of V , we define the associated central arrangement $\mathcal{A} = \mathcal{A}(V)$ of n hyperplanes in \mathbb{R}^p , each having v_j as its normal vector : $\mathcal{A}(V) = \{h_j^0 \mid j = 1, 2, \dots, n\}$ where $h_j^0 = \{x \in \mathbb{R}^p \mid v_j^\top x = 0\}$ for $j = 1, 2, \dots, n$. There is a duality relation between a zonotope and its corresponding hyperplane arrangement. Every d -dimensional open polyhedron in \mathcal{A} determined by its sign vectors corresponds to a $p - d$ -dimensional region of a zonotope where $d \leq p$ for e.g., a full-dimensional open polyhedron corresponds to the vertices of the zonotope.

We denote by $\Sigma = \Sigma(V)$ a set of sign vectors of cells of the arrangement where each vector corresponds to a cell. For example, Figure 5.24 illustrates an arrangement of 5 hyperplanes in \mathbb{R}^2 . Each cell is represented by a sign vector of dimension 5. Furthermore, two extreme points in \mathfrak{Z} are adjacent in \mathfrak{Z} if and only if the associated cells are adjacent, (i.e. sharing a $(p - 1)$ face). This notion can be extended to the fact that two tiles are adjacent if they share a whole facet⁷ which we will observe later. We assume certain regularities in the structure of the matrix V . Under those regularity assumptions, two cells are adjacent if and only if their sign vectors are different in exactly one component.

⁷A facet of a polytope of dimension p is a face which has dimension $p - 1$.

Assumption. There are no zero vectors in V and no two vectors are linearly dependent. If two vectors are parallel, one can add both the vectors without changing the combinatorial structure of the hyperplane arrangement. Provided that two vectors are multiples of each other, then they determine the same hyperplane in the arrangement which means one of them can be removed for sake of simplicity. Then this simplified hyperplane arrangement can be used to obtain the original sign vectors.

Enumerating sign vectors. Recall that finding the tiles for a zonotope is equivalent to enumerating the vertices of the tilings as sign vectors of the so-called hyperplane arrangement corresponding to a zonotope. Computing the sign vectors is thus a cell enumeration problem. There are several algorithms for the cell enumeration for a general arrangement. One of the widely used algorithm is reverse search. It is a framework for solving various enumeration problems on graphs. We use here a reverse search algorithm [FFL05] that has a time complexity of $\mathcal{O}(n \cdot p \cdot \text{LP}(n, p) \cdot |\Sigma|)$ to compute $\Sigma = \Sigma(V)$ for any given rational $p \times n$ matrix V , where $\text{LP}(n, p)$ is the time to solve a linear programming problem with n inequalities in p variables.

Reverse search algorithm. Let c^* be any cell in Σ . Finding one cell is relatively simple: for example by selecting an arbitrary point in \mathbb{R}^p . Without loss of generality, we may assume c^* is the vector $(-, +, +, +, +)$. Consider a cell c^* with the sign configuration $(-, +, +, +, +)$ from the hyperplane arrangement shown in Figure 5.25. One may also assume c^* is the vector of all $+$'s or $-$'s based on which we might have to replace some columns by their negatives or positives and it does not affect the arrangement. Now, we will use the reverse search algorithm to trace all the members of Σ by trying to reach the goal cell c^* from another cell c such that $c \in \Sigma \setminus c^*$. The principal idea is to use ray shooting. For this, we need two points, one interior point p^* of the goal cell c^* and one interior point p of a cell c . Then, shoot a ray from p to p^* . It will hit all hyperplanes separating c and c^* . Let the interior point p belongs to the cell with the sign configuration $(+, -, +, -, -)$. We select the first hyperplane hit by the ray. In here it is the second hyperplane, and we flip the sign of the second generator modifying the sign vector to $(+, +, +, -, -)$. In the similar manner the ray hits the fourth and first hyperplane consecutively before reaching p^* and we modify the signs accordingly. Note that while tracing the ray from p to p^* we have enumerated the three extremal points with the following sign vectors $(+, +, +, -, -)$, $(+, +, +, +, -)$, $(-, +, +, +, -)$. Notice that in order to find an interior of a cell, we use linear programming because we ensure that the interior point is defined uniquely. Thus, in the above manner we list all the cells in a hyperplane arrangement in \mathbb{R}^p and their sign vectors.

Notions whose sequel is the tiling algorithm. As already observed in Figure 5.17, a sub-zonotope can be enumerated from a given zonotope by fixing the sign of one of its generators and if we keep repeating the procedure then eventually we obtain a p -parallelotope. We believe that during the process of enumerating the first parallelotopic tile, we have enumerated sufficiently many non-existent vertices to obtain all the remaining tiles. This can be proved by mathematical

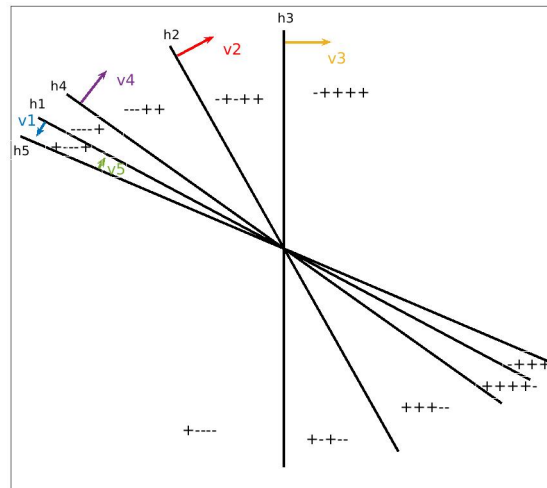


Figure 5.24 – Arrangement of hyperplanes.

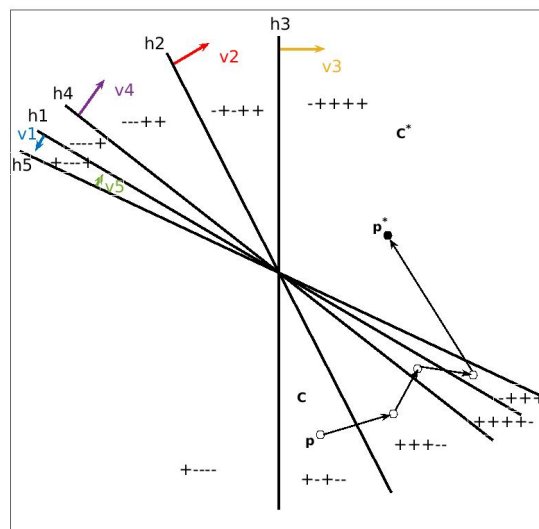


Figure 5.25 – Ray shooting and sign enumeration.

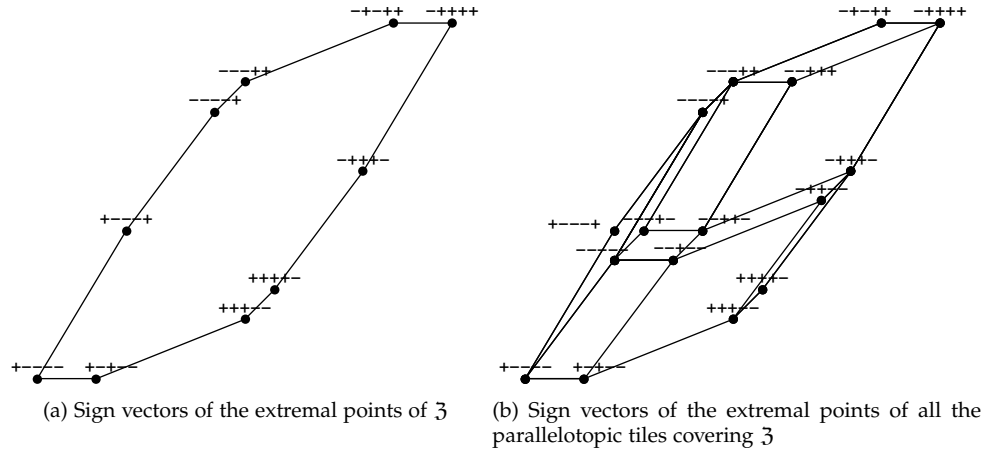


Figure 5.26 – The primitive zonotope and its tiling

induction, i.e., if every sub-problem of tiling is denoted by P_k then

$$(\forall P, [P_1 \vee (\forall k \geq 1)P_k \implies P_{k+1}]) \implies [\forall (k \geq 1)P_k].$$

LEMMA 5.33 *Let $\mathfrak{Z}(V)$ be a zonotope formed on a set of generators $V = (v_1, \dots, v_n) \in \mathbb{R}^{p \times n}$ where p is the dimension and $n \geq p$. Incrementally fixing the sign of the generators until we enumerate the first p -parallelotopic tile is equivalent to the fact that we have enumerated sufficient number of hidden vertices (with respect to the original zonotope) corresponding to the extremal points of each sub-zonotope to construct the remaining parallelotopic tiles.*

Proof. Base case, $n = p + 1$:

It is trivial that for a zonotope with the number of generators equal to $n = p + 1$, fixing the sign of one of its generators would produce a p -parallelotope.

Consider a zonotope in \mathbb{R}^2 shown in Figure 5.27a. Fixing the sign of one of its generators would immediately enumerate a parallelotope (extremal points marked in red in Figure 5.27b) with the two remaining parallelotopic tiles linked by an edge being enumerated implicitly.

Now, consider a zonotope in \mathbb{R}^3 illustrated in Figure 5.28a. As earlier, once you obtain a 3-parallelotope (shown in Figure 5.28b with extremal points noted in red), we have sufficient number of vertices to enumerate rest of the 3-parallelotopic tiles (all of it are illustrated one-by-one from Figure 5.28c-5.28e).

Hence, for $n = p + 1$ case, it is true that once we obtain a parallelotopic tile, we have enumerated sufficiently many non-existent vertices to obtain all the remaining parallelotopic tiles.

Inductive step, for all integers $k \geq 1$:

Hypothesis: true for $n = p + k$

To prove: true for $n = p + k + 1$

The problem of tiling the zonotope with $n = p + k + 1$ number of generators can be partitioned into the problem of tiling the zonotope with $n = p + k$ generators, plus tiling the zonotope with $n = p + 1$ generators and plus a

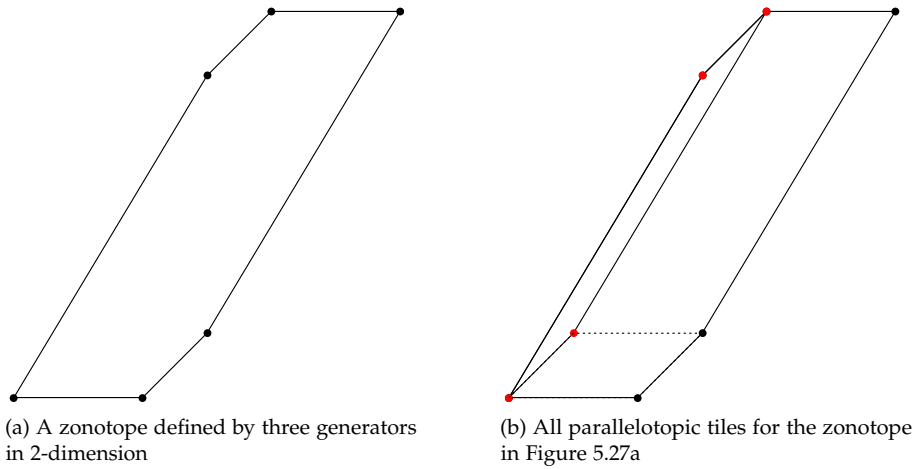


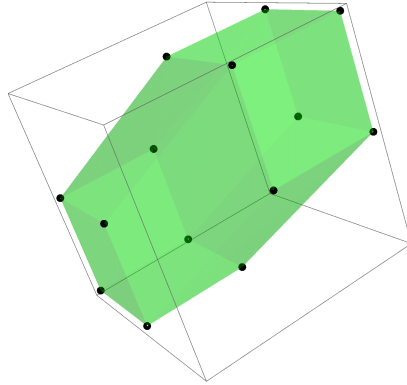
Figure 5.27 – Illustrating, how fixing the sign of a zonotope defined by 3 generators in 2-dimension implicitly enumerates all the tiles

p -parallelotopic tile, i.e., $n = p$. Tiling the zonotope with $n = p + 1$ generators into parallelotopes is the base case. For the zonotope with $n = p + k$, the tiling using parallelotopes can be accomplished by inductive hypothesis. \square

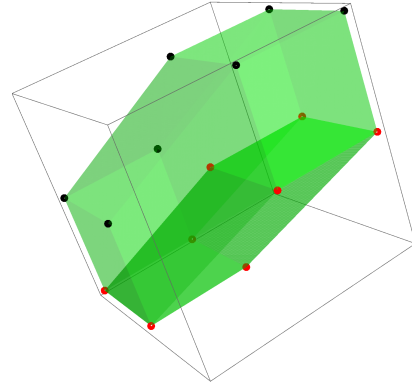
Remark 5.34. A consequence of the Lemma 5.33 is a recursive algorithm for tiling which we discuss below in detail.

Tiling algorithm. Algorithm 5.2 illustrates a recursive algorithm for computing all the p -parallelotopic tiles (also called p -dimensional parallelotopic tiling) characterizing a given p -dimensional zonotope $\mathfrak{Z} = \mathfrak{Z}(c, v_1, v_2, \dots, v_n)$. First it checks if the input is already a tile i.e., $n == p$, then it returns the singleton containing the zonotope itself, otherwise it arbitrarily chooses a sign to fix the first generator. Fixing v_1 will produce a sub-zonotope defined by: $\mathfrak{Z}_{\text{sub}} = \mathfrak{Z}((\sigma_1, 0, \dots, 0))$ computed according to (5.32) where σ_1 is either '+' or '-'. We consider the fact that tilings of a zonotope by zonotopes are in bijection with fixing the sign of a generator which is essentially Bohne-Dress theorem [Boh92, Dre89]. Then we make a recursive call of the tiling function on $\mathfrak{Z}_{\text{sub}}$ which computes its tiling and stores the result in \mathcal{T} . The remaining step consist in finding all the adjacent p -parallelotopic tiles of $\mathfrak{Z}_{\text{sub}}$. First we compute the sign vectors (Σ') of $\mathfrak{Z}_{\text{sub}}$ and prepend to them the sign of the first generator (σ_1) and add all the non-existent sign vectors to Σ . This corresponds to the extremal points of $\mathfrak{Z}_{\text{sub}}$ which are not extremal points of \mathfrak{Z} . We know that the first generator was fixed for computing $\mathfrak{Z}_{\text{sub}}$. Now, we free it and for all subset of generators $v_2 \dots v_n$ of length $p - 1$ we compute the parallelotopes for the p free generators $\{1\} \cup S$.

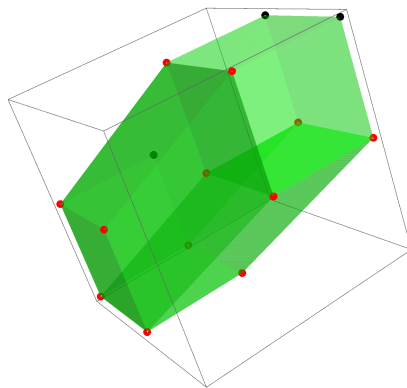
EXAMPLES Below, we consider few examples in 2 and 3-dimension zonotopes on which we illustrate step-by-step the tiling algorithm.



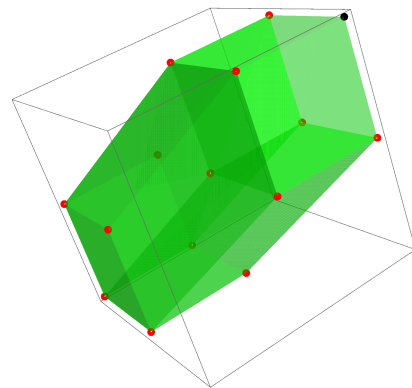
(a) A zonotope defined by four generators in 3-dimension



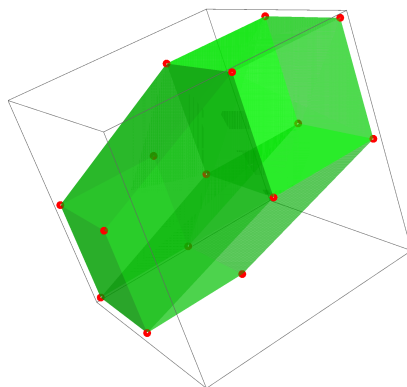
(b) The first parallelotopic tile obtained after fixing the sign of one of the generators



(c) Tile adjacent to the parallelotope enumerated in Figure 5.28b



(d) Another parallelotopic tile adjacent to the tile in Figure 5.28c



(e) All the parallelotopic tiles

Figure 5.28 – Illustrating, how fixing the sign of a zonotope defined by 4 generators in 4-dimension implicitly enumerates all the tiles

Algorithm 5.2 – Tiling Algorithm

```

Function{TILINGS}{Z}
IF n == p THEN
  return {Z}
ELSE
  Compute  $\Sigma = \Sigma(V)$ 
  Compute  $Z_{\text{sub}} = Z((\sigma_1, 0, \dots, 0))$ 
  T = TILINGS{Zsub}
   $\Sigma' = \Sigma(V')$  where  $V'$  are the generators of  $Z_{\text{sub}}$ 
  Prepend  $\sigma_1$  to  $\Sigma'$ , and add to  $\Sigma$ 
  // Find all tiles adjacent to  $Z_{\text{sub}}$ 
  FOR S in  $2^{\{2, \dots, n\}}$  of length  $p - 1$  DO
    Find all p-parallelotopic tiles for S in  $\Sigma$ 
    Add to T
  return T
EndFunction

```

EXAMPLE 5.35 Consider the zonotope shown in Figure 5.26a with its associated hyperplane arrangement in Figure 5.24. The region between two adjacent hyperplanes is a cell corresponding to a vertex of the zonotope. The signs for each of the vertices are illustrated in Figure 5.26a which are $(-, -, -, -, +)$, $(-, -, -, +, +)$, $(-, +, -, +, +)$, $(-, +, +, +, +)$, $(-, +, +, +, -)$. Recall that for a given zonotope the associated hyperplane arrangement is central, so we will only compute one half of the sign vectors and the rest is its negative counterpart. Subsequently, the sign for the vertices of all the tiles are shown in Figure 5.26b.

Fixing the sign of first generator. As an example, fixing the first generator of the zonotope Z or $Z(V)$ ($V = ((-3, 4), (5, 2), (2, 0), (1, 1), (3, 5))$ and the center is $(20, 10)$) to '-', we obtain a sub-zonotope (Z_{sub} or $Z(V \setminus 1^-)$) shown in Fig. 5.29a (the extremal points marked in red). The remaining generators characterize this sub-zonotope and the one fixed will be used to describe the translation with respect to the original center. The center of this sub-zonotope can be determined as

$$(20, 10) + (-1)(-3, -4) + 0(5, 2) + 0(2, 0) + 0(1, 1) + 0(3, 5)$$

where $(20, 10)$ is the center of the primitive zonotope $Z(V)$. Sign 0 is associated to vectors $(5, 2)$, $(2, 0)$, $(1, 1)$ and $(3, 5)$ because they belong to the generators of the sub-zonotope. Once we have generated this sub-zonotope the next step is to enumerate the sign vectors corresponding to its extremal points. However, it is not necessary to call the reverse search algorithm again for computing the signs of $Z(V \setminus 1^-)$. It can be enumerated from the signs of $Z(V)$ in the following way.

Enumerating the vertices of the first sub-zonotope. We know that the first generator of $Z(V)$ was fixed for determining $Z(V \setminus 1^-)$ which means the sign for the first generator will remain same for all the vertices of $Z(V \setminus 1^-)$ i.e., now there will be four free generators. So, certainly all the extremal points of $Z(V)$ which have '-' sign on the first generator are also going to be the extremal points of $Z(V \setminus 1^-)$ i.e. $(-, -, -, -, +)$, $(-, -, -, +, +)$, $(-, +, -, +, +)$,

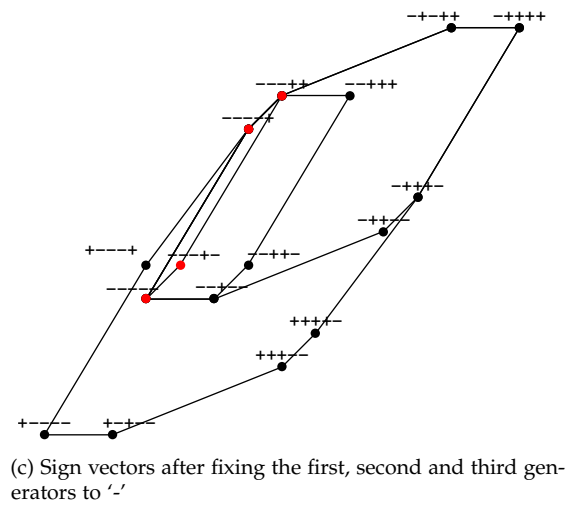
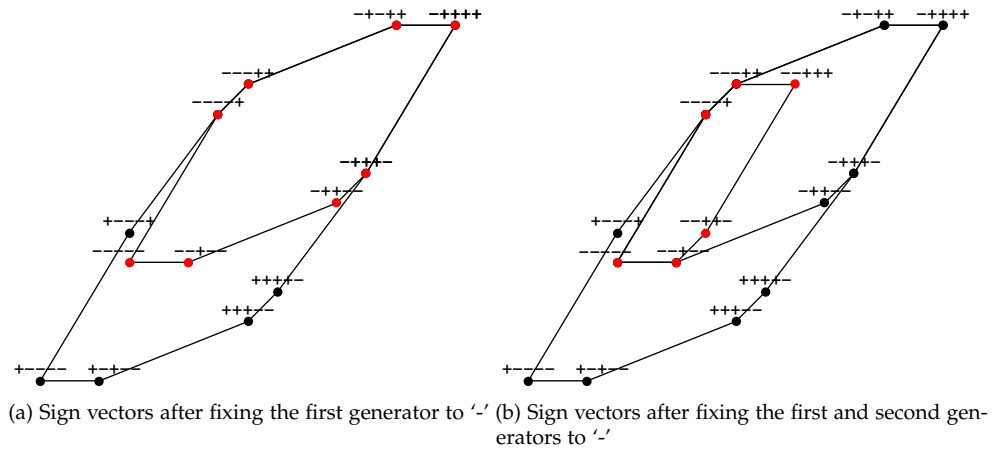


Figure 5.29 – Illustrating one-by-one all sub-zonotopes obtained after fixing the sign of generators

$(-, +, +, +, +)$, $(-, +, +, +, -)$ as shown in Figure 5.29a. If we take the complement of the sign of the first generator for the extremal points of $\mathfrak{Z}(V)$ which have a '+' sign on the first generator then we enumerate the sign vectors of the remaining vertices of $\mathfrak{Z}(V \setminus 1^-)$ i.e., $(+, -, -, -, +) \rightarrow (-, -, -, -, +)$, $(+, -, -, -, -) \rightarrow (-, -, -, -, -)$, $(+, -, +, -, -) \rightarrow (-, -, +, -, -)$, $(+, +, +, -, -) \rightarrow (-, +, +, -, -)$, $(+, +, +, +, -) \rightarrow (-, +, +, +, -)$. Among these sign vectors of $\mathfrak{Z}(V \setminus 1^-)$ there are few which were non-existent (corresponding to the vertices interior of zonotope $\mathfrak{Z}(V)$ and not on its boundary) in the sign vectors of the primitive zonotope $\mathfrak{Z}(V)$, for e.g., $(-, -, -, -, -)$, $(-, -, +, -, -)$ and $(-, +, +, -, -)$. They correspond to the extremal points of $\mathfrak{Z}(V \setminus 1^-)$ which are not extremal points of $\mathfrak{Z}(V)$.

Recall that the key idea of the algorithm is to enumerate sufficiently many of these non-existent sign vectors so as to find all the parallelotopic tiles. Notice that, since $\mathfrak{Z}(V \setminus 1^-)$ was obtained by fixing the first generator of $\mathfrak{Z}(V)$ it would be correct to use the notation that the sign vectors of $\mathfrak{Z}(V \setminus 1^-)$ are $(-, -, -, -, -)$, $(-, -, -, +)$, $(-, -, +, +)$, $(+, -, +, +)$, $(+, +, +, +)$, $(+, +, +, -)$, $(+, +, -, -)$ and $(-, +, -, -)$ respectively.

Fixing the sign of second generator. Now, fixing the first generator to '-' again, we obtain another sub-zonotope $\mathfrak{Z}(V \setminus \{1^- 2^-\})$ shown in Figure 5.29b (the extremal points marked in red). The center of this sub-zonotope can be determined as

$$(23, 14) + -1(5, 2) + 0(2, 0) + 0(1, 1) + 0(3, 5)$$

where $(23, 14)$ is the center of the sub-zonotope $\mathfrak{Z}(V \setminus 1^-)$, (the extremal points marked in red shown in Figure 5.29a) generated after fixing the first generator of the primitive zonotope to '-'. Sign 0 is associated to vectors $(2, 0)$, $(1, 1)$ and $(3, 5)$ because they belong to the generators of the sub-zonotope.

Enumerating the vertices of $\mathfrak{Z}(V \setminus \{1^- 2^-\})$. Similar to the previous case, we can obtain the non-existent sign vectors by just complimenting the sign for those vertices which has a '+' on the first generator for e.g., $(+, +, -, -) \rightarrow (-, +, -, -)$, $(+, +, +, -) \rightarrow (-, +, +, -)$, $(+, +, +, +) \rightarrow (-, +, +, +)$, $(+, -, +, +) \rightarrow (-, -, +, +)$. As earlier, the sign vectors corresponding to the vertices which have '-' sign on the first generator of $\mathfrak{Z}(V \setminus 1^-)$ also share the corresponding vertices as the extremal points of the new sub-zonotope $\mathfrak{Z}(V \setminus \{1^- 2^-\})$. Accordingly, once again we have enumerated all the sign vectors of $\mathfrak{Z}(V \setminus \{1^- 2^-\})$ including the non-existent ones with respect to the $\mathfrak{Z}(V \setminus 1^-)$ from which $\mathfrak{Z}(V \setminus \{1^- 2^-\})$ was obtained.

The first parallelotopic tile. Repeating this iterative procedure, we enumerate the sign vectors of another sub-zonotope $\mathfrak{Z}(V \setminus \{1^- 2^- 3^-\})$ shown in Figure 5.29c) which is a parallelotopic tile. The sign vectors corresponding to the vertices of this tile are $(-, -)$, $(+, -)$, $(+, +)$, $(-, +)$ with the sign of the first three generators being fixed to '-'. The center of the parallelotope tile can be determined as

$$(18, 12) - 1(2, 0) + 0(1, 1) + 0(3, 5)$$

where $(18, 12)$ is the center of the sub-zonotope $\mathfrak{Z}(V \setminus \{1^- 2^-\})$ and vectors $(1, 1)$ and $(3, 5)$ are the free generators corresponding to the parallelotopic tile.

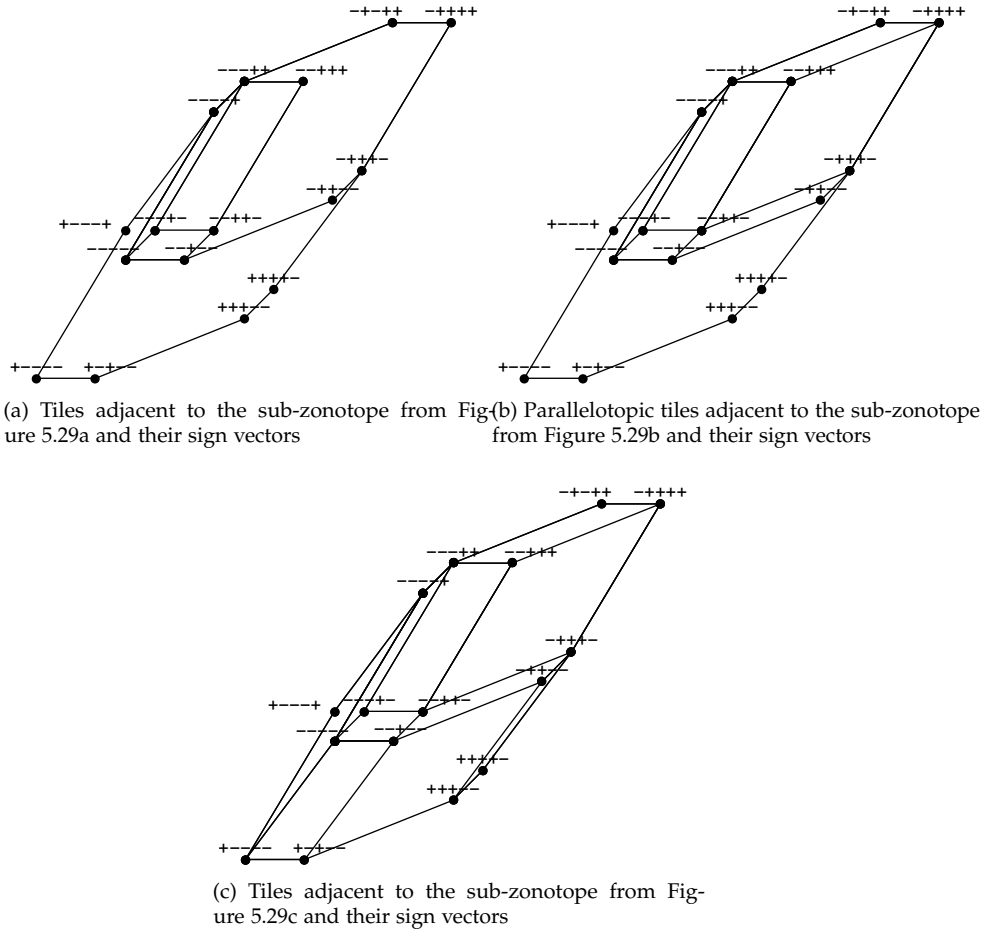


Figure 5.30 – Illustrating one-by-one all parallelotopic tiles being enumerated

Freeing the already fixed generators. Once we have generated a parallelotopic tile we stop fixing the sign of the generators since we have enumerated sufficiently many non-existent vertices (which are not extremal points of the primitive zonotope) to construct the remaining parallelotopic tiles. We shall be using this tile to generate the others which is the second part of the algorithm i.e., “finding all tiles adjacent to $\mathfrak{Z}_{\text{sub}}$ ” in Algorithm 5.2.

Enumerating the remaining tiles of $\mathfrak{Z}(V \setminus \{1^- 2^-\})$. The free generators of the sub-zonotope $\mathfrak{Z}(V \setminus \{1^- 2^-\})$ (the one whose extremal points are marked in red, shown in Figure 5.29b) from which the first parallelotopic tile is obtained are $(2, 0)$, $(1, 1)$ and $(3, 5)$ respectively. The third generator $(2, 0)$ of the primitive zonotope $\mathfrak{Z}(V)$ was fixed in order to generate the tile. Now, we free the third generator (v_3) and for all subset of generators v_4, v_5 of length $p - 1$ we compute the parallelotopes. That means we can generate two parallelotopic tiles with the following free generators’ combination, i.e., (v_3, v_4) and (v_3, v_5) . They correspond to the two parallelotopic tiles

adjacent to our first tile $\mathfrak{Z}(\mathbb{V} \setminus \{1^- 2^- 3^-\})$. Notice that finding the parallelotopes for a given free generator combination is straightforward and below we explain it in detail. Consider the above-mentioned free generation combination (v_3, v_4) with the vectors v_1, v_2 and v_5 being fixed. Now, in order to enumerate a parallelotope we examine the sign vectors corresponding to the vertices of the sub-zonotope $\mathfrak{Z}(\mathbb{V} \setminus \{1^- 2^-\})$ which are $(-, -, -, -, -)$, $(-, -, -, -, +)$, $(-, -, -, +, +)$, $(-, -, +, +, +)$, $(-, -, +, -, -)$, $(-, -, +, +, -)$, $(-, -, -, +, -)$. Observe that few of these sign vectors are marked in red and those characterize a parallelotope: $(-, -, -, -, -)$, $(-, -, +, -, -)$, $(-, -, +, +, -)$, $(-, -, -, +, -)$. That means we check for the fixed generators v_1, v_2 and v_5 which are the sign vectors which match among themselves so that they form a parallelotope⁸. Similarly, when vectors v_3 and v_5 are free, the matching sign vectors are: $(-, -, -, +, +)$, $(-, -, +, +, +)$, $(-, -, +, +, -)$, $(-, -, -, +, -)$, which enumerates the another parallelotopic tile. The center of these tiles can be determined as

$$(18, 12) + 0(2, 0) + 0(1, 1) + (-1)(3, 5)$$

and

$$(18, 12) + 0(2, 0) + (+1)(1, 1) + 0(3, 5)$$

where $(18, 12)$ is the center of the sub-zonotope $\mathfrak{Z}(\mathbb{V} \setminus \{1^- 2^-\})$ (the one whose extremal points are marked in red in Figure 5.29b) to which these tiles belong. The center of these two tiles can also be determined as

$$(20, 10) + (-1)(-3, -4) + (-1)(5, 2) + 0(2, 0) + 0(1, 1) + (-1)(3, 5)$$

and

$$(20, 10) + (-1)(-3, -4) + (-1)(5, 2) + 0(2, 0) + (+1)(1, 1) + 0(3, 5)$$

where $(20, 10)$ is the center of the primitive zonotope $\mathfrak{Z}(\mathbb{V})$. The sign vectors of the extremal points of these two tiles are $(-, -, -, -, -)$, $(-, -, +, -, -)$, $(-, -, +, +, -)$, $(-, -, -, +, -)$ (for free generators v_3, v_4) and $(-, -, -, +, -)$, $(-, -, +, +, -)$, $(-, -, +, +, +)$, $(-, -, -, +, +)$ (for free generators v_3, v_5) respectively as shown in Figure 5.30a.

Enumerating the remaining tiles of $\mathfrak{Z}(\mathbb{V} \setminus \{1^-\})$. As far, we have tiled the sub-zonotope $\mathfrak{Z}(\mathbb{V} \setminus \{1^- 2^-\})$ of Figure 5.29b (extremal points marked in red) into three possible parallelotopic tiles. Now, we shall find the other adjacent tiles of the sub-zonotope $\mathfrak{Z}(\mathbb{V} \setminus \{1^-\})$ (shown in Figure 5.29a with vertices noted in red) whose free generators are $(5, 2)$, $(2, 0)$, $(1, 1)$ and $(3, 5)$ respectively. Notice that the second generator, i.e., $(5, 2)$ relative to the primitive zonotope was fixed in order to generate the sub-zonotope $\mathfrak{Z}(\mathbb{V} \setminus \{1^- 2^-\})$. Now, we free this generator (v_2) and for all subset of generators v_3, v_4, v_5 of length $p-1$ we compute three parallelotopic tiles with the following combinations, i.e., $(v_2, v_5), (v_2, v_4)$ and (v_2, v_3) . They correspond to the three

⁸Here $p=2$, so there are four sign vectors corresponding to four vertices of the parallelotope. Accordingly, when $p=3$, there will be eight sign vectors corresponding to eight vertices of a 3-parallelotope.

parallelotopic tiles adjacent to the sub-zonotope $\mathfrak{Z}(V \setminus \{1^- 2^-\})$ (extremal points noted in red in Figure 5.29b). The center of these tiles can be determined as

$$(23, 14) + 0(5, 2) + (+1)(2, 0) + (+1)(1, 1) + 0(3, 5),$$

$$(23, 14) + 0(5, 2) + (+1)(2, 0) + 0(1, 1) + (-1)(3, 5)$$

and

$$(23, 14) + 0(5, 2) + 0(2, 0) + (+1)(1, 1) + (+1)(3, 5)$$

where $(23, 14)$ is the center of the sub-zonotope $\mathfrak{Z}(V \setminus 1^-)$ (the one whose extremal points are marked in red in Figure 5.29a) to which these tiles belong. The sign vectors of the vertices of these three tiles are $(-, -, +, +, +)$, $(-, -, +, +, -)$, $(-, +, +, +, -)$, $(-, +, +, +, +)$ (for free generators v_2, v_5), $(-, -, +, -, -)$, $(-, +, +, +, -)$, $(-, +, +, -, -)$, $(-, -, +, +, -)$ (for free generators v_2, v_4) and $(-, -, -, +, +)$, $(-, +, +, +, +)$, $(-, +, -, +, +)$, $(-, -, +, +, +)$ (for free generators v_2, v_3) respectively as shown in Figure 5.30b.

All the $\binom{5}{2}$ parallelotopic tiles. At this time, the only remaining parallelotopic tiles to be enumerated are the ones which are adjacent to the sub-zonotope $\mathfrak{Z}(V \setminus 1^-)$. We free the first generator v_1 of the primitive zonotope which leaves four combinations of free generators which are (v_1, v_5) , (v_1, v_3) , (v_1, v_2) and (v_1, v_4) respectively. These combinations will produce the remaining four parallelotopic tiles whose center can be characterized as

$$(20, 10) + 0(-3, -4) + (-1)(5, 2) + (-1)(2, 0) + (-1)(1, 1) + 0(3, 5),$$

$$(20, 10) + 0(-3, -4) + (+1)(5, 2) + (+1)(2, 0) + 0(1, 1) + (-1)(3, 5),$$

$$(20, 10) + 0(-3, -4) + (-1)(5, 2) + 0(2, 0) + (-1)(1, 1) + (-1)(3, 5)$$

and

$$(20, 10) + 0(-3, -4) + 0(5, 2) + (+1)(2, 0) + (-1)(1, 1) + (-1)(3, 5)$$

where $(20, 10)$ is the center of the primitive zonotope $\mathfrak{Z}(V)$. All these tiles are illustrated in Figure 5.30c whose vertices have the following sign vectors: $(-, -, -, -, +)$, $(-, -, -, -, -)$, $(+, -, -, -, -)$, $(+, -, -, -, +)$ (for free generators v_1, v_5), $(-, +, +, +, -)$, $(-, +, +, -, -)$, $(+, +, +, -, -)$, $(+, +, +, +, -)$ (for free generators v_1, v_4), $(-, -, -, -, -)$, $(-, -, +, -, -)$, $(+, -, -, -, -)$, $(+, -, +, -, -)$ (for free generators v_1, v_3) and $(-, -, +, -, -)$, $(-, +, +, -, -)$, $(+, -, +, -, -)$, $(+, +, +, -, -)$ (for free generators v_1, v_2).

EXAMPLE 5.36 Now, we illustrate the different steps of the tiling algorithm on a 3-dimensional zonotope shown in Figure 5.31a. For $n = 5$ and $p = 3$, the zonotope in Figure 5.31a is the concretization of the affine forms $X = (\hat{x}, \hat{y}, \hat{z})$

with $\hat{x} = 0 - 1\varepsilon_1 + 1\varepsilon_4 + 1\varepsilon_5$, $\hat{y} = 0 + 2\varepsilon_1 + 1\varepsilon_3 + 1\varepsilon_5$, $\hat{z} = 0 - 4\varepsilon_1 + 1\varepsilon_3 + 1\varepsilon_5$, that is,

$$A^T = \begin{pmatrix} 0 & -1 & 0 & 0 & 1 & 1 \\ 0 & 2 & 0 & 1 & 0 & 1 \\ 0 & -4 & 1 & 0 & 0 & 1 \end{pmatrix} \quad (5.37)$$

or we would write $A = ((0,0,0), (-1,2,4), (0,0,1), (0,1,0), (1,0,0), (1,1,1))$.

We denote the original zonotope by $\mathfrak{Z}(V)$ where $V = ((-1,2,4), (0,0,1), (0,1,0), (1,0,0), (1,1,1))$. The very first step is to compute the sign vectors corresponding to the extremal points of the zonotope using the reverse search algorithm. Then we keep fixing the sign of the generators prior to obtaining a 3-parallelotope or 3-dimensional parallelotope. Once a 3-parallelotopic tile is enumerated, we have listed sufficient number of interior vertices of the primitive zonotope. Fixing the first generator to '-' will generate a sub-zonotope $\mathfrak{Z}(V \setminus 1^-)$ (the zonotope in green with extremal points marked in red shown in Figure 5.31b) given by the vectors $((0,0,1), (0,1,0), (1,0,0), (1,1,1))$ whose center can be determined as

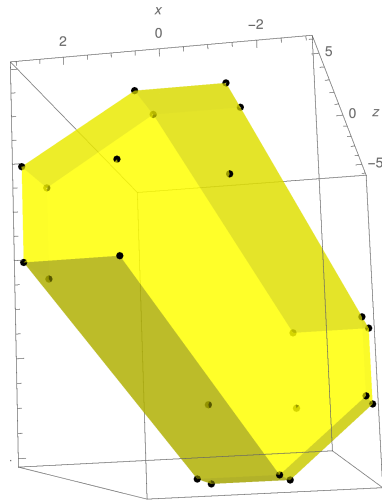
$$\begin{aligned} & (0, 0, 0) + (-1)(-1, 2, -4) + 0(0, 0, 1) + \\ & 0(0, 1, 0) + 0(1, 0, 0) + 0(1, 1, 1) \end{aligned}$$

where $(0, 0, 0)$ is the center of the primitive zonotope. We repeat the procedure again and this time we obtain a parallelotope $\mathfrak{Z}(V \setminus 1^- 2^-)$ (the zonotope in blue shown in Figure 5.31c) with the following vectors $((0,1,0), (1,0,0), (1,1,1))$. The center of the parallelotopic tile can be characterized as

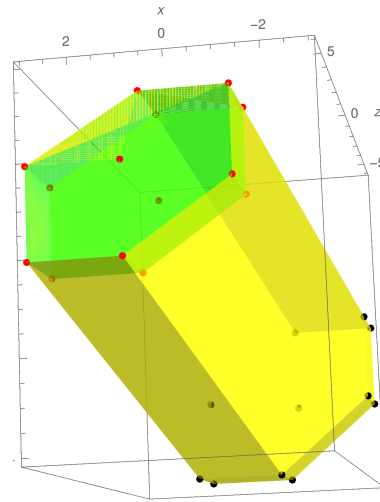
$$\begin{aligned} & (1, -2, 4) + (-1)(0, 0, 1) + 0(0, 1, 0) + 0(1, 0, 0) + \\ & 0(1, 1, 1) \end{aligned}$$

where $(1, -2, 4)$ is the center of the sub-zonotope $\mathfrak{Z}(V \setminus 1^-)$. Heretofore, we have enumerated enough hidden vertices to construct the remaining adjacent tiles. Henceforth, we free the generator which was fixed to obtain the first tile, and we enumerate its adjacent tiles which are illustrated one at a time in Figure 5.32a-5.32d.

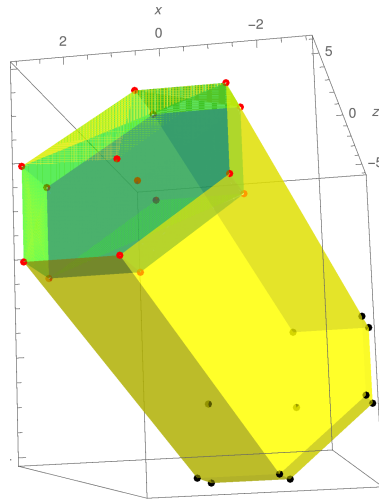
The free generators of the sub-zonotope $\mathfrak{Z}(V \setminus 1^-)$ (zonotope in green in Figure 5.31b) from which the first parallelotopic tile is obtained are $(0, 0, 1)$, $(0, 1, 0)$, $(1, 0, 0)$ and $(1, 1, 1)$ respectively. The sign of the second generator, i.e., $(0, 0, 1)$ with respect to the primitive zonotope $\mathfrak{Z}(V)$ was fixed to generate the tile. Now, we free this generator (v_2) and for all subset of generators v_2, v_3, v_4, v_5 of length $p - 1$ we compute the parallelotopes. That means we can generate three parallelotopic tiles with the following free generators' combination, i.e., (v_2, v_4, v_5) , (v_2, v_3, v_5) and (v_2, v_3, v_4) . They correspond to the three parallelotopic tiles in Figure 5.32a-5.32d adjacent to our first tile $\mathfrak{Z}(V \setminus \{1^- 2^-\})$. In the similar manner we construct all the remaining adjacent 3-parallelotopic tiles shown in Figure 5.33a-5.33f. The final tiling is illustrated in Figure 5.34.



(a) Zonotope concretization for the matrix in (5.37)

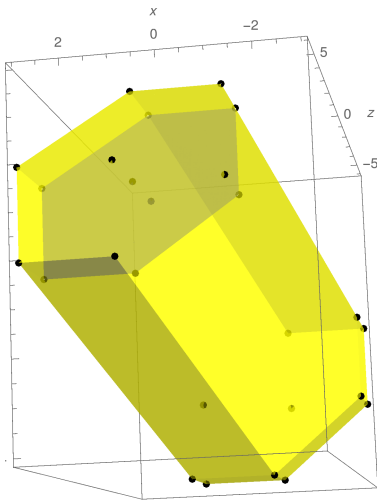


(b) Sub-zonotope in green after fixing the first generator to '-'

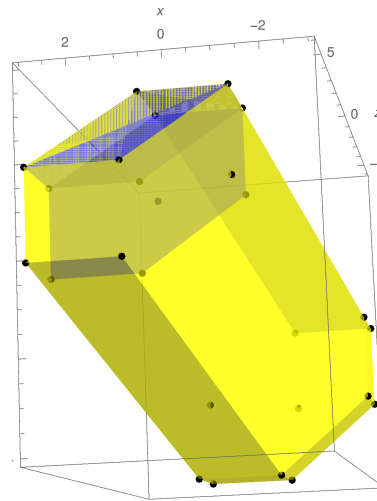


(c) Sub-zonotope in blue after fixing the first and second generators to '-'

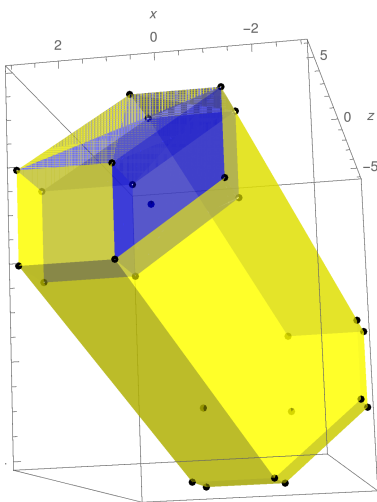
Figure 5.31 – Illustrating one-by-one all sub-zonotopes obtained after fixing the sign of generators



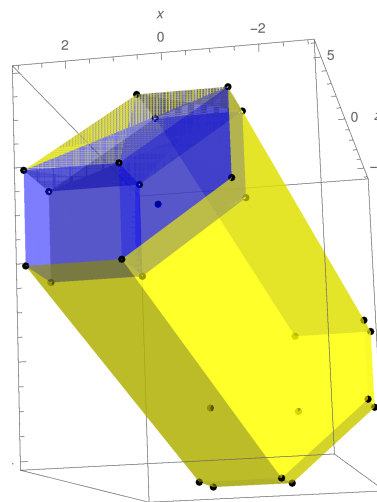
(a) First parallelotopic tile enumerated after fixing the first and second generators to \vec{v}



(b) Parallelotopic tile adjacent to the tile from Figure 5.32a

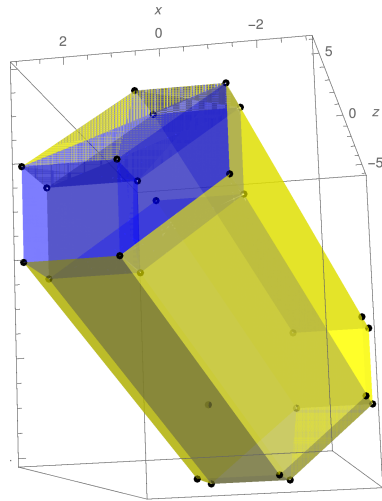


(c) Parallelotopic tile adjacent to the tile from Figure 5.32b

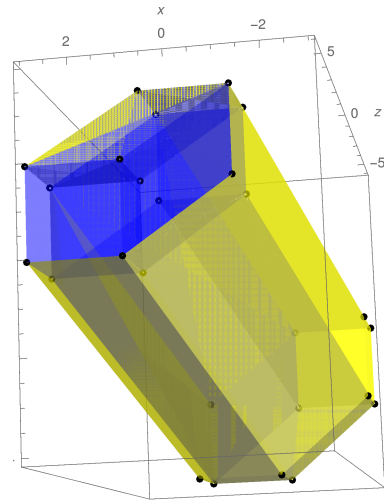


(d) Parallelotopic tile adjacent to the tile from Figure 5.32c

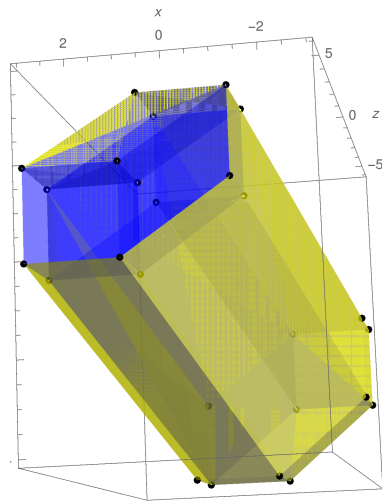
Figure 5.32 – Illustrating one-by-one all parallelotopic tiles being enumerated



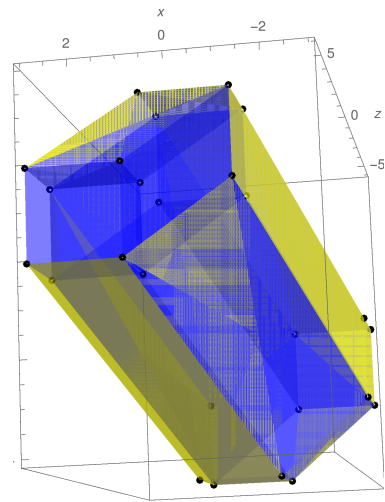
(a) First parallelotopic tile of the sub-zonotope in green from Figure 5.31b



(b) Parallelotopic tile adjacent to tile in Figure 5.33a

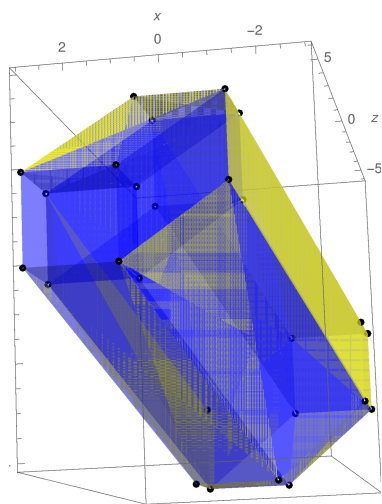


(c) Parallelotopic tile adjacent to tile in Figure 5.33b

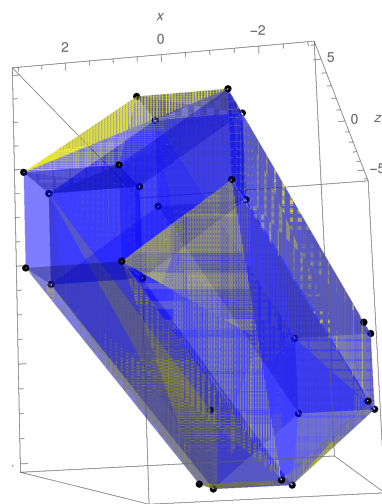


(d) Parallelotopic tile adjacent to tile in Figure 5.33c

Figure 5.33 – Illustrating one-by-one all parallelotopic tiles being enumerated



(e) Parallelotopic tile adjacent to tile in Figure 5.33d



(f) Parallelotopic tile adjacent to tile in Figure 5.33e

Figure 5.33 – Illustrating one-by-one all parallelotopic tiles being enumerated

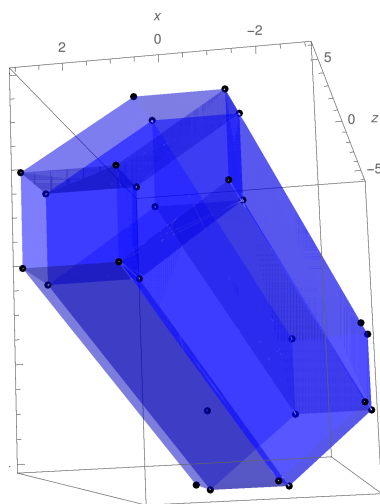
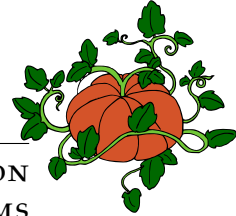


Figure 5.34 – 3-dimensional parallelotopic tiles delineating the zonotope in Figure 5.31a.



IMPLEMENTATION AND EXPERIMENTS ON PROGRAMS

All the elements presented in this chapter have been implemented as a prototype analyzer. This analyzer was already developed in [MBR16] which implemented the algorithm of Section 4.2 using the boxes and octagons abstract domains of the Apron [JM09] library. We adapted the prototype analyzer, as described in Section 5.7, in order to use the zonotopic abstract domain [GGP09] and polyhedra abstract domain New Polka of Apron. We added the operations split, intersection test, meet, coverage measure and the new inclusion test described in Section 5.1 in Apron for the zonotopic abstract domain. The analyzer is composed of two parts:

- The core mathematical computations are done with Apron library (a C library of numerical abstract domains), from an OCaml front-end. This part implements the set of functions, for e.g., test for inclusion, intersection, splitting for every abstract domain. Computation in Apron are done using double precision floats.
- The front-end is an OCaml code. It implements the algorithm presented in Section 4.2. It loads the different abstract elements and interacts with Apron to compute the different sequence of calls to Apron functions.

6.1 IMPLEMENTATION

6.1.1 *Apron*

One of the most important components of the analyzer is the Apron library. This C library presents a set of numerical abstract domains, namely Boxes, Octagons, Polyhedra and Zonotopes. It permits bindings for languages like C, C++ and OCaml. All the abstract domains in Apron come with a common interface, allowing to switch from one domain to another without any additional effort. In that way, one can easily compare results of different numerical domains, or combine them for a better precision.

6.1.2 *Taylor1+*

The domain for zonotopes in Apron is called Taylor1+ [GGP09, Gho11]. Its API (Application Programming Interface) is fully compliant with the Apron library. Any analyzer using Apron library can avail Taylor1+ with all its functions.

Data structure of Taylor1+. An affine set in Taylor1+ is represented by an array of pointers of size p (the number of the numerical variables) and a generic abstract object for the noise symbols. An affine form is encoded by a special structure pointed by each pointer. The data structure of an affine form is a coefficient plus a list of terms. Each term contains a non-zero coefficient and a pointer to a noise symbol. The data structure of an affine set is a matrix of size $p \times n$ (n is the number of noise symbols) where the first column of the matrix is the center corresponding to the geometric concretization of the affine set and the rest of the columns are the generators. The entries of the matrix are extracted from the terms of the affine forms using a hash table where the keys are the noise symbols and the values are the coefficients associated with each of these noise symbols.

New noise symbols are created for the affine forms after every split operation.

6.1.3 Our contribution with respect to implementation

- We had to implement test for inclusion and intersection, splitting and meet operators in order to extend the analyzer with zonotope abstract domain.
- Initially, the analyzer was designed to handle only two sub-parts after every split operation. Since, our split operator on zonotopes can produce more than two sub-zonotopes, we had to modify the analyzer to handle multiple sub-parts.
- Originally, the analyzer computed the volume of the abstract elements to calculate the coverage information. In our version, we have implemented the coverage metric and the test for benign, as described in Section 5.7. This modified version of the analyzer is compliant with different abstract domains (tested for boxes, octagons, polyhedra and zonotopes).
- Apron is a C library of numerical abstract domains mainly used for static analysis of programs by abstract interpretation. So, obviously there is no split operation implemented for any abstract domain in Apron. For boxes and octagons, the authors in [MBR16] implemented the respective split operation directly within the OCaml code of the analyzer. In an effort to make the split operator more generic, we implemented the tiling function for zonotopes inside Apron. However, in order to interact with the analyzer we had to write the OCaml bindings.

The code of the analyzer with the new elements and the benchmarks are released under a GPL license and is available at https://github.com/bibekkabi/Prototype_analyzerwithApron. The APRON library with all the new set-theoretic operations is available at <https://github.com/bibekkabi/taylor1plus>.

6.2 EXPERIMENTS

Experiments were conducted on a set of programs with non-linear loops that present complex, possibly non-convex, invariants. These programs were

extracted from [MBR16], [AGG10], [RG13], [DHKT12]. A number of these programs are typical model-checking benchmarks, for example from SV-COMP¹.

In Table 6.1, we compare on some small but challenging loops the results of the algorithm applied with boxes, octagons, polyhedra and zonotopes. For each abstract domain, we give the number of iterations and time until a first inductive invariant is found, and the number of elements that compose this invariant. Table 6.1 illustrates the results, with tightening (Equation (4.2)) used only during the first iteration.

For each example, we highlight in bold in Table 6.1 the entries corresponding to the smaller number of elements, iterations, or execution time.

Program	Boxes			Octagons			Zonotopes			Polyhedras		
	#elems.	#iters.	time(s)	#elems.	#iters.	time(s)	#elems.	#iters.	time(s)	#elems.	#iters.	time(s)
Octagon	752	2621	0.1042	752	2756	0.6115	1	1	0.0001	1	1	0.0001
Filter	238	1310	0.1029	74	736	0.2105	38	222	0.5020	42	312	0.2554
Arrow-Hurwicz	1784	1643	0.4033	369	931	0.5147	15	38	0.0235	134	484	1.0059
Filter2	14	58	0.0034	7	13	0.0013	8	16	0.0045	1	1	0.0009
Harm	87	438	0.0112	88	448	0.0647	60	254	0.5143	53	243	0.2442
Harm-reset	87	438	0.0204	88	446	0.1478	60	268	0.9717	53	253	0.3867
Harm-saturated	23	15	0.0011	24	16	0.0112	9	14	0.0157	5	9	0.0124
Lead-lag	-	-	-	-	-	-	-	-	-	-	-	-
Lead-lag-reset	-	-	-	-	-	-	-	-	-	-	-	-
Lead-lag-saturated	-	-	-	-	-	-	-	-	-	-	-	-
Sine	240	1448	0.4395	154	348	0.1102	21	33	0.0547	136	286	1.1145
Square root	7	10	0.0005	4	4	0.0016	1	1	0.0001	4	4	0.0066
Newton	200	102	0.1097	158	76	0.1785	11	17	0.0197	64	26	2.0660
Newton2	1806	499	6.6861	709	430	2.2207	8	6	0.0193	12	12	2.7498
Corner	129781	1847	646.8494	129767	1847	8850.8766	488	999	35.6245	2368	4248	126.7980

Table 6.1 – Experimental results with tightening applied only during first iteration.

Example *Octagon* in Table 6.1 corresponds to the motivational example from [MBR16]. Its loop body performs a 45-degree rotation around the origin, with a slight inward scaling. The initial element obtained after the first tightening step is already inductive with zonotopes and polyhedras. The classical abstract semantics for addition and subtraction on octagons is too coarse to prove this is an inductive invariant, explaining why the analyzer had to iterate a lot, contrarily to the zonotopic case.

Filter is a second-order digital filter (Figure 2.1) taken from [MBR16]. The candidate invariant provided to the algorithm is $[-4, 4]$ which is not inductive. For *Filter2* from [AGG10], our inductive invariant shows that x and y remain within $[-0.2, 1]^2$.

Figure 6.1a, 6.1b, 6.1c and 6.1d compare the result of the algorithm on the *Filter* program using intervals, octagons, polyhedras and zonotopes. Natural inductive invariants of such filters are ellipsoids. The inductive invariant found within each abstraction is indeed the approximation of an ellipsoid. It is composed of fewer zonotopes and polyhedra than boxes and, to a lesser extent, octagons, and requires fewer iterations to be synthesized. In order to compare the inductive invariants obtained, we computed the area covered by the invariant for each domain: they are respectively 28.8125, 28.875, 24.1343 and 20.732 for boxes, octagons, polyhedras and zonotopes (see Figures 6.1e–6.1g), so the inductive invariant inferred by zonotopes is tighter than that with boxes, octagons and polyhedras.

¹<https://github.com/sosy-lab/sv-benchmarks/tree/master/c/floats-cdfpl>

We analyzed the Arrow-Hurwicz² loop (taken from [AGG10]) as a two-variable program by eliminating the loop condition and the two variables u and v that are not needed in the body of the loop; the algorithm with boxes, octagons, polyhedras and zonotopes was able to verify that the variables x and y remain within the bound $[-1.73, 1.73]^2$. The analysis with zonotopes was faster and composed of far fewer abstract elements compared to other domains. We were not able to verify the bounds using boxes, octagons and polyhedras even with tightening. However, for a wider goal compared to $[-1.7, 1.7]^2$, the algorithm could infer an inductive invariant with all four abstract domains.

Harm is a harmonic oscillator program from [AGG10]. Its loop body is close to identity. The programs *Harm-reset* and *Harm-saturated* add some non-determinism in the body loop. The polyhedras require fewer elements and iterations, but more time, compared to boxes and octagons. On too simple cases, the use of complicated abstraction is not competitive, as expected.

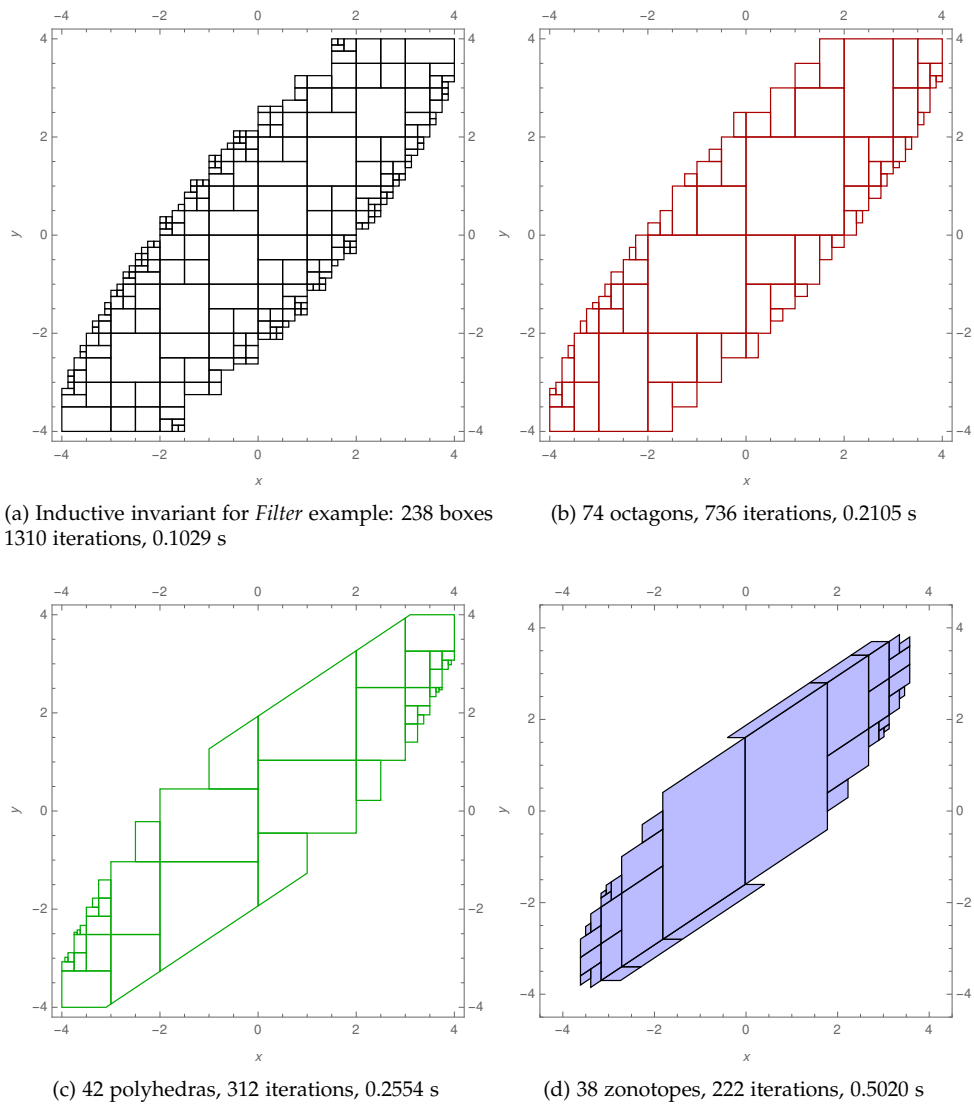
For the lead-lag controllers (program *Lead-lag*, *Lead-lag-reset* and *Lead-lag-saturated* from [RG15]), none of the abstract domains could find an inductive invariant before timeout while using tightening only during the first iteration.

Sine, *Square root*, *Newton*, *Newton2* (taken from [DHKT12]) and *Corner* are programs with non-linear loop bodies. *Sine* computes the corresponding mathematical function through Taylor expansion and *Square root* computes the polynomial interpolation function for square root, while *Newton* and *Newton2* perform one step and two steps of Newton solving respectively. In all the four cases, the inductive invariant (see Figure 6.2, 6.3 and 6.4) inferred by the algorithm matches closely the graph of the function, ensuring functional correctness.

Zonotopes are the fastest on all these examples: they require much fewer iterations and elements compared to intervals, octagons and polyhedras. For the *Sine* program, we check bounds on the result of computing a sine approximation under the input range $x = [-\frac{\pi}{2}, \frac{\pi}{2}]$ or $x = [-1.57079632679, 1.57079632679]$ in radians. The inductive invariant must include both the initial state, where $y = 0$, and the end state, where y is the approximate sine. The candidate invariant provided to the algorithm is $[-1.05, 1.05]$ (the results shown in Figures 6.2a, 6.2b, 6.2c and 6.2d are within the bound $[-1.05, 1.05]$ on the x -axis). Figure 6.2a, 6.2b, 6.2c and 6.2d compare the result of the algorithm on the *Sine* program using intervals, octagons, polyhedras and zonotopes. The area covered by the inductive invariant for boxes, octagons and polyhedras are 4.9957, 4.9752, 4.5822, which are tighter compared to 5.1207 obtained with zonotopes (see Figures 6.2e–6.2g). According to the authors in [DHKT12], the actual maximum of the function lies at about 1.00921. We also ran the experiment with the candidate invariant as $[-1.00921, 1.00921]$, and the algorithm was only able to find an inductive invariant with zonotopes.

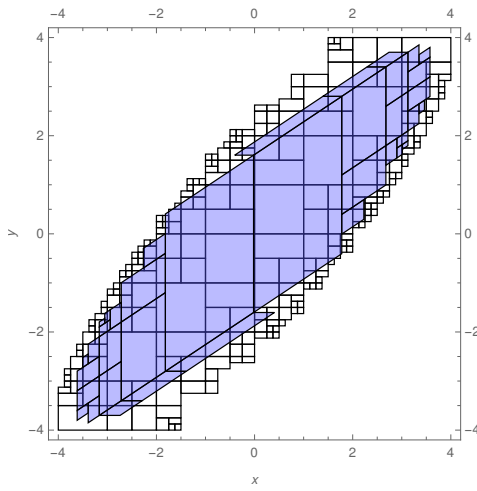
Similar is the analysis (inductive invariant for boxes, octagons and polyhedras are relatively tighter compared to zonotopes) in case of *Newton* (see Figures 6.3e–6.3g) and *Newton2* (see Figures 6.4e–6.4g). Our zonotopes-based method is better for the specific aim of proving as fast as possible that the ini-

²Arrow-Hurwicz is an algorithm to compute both primal and dual solutions for convex constrained optimization problems.

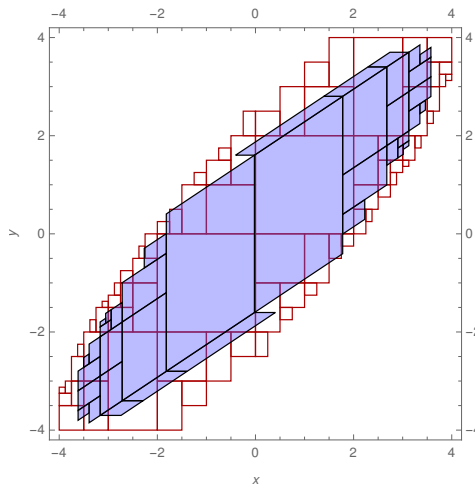
Figure 6.1 – Inductive invariant for *Filter* example

tial invariant holds, strengthening it into an inductive invariant. Indeed, with a better interpretation of non affine operations, less splitting steps are needed before getting an inductive invariant. For example, zonotopes enabled us to prove the initial invariant as inductive for the *Square root* program. Finally, zonotopes are again faster in the case of the *Newton* and *Newton2* programs compared to intervals, octagons and polyhedras.

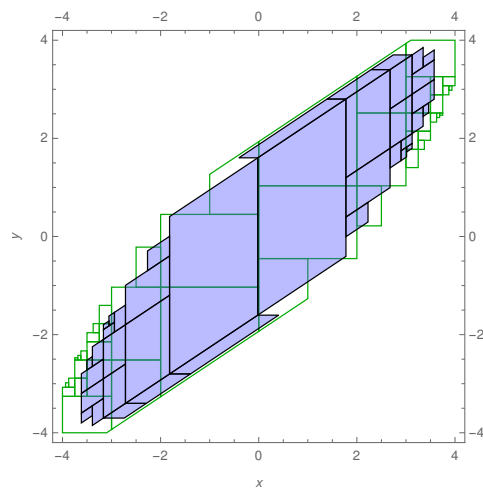
Corner (similar benchmarks are considered in [Mar14]) is the program in Figure 1.1 that presents a non-convex inductive invariant, with a hole in the middle. It is important to note that, unlike constraint solvers, but similarly to iteration with widening, the CP algorithm may fail to find an inductive



(e) Superimposing Fig. 6.1a and 6.1d



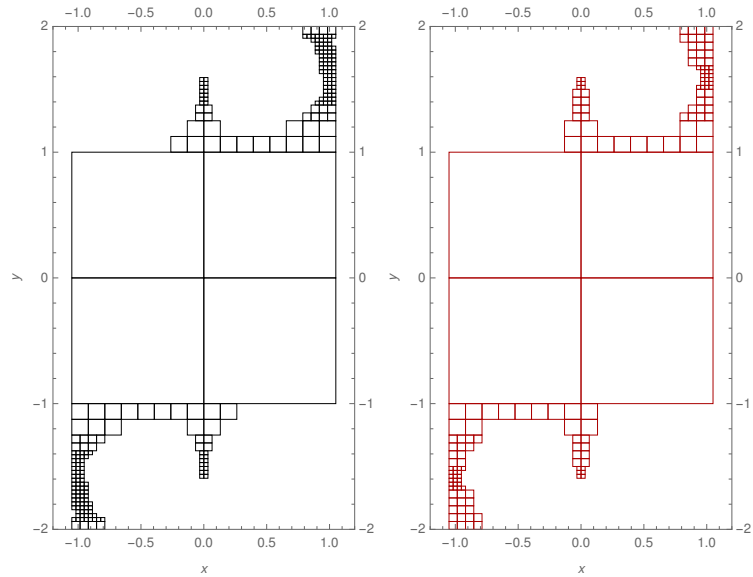
(f) Superimposing Fig. 6.1b and 6.1d



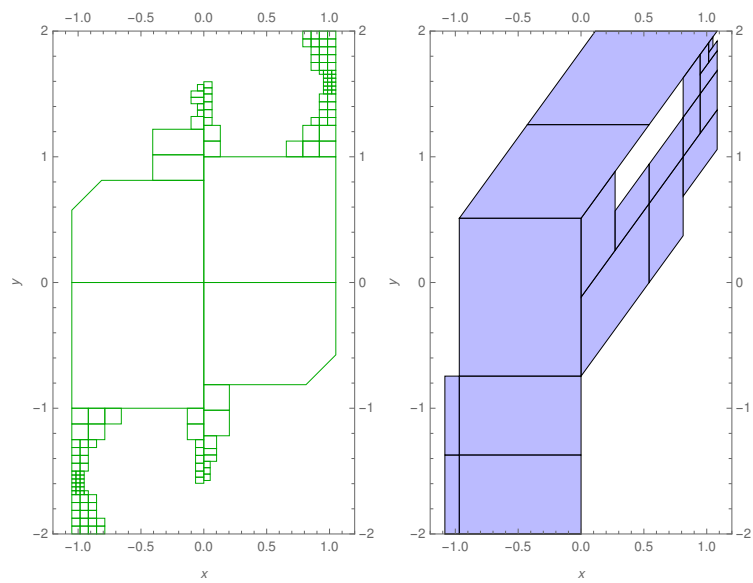
(g) Superimposing Fig. 6.1c and 6.1d

Figure 6.1 – Inductive invariant for *Filter* example

6. IMPLEMENTATION AND EXPERIMENTS ON PROGRAMS

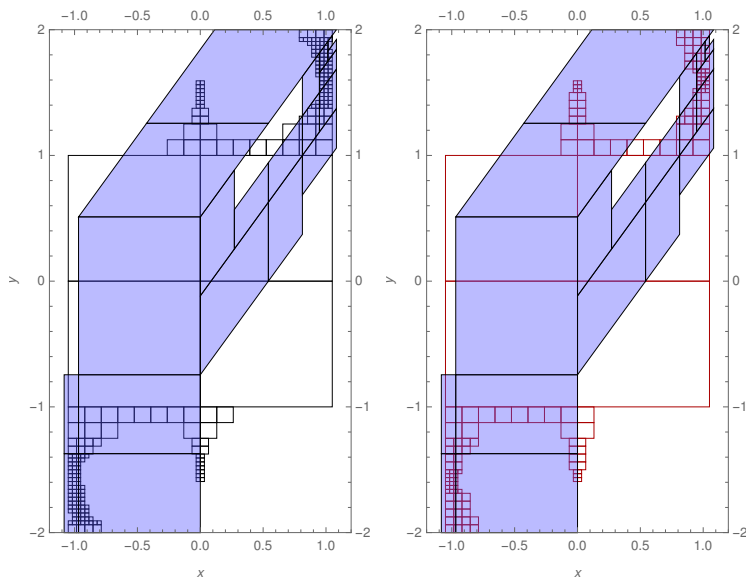


(a) Inductive invariant for *Sine* example (b) 154 octagons, 348 iterations, 0.1102 s
238 boxes 1448 iterations, 0.4395 s

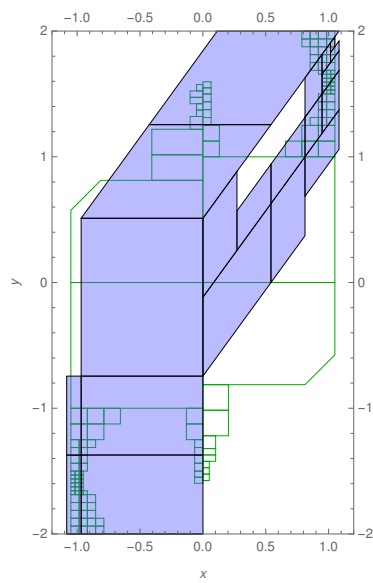


(c) 136 polyhedras, 286 iterations, 1.1145 s (d) 21 zonotopes, 33 iterations, 0.0547 s

Figure 6.2 – Inductive invariant for *Sine* example

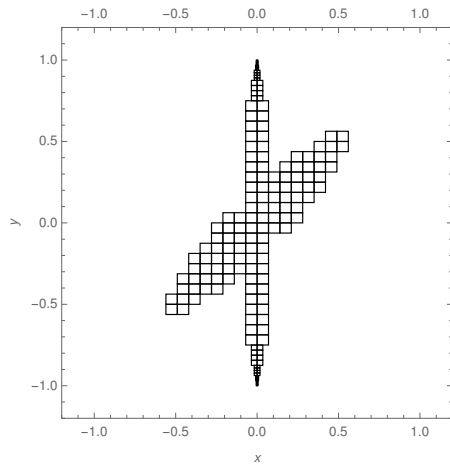


(e) Superimposing Fig. 6.2a and 6.2d (f) Superimposing Fig. 6.2b and 6.2d

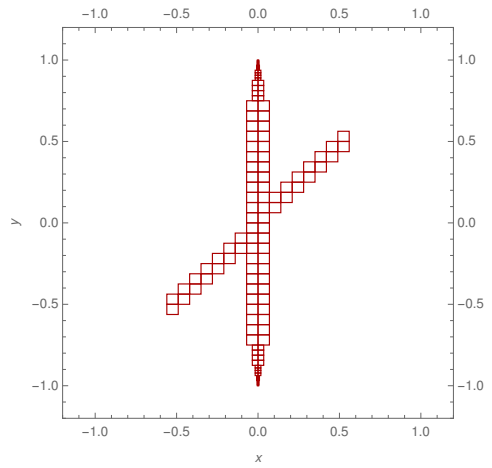


(g) Superimposing Fig. 6.2c and 6.2d

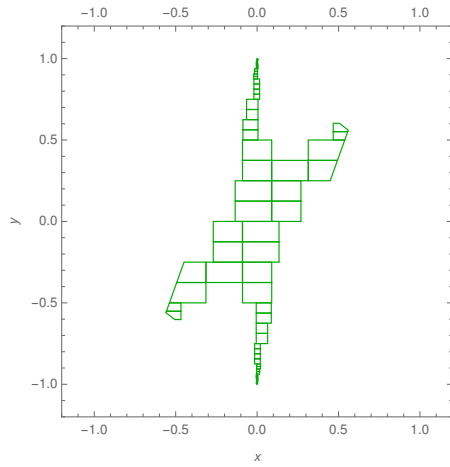
Figure 6.2 – Inductive invariant for *Sine* example



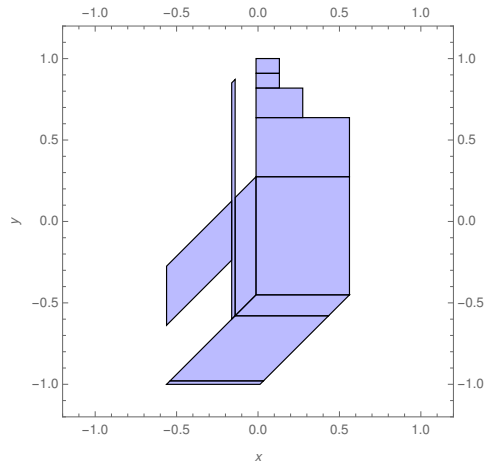
(a) Inductive invariant for *Newton* example: 200 boxes 102 iterations, 0.1029 s



(b) 158 octagons, 76 iterations, 0.1785 s

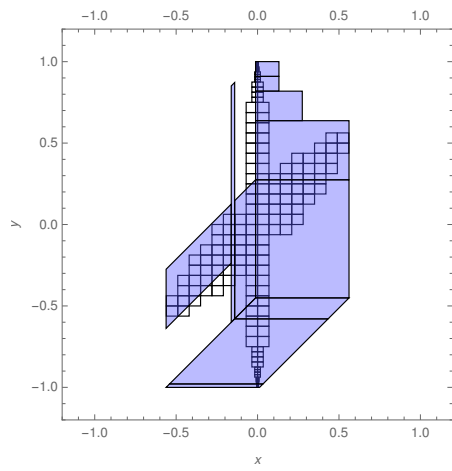


(c) 64 polyhedras, 26 iterations, 2.0660 s

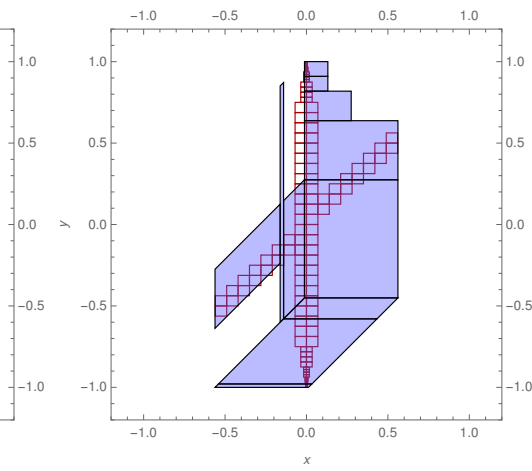


(d) 11 zonotopes, 17 iterations, 0.0197 s

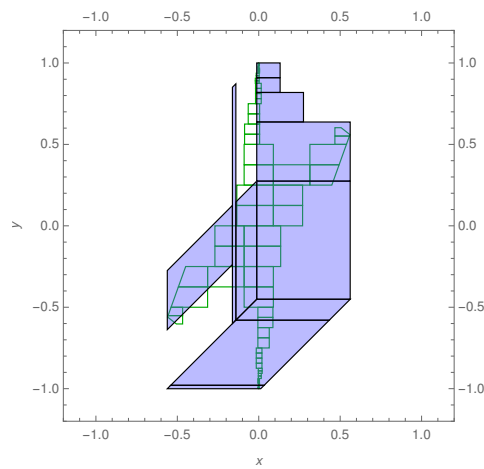
Figure 6.3 – Inductive invariant for *Newton* example



(e) Superimposing Fig. 6.3a and 6.3d



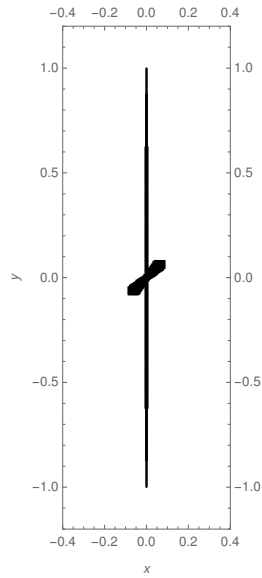
(f) Superimposing Fig. 6.3b and 6.3d



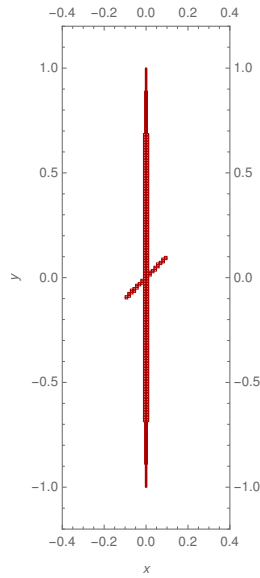
(g) Superimposing Fig. 6.3c and 6.3d

Figure 6.3 – Inductive invariant for *Newton* example

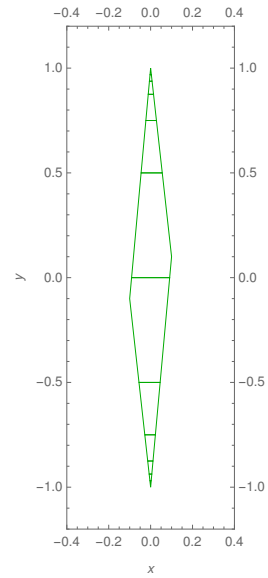
6. IMPLEMENTATION AND EXPERIMENTS ON PROGRAMS



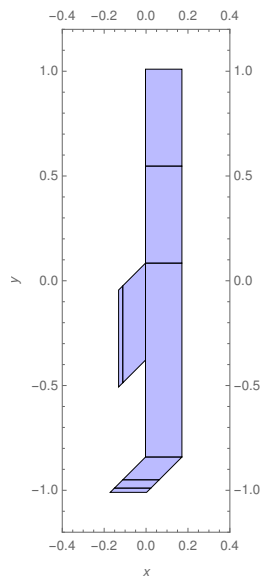
(a) Inductive invariant for *Newton2* example: 1806 boxes 499 iterations, 6.6861 s



(b) 709 octagons, 430 iterations, 2.2207 s

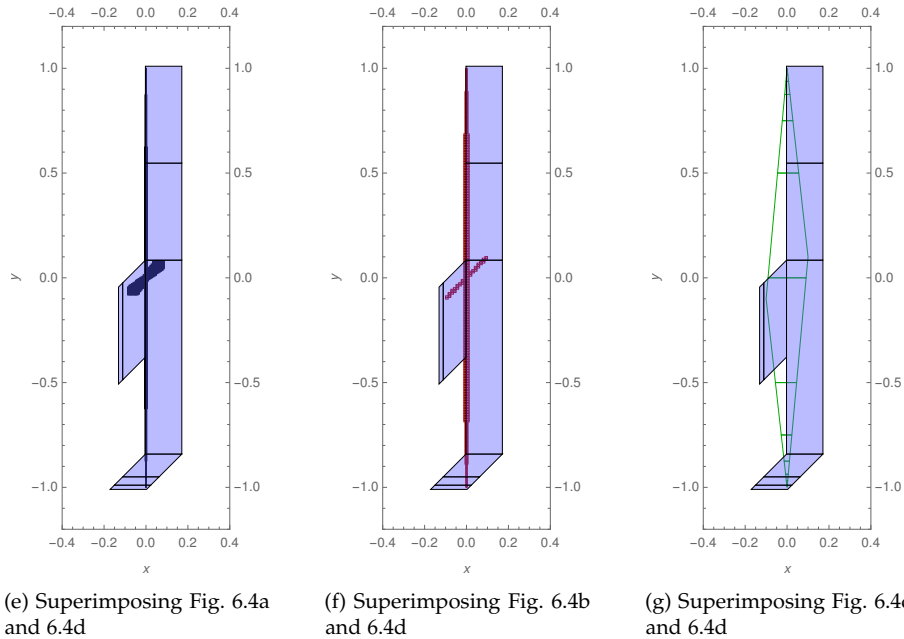


(c) 12 polyhedras, 12 iterations, 2.7498 s



(d) 8 zonotopes, 6 iterations, 0.0193 s

Figure 6.4 – Inductive invariant for *Newton2* example

Figure 6.4 – Inductive invariant for *Newton2* example

invariant, even if there exists one that can be represented with the abstract elements whose size is greater than a user-specified size. A way to avoid this is to lower the value of user-specified size, but this change can also make the algorithm slower, because it can now spend more time splitting boxes that cannot be part of an inductive invariant.

For the *Corner* program, while using the box/octagon abstract domain, the image by the loop body in the abstract domain, could not be within the collection of elements, and hence not proven invariant. A natural fix was to lower the value of user-specified threshold for size (ϵ_s) and apply the failure recovery steps. Thus, boxes and octagons rely heavily on re-splitting, hence output a large set of elements and have a comparatively higher run-time compared to polyhedra and zonotopes.

These experiments confirm that zonotopes provide a very interesting trade-off between a general purpose abstraction, that stands the comparison to simpler abstractions on basic linear examples, and the abstraction of more complex, non affine, behaviors where it is faster and more flexible. In particular, the fact that they allow to represent inductive invariants with fewer elements will be even more crucial for the scalability of the approach to higher dimension programs.

Tightening at every iteration. Recall that in Table 6.1 tightening was only applied during the first iteration. Now, we shall illustrate the results with tightening after each split operation for boxes, octagons and polyhedras except for zonotopes (only during the first iteration as shown in Table 6.1). As already discussed in Section 4.2, the precision and efficiency of the meet operation

plays an important role in the effectiveness of tightening. In Section 5.4, we proposed a meet operator that overapproximates the intersection of zonotopes. Applying this over-approximated meet operation is crucial for the initial states, but applying it in further iterations proved to gain little.

Table 6.2 highlights the results, with tightening applied at all iterations for boxes, octagons and polyhedra. Compared to Table 6.1, the number of abstract elements and the iterations have reduced compared to Table 6.1. In case of the *Lead-Lag controllers*, the algorithm is now able to find an inductive invariant. This indicates that a tightening at each iteration of the algorithm is actually useful in inferring an inductive invariant. However, if the aim of the analysis is simply to prove as fast as possible that the initial invariant holds, because we could strengthen it in an inductive invariant, then our zonotopic analysis is better in case of complex and non-affine behaviors.

Program	Boxes			Octagons			Polyhedra		
	#elems.	#iters.	time(s)	#elems.	#iters.	time(s)	#elems.	#iters.	time(s)
Octagon	619	2240	0.1001	560	2268	0.5268	1	1	0.0001
Filter	203	1115	0.0601	57	261	0.0493	4	4	0.0155
Arrow-Hurwicz	577	1297	0.1361	39	96	0.0318	4	5	0.0165
Filter2	9	45	0.0015	5	5	0.0012	1	1	0.0009
Harm	34	38	0.0071	47	51	0.0270	12	12	0.0504
Harm-reset	40	46	0.0063	- ⁴	- ⁴	- ⁴	12	12	0.0747
Harm-saturated	4	4	0.0006	3	3	0.0057	2	2	0.0035
Lead-lag	16154	10	20.3362	16154	10	157.5281	14	14	0.1181
Lead-lag-reset	1024	8	0.2236	1024	8	1.4561	10	10	0.0674
Lead-lag-saturated	16154	10	20.4759	16154	10	157.4786	14	14	0.1058
Sine	237	1290	0.3954	145	200	0.1546	113	149	1.3816
Square root	8	8	0.0005	4	4	0.0007	4	4	0.0066
Newton	128	77	0.0374	78	23	0.0513	56	17	0.5081
Newton2	2598	246	0.8878	350	103	0.2791	12	12	1.0878
Corner	- ⁴	- ⁴	- ⁴	- ⁴	- ⁴	- ⁴	- ⁴	- ⁴	- ⁴

Table 6.2 – Experimental results with tightening (tightening is applied after each split).

Programs. All the programs can be found in the Appendix. The programs are illustrated as shown in the Figure 6.5, a structure compliant with respect to the analyzer. In Figure 6.5, *init* stands for the initial states of the program, which the final inductive invariant must contain. If any abstract element does not intersect with the *init*, then it is not a *necessary* abstract element. *Body* denotes the loop of the program and the goal is the candidate invariant. All the input types for the variables representing the *init* and goal states are in the form of an interval and are abstracted accordingly depending on the domain.

6.3 CONCLUSION

The interest of zonotopes on the simpler examples is no doubt limited but on programs with non trivial mathematical operations the zonotopes allow inductive invariants to be calculated much faster than methods based on octagons or polyhedra. In addition, these inductive invariants contain much lesser elements than those obtained with these latter methods. Methods based on polyhedra are more effective if the contraction operation is repeated with each iteration. However, the zonotope-based approach remains the most effective in showing that the initial invariant is correct.

```
init
{
}
body
{
}
goal
{
}
```

Figure 6.5 – Structure of a program for the analyzer

PART III

INVARIANTS OF CONTINUOUS
SYSTEMS



Dynamical systems are mathematical objects used to model physical phenomena whose behavior (the state) changes over time. Their theory describes in form of mathematical equations, the changes over time that occur in biological, chemical, economical, financial and electronic systems. Examples of such systems include the physiological model of glucose metabolism in man and its use to design and assess improved insulin therapies for diabetes [ATS09, Sor85], the stock market, the long-term behavior of solar system (sun and planets) or galaxies.

When differential equations are used, the theory of dynamical systems is called continuous dynamical systems. One of the significant aspects of continuous dynamical system is the positive invariant sets. It has many applications such as verification of the safety of any physical phenomena modeled by dynamical systems. In this chapter, we will first introduce the notation and definitions of dynamical systems. Further we will prove that our CP algorithm will obtain an outer-approximation of the positive invariant. For deriving the proof, we will again consider discrete maps or the corresponding discrete system after discretizing a continuous system. We will illustrate through an example, how our zonotopic abstraction based CP algorithm can be used to find an outer-approximation of the positive invariant sets in dynamical systems.

7.1 PRELIMINARIES

Basic definitions and main concepts of invariant sets in dynamical systems are presented briefly hereafter, following [Ste10, HK02, Mis99, NP19].

DEFINITION 7.1 (Map.) Consider a function given by

$$F: X \rightarrow X \quad (7.1)$$

This function F is called a map which is an evolution rule with discrete time, and a continuous state space.

Remark 7.2. A dynamical system with discrete time is defined by a map,

$$x_n = F^n(x_0) \quad (7.2)$$

where x_n is the state resulting from the initial state x_0 after the n -th iterate of F .

DEFINITION 7.3 (**Flow map**.) Consider a function given by

$$\varphi: \mathbb{R} \times X \rightarrow X \quad (7.3)$$

We call the function φ , a flow map which is a continuous dynamical system on a manifold, X continuously differentiable with respect to time.

Remark 7.4. A dynamical system with continuous time is defined by a flow map,

$$x(t) = \varphi(x(0), t) \quad (7.4)$$

where $x(0)$ and $x(t)$ are the states at time 0 and t respectively.

DEFINITION 7.5 (**Forward or positive invariant**.) Consider a flow map given by $\varphi: \mathbb{R} \times X \rightarrow X$ and a subset B ($B \subseteq X$) of X . The set B is called forward invariant or positively invariant if

$$\varphi(B, t) \subseteq B, \forall t \geq 0 \quad (7.5)$$

Similarly, for a map $F: X \rightarrow X$, a forward invariant is the set B ($B \subseteq X$) such that $F(B) \subseteq B$

Remark 7.6. If a system reaches a positive invariant set, its future evolution remains inside this set. Positively invariant sets are often encountered as absorbing sets, which is a first step in order to prove the existence of an attractor.

DEFINITION 7.7 (**Backward invariant**.) Consider a flow map given by $\varphi: \mathbb{R} \times X \rightarrow X$ and a subset B ($B \subseteq X$) of X . The set B is called backward invariant if

$$B \subseteq \varphi(B, t), \forall t \geq 0 \quad (7.6)$$

Remark 7.8. For certain differential equations, the solution may also exist for negative time $t < 0$. If a set is invariant under the negative time flow map of a differential equation, we say the set is negatively invariant.

DEFINITION 7.9 (**Invariant**.) The set B is invariant for the flow map φ if

$$\varphi(B, t) = B, \forall t \in \mathbb{R} \quad (7.7)$$

and similarly for a map F if $F(B) = B$. If the map is invertible then it is analogous to $F^{-1}(B) = B$

Remark 7.10. Thus, an invariant set is a subset of the state-space which is positively invariant under the flow, and positively invariant under the opposite flow (i.e. it is also negatively invariant). In other words, a set is invariant if it is both positive and negative invariant.

A fundamental requirement for verifying safety properties in discrete, continuous and hybrid systems is to compute over-approximations of the reachable set [RSA14]. Among the number of tools and methods that have been developed for reachable set over-approximation in continuous systems, most are geared towards invariant computation and flowpipe construction.

The invariant computation methods capable of proving safety properties often rely on constructing continuous invariants [PC09, LZZ11, GT08, SSM04].

Such invariants can be considered as a generalization of positively invariant sets (see e.g. [Bla99]). Recall that positive invariant set is a classical notion in reachable set over-approximation. These sets are analogous to inductive invariants used in computer science to reason about the correctness of discrete programs (example: Chapter 6) using Hoare logic. The exact reachable sets of any given state x_0 of the system are the smallest positively invariant sets one can hope to find that include x_0 .

Methods geared towards flowpipe construction employ set reachability computations based on constructing over-approximating enclosures of the reachable states of ordinary differential equations [CK98, SNÁ17, SJJ17, CAS12, FLGD⁺11, ERNF15, GP17] because ODEs are used to describe the evolution in continuous time t of a dynamical system. Thus, a continuous time system can be defined as follows.

DEFINITION 7.11 (Continuous time systems) Consider a vector field f (See Definition 7.12) which maps each point x to the corresponding tangent space at time t . Such a vector field can be associated with an ordinary differential equation given by

$$\dot{x} = f(x, t). \quad (7.8)$$

Such an ODE also defines a continuous time system where x are the state variables and t is the time variable.

DEFINITION 7.12 (Vector field) A vector field f over \mathbb{R}^n associates each point $x \in \mathbb{R}^n$ with a derivative vector $f(x) \in \mathbb{R}^n$.

Remark 7.13. The flow map φ_f (Definition 7.3) is the general solution of Equation (7.8), i.e., the curve $\varphi_f(x_0, t)$ is the solution of Equation (7.8) with the initial condition $x(0) = x_0$. The problem of finding the solution $x(t)$ over some time interval T containing 0 for an ODE in Equation (7.8) with initial condition $x(0) = x_0$ is commonly known as an initial value problem (IVP). In the context of continuous systems defined by the ODE in Equation (7.8) the problem is analogous to computing the reachable set $\varphi_f(x_0, t)$.

It is often needed to solve an initial value problem for computing the reachable set of a continuous system. Moreover, it is well-known that numerical integration methods like Euler's method, Taylor's method and Runge-Kutta method are used to generate numerical solutions for IVPs. Recall that a safety verification problem is concerned with establishing that once the state of a system enters a set of safe states, the state of the system may not leave this set within time t . Unfortunately, this property may be only established when the ODEs have closed-form solutions because these integration methods provide only approximate values for the solution. In the absence of closed-form solutions, guaranteed bounds for the flow of an ODE, including all discretization and roundoff errors in the computation can be computed by validated integration methods [NJC99, BM98]. We can also compute successive over-approximating enclosures (known as flowpipes) of the reachable states in discrete time steps.

Flowpipe construction [CK98, SNÁ17, SJJ17, CAS12, FLGD⁺11, GP17] is one of the widely used techniques for over-approximating reachable sets for continuous time systems. It can be defined as follows.

DEFINITION 7.14 (Flowpipe) Given a time interval $[0, t]$, an over-approximation of the reachable set in $[0, t]$ is iteratively computed by dividing the time interval $[0, t]$ into N segments and enclosing each segment with a convex geometric object (example: convex polyhedra, zonotopes and ellipsoids). Thus, the complete over-approximation of the reachable set in $[0, t]$ is the union of N flowpipe segments.

Remark 7.15. Consider a continuous time system defined by $\dot{x} = f(x, t)$ and represented by a flow φ_f . Flowpipe is the set of states reachable from an initial state X_0 in a time interval Δ denoted by

$$\{\varphi_f(x_0, t) \mid x_0 \in X_0, t \in \Delta\}.$$

One of the popular methods for computing flowpipe over-approximations is based on Taylor models [SJJ17, RSÁ14, Che15, GP17].

In this work, we will only recall the classical Taylor models based on Taylor model arithmetic developed by Berz and Makino [Ber99, MB03, MB09, BM98, Jol11]. As a scope for future work, we would like to extend our algorithm of Section 5.1 for flowpipes to over-approximate the set of reachable states in continuous and hybrid systems. So, in order to encourage our work and motivate the readers, we will illustrate the step-by-step procedure for finding Taylor models of one and two dimensional ODEs. These examples are taken from the benchmarks of Flow* [CÁS13, Che15].

Recall that our CP algorithm is parameterized by a choice of abstract domain D^\sharp and an abstract version F^\sharp of a continuous map F . Before introducing Taylor models, we will prove that provided with a continuous map F , our algorithm will find an over-approximation of positive invariant set. We will illustrate this property by running the algorithm on a discrete time dynamical system: Hénon map.

We know that flow maps have been largely studied because of their usefulness as solutions of differential equations. Nevertheless, maps have also proven useful for a few reasons: they better represent discrete systems, and they help us understand some flow maps better. That is, we can break flow maps into their fixed time maps. In other words, we can discretize them. We will use Taylor based approximation and discretize the continuous time system to estimate an approximation of the flow map. We will illustrate this through Van-der-Pol oscillator. This is the first step towards extending our CP algorithm to flowpipes.

7.2 THE CP ALGORITHM REVISITED

In this chapter we will apply our CP algorithm with two different settings. The first one is, we will discretize the continuous time system using Taylor method and apply the CP algorithm on the discretized map. Recall that for a continuous time system Taylor based discretization method provides only approximate values for the solution. However, Taylor models provide guaranteed bounds for the flow of an ODE. Therefore, the second formulation is, we will evaluate a Taylor model on each time step and apply our CP algorithm on every new Taylor model. In the first case the map is a simple polynomial and in the second case it is a polynomial plus an interval term.

Nevertheless, in both cases they are discretized maps. So, in this section we will prove that our CP algorithm will find an over-approximation of positive invariant set by again considering maps.

Recall that the CP algorithm (Section 5.1) searches for an invariant that can be expressed as a finite collection of abstract elements, i.e., $\mathcal{S} = \{S_1, \dots, S_n\}$. We discard S_k if

$$F(S_k) \cap (\cup_i S_i) = \emptyset.$$

We would like to prove that our CP algorithm will obtain an outer-approximation of the positive invariant. For deriving the proof, we again consider discrete maps or the corresponding discrete system after discretizing a continuous system. We illustrate the proof on a discrete map because for a sufficiently small step size an invariant set for a continuous system, is also an invariant set for the corresponding derived discrete system (see [Bla99, HST18, HST17, BM96]).

LEMMA 7.16 *Consider a flow map $\varphi_f(t)$ representing a continuous time system. After discretization, we obtain a discrete map F . Our CP algorithm discards an abstract element S_k if*

$$F(S_k) \cap (\cup_i S_i) = \emptyset. \quad (7.9)$$

With the test shown in Equation (7.9) the CP algorithm will find an outer-approximation of the positive invariant set.

Before deriving the proof, we will recap the definition of a positive invariant set.

DEFINITION 7.17 The positive invariant ($\text{pinv}_Y F$) is equal to

$$\text{pinv}_Y F = \sqcup \{X : X \subseteq Y \wedge F(X) \subseteq X\} \quad (7.10)$$

Let $\cup_i S_i = G_k^\sharp$ at k^{th} iteration. Recall that we use Equation (7.9) to generate G_k^\sharp . Then,

$$\text{pinv}_Y F \subseteq \cup G_0^\sharp = Y \quad (7.11)$$

and

$$\text{pinv}_Y F \subseteq \cup G_k^\sharp \quad \forall k > 0. \quad (7.12)$$

We suppose that $\text{pinv}_Y F \subseteq \cup G_k^\sharp$ is true at iteration k . Now, in order to prove the lemma we must show that $\text{pinv}_Y F \subseteq \cup G_{k+1}^\sharp$ is true at $k+1$.

Proof. Suppose $\exists x \in \text{pinv}_Y F$ which is not present in $\cup G_{k+1}^\sharp$ (not G_k^\sharp). Then as it was (by induction) in $\cup G_k^\sharp$, that means it has been removed. There is a $[x] \in G_k^\sharp$ such that $x \in [x]$ wherein $[x]$ is a box. Recall that the test for discarding elements is: we remove an abstract element if

$$F(S_k) \cap (\cup_i S_i) = \emptyset \quad (7.13)$$

Now, assume that $[x]$ does satisfy the condition in Equation (7.13) and can be removed which implies

$$F([x]) \cap \cup G_k^\sharp = \emptyset \quad (7.14)$$

Recall that x was in $\cup G_k^\sharp$ by induction but has been removed, i.e., Equation (7.13) thus Equation (7.14) hold, and we know that $\cup G_k^\sharp \supseteq \text{pinv}_\gamma F$. So, we can say that $F([x])$ does not intersect $\text{pinv}_\gamma F$, i.e.,

$$F([x]) \cap \text{pinv}_\gamma F = \emptyset,$$

thus $F(x)$ is not in $\text{pinv}_\gamma F$, i.e.,

$$F(x) \cap \text{pinv}_\gamma F = \emptyset \rightarrow F(x) \notin \text{pinv}_\gamma F.$$

This contradicts the fact that $x \in \text{pinv}_\gamma F$ since this implies

$$F(x) \in \text{pinv}_\gamma F.$$

Thus, $\text{pinv}_\gamma F \subseteq \cup G_k^\sharp$ is also true at iteration $k + 1$. \square

EXAMPLE 7.18 (Hénon attractor) Before we introduce Hénon attractor, we will discuss attractors and their relationship to positive invariant sets.

Consider a map F on a space X . Let N be a subset of X such that $N \subseteq X$. The set N can be called as an isolating neighborhood if the invariant set $\text{INV}(N, F)$ defined as

$$\text{INV}(N, F) = \bigcap_{m=-\infty}^{\infty} F^m(N) \quad (7.15)$$

is a subset of the interior ($\text{Int } N$) of N .

It can also be defined as

$$\text{INV}(N, F) = \{x \in N \mid F^n(x) \subseteq N, n \in \mathbb{Z}\} \subseteq \text{Int } N \quad (7.16)$$

or for a flow φ on a space X as

$$\text{INV}(N, \varphi) = \{x \in N \mid \varphi_t(x) \subseteq N, t \in \mathbb{R}\} \subseteq \text{Int } N \quad (7.17)$$

A set S is called isolated invariant set if there exists an isolating neighborhood N with $S = \text{INV}(N, F)$.

An attractor is a special case of an isolated invariant set. Consider $A \subseteq X$ is a subset and there is a neighborhood N of A such that $F(N) \subseteq \text{Int } N$ and A is the intersection of forward images of N , i.e.,

$$\bigcap_{n \geq 0} F^n(N) = A, \quad (7.18)$$

then A is called an attractor and N is its isolating neighborhood. Consider the set V such that $V = X \setminus \text{Int } N$. As a model of Poincaré map to study the dynamics of the Lorenz system, Hénon in 1976 [Hén76] proposed the famous two-dimensional Hénon map that is defined by the following equation:

$$\begin{aligned}x_{n+1} &= 1 - \alpha x_n^2 + y_n \\ y_{n+1} &= \beta x_n\end{aligned}\tag{7.19}$$

It is an invertible map. The inverse of the Hénon map or the backward iteration in time is given by

$$\begin{aligned}x_{n+1} &= \frac{y_n}{\beta} \\ y_{n+1} &= x_n - 1 + \frac{\alpha y_n^2}{\beta^2}\end{aligned}\tag{7.20}$$

Hénon claimed that for the parameters $(a, b) = (1.4, 0.3)$ the Hénon map converges to a strange attractor. Note that Hénon map does not have a strange attractor for all values of the parameters a and b . The parameter a controls the amount of stretching and the parameter b controls the thickness of folding. A range of values of a and b for which the Hénon map preserves the Hénon attractor is $a \in [1.16, 1.41]$ and $b \in [0.2, 0.3]$. From these ranges, a slight change in the values of a and b can affect the Hénon attractor, but a small change of F does not affect the attractor [Rue06, Wen14].

A visualization of the Hénon attractor can be done in the following manner.

Consider an initial point $(x_0, y_0) = (0, 0)$. Taking this point and iterating Equation (7.19) and plotting the results on the (x_{n+1}, y_{n+1}) plane, we can get a sketch of the dynamics of the Hénon map. Note that if we plot the points $x_n = F^n(0, 0)$ they accumulate, for $n \rightarrow \infty$, on a convoluted fractal set A known as the Hénon attractor. An iteration of 20,000 with initial point $(0, 0)$ is plotted in Figure 7.1 (See Figure 1 in [SWYZ09], [Rue06], Fig. 1 in [Wen14]). An Hénon attractor is also illustrated in a Mathematica notebook in the appendix.

In the remaining part of this example, we will discuss the experiments related to computing an over-approximation of the positive invariant set using our zonotope abstraction based CP algorithm. The CP algorithm has an abstract version F^\sharp of the Hénon map F , already provided by abstract interpretation. The remaining operations of the algorithm remain the same. For instance, the algorithm requires the split operator, as well as a notion of size of abstract elements (to ensure termination by avoiding splitting beyond a certain size), intersection test, inclusion test and the coverage heuristic for splitting. The candidate invariant provided to the algorithm is the box $[-2, 2] \times [-2, 2]$.

Recall that, we proved through Lemma 7.16 that our CP algorithm finds an outer-approximation of the positive invariant. We ran the algorithm and removed the abstract elements which are not part of the positive invariant set and the result we obtained is illustrated in Figure 7.2. The figure shows boxes with three colors: the ones in blue belong to the positive invariant set; the ones in pink, their state could not be decided by the algorithm and the red ones are the images of abstract elements (blue and pink) by an iteration of the Hénon map in the abstract domain used for computing the positive invariant set. Thus, we get an outer-approximation of the positive invariant set upto a precision criterion. Note that the map applied is only F . Later, we will illustrate some experiments with F^2 , F^3 and F^4 maps. We adapted our CP algorithm for using the functions F^n where $n \in \mathbb{N}$. We present the modified algorithm in Algorithm 7.1.

Algorithm 7.1 – The zonotopic variant of the CP based AI algorithm 4.1 for inferring inductive invariants while considering an iterated map sequence F, F^2, \dots, F^n

```

Function{SearchInvariant}{I#, F#, T#}
T# ← T# ∩ F#(T#) // Tightening the target invariant
search space & final solution set, a set of zonotopes G# := {T#}
WHILE G# ≠ ∅ DO
    S# ← pop a zonotope from G# // Based on minimum coverage
    IF F#(S#) ⊆ ∪iSi# and F#(F#(S#)) ⊆ ∪iSi# ⋯ and F#n(S#) ⊆ ∪iSi# THEN
        return G# ∪ {S#}
    ELSE IF (coverage(S#, G# ∪ {S#}) = 0 or size(S#) < εs or S# is not useful)
    THEN
        remove S#
    ELSE IF size(S#) < εs THEN
        return failure
    ELSE
        split S# into a set {S1#, S2#, ⋯} such that S# = ∪iSi#
        push {S1#, S2#, ⋯} into G#
EndFunction
    
```

Note that in Algorithm 7.1 when an iterated map sequence the map F, F^2, \dots, F^n is considered, an element is benign only if it satisfies the condition:

$$F^{\#}(S^{\#}) \subseteq \cup_i S_i^{\#} \text{ and } F^{\#}(F^{\#}(S^{\#})) \subseteq \cup_i S_i^{\#} \dots \text{ and } F^{\#k}(S^{\#}) \subseteq \cup_i S_i^{\#}.$$

Recall that for splitting we still need a sound coverage measure, and we compute it as explained below.

Coverage Recall the coverage measure from Chapter 4 in Section 4.3. We rewrite below the Equation (4.4) that defines coverage

$$\text{coverage}(S_k^{\#}) := \frac{\sum \{\text{vol}(F^{\#}(S_k^{\#}) \cap \text{cnt}(P^{\#})) \mid P^{\#} \in \text{post}(S_k^{\#})\}}{\text{vol}(F^{\#}(S_k^{\#}))} \quad (7.21)$$

Recall that the map post in Equation (7.21) is computed as

$$\text{post}(S_k^{\#}) := \{P^{\#} \in B^{\#} \mid F^{\#}(S_k^{\#}) \cap P^{\#} \neq \emptyset\} \quad (7.22)$$

Please follow Chapter 4 and 5 for the definition of partitions $P^{\#} \in B^{\#}$. In Chapter 5 under Section 5.7 we developed a heuristic measure for the coverage computation that is sound enough to select abstract elements which require immediate action. This heuristic measure (already defined in Equation (5.21)) is

$$\text{coverage}(S_k^{\#}) := \frac{\#\{P^{\#} \mid \text{cnt}(P^{\#}) \neq \emptyset, P^{\#} \in \text{post}(S_k^{\#})\}}{\#\{P^{\#} \mid P^{\#} \in \text{post}(S_k^{\#})\}} \quad (7.23)$$

Now, for instance consider that we will be using the function F^2 . First, we will compute the map post with respect to F^2 . We will denote this new post as $\text{post}(\text{post}(S_k^{\#}))$ and can be defined as

$$\text{post}(\text{post}(S_k^{\#})) := \{P^{\#} \in B^{\#} \mid F^{\#}(F^{\#}(S_k^{\#})) \cap P^{\#} \neq \emptyset\} \quad (7.24)$$

Now, we compute the coverage with respect to F and F^2 . They are:

$$\text{coverage}_1(S_k^\sharp) := \frac{\#\{P^\sharp \mid \text{cnt}(P^\sharp) \neq \emptyset, P^\sharp \in \text{post}(S_k^\sharp)\}}{\#\{P^\sharp \mid P^\sharp \in \text{post}(S_k^\sharp)\}} \quad (7.25)$$

$$\text{coverage}_2(S_k^\sharp) := \frac{\#\{P^\sharp \mid \text{cnt}(P^\sharp) \neq \emptyset, P^\sharp \in \text{post}(\text{post}(S_k^\sharp))\}}{\#\{P^\sharp \mid P^\sharp \in \text{post}(\text{post}(S_k^\sharp))\}} \quad (7.26)$$

Then we take the minimum from the two coverage values to prioritize the splitting operation. Thus, in this manner we compute the coverage in Algorithm 7.1. We remove an abstract element with the useless test if it is either way useless with respect to F^\sharp or $F^\sharp(F^\sharp)$.

Observe that the images (in red) of the abstract elements shown in Figures 7.2 are only the forward images of the abstract elements. If we compare the images (in red) of the abstract elements shown in Figures 7.2 with the Hénon attractor in Figure 7.1, the images of the set $F^\sharp(\cup_i S_i)$ is related to the Hénon attractor (fractal set).

The CP Algorithm had to rely heavily on splitting to obtain the set shown in Figure 7.2. So, further we will experiment with the CP Algorithm 7.1 on the map sequence F, F^2, \dots, F^n and see if it helps to remove abstract elements and obtain the attractor. For instance, the set in Figure 7.3 is obtained on the map F, F^2 . We consider an abstract element is benign if $F^\sharp(S^\sharp) \subseteq \cup_i S_i^\sharp$ and $F^\sharp(F^\sharp(S^\sharp)) \subseteq \cup_i S_i^\sharp$. In blue are the abstract elements which are part of the positive invariant set, in pink are the ones whose state could not be decided by the algorithm and in red is the image of the abstract elements by a loop iteration in the abstract domain used for computing the positive invariant set. Similarly, we also experimented with F, F^2, F^3 and F, F^2, F^3, F^4 . Comparing the set in Figures 7.3, 7.4 and 7.5 with the one in Figure 7.2, we observe that the algorithm manages to further remove abstract elements which are not part of the positive invariant set. However, this does not help much because the over-approximation due to the abstraction becomes larger (see the images of the abstract elements in red in Figure 7.5) and eventually does not help the tests in discarding elements.

EXAMPLE 7.19 (Van-der-Pol oscillator) Now, we will consider a well-known continuous time system, the Van-der-Pol oscillator. It is defined by

$$\begin{aligned} \dot{x} &= 2y \\ \dot{y} &= -0.8x - 10(x^2 - 0.21)y \end{aligned} \quad (7.27)$$

with the initial set $x_0 \in [-1.2, 1.2]$ and $y_0 \in [-1.2, 1.2]$. This example is taken from [HK13].

We will use our CP algorithm for finding an over-approximation of the positive invariant for Van-der-Pol oscillator. We derived a 2 order polynomial after Taylor discretization for a step size $t = 0.05$ as follows. Before we illustrate the derivation, we will introduce briefly about Taylor approximation. We detail it in Section 7.3. Consider a univariate function f which is k times differentiable over the domain $[a, b] \in \mathbb{R}$. The k order Taylor approximation

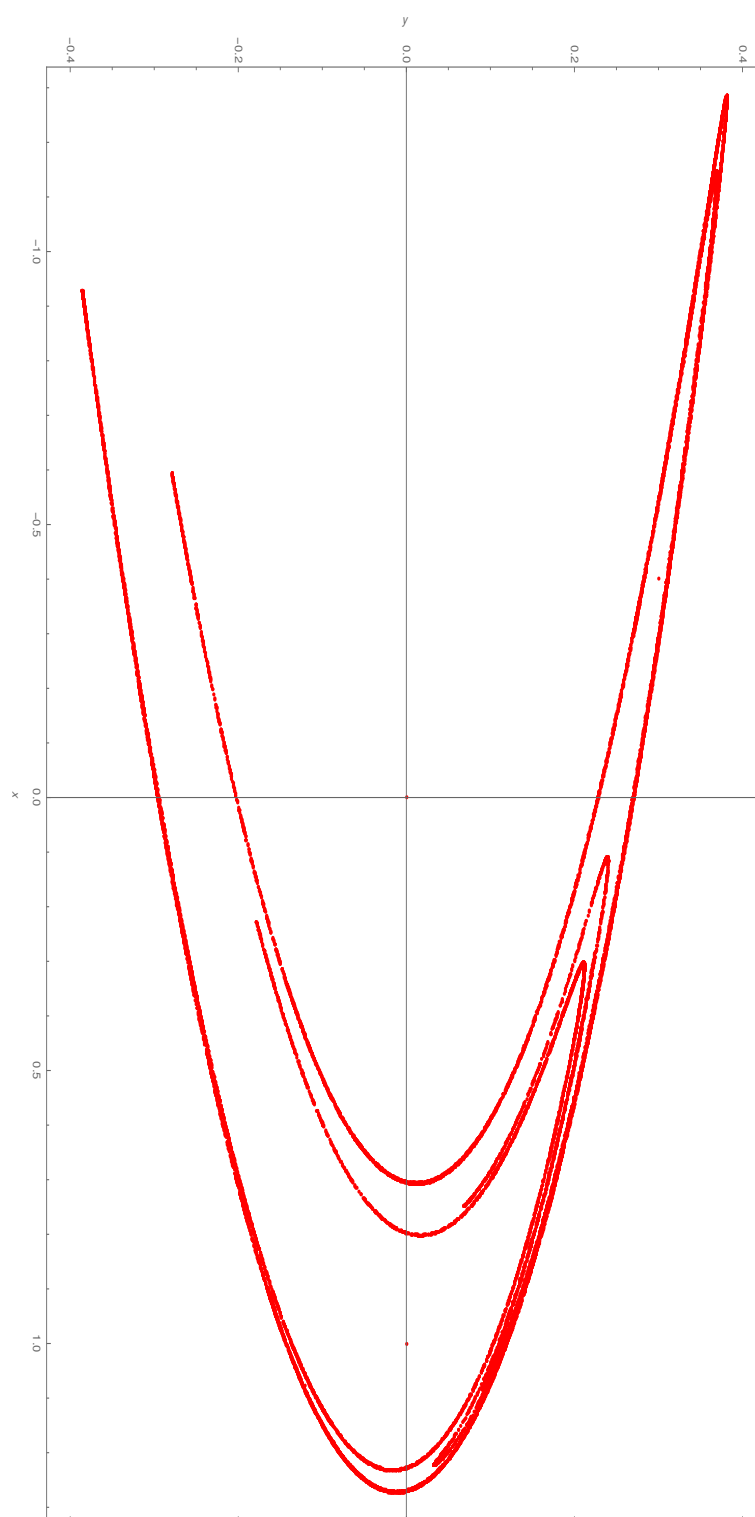


Figure 7.1 – Hénon attractor

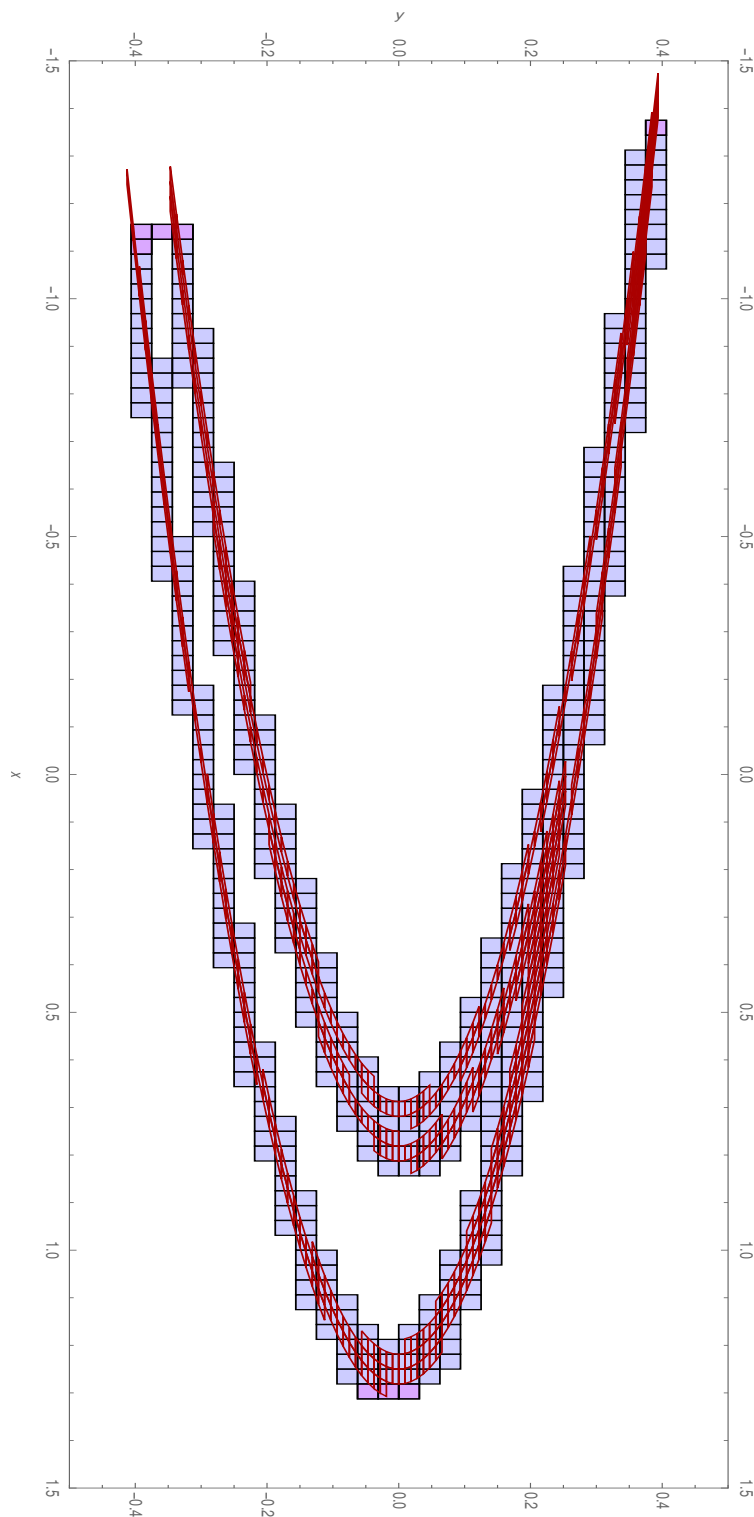


Figure 7.2 – An outer-approximation of the positive invariant set of Hénon map (in blue are the abstract elements which are benign, in pink are the ones whose state cannot be decided by the algorithm, and in red is the image of the abstract elements by a loop iteration in the abstract domain used for computing the positive invariant set)

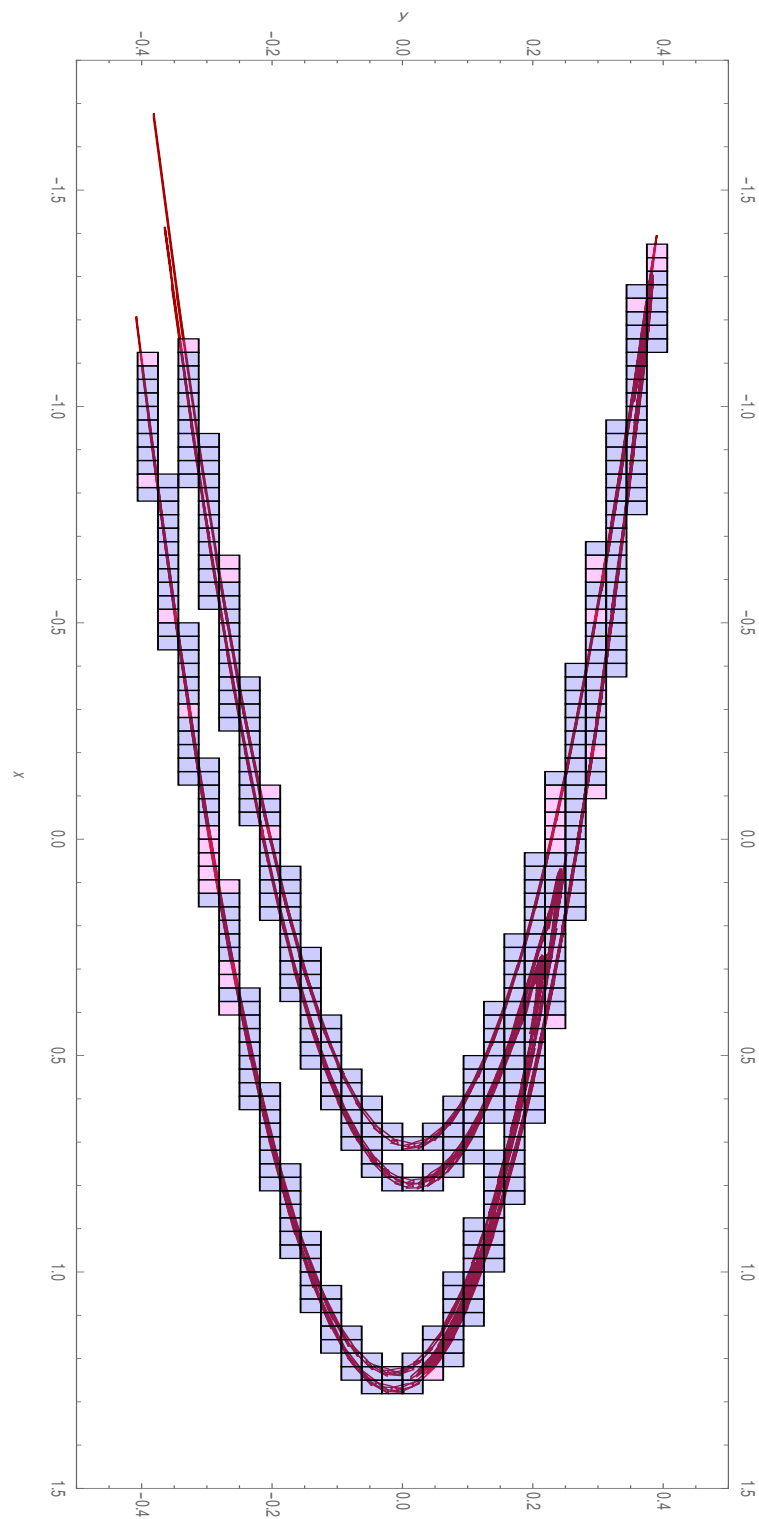


Figure 7.3 – The set obtained using the CP Algorithm 7.1 upto a size criterion for an iterated function sequence F, F^2 .

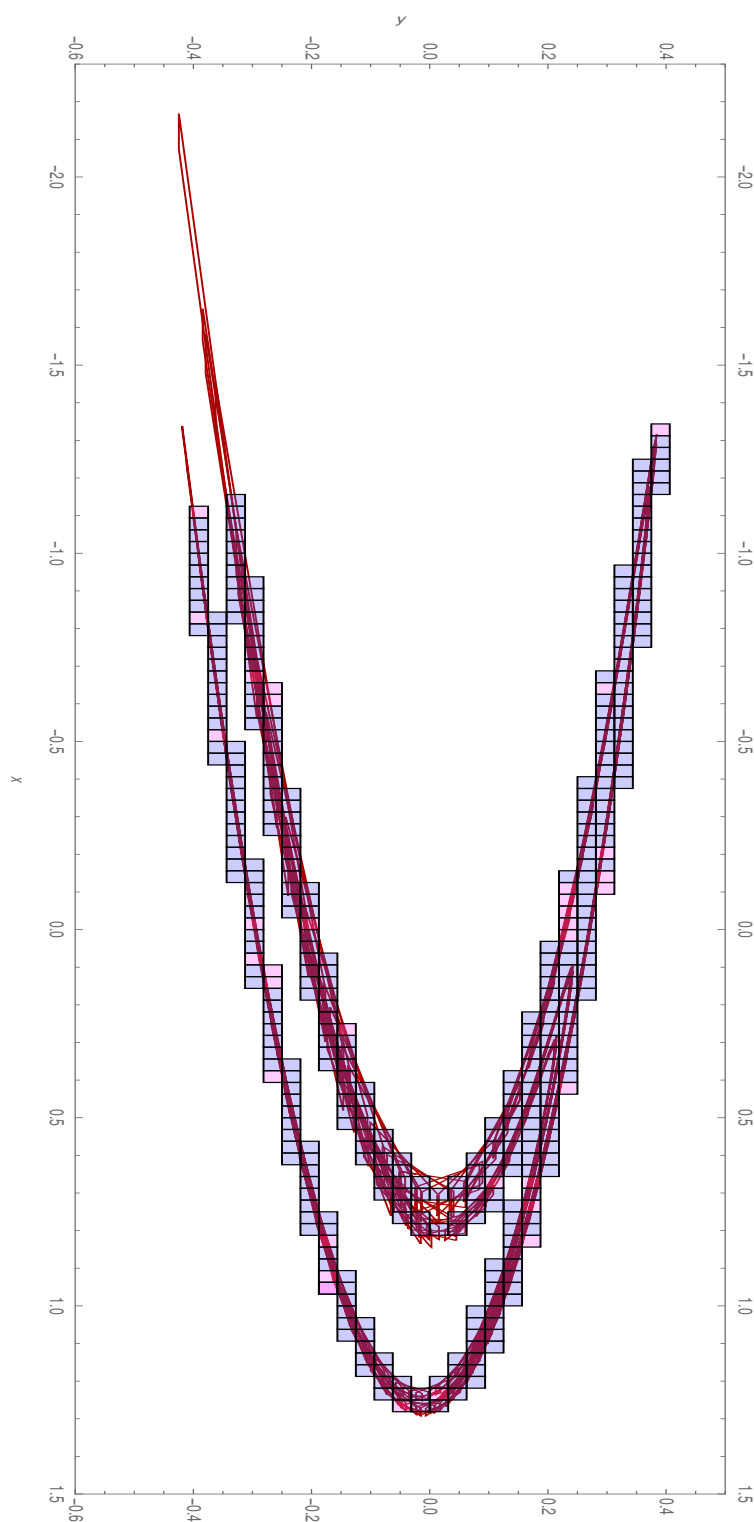


Figure 7.4 – The set obtained using the CP Algorithm 7.1 upto a size criterion for an iterated function sequence F, F^2, F^3 .

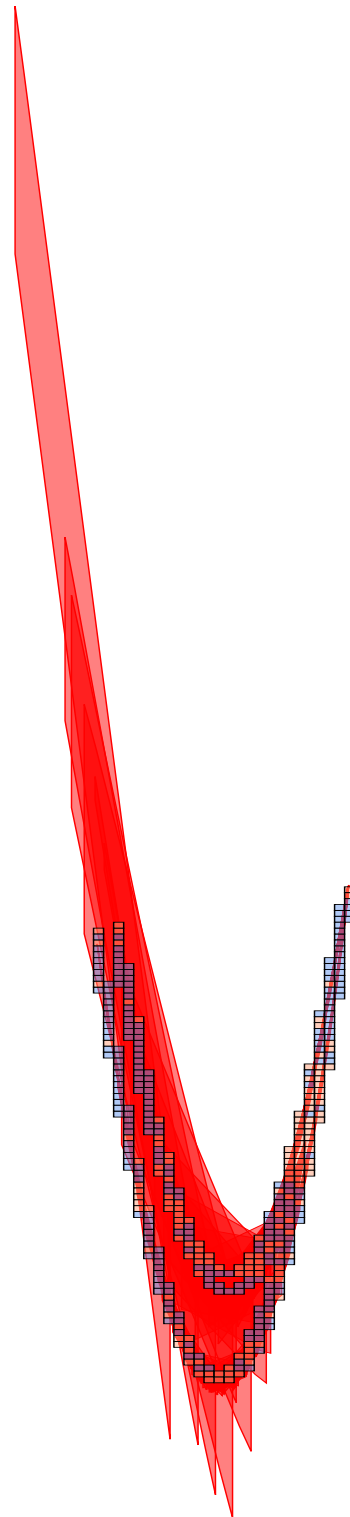


Figure 7.5 – The set obtained using the CP Algorithm 7.1 upto a size criterion for an iterated function sequence F, F^2, F^3, F^4 .

of f at $x = c$ for some $c \in [a, b]$ is

$$p_k(x) = f(x) = f(c) + f^{(1)}(c)(x - c) + \frac{1}{2}f^{(2)}(c)(x - c)^2 + \dots + \frac{1}{k!}f^{(k)}(c)(x - c)^k \quad (7.28)$$

such that $f^{(i)}(c)$ denotes the i -th order derivative of f at $x = c$.

Similarly, for multivariate functions, the k order Taylor approximation of a multivariate function f at $\vec{x} = \vec{c}$ is given by

$$p_k(\vec{x}) = f(\vec{c}) + \sum_{i=1}^n \left(\frac{\partial f}{\partial x_i}(\vec{c}) \cdot (x_i - c_i) \right) + \dots + \frac{1}{k!} \sum_{j_1+j_2+\dots+j_n=k} \left(\frac{\partial^k f}{\partial x_1^{j_1} \dots \partial x_n^{j_n}}(\vec{c}) \cdot \prod_{i=1}^n (x_i - c_i)^{j_i} \right) \quad (7.29)$$

Now, we will generate a second order Taylor approximation for the Van-der-Pol oscillator defined by Equation (7.27) for $t = 0.05$. The first order derivative is given by

$$\begin{pmatrix} 2y \\ -0.8x - 10x^2y + 2.1y \end{pmatrix}$$

which is the Equation (7.27) defining the Van-der-pol oscillator.

The second order derivative or the Jacobian is derived as:

$$\begin{pmatrix} \frac{\partial \dot{x}}{\partial x} & \frac{\partial \dot{x}}{\partial y} \\ \frac{\partial \dot{y}}{\partial x} & \frac{\partial \dot{y}}{\partial y} \end{pmatrix} \begin{pmatrix} 2y \\ -0.8x - 10x^2y + 2.1y \end{pmatrix} \\ \begin{pmatrix} 0 & 2 \\ -0.8 - 20xy & -10x^2 + 2.1 \end{pmatrix} \begin{pmatrix} 2y \\ -0.8x - 10x^2y + 2.1y \end{pmatrix} \\ \begin{pmatrix} -1.6x - 20x^2y + 4.2y \\ -1.68x + 8x^3 + 2.81y - 42x^2y + 100x^4y - 40xy^2 \end{pmatrix}.$$

To construct the polynomial we consider the following expansion:

$$\begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 2y \\ -0.8x - 10x^2y + 2.1y \end{pmatrix} t + \begin{pmatrix} -1.6x - 20x^2y + 4.2y \\ -1.68x + 8x^3 + 2.81y - 42x^2y + 100x^4y - 40xy^2 \end{pmatrix} \frac{t^2}{2}. \quad (7.30)$$

Hence, the discretized map obtained for $t = 0.05$ is given by

$$\begin{aligned} x &= 0.998x + 0.10525y - 0.025x^2y, \\ y &= 0.01x^3 + 1.10851y - 0.5525x^2y + 0.125x^4y + x(-0.0421 - 0.05y^2). \end{aligned} \quad (7.31)$$

We also have illustrated the computation of this polynomial pictorially in a Mathematica notebook shown in the appendix. Recall that by proof of Lemma 7.16, our algorithm finds an over-approximation of positive invariant

set. Thus, when we ran the algorithm using the map provided in Equation (7.31), we obtain the set shown in Figure 7.6 (See the yellow region in Fig. 10 in [LMJZ17] which the authors claim to be the first positive invariant).

We ran the algorithm and removed the zonotopes which are not part of the positive invariant set and the result we obtained is illustrated in Figures 7.6–7.8. The figure shows boxes with three colors: the ones in blue belong to the positive invariant set; the ones in pink, their state could not be decided by the algorithm and the red ones are the images of abstract elements (blue and pink) by an iteration of the map (Equation (7.46)) in the zonotope abstract domain used for computing the invariant set.

Similar to the Hénon example, we conducted experiments with maps F^2 , F^3 , F^4 and F^5 . Again the aim of the experiment was the same, to see if using the map sequence F, F^2, \dots, F^n helps in removing abstract elements which are not part of the positive invariant set. We use the CP Algorithm 7.1 and the set obtained with map sequence F, F^2 ; F, F^2, F^3 ; F, F^2, F^3, F^4 and F, F^2, F^3, F^4, F^5 are shown in Figures 7.9–7.12. In blue are the abstract elements which are benign under the condition:

$$F^\sharp(S^\sharp) \subseteq \cup_i S_i^\sharp \text{ and } F^\sharp(F^\sharp(S^\sharp)) \subseteq \cup_i S_i^\sharp \dots \text{ and } F^{\sharp k}(S^\sharp) \subseteq \cup_i S_i^\sharp,$$

in pink are the ones whose state cannot be decided by the algorithm, and in red is the image of the abstract elements by a loop iteration in the abstract domain used for computing the positive invariant set. Comparing the set in Figure 7.8 with the ones in Figures 7.9–7.12 we observe that the algorithm manages to remove abstract elements which are not part of the positive invariant set. Also, the set in Figure 7.12 is related to the outer-approximation of the region of attraction reported in Figure 2 by the authors in [HK13].

Remark 7.20. So, far we have conducted experiment on a continuous time system by using its map obtained after Taylor based discretization. However, we know that numerical integration methods provide only approximate values for the solution. So, further in this chapter we will introduce Taylor models and illustrate how they can be used to provide guaranteed bounds for the flow of an ODE. Later we will evaluate a Taylor model for the Vanderpol oscillator on each time step and apply our CP algorithm on every new Taylor model obtained to compute the positive invariant set.

7.3 TAYLOR MODEL APPROXIMATION OF FLOW MAP

Consider a function f defined over an interval domain D . Assume that f can be expanded in the form a polynomial approximation. In order to find an accurate or rigorous approximation for the function f one must also consider approximation error. Thus, the approximation problem is: finding the coefficients of a polynomial approximation p along with a rigorous bound I such that $\|f - p\| \leq I$. Developed by Berz and Makino [Ber99, MB03, MB09, BM98, Jol11] a well-known tool for obtaining rigorous polynomial approximations based on Taylor approximations is Taylor models. It can be defined as follows.

DEFINITION 7.21 (Taylor model) A Taylor model of order $n > 0$ for a function f defined over an interval domain D is represented by a pair (p, I) formed by a polynomial of degree n and an interval I , such that $f(x) - P(x) \in I, \forall x \in D$.

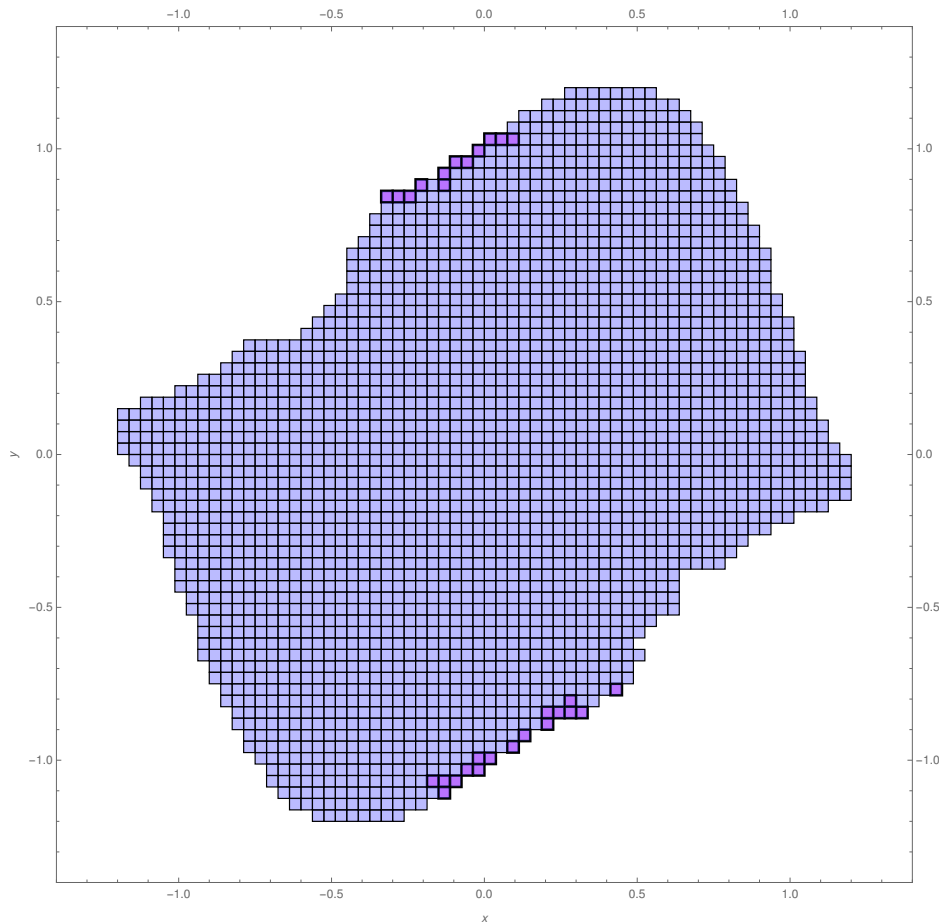


Figure 7.6 – An outer-approximation of the positive invariant for the Van-der-Pol oscillator described by the map shown in Equation 7.31

Remark 7.22. The degree n polynomial P can be computed by the Taylor expansion of the function f at a given point. The interval I is the remainder interval which provides an enclosure for the approximation errors due to truncation or rounding. Thus, Taylor model can be seen as a tube around the function f .

For computing the Taylor model approximation of a flow map, the steps are:

- computing a Taylor expansion p_n of order n for the flow map either by using Lie derivatives or by Picard iteration,
- and finding the remainder interval for the Taylor polynomial [RSÁ14, Che15, Jol11, GP17, CAS12, SJJ17].

Remark 7.23. For the Example 7.19, a Taylor model can be derived which would hold for just times in $[0, 0.05]$. As earlier, we can still apply our CP

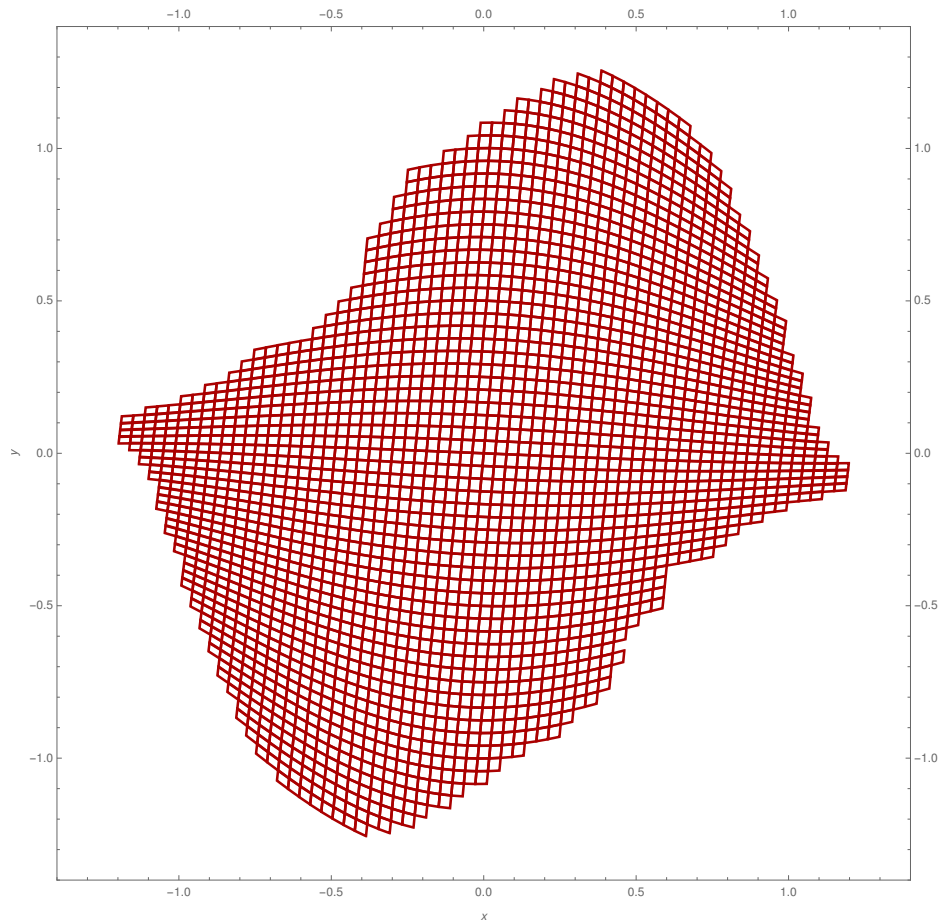


Figure 7.7 – The image of abstract elements in Figure 7.6 by a loop iteration in the abstract domain used for computing the positive invariant set

algorithm on this Taylor model and the test in Equation (7.9) should work. We will derive a Taylor model for Van-der-Pol oscillator later in this chapter.

An ideal solution for a given IVP is to compute the map φ_f by solving the given ODE analytically. However, this cannot be done exactly, since most of the ODEs do not have closed form solutions. A classic approach is to approximate the solution by a conservative Taylor model (p, I) wherein the Taylor polynomial is computed based on the higher-order Lie derivatives of the vector field. Thus, a Lie derivative can be defined as follows.

DEFINITION 7.24 (Lie derivative) Consider an ODE $\dot{x} = f(x, t)$ (a continuous time dynamical systems) and g be a variable in the state space such that $g(x(t))$ is a solution to the ODE. The Lie derivative $\mathcal{L}_f = f \cdot \nabla + \partial/\partial t$ of the differentiable function $g(x(t))$ is given by

$$\mathcal{L}_f(g) = \frac{\partial g}{\partial x} \cdot f + \frac{\partial g}{\partial t} \tag{7.32}$$

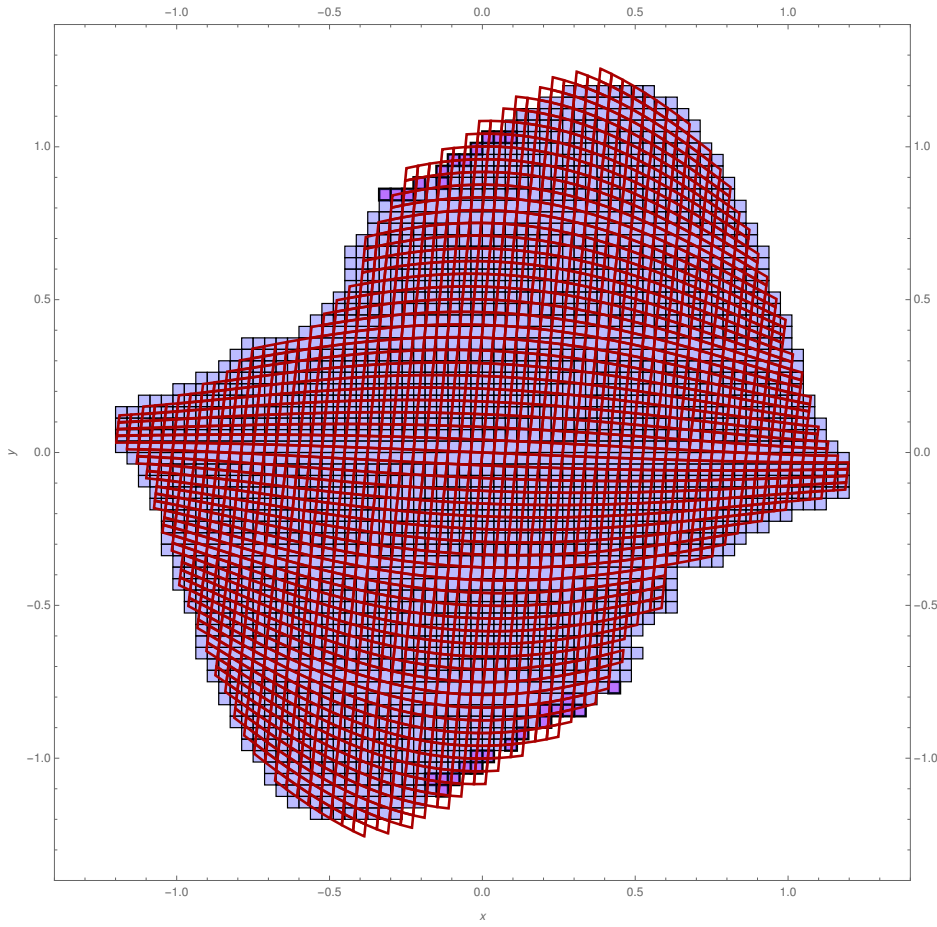


Figure 7.8 – Superposition of the two figures 7.6 and 7.7 showing the abstract elements which belong to the invariant set

Remark 7.25. If g is k times differentiable, the higher order Lie derivatives of it are defined recursively as

$$\mathcal{L}_f^{m+1}(g) = \mathcal{L}_f(\mathcal{L}_f^m(g))$$

for $m = 1, 2, \dots, k - 1$.

Thus, the polynomial p_n of the order n Taylor model that over-approximates the flow map can be computed as

$$g(x(0)) + \mathcal{L}_f(g(x(0)))t + \mathcal{L}_f^2(g(x(0)))\frac{t^2}{2!} + \dots + \mathcal{L}_f^n(g(x(0)))\frac{t^n}{n!} \quad (7.33)$$

In other words, Equation 7.33 is a Taylor expansion (see Definition 7.36) of $g(x(t))$. We will recap the definitions of Taylor approximation later in this section.

The polynomial p can also be computed by applying Picard iteration. We will discuss it in the following section.

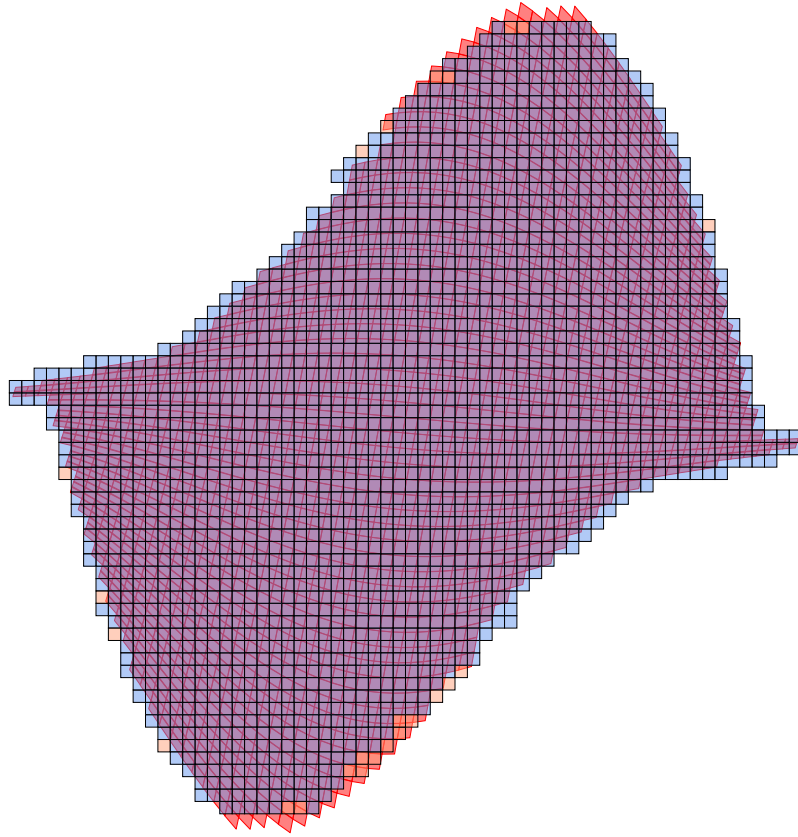


Figure 7.9 – The set obtained using the CP Algorithm 7.1 upto a size criterion for an iterated function sequence F, F^2 .

7.3.1 Picard iteration

One of the most important theorems in ODEs is to prove the existence of a solution to an ordinary differential equation. Picard iteration plays a significant role in proving the existence and it can be defined as follows.

DEFINITION 7.26 (Picard iteration) Consider an ODE $\dot{x} = f(x(t), t)$ and an initial condition $x(0) = x_0$, the picard iteration sequence is defined as

$$\begin{aligned} x_0(t) &= x_0 \\ x_{n+1}(t) &= \mathcal{P}_f(x) = x_0 + \int_0^t f(x_n(s), s) ds \end{aligned} \quad (7.34)$$

where $\mathcal{P}_f(x)$ is called the Picard-Lindelöf operator.

EXAMPLE 7.27 Consider an ODE $\dot{x} = 2t(1 - x)$ with the initial condition $x(0) = 2$. The Picard iteration sequence with $x_0(t) = 2$ is given by

$$x_1(t) = 2 + \int_0^t 2s(1 - 2) ds = 2 - t^2$$

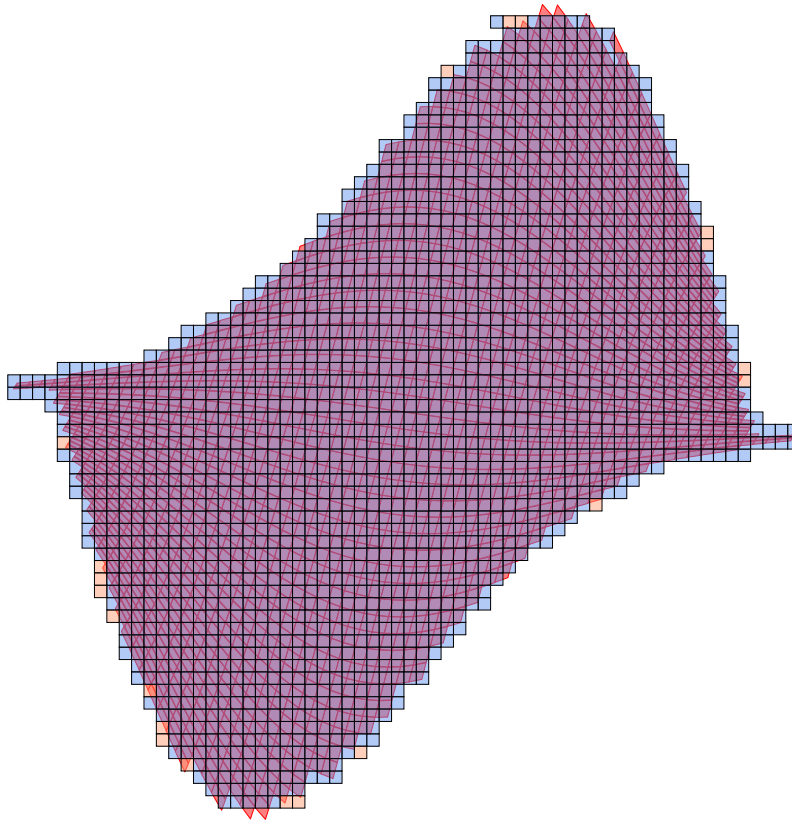


Figure 7.10 – The set obtained using the CP Algorithm 7.1 upto a size criterion for an iterated function sequence F, F^2, F^3 .

$$x_2(t) = 2 + \int_0^t 2s(s^2 - 1)ds = 2 - t^2 + \frac{t^4}{2}$$

$$x_3(t) = 2 + \int_0^t 2s(s^2 - \frac{s^4}{2} - 1)ds = 2 - t^2 + \frac{t^4}{2} - \frac{t^6}{6}$$

Remark 7.28. Given an ODE, a n -order approximation to flow map (which is assumed not to be known) can be computed by applying at most n Picard iterations.

We will discuss further how Picard-Lindelöf operator can also be used to compute the remainder interval of a Taylor model while over-approximating a flow map over a time interval.

Consider an ODE given by

$$\dot{x} = f(x(t), t) \tag{7.35}$$

with the initial condition $x(0) = x_0$. Equation (7.35) is an initial value problem in its general form. We are interested in solving this IVP to find $x(t)$. Note that the Picard operator in Equation (7.34) is related to the integral form of the IVP. So, the solution of the IVP corresponds to the fixpoint of the Picard operator $\mathcal{P}_f(x)$. In other words, the solution of the Picard operator in Equation (7.34) is

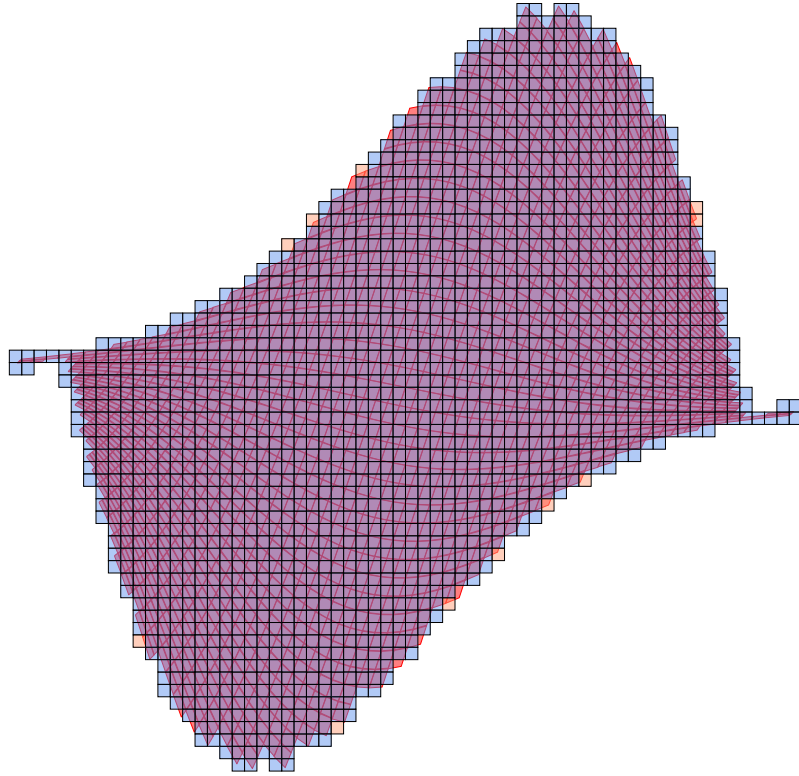


Figure 7.11 – The set obtained using the CP Algorithm 7.1 upto a size criterion for an iterated function sequence F, F^2, F^3, F^4 .

also the solution of IVP. To find a fixpoint of the transformation $\mathcal{P}_f(x)$ using Picard iteration, we start with the function $x_0(t) = x_0$ and then iterate using Equation (7.34) to produce the sequence of functions $x_0(t), x_1(t), x_2(t), \dots$. If this sequence converges, the limit function will be a fixpoint of $\mathcal{P}_f(x)$. Thus, the problem of finding a solution to the differential equation is reduced to a fixpoint problem

$$x = \mathcal{P}_f(x).$$

If f is continuous on $[0, t] \times \mathbb{R}^n$ and bounded there then Schauder's fixpoint theorem asserts the existence of a solution of an ODE over the interval $[0, t]$. Schauder's fixed-point theorem can be spelled as follows.

DEFINITION 7.29 (Schauder's Fixpoint Theorem [Con13]) Let K be a locally convex topological vector space, and $X \subset K$ be a non-empty, compact, and convex set. Then given any continuous mapping $f : X \rightarrow X$ there exists $x \in X$ such that $f(x) = x$.

The Definition 7.29 can be generalized to Taylor models as follows.

DEFINITION 7.30 (Schauder's Fixpoint Theorem to obtain a Taylor Model for the flow map) Let K be a Banach space, and $X \subset K$ be a non-empty, compact, and convex set. Let \mathcal{P}_f be a continuous operator on the Banach space K , and

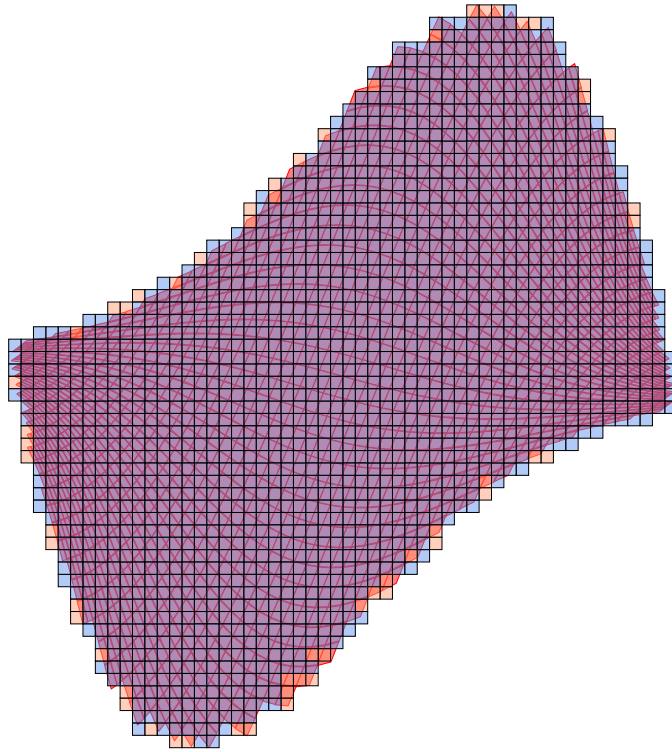


Figure 7.12 – The set obtained using the CP Algorithm 7.1 upto a size criterion for an iterated function sequence F, F^2, F^3, F^4, F^5 .

$\mathcal{P}_f(X) \subset X$. Then there exists $x \in X$ such that \mathcal{P}_f has a fixpoint in X , i.e., $\mathcal{P}_f(x) = x$.

If f is Lipschitz continuous (see Definition 7.31) then the operator $\mathcal{P}_f(x)$ is contractive and its unique fixpoint defines the solution to the ODE. This is asserted by Banach's fixpoint theorem (See Definition 7.32).

DEFINITION 7.31 (Lipschitz continuity) Consider a function $f : X \rightarrow \mathbb{R}$. The function f can be called Lipschitz continuous if there exists $K \in \mathbb{R}, K > 0$ such that $\forall x, y \in X$ the following inequality holds.

$$|f(x) - f(y)| \leq K|x - y| \quad (7.36)$$

DEFINITION 7.32 (Banach's fixpoint theorem) Let (X, d) be a complete metric space and $g : X \rightarrow X$ such that

$$d(g(x), g(y)) \leq cd(x, y)$$

for $c \in [0, 1]$ and $\forall x, y \in X$. Then g has a unique fixed-point in X

Remark 7.33. The condition $d(g(x), g(y)) \leq cd(x, y)$ makes g a contraction. Hence, Banach's fixpoint theorem is otherwise known as Contraction mapping theorem.

Remark 7.34. As stated by the Banach's fixpoint theorem (Definition 7.32) Picard-Lindelöf operator is used to check the contraction of the solution on an integration step in order to prove the existence and the uniqueness of the solution of an IVP. Hence, this operator is to compute the remainder interval of a Taylor model while over-approximating a flow map over a time interval.

Remark 7.35. Note that one of the challenging tasks in flowpipe construction is to find a safe remainder interval such that the Taylor model $(p_f(x, t), I_f)$ is an over-approximation of the flow map $\varphi_f(x, t)$ over $t \in [0, \delta]$. In order to find such a remainder interval one may start with an estimation, and then conservatively check the contractiveness of the Picard operation. Note that, the initial estimation may be incorrect. The initial estimate of the remainder interval for an order n Taylor model approximation of a flow map can be computed by Taylor-Lagrange formula (Definition 7.37):

$$\frac{1}{(n+1)!} \mathcal{L}_f^{k+1}(\varphi_f(x, \xi)) \cdot t^{n+1} \quad (7.37)$$

for some constant ξ between 0 and t .

Later, in this section we will illustrate how to derive a Taylor model. Before that, we will introduce the definitions of Taylor approximation. An order n Taylor expansion can be defined as follows.

DEFINITION 7.36 (Taylor series) Consider a univariate function f which is n times differentiable over the domain $D = [a, b] \in \mathbb{R}$. The order n Taylor approximation of f at $x = x_0$ for some $x_0 \in D$ is given by

$$p_n(x) = f(x) = f(x_0) + f^{(1)}(x_0)(x - x_0) + \frac{1}{2}f^{(2)}(x_0)(x - x_0)^2 + \dots + \frac{1}{n!}f^{(n)}(x_0)(x - x_0)^n \quad (7.38)$$

such that $f^{(i)}(x_0)$ denotes the i -th order derivative of f at $x = x_0$.

Definition 7.36 can be extended to function f that is $n+1$ times continuously differentiable and it is defined as follows.

DEFINITION 7.37 (Taylor-Lagrange Formula.) If f is $n+1$ times continuously differentiable on the domain D , then we can expand f in its Taylor series around any point $x_0 \in D$ and we have according to Lagrange formula:

$$\forall x \in I, \exists \xi \text{ between } x_0 \text{ and } x_1, \text{ s.t } f(x) = \underbrace{\left(\sum_{i=0}^n \frac{f^{(i)}(x_0)}{i!} (x - x_0)^i \right)}_{p_n(x)} + \underbrace{\frac{f^{(n+1)}(\xi)}{(n+1)!} (x - x_0)^{n+1}}_{r_n(x)} \quad (7.39)$$

wherein $r_n(x) = f(x) - p_n(x)$ is the Lagrange remainder.

Remark 7.38. If f is a continuous function and differentiable at least to order $n+1$ then the last term of the Taylor series is the remainder $r_n(x)$.

EXAMPLE 7.39 Let $I \in [-1, 1]$, $f(x) = \exp(x)$. Its Taylor series around $x_0 = 0$ is: $f(x) = \sum_{i=0}^{\infty} \frac{1}{i!} (x^i)$. Now consider its order 2 Taylor approximation: $p_2(x) =$

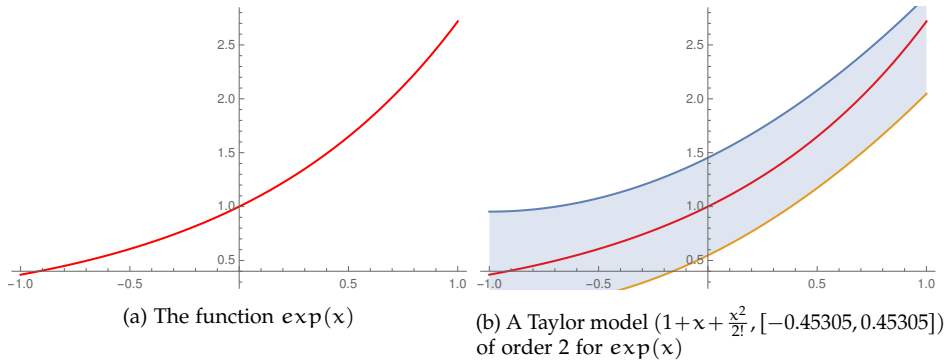


Figure 7.13 – Taylor model over-approximation for the function $\exp(x)$

$1 + x + 0.5x^2$. Thus, an order 2 Taylor model of the function $\exp(x)$ over is given by $[-1, 1]$

$$\left(1 + x + \frac{x^2}{2!}, [-0.45305, 0.45305]\right) \quad (7.40)$$

such that remainder interval is obtained using the Lagrange formula (Equation 7.39) as $\frac{1}{(2+1)!} \cdot \exp([-0.5, 0.5]) \cdot [-0.5, 0.5]^3$

In Figure 7.13b, the Taylor model (Equation 7.40) can be seen as a tube around the function $\exp(x)$.

Operations on Taylor models. Consider two n -th order Taylor models $T_1 = (p_1, I_1)$ and $T_2 = (p_2, I_2)$ around x_0 over the domain D . The addition operation is defined as

$$T_1 + T_2 = (p_1 + p_2, I_1 + I_2).$$

The multiplication operation is defined as

$$T_1 \cdot T_2 = (p_{1,2}, I_{1,2})$$

where $p_{1,2}$ is the part of the polynomial $p_1 \cdot p_2$ up to order n and

$$I_{1,2} = B(p_e) + B(p_1) \cdot I_2 + B(p_2) \cdot I_1 + I_1 \cdot I_2$$

where p_e is the part of the polynomial $p_1 \cdot p_2$ of orders $(n + 1)$ to $2n$, and $B(p)$ denotes a bound of p on the domain D .

The inverse derivation operation on Taylor models is given by

$$\partial^{-1}(p, I) = \int_{a_i}^{b_i} (p(x) - p_e(x), (B(p_e) + I) \cdot [a_i, b_i]) \quad (7.41)$$

EXAMPLE 7.40 Consider the 2 order Taylor model shown in Equation (7.40) for the function $\exp(x)$. We would like to compute:

$$\left(1 + x + \frac{x^2}{2!}, [-0.45305, 0.45305]\right) \cdot \left(1 + x + \frac{x^2}{2!}, [-0.45305, 0.45305]\right)$$

wherein $(1 + x + \frac{x^2}{2!}, [-0.45305, 0.45305]) = (p_1, I_1)$ and $(p_2, I_2) = (p_1, I_1)$. The resulting polynomial after removing the terms $(P_e = x^3 + 0.25x^4)$ having order > 2 is

$$1 + 2x + 2x^2.$$

The remainder interval can be computed by evaluating the polynomial p_e over the interval $[-1, 1]$:

$$B(p_e) = ([-1, 1])^3 + 0.25([-1, 1])^4 = [-1, 1.25]$$

$$B(p_1) \cdot I_2 = ([1, 1] + [-1, 1] + 0.5([-1, 1])^2) \cdot [-0.45305, 0.45305] = [-1.13262, 1.13262]$$

$$B(p_2) \cdot I_1 = ([1, 1] + [-1, 1] + 0.5([-1, 1])^2) \cdot [-0.45305, 0.45305] = [-1.13262, 1.13262]$$

$$I_1 \cdot I_2 = [-0.45305, 0.45305] \cdot [-0.45305, 0.45305] = [-0.20525, 0.20525]$$

$$B(p_e) + B(p_1) \cdot I_2 + B(p_2) \cdot I_1 + I_1 \cdot I_2 = [-3.47050, 3.72050]$$

Thus, the 2 order Taylor model approximation for

$$(1 + x + \frac{x^2}{2!}, [-0.45305, 0.45305]) \cdot (1 + x + \frac{x^2}{2!}, [-0.45305, 0.45305])$$

is

$$(1 + 2x + 2x^2, [-3.47050, 3.72050]).$$

7.4 EXAMPLES: ILLUSTRATING EVALUATION OF REMAINDER INTERVAL BY PICARD OPERATOR

EXAMPLE 7.41 In the following examples we will compute an n -th order Taylor polynomial of the flow either by using Lie derivatives or by Picard iteration. Thereupon we will find an interval I such that evaluating the Picard operator over (p_n, I) yields a Taylor model of the form (p_n, J) wherein $J \subseteq I$. In other words, we wish to find an interval I over which the Picard iteration is contractive.

Consider the ODE $\dot{x} = f(x) = \sin(x)$. We want to illustrate that the 3 order Taylor model given by

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

is an over-approximation of the solution $x(t)$ over the time $[0, 0.02]$ with the initial condition $x(0) = x_0$ where $x_0 \in [-1, 1]$. The order 3 Taylor approximation of $f(x)$ is $p(x) = x - \frac{x^3}{6}$.

We wish to compute the polynomial p_3 representing the Taylor expansion of degree 3 for the flow map. Recall that this polynomial can be computed by Lie derivatives or Picard iteration. We will compute the polynomial by Lie derivatives which are:

$$\mathcal{L}(x) = x - \frac{x^3}{6},$$

$$\mathcal{L}^2(x) = \mathcal{L}(x - \frac{x^3}{6}) = x - \frac{2}{3}x^3 + \frac{x^5}{2},$$

and

$$\mathcal{L}^3(x) = \mathcal{L}(x - \frac{2}{3}x^3 + \frac{x^5}{2}) = x - \frac{x^3}{6} - 2x^3 + \frac{x^5}{3} + \frac{5}{2}x^5 - \frac{5}{12}x^7.$$

The polynomial p_3 is computed by considering the expansion around $x = x_0$ and $t = 0$, i.e.,

$$p_3(x_0, t) = x_0 + (x_0 - \frac{x_0^3}{6})t + (x_0 - \frac{2}{3}x_0^3 + \frac{x_0^5}{12})\frac{t^2}{2!} + (x_0 - \frac{13}{6}x_0^3 - \frac{1}{12}x_0^5 + \frac{5}{72}x_0^7)\frac{t^3}{3!}$$

Next, we compute a polynomial of order 3 for the truncated Lie series by removing monomials of degree greater than 3. As a result, the polynomial obtained is

$$p_3 = x_0 + x_0 t + 0.5x_0 t^2.$$

We wish to consider the Taylor model $(p_3, [-0.1, 0.1])$. We then evaluate the Picard operator over $(p_3, [-0.1, 0.1])$ to compute a Taylor model (p_3, J) . If the Picard operator is contractive on $(p_3, [-0.1, 0.1])$, i.e., $J \subseteq [-0.1, 0.1]$ then $(p_3; [-0.1, 0.1])$ is a valid over-approximation for the solution $x(t)$ over the time interval $[0, 0.02]$.

Thus, the Taylor model Picard operator is given by

$$\mathcal{P}_f(p_3, [-0.1, 0.1]) = x_0 + \int_0^t ((p_3, [-0.1, 0.1]) - \frac{(p_3, [-0.1, 0.1])^3}{6}) \quad (7.42)$$

In Equation (7.42), we will break the integration step into several sub-steps. For instance, first we will compute the 3 order Taylor model approximation for

$$(p_3, [-0.1, 0.1])^2$$

followed by

$$1 - \frac{1}{6}(p_3, [-0.1, 0.1])^2$$

and finally

$$(p_3, [-0.1, 0.1])(1 - \frac{1}{6}(p_3, [-0.1, 0.1])^2).$$

For computing the 3 order Taylor approximation of $((p_3, [-0.1, 0.1])^2)$ we drop the terms with degree greater than 3 in

$$x_0^2 + x_0^2 t^2 + 0.25x_0^2 t^4 + 2x_0^2 t + x_0^2 t^2 + x_0^2 t^3$$

and we obtain the 3 order polynomial as

$$x_0^2 + 2x_0^2 t,$$

and the remainder polynomial is given by

$$p_e = 2x_0^2 t^2 + 0.25x_0^2 t^4 + x_0^2 t^3.$$

We will use the Taylor model arithmetic of multiplication operator for two Taylor models to compute the remainder interval:

$$B(p_e) + B(p) \cdot I + B(p) \cdot I + I \cdot I.$$

The bound of p_e over the interval $[-1, 1]$ can be computed as

$$B(p_e) = 2([-1, 1])^2([0, 0.02])^2 + 0.25([-1, 1])^2([0, 0.02])^4 + ([-1, 1])^2([0, 0.02])^3 = [0.00000, 0.00081]$$

The bound of polynomial p_3 over the interval $I = [-1, 1]$ is

$$B(p) = [-1, 1] + ([-1, 1])([0, 0.02]) + 0.5([-1, 1])([0, 0.02])^2 = [-1.02020, 1.02020],$$

$$B(p) \cdot I = [-0.10202, 0.10202],$$

and

$$I_1 \cdot I_2 = [-0.1, 0.1] \cdot [-0.1, 0.1] = [0, 0.01].$$

Thus, the remainder interval is

$$B(p_e) + B(p) \cdot I + B(p) \cdot I + I \cdot I = [-0.20404, 0.21485],$$

and the Taylor model approximation for $((p_3, [-0.1, 0.1]))^2$ is

$$(x_0^2 + 2x_0^2t, [-0.20404, 0.21485]).$$

Using this we can compute the Taylor approximation of $1 - \frac{1}{6}(p_3, [-0.1, 0.1])^2$ which is

$$1 - \frac{1}{6}(x_0^2 + 2x_0^2t, [-0.03581, 0.03401]).$$

Next we will compute the 3 order Taylor approximation of

$$(p_3, [-0.1, 0.1])(1 - \frac{1}{6}(p_3, [-0.1, 0.1])^2),$$

or

$$(x_0 + x_0t + 0.5x_0t^2, [-0.1, 0.1]) \cdot (1 - \frac{1}{6}(x_0^2 + 2x_0^2t, [-0.03581, 0.03401]))$$

wherein

$$(x_0 + x_0t + 0.5x_0t^2, [-0.1, 0.1]) = (p_1, I_1),$$

and

$$(1 - \frac{1}{6}(x_0^2 + 2x_0^2t, [-0.03581, 0.03401])) = (p_2, I_2).$$

The multiplication of two polynomials yield

$$x_0 - \frac{x_0^3}{6} - \frac{x_0^3t}{3} + x_0t - \frac{x_0^3t}{6} - \frac{x_0^3t^2}{3} + 0.5x_0t^2 - \frac{0.5}{6}x_0^3t^2 - \frac{0.5}{3}x_0^3t^3$$

wherein the remainder polynomial is given by

$$p_e = -\frac{x_0^3t}{3} - \frac{x_0^3t}{6} - \frac{x_0^3t^2}{3} - \frac{0.5}{6}x_0^3t^2 - \frac{0.5}{3}x_0^3t^3.$$

The bound of the remainder polynomial can be computed as

$$B(p_e) = -\frac{1}{3}([-1, 1])^3([0, 0.02]) - \frac{1}{6}([-1, 1])^3([0, 0.02]) - \frac{1}{3}([-1, 1])^3([0, 0.02])^2 - \frac{0.5}{6}([-1, 1])^3([0, 0.02])^2 - \frac{0.5}{3}([-1, 1])^3([0, 0.02])^3 = [-0.01017, 0.01017]$$

and the remaining bounds for calculating the remainder interval for the multiplication operation are computed as follows:

$$B(p_1) = [-1, 1] + ([-1, 1])([0, 0.02]) + 0.5([-1, 1])([0.02])^2 = [-1.02020, 1.02020]$$

$$B(p_1) \cdot I_2 = [-1.02020, 1.02020] \cdot [-0.03581, 0.03401]$$

$$B(p_2) \cdot I_1 = ([-1, 1] - \frac{1}{6}([-1, 1])^2 - \frac{1}{3}([-1, 1])^2([0, 0.02])) \cdot [-0.1, 0.1]$$

$$I_1 \cdot I_2 = [-0.03581, 0.03401] \cdot [-0.1, 0.1]$$

Thus, the remainder interval of 3 order Taylor approximation for $(p_3, [-0.1, 0.1])(1 - \frac{1}{6}(p_3, [-0.1, 0.1])^2)$ is

$$B(p_e) + B(p_1) \cdot I_2 + B(p_2) \cdot I_1 + I_1 \cdot I_2 = [-0.15028, 0.15028],$$

and the Taylor model approximation is given by

$$(x_0 - \frac{x_0^3}{6} + x_0 t + 0.5x_0 t^2, [-0.15028, 0.15028]) \quad (7.43)$$

We substitute the Taylor model (Equation (7.43)) in Equation (7.42) and the Picard operator becomes

$$\begin{aligned} \mathcal{P}_f(p_3, [-0.1, 0.1]) &= x_0 + \int_0^t (x_0 - \frac{x_0^3}{6} + x_0 s + 0.5x_0 s^2, [-0.15028, 0.15028]) \\ &= (x_0 + x_0 t + x_0 \frac{t^2}{2}, (B(-\frac{x_0^3}{6} t + 0.5x_0 \frac{t^3}{3}) + [-0.15028, 0.15028]) \cdot [0, 0.02]) \\ &= (p_3, (B(-\frac{x_0^3}{6} t + 0.5x_0 \frac{t^3}{3}) + [-0.15028, 0.15028]) \cdot [0, 0.02]) \end{aligned}$$

The bound of $-\frac{x_0^3}{6} t + 0.5x_0 \frac{t^3}{3}$ can be computed as

$$\frac{0.5}{3}[-1, 1]([0, 0.02])^3 - \frac{1}{6}([-1, 1])^3[0, 0.02] = [0.00333, 0.00333].$$

Thus, the Picard operator over $p_3 + [-0.1, 0.1]$ yield (p_3, J) , i.e.,

$$(p_3, [-0.00307, 0.00307]).$$

As a result, the Picard operator is contractive on $(p_3, [-0.1, 0.1])$. This suggests that $(p_3, [-0.1, 0.1])$ is a valid over-approximation over the time interval $[0, 0.02]$.

Remark 7.42. The steps illustrated in the above example for constructing a flowpipe produces a Taylor model that over-approximates the flow map $\varphi_f(x_0, t)$ for $t \in [0, \delta]$.

EXAMPLE 7.43 As of now we saw how to evaluate if a Taylor model is an over-approximation of a flow map. Now, we consider a two dimensional ODE $\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = f(x, y) = \begin{pmatrix} 1 + y \\ -x^2 \end{pmatrix}$ over the domain $x \in [-1, 1]$, $y \in [-0.5, 0.5]$. We want to compute a 3 order Taylor model and then evaluate using the Picard operator if this model is an over-approximation of the flow map over a time

interval $t \in [0, 0.02]$. First, we will compute an order 3 polynomial. Recall that the polynomial can be computed in two ways: (a) using a truncated Lie series by computing Lie derivatives or (b) applying Picard iteration. Here, we will use Picard iteration and the sequence is

$$p_0(x, y, t) = \begin{pmatrix} x \\ y \end{pmatrix}$$

$$p_1(x, y, t) = \begin{pmatrix} x \\ y \end{pmatrix} + \int_0^t \begin{pmatrix} 1 + y \\ -x^2 \end{pmatrix} ds = \begin{pmatrix} x + t \\ y \end{pmatrix}$$

$$p_2(x, y, t) = \begin{pmatrix} x \\ y \end{pmatrix} + \int_0^t \begin{pmatrix} 1 + y \\ -(x + s)^2 \end{pmatrix} ds = \begin{pmatrix} x + t + yt \\ y \end{pmatrix}$$

$$p_3(x, y, t) = \begin{pmatrix} x \\ y \end{pmatrix} + \int_0^t \begin{pmatrix} 1 + y \\ -(x + s + ys)^2 \end{pmatrix} ds = \begin{pmatrix} x + t + yt \\ y - x^2t - xt^2 - \frac{1}{3}t^3 \end{pmatrix}$$

Consider an initial estimate of the remainder interval as $I_0 = \begin{pmatrix} [-0.1, 0.1] \\ [-0.1, 0.1] \end{pmatrix}$

The Picard operator is given by

$$\mathcal{P}_f(p_3, I_0) = \begin{pmatrix} x \\ y \end{pmatrix} + \int_0^t \begin{pmatrix} 1 + (y - x^2s - xs^2 - \frac{1}{3}s^3, [-0.1, 0.1]) \\ -((x + s + ys, [-0.1, 0.1]))^2 \end{pmatrix} ds \quad (7.44)$$

First, we will compute the 3 order Taylor model approximation for $((x + t + yt, [-0.1, 0.1]))^2$.

We will use the Taylor order arithmetic of multiplication operator for Taylor models to compute the remainder interval.

The polynomial obtained after the multiplication is given by

$$x^2 + t^2 + y^2t^2 + 2xt + 2xyt + 2yt^2,$$

wherein the remainder polynomial is

$$P_e = y^2t^2.$$

The bound of the remainder polynomial can be computed as

$$B(p_e) = ([-0.5, 0.5])^2([0, 0.02])^2 = [0.00000, 0.00010],$$

and the other bounds for calculating the remainder interval for the multiplication operation are computed as follows:

$$B(p) = [-1, 1] + [0, 0.02] + [-0.5, 0.5] \cdot [0, 0.02] = [-1.01000, 1.03000]$$

$$B(p) \cdot I = [-0.10300, 0.10300]$$

$$I_1 \cdot I_2 = [-0.1, 0.1] \cdot [-0.1, 0.1] = [0, 0.01]$$

Hence the remainder interval is

$$B(p_e) + B(p_1) \cdot I_2 + B(p_2) \cdot I_1 + I_1 \cdot I_2 = [-0.20600, 0.21610],$$

and the 3 order Taylor approximation of $(x + t + yt, [-0.1, 0.1])^2$ is

$$(x^2 + t^2 + 2xt + 2xyt + 2yt^2, [-0.20600, 0.21610]).$$

As a result, the Picard operator in Equation (7.44) becomes

$$\begin{aligned} \mathcal{P}_f(p_3, I_0) &= \begin{pmatrix} x \\ y \end{pmatrix} + \int_0^t \begin{pmatrix} 1 + (y - x^2s - xs^2 - \frac{1}{3}s^3, [-0.1, 0.1]) \\ -(x^2 + s^2 + 2xs + 2xys + 2ys^2, [-0.20600, 0.21610]) \end{pmatrix} ds \\ &= \begin{pmatrix} x \\ y \end{pmatrix} + \int_0^t \begin{pmatrix} 1 + (y - x^2s - xs^2 - \frac{1}{3}s^3, [-0.1, 0.1]) \\ -x^2 - s^2 - 2xs - 2xys - 2ys^2, [-0.21610, 0.20600] \end{pmatrix} ds \\ &= \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t + yt, ((B(-x^2\frac{t^2}{2} - x\frac{t^3}{3} - \frac{1}{3}\frac{t^3}{12}) + [-0.1, 0.1]) \cdot [0, 0.02]) \\ -x^2t - \frac{t^3}{3} - xt^2, ((B(-xyt^2 - \frac{2}{3}yt^3) + [-0.21610, 0.20600]) \cdot [0, 0.02]) \end{pmatrix} \\ &= \begin{pmatrix} (x + t + yt, [-0.00200, 0.00200]) \\ (y - x^2t - \frac{t^3}{3} - xt^2, [-0.00433, 0.00412]) \end{pmatrix} \\ &= \begin{pmatrix} (x + t + yt, [-0.00200, 0.00200]) \\ (y - x^2t - \frac{t^3}{3} - xt^2, [-0.00433, 0.00412]) \end{pmatrix}. \end{aligned}$$

Accordingly, the Picard operator is contractive on

$$\begin{pmatrix} (x + t + yt, [-0.00200, 0.00200]) \\ (y - x^2t - \frac{t^3}{3} - xt^2, [-0.00433, 0.00412]) \end{pmatrix},$$

and hence it is a valid over-approximation over the time interval $[0, 0.02]$.

Remark 7.44. Recall that one of the challenges in flowpipe construction is the evaluation of a safe remainder interval I for a time interval such that the Taylor model (p, I) is an over-approximation of the flow map. By Banach fixed-point theorem, this remainder interval is sufficient if the Picard operator is contractive on (p, I) . Initially, we estimate this interval and check the contractiveness of Picard operator. However, this initial estimation can be incorrect and may not result in a contractive interval.

EXAMPLE 7.45 (Taylor model for Van-der-Pol oscillator) We will use the same Vanderpol example described earlier in this chapter and was defined by

$$\begin{aligned} \dot{x} &= 2y \\ \dot{y} &= -0.8x - 10(x^2 - 0.21)y \end{aligned} \tag{7.45}$$

with the initial set $x_0 \in [-1.2, 1.2]$ and $y_0 \in [-1.2, 1.2]$. This example is taken from [HK13]. We want to compute a 3 order Taylor model and then evaluate using the Picard operator if this model is an over-approximation of the flow map over a time interval $t \in [0, 0.05]$ with an initial estimate of the remainder interval as $I_0 = \begin{pmatrix} [-0.1, 0.1] \\ [-0.1, 0.1] \end{pmatrix}$.

We first generate the Taylor polynomial by Lie derivatives sequence given by

$$\mathcal{L}_f \left(\begin{pmatrix} x \\ y \end{pmatrix} \right) = \begin{pmatrix} 2y \\ -0.8x - 10x^2y + 2.1y \end{pmatrix}$$

$$\mathcal{L}_f^2 \left(\begin{pmatrix} x \\ y \end{pmatrix} \right) = \begin{pmatrix} -1.6x - 20x^2y + 4.2y \\ -1.68x + 8x^3 + 2.81y - 42x^2y + 100x^4y - 40xy^2 \end{pmatrix}$$

To construct the polynomial we consider the following expansion

$$\begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 2y \\ -0.8x - 10x^2y + 2.1y \end{pmatrix} t + \begin{pmatrix} -1.6x - 20x^2y + 4.2y \\ -1.68x + 8x^3 + 2.81y - 42x^2y + 100x^4y - 40xy^2 \end{pmatrix} \frac{t^2}{2}. \quad (7.46)$$

After removing the monomials of degree greater than 3 the polynomial obtained is

$$p_3(x, y, t) = \begin{pmatrix} x + 2yt - 1.6x \frac{t^2}{2} + 4.2y \frac{t^2}{2} \\ y - 0.8xt + 2.1yt - 1.68x \frac{t^2}{2} + 2.81y \frac{t^2}{2} \end{pmatrix}.$$

We start with the remainder estimate $I_0 = \begin{pmatrix} [-0.1, 0.1] \\ [-0.1, 0.1] \end{pmatrix}$ for the order 3 Taylor expansion, and we compute the following order 3 Taylor model extension of the Picard operation, i.e.,

$$\mathcal{P}_f(p_3, I_0) = \begin{pmatrix} x \\ y \end{pmatrix} + \int_0^t \begin{pmatrix} 2b \\ -0.8a - 10a^2b + 2.1b \end{pmatrix} ds \quad (7.47)$$

wherein

$$a = (x + 2ys - 1.6x \frac{s^2}{2} + 4.2y \frac{s^2}{2}, [-0.1, 0.1]),$$

and

$$b = (y - 0.8xs + 2.1ys - 1.68x \frac{s^2}{2} + 2.81y \frac{s^2}{2}, [-0.1, 0.1]).$$

We will simplify Equation (7.47) by computing the 3 order Taylor model approximations of a^2 and $10a^2b$ before performing the integration step. We will begin with

$$((x + 2ys - 1.6x \frac{s^2}{2} + 4.2y \frac{s^2}{2}, [-0.1, 0.1]))^2.$$

After multiplying the polynomials and dropping the monomials of degree greater than 3 the polynomial obtained is

$$p = x^2 + 4txy,$$

and the remainder polynomial is

$$p_e = -1.6t^2x^2 + 0.64t^4x^2 + 4.2t^2xy - 3.2t^3xy - 3.36t^4xy + 4t^2y^2 + 8.4t^3y^2 + 4.41t^4y^2$$

The bound of the p_e over $x, y \in [-1.2, 1.2]$ and $t \in [0, 0.05]$ is

$$B(p_e) = [-0.28723, 0.30741].$$

The remaining bounds for computing the remainder interval for the Taylor model approximation of a^2 are

$$\begin{aligned} & B(x + 2ys - 1.6x \frac{s^2}{2} + 4.2y \frac{s^2}{2}) \cdot [-0.1, 0.1] \\ &= [-1.32870, 1.32870] \cdot [-0.1, 0.1] = [-0.13287, 0.13287]. \end{aligned}$$

and hence the remainder interval can be computed as

$$\begin{aligned} & [-0.28723, 0.30741] + [-0.13287, 0.13287] + [-0.13287, 0.13287] + [-0.1, 0.1] \cdot [-0.1, 0.1] \\ &= [-0.28723, 0.30741] \end{aligned}$$

So, the 3 order Taylor model approximation of a^2 is

$$(x^2 + 4txy, [-0.28723, 0.30741]).$$

The next step is to compute the 3 order Taylor model approximation for a^2b or

$$(x^2 + 4txy, [-0.28723, 0.30741])(y - 0.8xt + 2.1yt - 1.68x \frac{t^2}{2} + 2.81y \frac{t^2}{2}, [-0.1, 0.1])$$

The 3 degree polynomial obtained is x^2y with the remainder polynomial is $p_e = -0.8tx^3 - 0.84t^2x^3 + 2.1tx^2y - 1.795t^2x^2y - 3.36t^3x^2y + 4txy^2 + 8.4t^2xy^2 + 5.62t^3xy^2$, whose bound evaluated over $x, y \in [-1.2, 1.2]$ and $t \in [0, 0.05]$ is

$$B(p_e) = [-0.64577, 0.64577].$$

We compute the following bounds:

$$B(x^2 + 4txy) \cdot [-0.1, 0.1] = [-0.28800, 1.72800] \cdot [-0.1, 0.1] = [-0.17280, 0.17280]$$

$$\begin{aligned} & B(y - 0.8xt + 2.1yt - 1.68x \frac{t^2}{2} + 2.81y \frac{t^2}{2}) \cdot [-0.28723, 0.30741] \\ &= [-1.38074, 1.38074] \cdot [-0.28723, 0.30741] = [-0.42445, 0.42445] \end{aligned}$$

for calculating the remainder interval for the Taylor model approximation of $10a^2b$ which is

$$\begin{aligned} & [-0.64577, 0.64577] + [-0.17280, 0.17280] + [-0.42445, 0.42445] + \\ & [-0.28723, 0.30741] \cdot [-0.1, 0.1] = [-1.27376, 1.27376], \end{aligned}$$

and hence the Taylor model approximation is

$$(10x^2y, [-12.7376, 12.7376]).$$

Substituting the Taylor model approximation of $10a^2b$ in Equation (7.47) we perform the integration step and the Picard operator is

$$\left(\begin{aligned} & (x + 2yt - 1.6x \frac{t^2}{2} + 4.2y \frac{t^2}{2}, (B(-1.68xt^3 + 2.81yt^3) + [-0.2, 0.2]) \cdot [0, 0.05]) \\ & (y - 0.8xt - 0.8yt^2 + 2.1yt - 1.68x \frac{t^2}{2} + 2.81y \frac{t^2}{2}, (B) \cdot [0, 0.05]) \end{aligned} \right)$$

wherein $B = B(p_e) + [-0.1, 0.1] + [-0.1, 0.1] + [-12.7376, 12.7376]$ and

$$p_e = 0.8 \cdot 0.8x \frac{t^3}{3} - 2.1 \cdot 0.8y \frac{t^3}{3} - 1.68x \frac{t^3}{6} + 2.81y \frac{t^3}{6} - 10x^2yt.$$

The Picard operator yields a Taylor model (p_3, I_1) given by

$$\left(\begin{array}{l} (x + 2yt - 1.6x\frac{t^2}{2} + 4.2y\frac{t^2}{2}, [-0.01014, 0.01014]) \\ (y - 0.8xt - 0.8yt^2 + 2.1yt - 1.68x\frac{t^2}{2} + 2.81y\frac{t^2}{2}, [-0.69012, 0.69012]) \end{array} \right).$$

As a result, the Picard operator is not contractive on (p_3, I_0) .

Remark 7.46. If the Picard operator is not contractive on the initial estimate of the remainder interval, normally the interval is shrunk and verified again [RSÁ14, SC16]. We used Flow* [CÁS13] and also the prototype from [GP17] to derive a Taylor model for the Vanderpol oscillator defined by Equation (7.45), but could not evaluate an order 2 Taylor model for a time step $t = 0.05$. The reason for the remainder interval not being contractive is that the time interval is too wide. Also, the initial estimate is too bounded.

We managed to derive an order 8 Taylor model (shown in Equations (7.48) and (7.49)) for a step size of $t = 0.0007$ and an initial remainder estimate of $[-0.00001, 0.00001]$.

$$\begin{aligned} x = & 1.2 * x + 2.4 * t * y + 2.52 * t^2 * y - 9.6e - 1 * t^2 * x + \\ & 1.124 * t^3 * y - 6.72e - 1 * t^3 * x - 1.728e1 * t^2 * x^2 * y + \\ & 2.541e - 1 * t^4 * y - 2.248e - 1 * t^4 * x - 2.304e1 * t^3 * x * y^2 - \\ & 2.4192e1 * t^3 * x^2 * y + 4.608 * t^3 * x^3 + 1.6802e - 2 * t^5 * y - \\ & 4.0656e - 2 * t^5 * x + [-8.7694e - 8, 8.7694e - 8] \end{aligned} \quad (7.48)$$

$$\begin{aligned} y = & 1.2 * y + 2.52 * t * y - 9.6e - 1 * t * x + 1.686 * t^2 * y - \\ & 1.008 * t^2 * x - 1.728e1 * t * x^2 * y + 5.082e - 1 * t^3 * y - \\ & 4.496e - 1 * t^3 * x - 3.456e1 * t^2 * x * y^2 - 3.6288e1 * t^2 * x^2 * y + \\ & 6.912 * t^2 * x^3 + 4.2005e - 2 * t^4 * y - 1.0164e - 1 * t^4 * x - \\ & 2.304e1 * t^3 * y^3 - 9.6768e1 * t^3 * x * y^2 - 1.2384 * t^3 * x^2 * y + \\ & 9.6768 * t^3 * x^3 - 2.30139e - 2 * t^5 * y - 6.7208e - 3 * t^5 * x + \\ & 1.24416e2 * t^2 * x^4 * y - 8.4672e1 * t^4 * y^3 - 8.90208e1 * t^4 * x * y^2 + \\ & 5.074272e1 * t^4 * x^2 * y + 2.4768e - 1 * t^4 * x^3 - 1.0295e - 2 * t^6 * y + \\ & 3.06852e - 3 * t^6 * x + [-1.6266e - 6, 1.6266e - 6] \end{aligned} \quad (7.49)$$

Further on we will use the Vanderpol oscillator from [CAS12] and defined by

$$\begin{aligned} \dot{x} &= y \\ \dot{y} &= y - x - x^2y \end{aligned} \quad (7.50)$$

with the initial set $x_0 \in [1.1, 1.4]$ and $y_0 \in [2.35, 2.45]$.

We compute a 6 order Taylor model that is an over-approximation of the flow map over a time interval $t \in [0, 0.02]$ with an initial estimate of the remainder interval as $I_0 = \left(\begin{array}{l} [-0.00001, 0.00001] \\ [-0.00001, 0.00001] \end{array} \right)$.

The Taylor model obtained is given by the following Equations (7.51) and (7.52)

$$\begin{aligned} x = & 1.25 + 0.14999x + 2.4 * 0.02 + 0.05 * 0.02 * y - 1.3 * 0.0004 - \\ & 0.0140625 * 0.0004 * y - 0.525 * 0.0004 * x - 2.55625 * 0.000008 - \\ & 0.009375 * 0.0004 * x * y - 0.027 * 0.0004 * x * x - \quad (7.51) \\ & 0.1056966 * 0.000008 * y - 0.0270625 * 0.000008 * x + \\ & 1.265805 * 0.00000016 + [-0.00000187, 0.00000391] \end{aligned}$$

$$\begin{aligned} y = & 2.4 + 0.05 * y - 2.60 * 0.02 - 0.028125 * 0.02 * y - 1.05 * 0.02 * x - \\ & 7.66875 * 0.0004 - 0.01875 * 0.02 * x * y - 0.054 * 0.02 * x * x - \\ & 0.3170898 * 0.0004 * y - 0.0811875 * 0.0004 * x + 5.06322395 * 0.000008 - \\ & 0.001125 * 0.02 * x * x * y - 0.003125 * 0.0004 * y * y - 0.02545312 * 0.0004 * x * y + \\ & 0.2413125 * 0.0004 * x * x + 0.02301684 * 0.000008 * y + 5.2348164 * 0.000008 * x + \\ & 20.6400466 * 0.00000016 - 0.000375 * 0.0004 * x * y * y + \\ & 0.0041484 * 0.0004 * x * x * y + 0.0219375 * 0.0004 * x * x * x - \\ & 0.00365625 * 0.000008 * y * y + 0.1990336 * 0.000008 * x * y + \\ & 0.57141796 * 0.000008 * x * x + 1.3241952 * 0.00000016 * y + \\ & 1.863067 * 0.00000016 * x - 7.8098148 * 0.0000000032 + [-0.00000497, 0.00000411] \quad (7.52) \end{aligned}$$

In Equations (7.51) and (7.52) is the first Taylor model in time $[0, 0.02]$. Recall that methods relying on flowpipe construction approach compute a flowpipe segment for every time step in an interval $[t_1, t_2]$ of interest. Thus, the solution to an IVP problem is the set union of these flowpipe segments computed during the time interval $[t_1, t_2]$. For our experiment, we will do something similar. We will evaluate a Taylor model on each time step and apply the CP algorithm on it. For instance, starting with the initial set $x_0 \in [1.1, 1.4]$ and $y_0 \in [2.35, 2.45]$, we compute an approximation (collection of zonotopes) for the states reachable from the initial set in time $[0, 0.02]$. We advance similarly for a time interval $[0, 6.70]$. Thus, the positive invariant is the set union of the zonotopes that cover the time interval $[0, 6.70]$.

The Taylor models over the time interval $[0, 0.08]$ for a step size of 0.02 is shown in the appendix. With a time step of 0.02 and order 6 Taylor models, we managed to evaluate 335 Taylor model flowpipes over a time interval of $[0, 6.70]$. All the Taylor models are shown in https://github.com/bibekabi/Prototype_analyzerwithApron/blob/master/Taylormodels_520.txt.

We apply our CP algorithm on every new Taylor model we evaluate on each time step. For the time interval $[0, 5.20]$ and $[0, 6.70]$ the set obtained are shown in Figure 7.14 and 7.15. Each time we apply the CP algorithm, we stop as soon as we find a set of abstract elements whose image by a loop iteration in the abstract domain used for computing the positive invariant set is included inside this set. Then, we again apply the algorithm on a new Taylor model being evaluated. If we compare the Figure 7.15 with the Figure 1 in [CAS12] (which is illustrating the Taylor model flowpipes for the oscillator) they are related. The authors in [CAS12] compute the Taylor model flowpipes

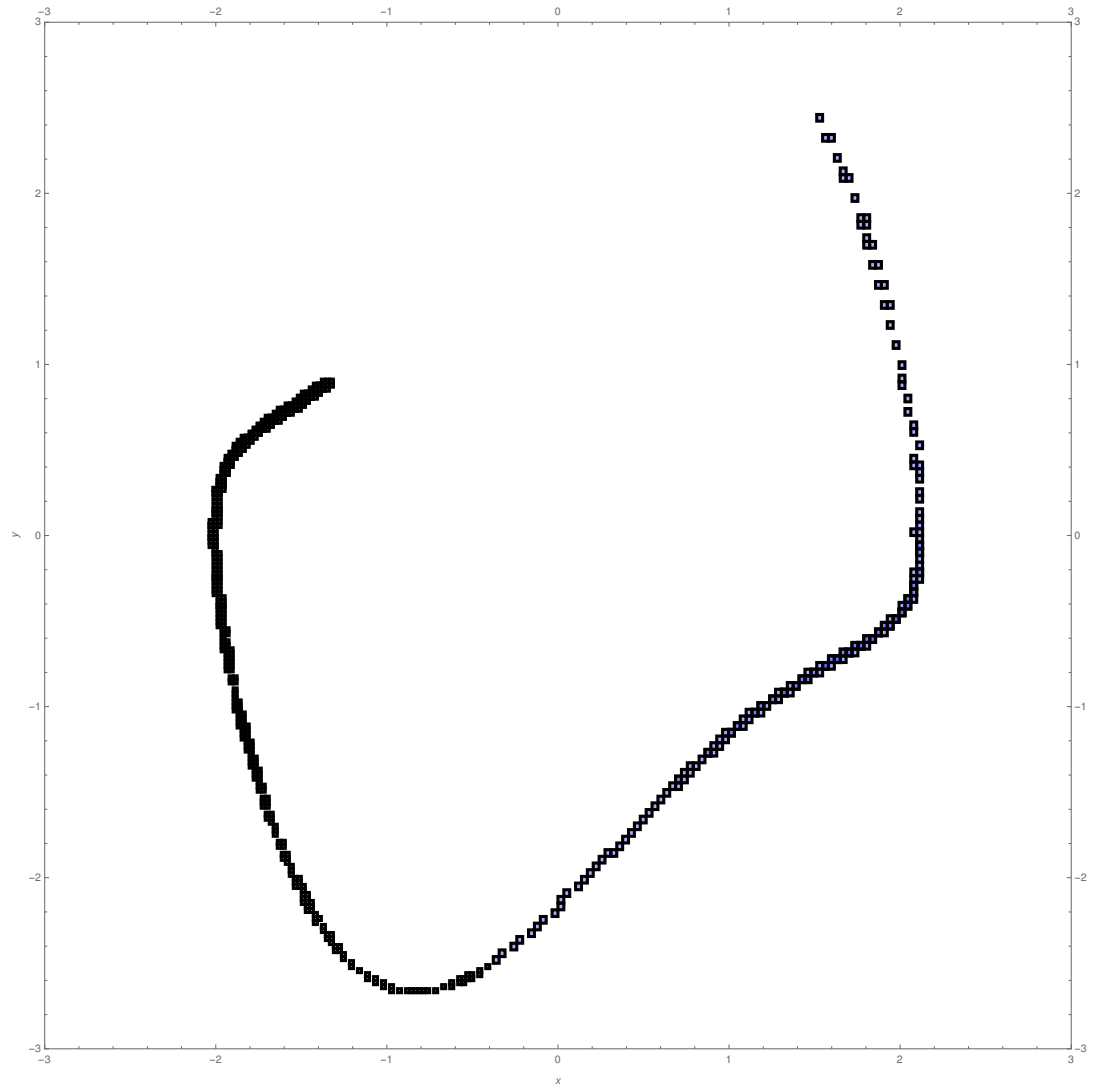


Figure 7.14 – The set obtained after using our CP Algorithm on every new Taylor model (6 order) evaluated at each time step over an interval $[0, 5.20]$.

until oscillation is detected. We can do the same and evaluate Taylor model flowpipes beyond 6.70 either by using smaller time step or by increasing the order of Taylor models. For instance, we computed 340 Taylor models of order 7 over a time interval $[0, 6.80]$ for a time step $t = 0.02$ and applied the CP algorithm on these Taylor models. The set union of zonotopes that cover the time interval $[0, 6.80]$ approximating the positive invariant is shown in Figure 7.16. A prototype analyzer, written in OCaml connected to APRON Abstract Domain Library for finding positive invariant of continuous time systems is available at https://github.com/bibekabi/Continuous_time_system_analyzer.



Figure 7.15 – The set obtained after using our CP Algorithm on every new Taylor model (6 order) evaluated at each time step over an interval $[0, 6.70]$.

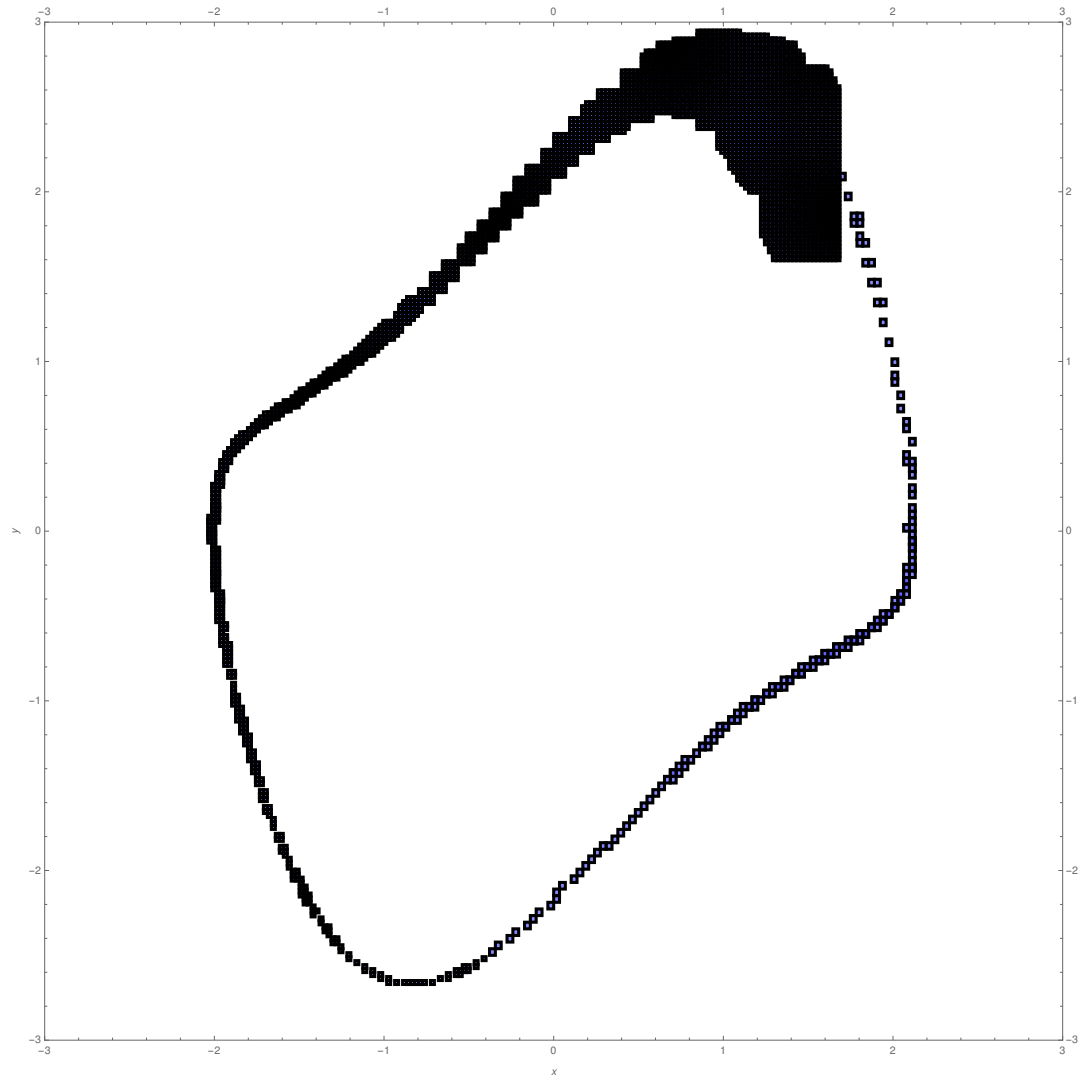
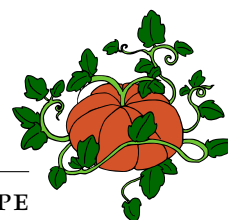


Figure 7.16 – The set obtained after using our CP Algorithm on every new Taylor model (7 order) evaluated at each time step over an interval $[0, 6.80]$.

7.5 CONCLUSION

In this chapter, we have illustrated that it is possible to find an outer-approximation of the positive invariant set for continuous dynamical systems using our new algorithm based on constraint programming and zonotopes. This time the functions analyzed by our method are developed in Taylor models, which seems particularly well suited to use zonotopes.



CONCLUSION AND FUTURE SCOPE

In this thesis, we extend an existing framework combining abstract interpretation and continuous constraint programming for numerical invariant synthesis, by using more expressive underlying abstract domains, such as zonotopes. As zonotopes are not closed under intersection, we had to extend the existing framework, in addition to designing new operations on zonotopes, such as a novel splitting algorithm based on paving zonotopes by sub-zonotopes and parallelotopes. We implemented this method on top of the APRON library, and tested it on programs with non-linear loops that present complex, possibly non-convex, invariants. We present some results demonstrating both the interest of this splitting-based algorithm to synthesize invariants on such programs, and the good compromise presented by its use in combination with zonotopes with respect to its use with both simpler domains such as boxes and octagons, and more expressive domains such as polyhedra. The main contribution of the thesis is briefly reviewed as below.

- We explored in detail the CP based AI approaches for finding invariants. We investigated the use of constraint-solving inspired algorithms for inferring inductive invariants. The CP algorithm is characterized on a numeric abstract domain, which takes care of the evaluation of the body loop semantics and it works by iteratively splitting and tightening a set of abstract elements until an inductive invariant is found.
- Our main contribution was to extend the type of abstract elements this algorithm can rely on, in particular when there is no natural bisection method. We instantiated it to the case of zonotopes, and demonstrated that they provide a good tradeoff, in particular on non-linear programs, and scale up much better than the same algorithm based on simpler domains, such as boxes. These experiments conducted confirm that zonotopes provide a very interesting trade-off between a general purpose abstraction, that stand the comparison with simpler abstractions on basic linear examples, but are also faster and more flexible for the abstraction of more complex, non affine behaviors. In particular, the fact that they allow representing inductive invariants with fewer elements will be even more crucial for the scalability of the approach to higher dimension programs.
- We also explored the use of such techniques for finding an over-approximation of the reachable sets of continuous systems. This serves as a good promise for hybrid system verification in future.

As a scope of future work there are several directions in which we would like to extend our work. They are listed as follows.

- Using the CP framework based on zonotopic abstraction for finding invariants in higher dimensional programs.
- Refining our method using tightening at each iteration of our algorithm, possibly, for non-lattice abstract domains, based on splitting as well as zonotope intersections.
- Recently complex zonotopes have had strong implication in proving safety properties of hybrid systems providing a wider set of representation: non-polytopic as well as polytopic zonotopes [Adi18]. Thus, one of our future endeavors is to extend the set-theoretic operations like inclusion checking and splitting (introduced in Chapter 5), for complex zonotopes.



BIBLIOGRAPHY

- [ABC05] Teodoro Alamo, José Manuel Bravo, and Eduardo F Camacho, *Guaranteed state estimation by zonotopes*, *Automatica* **41** (2005), no. 6, 1035–1043.
- [Adi18] Santosh Arvind Adimoolam, *A calculus of complex zonotopes for invariance and stability verification of hybrid systems*, Ph.D. thesis, 2018.
- [AF92] David Avis and Komei Fukuda, *A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra*, *Discrete & Computational Geometry* **8** (1992), no. 3, 295–313.
- [AGG10] Assalé Adjé, Stéphane Gaubert, and Eric Goubault, *Coupling policy iteration with semi-definite relaxation to compute accurate numerical invariants in static analysis*, *European Symposium on Programming*, Springer, 2010, pp. 23–42.
- [AK11] Matthias Althoff and Bruce H Krogh, *Zonotope bundles for the efficient computation of reachable sets*, 2011 50th IEEE conference on decision and control and European control conference, IEEE, 2011, pp. 6814–6821.
- [Alt15] Matthias Althoff, *An introduction to cora 2015*, *Proc. of the Workshop on Applied Verification for Continuous and Hybrid Systems*, 2015.
- [Arb16] Matthew Arbo, *Zonotopes and hypertoric varieties*.
- [AS12a] Gianluca Amato and Francesca Scozzari, *The abstract domain of parallelotopes*, *Electronic Notes in Theoretical Computer Science* **287** (2012), 17–28.
- [AS12b] ———, *Random: R-based analyzer for numerical domains*, *International Conference on Logic for Programming Artificial Intelligence and Reasoning*, Springer, 2012, pp. 375–382.
- [ASB08] Matthias Althoff, Olaf Stursberg, and Martin Buss, *Reachability analysis of nonlinear systems with uncertain parameters using conservative linearization*, *Decision and Control*, 2008. CDC 2008. 47th IEEE Conference on, IEEE, 2008, pp. 4042–4048.
- [ASB10] ———, *Computing reachable sets of hybrid systems using a combination of zonotopes and polytopes*, *Nonlinear analysis: hybrid systems* **4** (2010), no. 2, 233–249.

-
- [ATS09] Alessandro Abate, Ashish Tiwari, and Shankar Sastry, *Box invariance in biologically-inspired dynamical systems*, *Automatica* **45** (2009), no. 7, 1601–1610.
- [Bai97] Guy David Bailey, *Tilings of zonotopes: Discriminantal arrangements, oriented matroids and enumeration*, University of Minnesota, 1997.
- [Bap96] Philippe Baptiste, *Disjunctive constraints for manufacturing scheduling: Principles and extensions*, *International Journal of Computer Integrated Manufacturing* **9** (1996), no. 4, 306–310.
- [BCC⁺] Julien Bertrane, Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, and Xavier Rival, *Static analysis and verification of aerospace software by abstract interpretation*, *AIAA Infotech@ Aerospace 2010*, p. 3385.
- [BCC⁺09] Olivier Bouissou, Eric Conquet, Patrick Cousot, Radhia Cousot, Jérôme Feret, Khalil Ghorbal, Eric Goubault, David Lesens, Laurent Mauborgne, Antoine Miné, et al., *Space software validation using abstract interpretation*, *The International Space System Engineering Conference: Data Systems in Aerospace-DASIA 2009*, vol. 1, European Space Agency, 2009, pp. 1–7.
- [BCC⁺15] Julien Bertrane, Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, Xavier Rival, et al., *Static analysis and verification of aerospace software by abstract interpretation*, *Foundations and Trends® in Programming Languages* **2** (2015), no. 2-3, 71–190.
- [BCT12] Bruno Belin, Marc Christie, and Charlotte Truchet, *Interactive urban planning with local search techniques: the sustains project*, *CompSust’12-3rd International Conference on Computational Sustainability*, 2012.
- [BCT14] ———, *Interactive design of sustainable cities with a distributed local search solver*, *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, Springer, 2014, pp. 104–119.
- [Ber99] M Berz, *Modern map methods in particle beam physics*, *Adv. Imaging Electron Phys.* **108** (1999), 1–318.
- [BHZ08] Roberto Bagnara, Patricia M Hill, and Enea Zaffanella, *The parma polyhedra library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems*, *Science of Computer Programming* **72** (2008), no. 1-2, 3–21.
- [BKV96] Bernard Billoud, Milutin Kontic, and Alain Viari, *Palingol: a declarative programming language to describe nucleic acids’ secondary structures and to scan sequence databases*, *Nucleic acids research* **24** (1996), no. 8, 1395–1403.

BIBLIOGRAPHY

- [Bla99] Franco Blanchini, *Set invariance in control*, *Automatica* **35** (1999), no. 11, 1747–1767.
- [BM96] Franco Blanchini and Stefano Miani, *Constrained stabilization of continuous-time linear systems*, *Systems & control letters* **28** (1996), no. 2, 95–102.
- [BM98] Martin Berz and Kyoko Makino, *Verified integration of odes and flows using differential algebraic methods on high-order taylor models*, *Reliable computing* **4** (1998), no. 4, 361–369.
- [Boh92] Jochen Bohne, *Eine kombinatorische analyse zonotopaler raumaufteilungen. univ., diss*, 1992.
- [Bra11] Aaron R Bradley, *Sat-based model checking without unrolling*, *International Workshop on Verification, Model Checking, and Abstract Interpretation*, Springer, 2011, pp. 70–87.
- [CAS12] Xin Chen, Erika Abraham, and Sriram Sankaranarayanan, *Taylor model flowpipe construction for non-linear hybrid systems*, *2012 IEEE 33rd Real-Time Systems Symposium*, IEEE, 2012, pp. 183–192.
- [CÁ13] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan, *Flow*^{*}: An analyzer for non-linear hybrid systems*, *International Conference on Computer Aided Verification*, Springer, 2013, pp. 258–263.
- [CC76] Patrick Cousot and Radhia Cousot, *Static determination of dynamic properties of programs*, *Proceedings of the 2nd International Symposium on Programming*, Paris, France, Dunod, 1976.
- [CC77] ———, *Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints*, *Proceedings of POPL*, ACM, 1977, pp. 238–252.
- [CC79] ———, *Systematic design of program analysis frameworks*, *Proceedings of the 6th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, ACM, 1979, pp. 269–282.
- [CCF⁺05] Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, and Xavier Rival, *The astrée analyzer*, *European Symposium on Programming*, Springer, 2005, pp. 21–30.
- [CCF⁺06] ———, *Combination of abstractions in the astrée static analyzer*, *Annual Asian Computing Science Conference*, Springer, 2006, pp. 272–300.
- [CCM11] Patrick Cousot, Radhia Cousot, and Laurent Mauborgne, *The reduced product of abstract domains and the combination of decision procedures*, *International Conference on Foundations of Software Science and Computational Structures*, Springer, 2011, pp. 456–472.

-
- [CDR98] Hélène Collavizza, François Delobel, and Michel Rueher, *A note on partial consistencies over continuous domains*, International Conference on Principles and Practice of Constraint Programming, Springer, 1998, pp. 147–161.
- [CDR99] Hélène Collavizza, François Delobel, and Michel Rueher, *Comparing partial consistencies*, *Reliable computing* 5 (1999), no. 3, 213–228.
- [CH78] Patrick Cousot and Nicolas Halbwachs, *Automatic discovery of linear restraints among variables of a program*, Proceedings of the 5th ACM SIGACT-SIGPLAN symposium on Principles of programming languages, ACM, 1978, pp. 84–96.
- [Che68] NV Chernikoba, *Algorithm for discovering the set of all the solutions of a linear programming problem*, *USSR Computational Mathematics and Mathematical Physics* 8 (1968), no. 6, 282–293.
- [Che15] Xin Chen, *Reachability analysis of non-linear hybrid systems using taylor models*, Ph.D. thesis, Fachgruppe Informatik, RWTH Aachen University, 2015.
- [CJ09] Gilles Chabert and Luc Jaulin, *Contractor programming*, *Artificial Intelligence* 173 (2009), 1079–1100.
- [CK98] Alongkri Chutinan and Bruce H Krogh, *Computing polyhedral approximations to flow pipes for dynamic systems*, Proceedings of the 37th IEEE Conference on Decision and Control (Cat. No. 98CH36171), vol. 2, IEEE, 1998, pp. 2089–2094.
- [Com03] C Combastel, *A state bounding observer based on zonotopes*, 2003 European Control Conference (ECC), IEEE, 2003, pp. 2589–2594.
- [Con13] John B Conway, *A course in functional analysis*, vol. 96, Springer Science & Business Media, 2013.
- [Cou81] Patrick Cousot, *Semantic foundations of program analysis*, Program flow analysis: theory and applications, Prentice Hall, 1981.
- [CR06] Frédéric Chavanon and Eric Remila, *Rhombus tilings: decomposition and space structure*, *Discrete & Computational Geometry* 35 (2006), no. 2, 329–358.
- [CSS03] Michael A Colón, Sriram Sankaranarayanan, and Henny B Sipma, *Linear invariant generation using non-linear constraint solving*, Proceedings of CAV, Springer, 2003, pp. 420–432.
- [CZL08] Christophe Combastel, Qinghua Zhang, and Abdelhalim Lalami, *Fault diagnosis based on the enclosure of parameters estimated with an adaptive observer*, *IFAC Proceedings Volumes* 41 (2008), no. 2, 7314–7319.
- [dB81] Nicolaas Govert de Bruijn, *Algebraic theory of penrose’s non-periodic tilings of the plane*, *Kon. Nederl. Akad. Wetensch. Proc. Ser. A* 43 (1981), no. 84, 1–7.

BIBLIOGRAPHY

- [DB86] NG De Bruijn, *Dualization of multigrids*, Le Journal de Physique Colloques **47** (1986), no. C3, C3–9.
- [DD05] Vianney Desoutter and Nicolas Destainville, *Flip dynamics in three-dimensional random tilings*, Journal of Physics A: Mathematical and General **38** (2005), no. 1, 17.
- [DDP16] Tommaso Dreossi, Thao Dang, and Carla Piazza, *Parallelotope bundles for polynomial reachability*, Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control, ACM, 2016, pp. 297–306.
- [DFS04] Luiz Henrique De Figueiredo and Jorge Stolfi, *Affine arithmetic: concepts and applications*, Numerical Algorithms **37** (2004), no. 1-4, 147–158.
- [DGP⁺09] David Delmas, Eric Goubault, Sylvie Putot, Jean Souyris, Karim Tekkal, and Franck Védryne, *Towards an industrial use of fluctuat on safety-critical avionics software*, International Workshop on Formal Methods for Industrial Critical Systems, Springer, 2009, pp. 53–69.
- [DHK12] Vijay D’Silva, Leopold Haller, and Daniel Kroening, *Satisfiability solvers are static analysers*, International Static Analysis Symposium, Springer, 2012, pp. 317–333.
- [DHK14] Vijay D’Silva, Leopold Haller, and Daniel Kroening, *Abstract satisfaction*, ACM SIGPLAN Notices **49** (2014), no. 1, 139–150.
- [DHKT12] Vijay D’Silva, Leopold Haller, Daniel Kroening, and Michael Tautschnig, *Numeric bounds analysis with conflict-driven learning*, International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Springer, 2012, pp. 48–63.
- [DMB97] N Destainville, R Mosseri, and F Bailly, *Configurational entropy of codimension-one tilings and directed membranes*, Journal of statistical physics **87** (1997), no. 3-4, 697–754.
- [DMB08] Leonardo De Moura and Nikolaj Bjørner, *Z3: An efficient smt solver*, International conference on Tools and Algorithms for the Construction and Analysis of Systems, Springer, 2008, pp. 337–340.
- [DO18] Steven De Oliveira, *Finding constancy in linear routines*, Ph.D. thesis, Université Paris-Saclay, 2018.
- [dOBP16] Steven de Oliveira, Saddek Bensalem, and Virgile Prevosto, *Polynomial invariants by linear algebra*, International Symposium on Automated Technology for Verification and Analysis, Springer, 2016, pp. 479–494.
- [dOBP17] ———, *Synthesizing invariants by solving solvable loops*, Proceedings of ATVA, Springer, 2017, pp. 327–343.

-
- [Dre89] Andreas WM Dress, *Oriented matroids and penrose-type tilings*, Lecture at the "Symposium on Combinatorics and Geometry", organized by A. Björner, KTH Stockholm, 1989.
- [DS07] David Delmas and Jean Souyris, *Astrée: from research to industry*, International Static Analysis Symposium, Springer, 2007, pp. 437–451.
- [eeBO97] Frédéric Benhamou and William J Older, *Interval constraints*, Journal of logic Programming **32** (1997), no. 1, 1–24.
- [een11] *Efficient implementation of property directed reachability*, Proceedings of the International Conference on Formal Methods in Computer-Aided Design, FMCAD Inc, 2011, pp. 125–134.
- [Eln97] Serge Elnitsky, *Rhombic tilings of polygons and classes of reduced words in coxeter groups*, journal of combinatorial theory, Series A **77** (1997), no. 2, 193–221.
- [ERNF15] Andreas Eggers, Nacim Ramdani, Nediako S Nediakov, and Martin Fränzle, *Improving the sat modulo ode approach to hybrid systems analysis by combining different enclosure methods*, Software & Systems Modeling **14** (2015), no. 1, 121–148.
- [Fel12] Stefan Felsner, *Geometric graphs and arrangements: some chapters from combinatorial geometry*, Springer Science & Business Media, 2012.
- [Fer04] Jérôme Feret, *Static analysis of digital filters*, European Symposium on Programming, Springer, 2004, pp. 33–48.
- [FFL05] J-A Ferrez, Komei Fukuda, and Th M Lieblich, *Solving the fixed rank convex quadratic maximization in binary variables by a parallel zonotope construction algorithm*, European Journal of Operational Research **166** (2005), no. 1, 35–50.
- [FLGD⁺11] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler, *Spaceex: Scalable verification of hybrid systems*, International Conference on Computer Aided Verification, Springer, 2011, pp. 379–395.
- [Flo67] Robert W Floyd, *Assigning meanings to programs*, Mathematical aspects of computer science **19** (1967), no. 19-32, 1.
- [GGP09] Khalil Ghorbal, Eric Goubault, and Sylvie Putot, *The zonotope abstract domain taylor1+*, International Conference on Computer Aided Verification, Springer, 2009, pp. 627–633.
- [GGP10a] ———, *A logical product approach to zonotope intersection*, International Conference on Computer Aided Verification, Springer, 2010, pp. 212–226.

- [GGP10b] Khalil Ghorbal, Eric Goubault, and Sylvie Putot, *A logical product approach to zonotope intersection*, Proceedings of CAV, 2010, pp. 212–226.
- [GGTZ07] Stephane Gaubert, Eric Goubault, Ankur Taly, and Sarah Zenou, *Static analysis by policy iteration on relational domains*, European symposium on programming, Springer, 2007, pp. 237–252.
- [Gho11] Khalil Ghorbal, *Static analysis of numerical programs: constrained affine sets abstract domain*, Ph.D. thesis, Ecole Polytechnique X, 2011.
- [GK10] Eugene Gover and Nishan Krikorian, *Determinants and the volumes of parallelotopes and zonotopes*, Linear Algebra and its Applications **433** (2010), no. 1, 28–40.
- [GLG08] Antoine Girard and Colas Le Guernic, *Zonotope/hyperplane intersection for hybrid systems reachability analysis*, International Workshop on Hybrid Systems: Computation and Control, Springer, 2008, pp. 215–228.
- [GLMN14] Pranav Garg, Christof Löding, P Madhusudan, and Daniel Neider, *Ice: A robust framework for learning invariants*, Proceedings of CAV, Springer, 2014, pp. 69–87.
- [GMS16] David Gerault, Marine Minier, and Christine Solnon, *Constraint programming models for chosen key differential cryptanalysis*, International Conference on Principles and Practice of Constraint Programming, Springer, 2016, pp. 584–601.
- [GNMR16] Pranav Garg, Daniel Neider, Parthasarathy Madhusudan, and Dan Roth, *Learning invariants using decision trees and implication counterexamples*, ACM Sigplan Notices, vol. 51, ACM, 2016, pp. 499–512.
- [GNZ03] Leonidas J Guibas, An Nguyen, and Li Zhang, *Zonotopes as bounding volumes*, Proceedings of the ACM-SIAM symposium on Discrete algorithms, 2003, pp. 803–812.
- [GP06] Eric Goubault and Sylvie Putot, *Static analysis of numerical algorithms*, International Static Analysis Symposium, Springer, 2006, pp. 18–34.
- [GP09] ———, *A zonotopic framework for functional abstractions*, arXiv preprint arXiv:0910.1763 (2009).
- [GP15] ———, *A zonotopic framework for functional abstractions*, Formal Methods in System Design **47** (2015), no. 3, 302–360.
- [GP17] ———, *Forward inner-approximated reachability of non-linear continuous systems*, Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control, ACM, 2017, pp. 1–10.

-
- [GPBG07] Eric Goubault, Sylvie Putot, Philippe Baufreton, and Jean Gassino, *Static analysis of the accuracy in control systems: Principles and experiments*, International Workshop on Formal Methods for Industrial Critical Systems, Springer, 2007, pp. 3–20.
- [GPV12] Eric Goubault, Sylvie Putot, and Franck Védryne, *Modular static analysis with zonotopes*, International Static Analysis Symposium, Springer, 2012, pp. 24–40.
- [GS07a] Thomas Gawlitza and Helmut Seidl, *Precise fixpoint computation through strategy iteration*, European symposium on programming, Springer, 2007, pp. 300–315.
- [GS07b] ———, *Precise relational invariants through strategy iteration*, International Workshop on Computer Science Logic, Springer, 2007, pp. 23–40.
- [GSA⁺12] Thomas Martin Gawlitza, Helmut Seidl, Assalé Adjé, Stéphane Gaubert, and Éric Goubault, *Abstract interpretation meets convex optimization*, Journal of Symbolic Computation **47** (2012), no. 12, 1416–1446.
- [GSV08] Sumit Gulwani, Saurabh Srivastava, and Ramarathnam Venkatesan, *Program analysis as constraint solving*, ACM SIGPLAN Notices **43** (2008), no. 6, 281–292.
- [GT06] Sumit Gulwani and Ashish Tiwari, *Combining abstract interpreters*, ACM SIGPLAN Notices, vol. 41, ACM, 2006, pp. 376–386.
- [GT08] ———, *Constraint-based approach for analysis of hybrid systems*, International Conference on Computer Aided Verification, Springer, 2008, pp. 190–203.
- [HB12] Kryštof Hoder and Nikolaj Bjørner, *Generalized property directed reachability*, International Conference on Theory and Applications of Satisfiability Testing, Springer, 2012, pp. 157–171.
- [Hén76] Michel Hénon, *A two-dimensional mapping with a strange attractor*, The Theory of Chaotic Attractors, Springer, 1976, pp. 94–102.
- [HK02] Boris Hasselblatt and Anatole Katok, *Handbook of dynamical systems*, Elsevier, 2002.
- [HK13] Didier Henrion and Milan Korda, *Convex computation of the region of attraction of polynomial control systems*, IEEE Transactions on Automatic Control **59** (2013), no. 2, 297–312.
- [HLZ96] Martin Henz, Stefan Lauer, and Detlev Zimmermann, *Compoze-intention-based music composition through constraint programming*, Tools with Artificial Intelligence, 1996., Proceedings Eighth IEEE International Conference on, IEEE, 1996, pp. 118–121.
- [Hoa69] Charles Antony Richard Hoare, *An axiomatic basis for computer programming*, Communications of the ACM **12** (1969), no. 10, 576–580.

- [HR11] Olga Holtz and Amos Ron, *Zonotopal algebra*, *Advances in Mathematics* **227** (2011), no. 2, 847–894.
- [HST17] Zoltán Horváth, Yunfei Song, and Tamás Terlaky, *A novel unified approach to invariance conditions for a linear dynamical system*, *Applied Mathematics and Computation* **298** (2017), 351–367.
- [HST18] ———, *Invariance preserving discretization methods of dynamical systems*, *Vietnam Journal of Mathematics* **46** (2018), no. 4, 803–823.
- [Ilo99] S Ilog, *Revising hull and box consistency*, *Logic Programming: Proceedings of the 1999 International Conference on Logic Programming*, MIT press, 1999, p. 230.
- [Jau12] Luc Jaulin, *Solving set-valued constraint satisfaction problems*, *Computing* **94** (2012), no. 2-4, 297–311.
- [JM09] Bertrand Jeannet and Antoine Miné, *Apron: A library of numerical abstract domains for static analysis*, *Proceedings of CAV*, Springer, 2009, pp. 661–667.
- [Jol11] Mioara Maria Joldes, *Rigorous polynomial approximations and applications*, Ph.D. thesis, 2011.
- [KdBdGZ52] Stephen Cole Kleene, NG de Bruijn, J de Groot, and Adriaan Cornelis Zaanen, *Introduction to metamathematics*, vol. 483, van Nostrand New York, 1952.
- [Ken93] Richard Kenyon, *Tiling a polygon with parallelograms*, *Algorithmica* **9** (1993), no. 4, 382–397.
- [KGMP20] Bibek Kabi, Eric Goubault, Antoine Mine, and Sylvie Putot, *Combining zonotope abstraction and constraint programming for synthesizing inductive invariants*, *International Workshop on Numerical Software Verification*, Springer, 2020.
- [KGP16] Bibek Kabi, Eric Goubault, and Sylvie Putot, *A concoction of zonotope abstraction and constraint programming for finding an invariant*, https://swim2016.sciencesconf.org/data/pages/Kabi_Goubault_Putot.pdf, 2016.
- [KGP17] ———, *Combining zonotope abstraction and constraint programming for finding an invariant*, https://asimod.in.tum.de/2017/posters/Kabi_Bibek.pdf, 2017.
- [KKP⁺15] Florent Kirchner, Nikolai Kosmatov, Virgile Prevosto, Julien Signoles, and Boris Yakobowski, *Frama-c: A software analysis perspective*, *Formal Aspects of Computing* **27** (2015), no. 3, 573–609.
- [KMW16] Egor George Karpenkov, David Monniaux, and Philipp Wendler, *Program analysis with local policy iteration*, *International Conference on Verification, Model Checking, and Abstract Interpretation*, Springer, 2016, pp. 127–146.

- [Lat00] Matthieu Latapy, *Generalized integer partitions, tilings of zonotopes and lattices*, Formal Power Series and Algebraic Combinatorics, Springer, 2000, pp. 256–267.
- [LCCG96] Sandrine Lyonnard, Gerrit Coddens, Yvonne Calvayrac, and Denis Gratias, *Atomic (phason) hopping in perfect icosahedral quasicrystals at 70.3 ps 21.4 ms 8.3 by time-of-flight quasielastic neutron scattering*, Physical Review B **53** (1996), no. 6, 3150.
- [LMJZ17] Thomas Le Mézo, Luc Jaulin, and Benoit Zerr, *An interval approach to compute invariant sets*, IEEE Transactions on Automatic Control **62** (2017), no. 8, 4236–4242.
- [LMR⁺98] Ph Lacan, Jean Noel Monfort, LVQ Ribal, A Deutsch, and G Gonthier, *Ariane 5-the software reliability verification process*, DASIA 98-Data Systems in Aerospace, vol. 422, 1998, p. 201.
- [LP94] Claude Le Pape, *Using a constraint-based scheduling library to solve a specific scheduling problem*, 1994.
- [LSA⁺13] Vu Tuan Hieu Le, Cristina Stoica, Teodoro Alamo, Eduardo F Camacho, and Didier Dumur, *Zonotope-based set-membership estimation for multi-output uncertain systems*, 2013 IEEE International Symposium on Intelligent Control (ISIC), IEEE, 2013, pp. 212–217.
- [LV92] Hervé Le Verge, *A note on chernikova’s algorithm*, Ph.D. thesis, INRIA, 1992.
- [LZZ11] Jiang Liu, Naijun Zhan, and Hengjun Zhao, *Computing semi-algebraic invariants for polynomial dynamical systems*, Proceedings of the ninth ACM international conference on Embedded software, ACM, 2011, pp. 97–106.
- [M⁺17] Antoine Miné et al., *Tutorial on static inference of numeric invariants by abstract interpretation*, Foundations and Trends® in Programming Languages **4** (2017), no. 3-4, 120–372.
- [Mac77a] Alan K Mackworth, *Consistency in networks of relations*, Artificial intelligence **8** (1977), no. 1, 99–118.
- [Mac77b] ———, *On reading sketch maps*, Department of Computer Science, University of British Columbia, 1977.
- [Mar14] Benjamin Martin, *Rigorous algorithms for nonlinear biobjective optimization*, Ph.D. thesis, Université de Nantes, 2014.
- [Mau04] Laurent Mauborgne, *Astrée: Verification of absence of runtime error*, Building the Information Society, Springer, 2004, pp. 385–392.
- [MB03] Kyoko Makino and Martin Berz, *Taylor models and other validated functional inclusion methods*, International Journal of Pure and Applied Mathematics **6** (2003), 239–316.

- [MB09] ———, *Rigorous integration of flows and odes using taylor models*, Proceedings of the 2009 conference on Symbolic Numeric Computation, ACM, 2009, pp. 79–84.
- [MBR16] Antoine Miné, Jason Breck, and Thomas Reps, *An algorithm inspired by constraint solvers to infer inductive invariants in numeric programs*, Proceedings of ESOP, 2016, pp. 560–588.
- [McM71] Peter McMullen, *On zonotopes*, Transactions of the American Mathematical Society **159** (1971), 91–109.
- [McM76] PETER McMullen, *Polytopes with centrally symmetric facets*, Israel Journal of Mathematics **23** (1976), no. 3-4, 337–338.
- [Min01] Antoine Miné, *A new numerical abstract domain based on difference-bound matrices*, Programs as Data Objects, Springer, 2001, pp. 155–172.
- [Min04] ———, *Weakly relational numerical abstract domains.(domaines numériques abstraits faiblement relationnels).*, Ph.D. thesis, École Polytechnique, Palaiseau, France, 2004.
- [Min06] ———, *The octagon abstract domain*, Higher-order and symbolic computation **19** (2006), no. 1, 31–100.
- [Mis99] Konstantin Mischaikow, *The conley index theory: a brief introduction*, Banach Center Publications **47** (1999), no. 1, 9–19.
- [Mon74] Ugo Montanari, *Networks of constraints: Fundamental properties and applications to picture processing*, Information sciences **7** (1974), 95–132.
- [Moo69] Ramon E Moore, *Interval analysis. 1966*, Prince-Hall, Englewood Cliffs, NJ (1969).
- [NJC99] Nedialko S Nedialkov, Kenneth R Jackson, and George F Corliss, *Validated solutions of initial value problems for ordinary differential equations*, Applied Mathematics and Computation **105** (1999), no. 1, 21–68.
- [NO79] Greg Nelson and Derek C Oppen, *Simplification by cooperating decision procedures*, ACM Transactions on Programming Languages and Systems (TOPLAS) **1** (1979), no. 2, 245–257.
- [NP19] Shannon Negaard-Paper, *Attractors and attracting neighborhoods for multiflows*, arXiv preprint arXiv:1905.06473 (2019).
- [NQ10] Duong Nguyen Que, *Robust and generic abstract domain for static program analyses: the polyhedral case*, Ph.D. thesis, Paris, ENMP, 2010.
- [PC09] André Platzer and Edmund M Clarke, *Computing differential invariants of hybrid systems as fixedpoints*, Formal Methods in System Design **35** (2009), no. 1, 98–120.

-
- [Pel15] Marie Pelleau, *Abstract domains in constraint programming*, Elsevier, 2015.
- [PMR12] Olivier Ponsini, Claude Michel, and Michel Rueher, *Combining constraint programming and abstract interpretation for value analysis of floating-point programs*, 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation, IEEE, 2012, pp. 775–776.
- [PMTB13] Marie Pelleau, Antoine Miné, Charlotte Truchet, and Frédéric Benhamou, *A constraint solver based on abstract domains*, International Workshop on Verification, Model Checking, and Abstract Interpretation, Springer, 2013, pp. 434–454.
- [PTB11] Marie Pelleau, Charlotte Truchet, and Frédéric Benhamou, *Octagonal domains for continuous constraints*, International Conference on Principles and Practice of Constraint Programming, Springer, 2011, pp. 706–720.
- [PTB14] ———, *The octagon abstract domain for continuous constraints*, Constraints **19** (2014), no. 3, 309–337.
- [Rei99] Victor Reiner, *The generalized baues problem*, New perspectives in algebraic combinatorics **38** (1999), 293–336.
- [RG13] Pierre Roux and Pierre-Loïc Garoche, *Integrating policy iterations in abstract interpreters*, Automated Technology for Verification and Analysis, Springer, 2013, pp. 240–254.
- [RG14] ———, *Computing quadratic invariants with min-and max-policy iterations: A practical comparison*, International Symposium on Formal Methods, Springer, 2014, pp. 563–578.
- [RG15] ———, *Practical policy iterations*, Formal Methods in System Design **46** (2015), no. 2, 163–196.
- [RGZ94] Jürgen Richter-Gebert and Günter M Ziegler, *Zonotopal tilings and the bohne-dress theorem*, Contemporary Mathematics **178** (1994), 211–211.
- [Ric53] Henry Gordon Rice, *Classes of recursively enumerable sets and their decision problems*, Transactions of the American Mathematical Society **74** (1953), no. 2, 358–366.
- [RJGF12] Pierre Roux, Romain Jobredeaux, Pierre-Loïc Garoche, and Éric Féron, *A generic ellipsoid abstract domain for linear time invariant systems*, Proceedings of the 15th ACM international conference on Hybrid Systems: Computation and Control, ACM, 2012, pp. 105–114.
- [RSÁ14] Xin Chen Rwth, Sriram Sankaranarayanan, and Erika Ábrahám, *Under-approximate flowpipes for non-linear continuous systems*, 2014 Formal Methods in Computer-Aided Design (FMCAD), IEEE, 2014, pp. 59–66.

- [RSY04] Thomas Reps, Mooly Sagiv, and Greta Yorsh, *Symbolic implementation of the best transformer*, International Workshop on Verification, Model Checking, and Abstract Interpretation, Springer, 2004, pp. 252–266.
- [RT16] Thomas Reps and Aditya Thakur, *Automating abstract interpretation*, International Conference on Verification, Model Checking, and Abstract Interpretation, Springer, 2016, pp. 3–40.
- [Rub18] Gianluca Amato Marco Rubino, *Experimental evaluation of numerical domains for inferring ranges*, Electronic Notes in Theoretical Computer Science **334** (2018), 3–16.
- [Rue05] Michel Rueher, *Solving continuous constraint systems*, International Conference on Computer Graphics and Artificial Intelligence, vol. 1, 2005, pp. 2–2.
- [Rue06] David Ruelle, *What is a strange attractor*, Notices of the AMS **53** (2006), no. 7, 764–765.
- [S⁺04] Richard P Stanley et al., *An introduction to hyperplane arrangements*, Geometric combinatorics **13** (2004), 389–496.
- [SBGC84] Dan Shechtman, Ilan Blech, Denis Gratias, and John W Cahn, *Metallic phase with long-range orientational order and no translational symmetry*, Physical review letters **53** (1984), no. 20, 1951.
- [SC16] Julien Alexandre Dit Sandretto and Alexandre Chapoutot, *Validated simulation of differential algebraic equations with runge-kutta methods*.
- [Sch98] Alexander Schrijver, *Theory of linear and integer programming*, John Wiley & Sons, 1998.
- [SD07] Jean Souyris and David Delmas, *Experimental assessment of astrée on safety-critical avionics software*, International Conference on Computer Safety, Reliability, and Security, Springer, 2007, pp. 479–490.
- [SDF97] Jorge Stolfi and Luiz Henrique De Figueiredo, *Self-validated numerical methods and applications*, Monograph for 21st Brazilian Mathematics Colloquium, IMPA, 1997.
- [SDR⁺18] Xujie Si, Hanjun Dai, Mukund Raghothaman, Mayur Naik, and Le Song, *Learning loop invariants for program verification*, Advances in Neural Information Processing Systems, 2018, pp. 7762–7773.
- [Sha98] Paul Shaw, *Using constraint programming and local search methods to solve vehicle routing problems*, International conference on principles and practice of constraint programming, Springer, 1998, pp. 417–431.
- [She74] Geoffrey C Shephard, *Combinatorial properties of associated zonotopes*, Canadian Journal of Mathematics **26** (1974), no. 2, 302–321.

-
- [SHF96] Djamila Sam-Haroud and Boi Faltings, *Consistency techniques for continuous constraints*, *Constraints* **1** (1996), no. 1-2, 85–118.
- [SJJ17] Andrew Sogokon, Paul B Jackson, and Taylor T Johnson, *Verifying safety and persistence properties of hybrid systems using flowpipes and continuous invariants*, *NASA Formal Methods Symposium*, Springer, 2017, pp. 194–211.
- [SNÁ17] Stefan Schupp, Johanna Nellen, and Erika Ábrahám, *Divide and conquer: Variable set separation in hybrid systems reachability analysis*, arXiv preprint arXiv:1707.04851 (2017).
- [Sor85] John Thomas Sorensen, *A physiologic model of glucose metabolism in man and its use to design and assess improved insulin therapies for diabetes*, Ph.D. thesis, Massachusetts Institute of Technology, 1985.
- [SPV17] Gagandeep Singh, Markus Püschel, and Martin Vechev, *Fast polyhedra abstract domain*, *ACM SIGPLAN Notices*, vol. 52, ACM, 2017, pp. 46–59.
- [SSM04] Sriram Sankaranarayanan, Henny B Sipma, and Zohar Manna, *Constructing invariants for hybrid systems*, *International Workshop on Hybrid Systems: Computation and Control*, Springer, 2004, pp. 539–554.
- [SSM05] ———, *Scalable analysis of linear systems using mathematical programming*, *International Workshop on Verification, Model Checking, and Abstract Interpretation*, Springer, 2005, pp. 25–41.
- [Sta98] Richard P Stanley, *Hyperplane arrangements, parking functions and tree inversions*, *Mathematical essays in honor of Gian-Carlo Rota*, Springer, 1998, pp. 359–375.
- [Ste10] Shlomo Sternberg, *Dynamical systems*, Courier Corporation, 2010.
- [SWYZ09] Liyong Shen, Min Wu, Zhengfeng Yang, and Zhenbing Zeng, *Finding positively invariant sets of a class of nonlinear loops via curve fitting*, *Proceedings of the 2009 conference on Symbolic numeric computation*, ACM, 2009, pp. 185–190.
- [TAC01] Charlotte Truchet, Gérard Assayag, and Philippe Codognet, *Visual and adaptive constraint programming in music.*, *ICMC*, 2001.
- [Tar55] Alfred Tarski, *A lattice-theoretical fixpoint theorem and its applications*, *Pacific journal of Mathematics* **5** (1955), no. 2, 285–309.
- [TER12] Aditya Thakur, Matt Elder, and Thomas Reps, *Bilateral algorithms for symbolic abstraction*, *International Static Analysis Symposium*, Springer, 2012, pp. 111–128.
- [TLLR15] Aditya Thakur, Akash Lal, Junghee Lim, and Thomas Reps, *Posthat and all that: Automating abstract interpretation*, *Electronic Notes in Theoretical Computer Science* **311** (2015), 15–32.

BIBLIOGRAPHY

- [TS13] S Mojtaba Tabatabaeipour and Jakob Stoustrup, *Set-membership state estimation for discrete time piecewise affine systems using zonotopes*, 2013 European Control Conference (ECC), IEEE, 2013, pp. 3143–3148.
- [Tur37] Alan M Turing, *On computable numbers, with an application to the entscheidungsproblem*, Proceedings of the London mathematical society 2 (1937), no. 1, 230–265.
- [Urb15] Caterina Urban, *Static analysis by abstract interpretation of functional temporal properties of programs*, Ph.D. thesis, Paris, Ecole normale supérieure, 2015.
- [VGSM17] Yakir Vizel, Arie Gurfinkel, Sharon Shoham, and Sharad Malik, *Ic3-flipping the e in ice*, International Conference on Verification, Model Checking, and Abstract Interpretation, Springer, 2017, pp. 521–538.
- [Wen14] Haoran Wen, *A review of the hénon map and its physical interpretations*, Sch. Phys. Georg. Inst. Technol. Atlanta, GA 30332–0430 (2014), 1–9.
- [Wey34] Hermann Weyl, *Elementare theorie der konvexen polyeder*, Commentarii Mathematici Helvetici 7 (1934), no. 1, 290–306.
- [WVL09] Jian Wan, Josep Vehi, and Ningsu Luo, *A numerical approach to design control invariant sets for constrained nonlinear discrete-time systems with guaranteed optimality*, Journal of Global Optimization 44 (2009), no. 3, 395–407.
- [Zie12] Günter M Ziegler, *Lectures on polytopes*, vol. 152, Springer Science & Business Media, 2012.
- [ZRG17] Günter M Ziegler and Jürgen Richter-Gebert, *6: Oriented matroids*, Handbook of Discrete and Computational Geometry, Third Edition, Chapman and Hall/CRC, 2017, pp. 159–184.

APPENDICES

Programs

Octagon

```
init {
  x=[-1,1];
  y=[-1,1];
}

body {
  t=0.7*(x+y);
  y=0.7*(x-y);
  x=t;
}

goal {
  x=[-2,2];
  y=[-2,2];
}
```

Filter

```
init
{
  x=[-0.1,0.1];
  y=[-0.1,0.1];
}
body
{
  r = 1.5*x - 0.7*y + [-0.1,0.1];
  y = x;
  x = r;
}
goal
{
  x=[-4,4];
  y=[-4,4];
}
```

Filter2

```
init {
  x=[0,1];
  y=[0,1];
}

body {
  x = (3.0/4.0) * x - (1.0/8.0)* y;
  y = x;
}

goal {
  // This goal was not taken from the paper itself,
  // but was chosen as a reasonable property to prove.
  x=[-0.2,1];
  y=[-0.2,1];
}

/*
Example file: from LMCS '12: ACCURATE NUMERICAL INVARIANTS
              by Adje, Gaubert, and Goubault
              Figure 11: The program Filter
*/

/*
// Code from the paper:
x = [0,1];
y = [0,1];
while ( true ) {
  x = (3/4)* x-(1/8)*y;
  y = x;
}
*/
```

Arrow-Hurwicz

```
/* Example file: from ESOP'10: Coupling Policy Iteration with
   Semi-definite Relaxation to Compute
   Accurate Numerical Invariants in Static
   Analysis by Adje, Gaubert, and Goubault
   *****
NOTE: This is a *modified* version of the Arrow-Hurwicz example
   because it does not represent u and v as separate
   variables, and it has no loop condition.
   *****
*/
/*
// Original code from
// http://www.lix.polytechnique.fr/~adje/uploads/Codes.pdf
a = 1;b = -1;c = -1;r = 1/2;
u = [0, 1];v = [0, 1];x = [0, 3/2];y = [3/8, 11/8];
while (max(|x-u|, |y-v|) > 1e-9) {
    u = x;
    v = y;
    x = u - r * (a * u + b * v);
    y = v + (r / 2) * (b * u - c);
    if (y <= 0) {
        y = 0;
    }
}
*/
init {
    x = [0,1.5];
    y = [0.375,1.375];
}
body {
// Note: We have removed the loop condition,
// so u and v are temporary variables,
// not carried from one iteration to the next.
// We have also substituted in constants a,b,c,r
u = x;
v = y;
x = u - 0.5 * (1 * u + (-1) * v);
y = v + (0.5 / 2) * ((-1) * u - (-1));
if (y <= 0) {
    y = 0;
}
}
goal {
// Note: these bounds are not from the paper,
// they are simply a reasonable guess.
x = [-1.73,1.73];
y = [-1.73,1.73];
}
```

Harm

```
init {
  x0=[0,1];
  x1=[0,1];
}

body {
  x0p = x0; x1p = x1;
  x0 = 0.95*x0p + 0.09975*x1p;
  x1 = -0.1 *x0p + 0.95 *x1p;
}

goal {
  // Bounds given in the Roux paper (Table 3)
  x0=[-1.27,1.27];
  x1=[-1.27,1.27];
  // Goal from ESOP '10 paper (policy iteration goal)
  //x=[-1.42,1.42];
  //v=[-1.42,1.42];
}

/*
Example file:
  Practical policy iterations
  Roux and Garoche
  Form Methods Syst Des 2015
  DOI 10.1007/s10703-015-0230-7
  Also known as Ex.8 (Harmonic oscillator)
  ultimately from LMCS '12: ACCURATE NUMERICAL INVARIANTS
  by Adje, Gaubert, and Goubault
  Figure 12: An implementation of the Symplectic method
*/
/*
// Code from benchmarks tarball
node top(ix0, ix1 : real) returns (x0, x1 : real);
let
  assert(ix0 > 0. and ix0 < 1.);
  assert(ix1 > 0. and ix1 < 1.);
  x0 = ix0 -> 0.95 * pre x0 + 0.09975 * pre x1;
  x1 = ix1 -> -0.1 * pre x0 + 0.95 * pre x1;
tel
*/
/*
// Original code from the Goubault paper
tau = 0.1;
x = [0,1];
v = [0,1];
while (true) {
  x = (1-( tau / 2 ) ) * x + (tau -(( tau^3 ) / 4 ) ) * v ;
  v = -tau *x+(1-( tau / 2 ) ) * v ;
}
*/
```

Harm-reset

```

init {
  x0=[0,1];
  x1=[0,1];
}
body {
  x0p = x0; x1p = x1;
  x0 = 0.95*x0p + 0.09975*x1p;
  x1 = -0.1 *x0p + 0.95 *x1p;
  if ([0,1] > 0.5) {
    x0 = 1;
    x1 = 1;
  }
}
goal {
  // Bounds given in the Roux paper (Table 3) (for reset Ex8)
  //x0=[-1.00,1.00];
  //x1=[-1.01,1.01];
  // Bounds given in the Roux paper (Table 3) (for original Ex8)
  x0=[-1.27,1.27];
  x1=[-1.27,1.27];
  // Goal from ESOP'10 paper (policy iteration goal)
  //x=[-1.42,1.42];
  //v=[-1.42,1.42];
}
/* Example file:
   Practical policy iterations
   Roux and Garoche
   Form Methods Syst Des 2015
   DOI 10.1007/s10703-015-0230-7
   Also known as Ex.8 (Harmonic oscillator reset)
   ultimately from LMCS '12: ACCURATE NUMERICAL INVARIANTS
   by Adje, Gaubert, and Goubault
   Figure 12: An implementation of the Symplectic method */
/*
  // Code from benchmarks tarball
  node top(r : bool; ix0, ix1 : real) returns (x0, x1 : real);
  let
    assert(ix0 > 0. and ix0 < 1.);
    assert(ix1 > 0. and ix1 < 1.);
    x0 = ix0 -> if r then 1. else 0.95*pre x0 + 0.09975*pre x1;
    x1 = ix1 -> if r then 1. else -0.1*pre x0 + 0.95*pre x1;
  tel
*/
/* Original code from the Goubault paper
  tau = 0.1;
  x = [0,1];
  v = [0,1];
  while (true) {
    x = (1-( tau / 2 ) ) * x + (tau -(( tau^ 3 ) / 4 ) ) * v ;
    v = -tau *x+(1-( tau / 2 ) ) * v ;
  }
*/

```

Harm-saturated

```

init {
  x0=[0,1];
  x1=[0,1];
}
body {
  x0p = x0; x1p = x1;
  x0 = 0.95*x0p + 0.09975*x1p;
  x1 = -0.1 *x0p + 0.95*x1p;

  if (x0 > 0.5) { x0 = 0.5; }
  if (x0 < -0.5) { x0 = -0.5; }
}
goal {
  // Bounds given in the Roux paper (Table 3)
  x0=[-1.27,1.27];
  x1=[-1.27,1.27];
  // Goal from ESOP '10 paper (policy iteration goal)
  //x=[-1.42,1.42];
  //v=[-1.42,1.42];
}
/* Example file:
   Practical policy iterations
   Roux and Garoche
   Form Methods Syst Des 2015
   DOI 10.1007/s10703-015-0230-7
   Also known as Ex.8 (Harmonic oscillator saturate)
   ultimately from LMCS '12: ACCURATE NUMERICAL INVARIANTS
   by Adje, Gaubert, and Goubault
   Figure 12: An implementation of the Symplectic method
*/
/* Code from benchmarks tarball
node top(ix0, ix1 : real) returns (sx0, x0, x1 : real);
let
  assert(ix0 > 0. and ix0 < 1.);
  assert(ix1 > 0. and ix1 < 1.);
  x0=ix0 -> 0.95 * pre sx0 + 0.09975 * pre x1;
  x1=ix1 -> -0.1 * pre sx0 + 0.95 * pre x1;
  sx0=if x0 > 0.5 then 0.5 else if x0<-0.5 then -0.5 else x0;
tel
*/
/*
// Original code from the Goubault paper
tau = 0.1;
x = [0,1];
v = [0,1];
while (true) {
  x = (1-( tau / 2 ) ) * x + (tau -(( tau^3 ) / 4 ) ) * v ;
  v = -tau *x+(1-( tau / 2 ) ) * v ;
}
*/

```

Lead-lag

```
init {
  // True initial conditions from paper:
  x0 = [0,0];
  x1 = [0,0];
}

body {
  in0 = [-1,1];
  x0p = x0; x1p = x1;

  x0 = 0.499*x0p - 0.05*x1p + in0;
  x1 = 0.010*x0p +          x1p;
}

goal {
  // Bounds given in the Roux paper (Table 3)
  x0 = [-4.03,4.03];
  x1 = [-20.41,20.41];
}

/* Example file:
   Practical policy iterations
   Roux and Garoche
   Form Methods Syst Des 2015
   DOI 10.1007/s10703-015-0230-7
   Also known as Ex.3 (Discretized lead-lag controller)*/

/*
// Code from benchmarks tarball
node top(in0 : real) returns (x, y : real);
let
  assert(in0 >= -1. and in0 <= 1.);
  x = 0. -> 1.5 * pre x - 0.7 * pre y + 1.6 * in0;
  y = 0. -> pre x;
tel
*/
```

Lead-lag-reset

```
init {
  // True initial conditions from paper:
  x0 = [0,0];
  x1 = [0,0];
}

body {
  in0 = [-1,1];
  x0p = x0; x1p = x1;

  x0 = 0.499*x0p - 0.05*x1p + in0;
  x1 = 0.010*x0p +      x1p;

  if ([0,1] > 0.5) {
    x0 = 1;
    x1 = 1;
  }
}

goal {
  // Bounds given in the Roux paper (Table 3)
  x0 = [-4.14,4.14];
  x1 = [-21.41,21.41];
}

/* Example file:
   Practical policy iterations
   Roux and Garoche
   Form Methods Syst Des 2015
   DOI 10.1007/s10703-015-0230-7
   Also known as Ex.3 (Discretized lead-lag controller)
*/

/*
// Code from benchmarks tarball
node top(r : bool; in0 : real) returns (x0, x1 : real);
let
  assert(in0 >= -1. and in0 <= 1.);
  x0=0. -> if r then 1. else 0.499*pre x0 - 0.05*pre x1 + in0;
  x1=0. -> if r then 1. else 0.01*pre x0 + pre x1;
tel
*/
```


Lead-lag-saturated

```
init {
  // True initial conditions from paper:
  x0 = [0,0];
  x1 = [0,0];
}

body {
  in0 = [-1,1];
  x0p = x0; x1p = x1;

  x0 = 0.499*x0p - 0.05*x1p + in0;
  x1 = 0.010*x0p +          x1p;

  if (x0 > 50) { x0 = 50; }
  if (x0 < -50) { x0 = -50; }
}

goal {
  // Bounds given in the Roux paper (Table 3)
  x0 = [-4.03,4.03];
  x1 = [-20.41,20.41];
}

/* Example file:
   Practical policy iterations
   Roux and Garoche
   Form Methods Syst Des 2015
   DOI 10.1007/s10703-015-0230-7
   Also known as Ex.3 (Discretized lead-lag controller)*/

/* Code from benchmarks tarball:
node top(in0 : real) returns (sx0, x0, x1 : real);
let
  assert(in0 >= -1. and in0 <= 1.);
  x0=0. -> 0.499 * pre sx0 - 0.05 * pre x1 + in0;
  x1=0. -> 0.01 * pre sx0 + pre x1;
  sx0=if x0 > 50. then 50. else if x0 < -50. then -50. else x0;
tel
*/
```

Sine

```
init
{
  x = [-1.57079632679,1.57079632679];
  y=[0,0];
}
body
{
  y=x - x^3/6 + x^5/120 - x^7/5040;
}
goal
{
  x=[-2,2];
  y=[-1.05,1.05];
}
/*
Example file from Leopold Haller's benchmark
Original code from
http://www.cprover.org/cdfpl/
Simple Taylor expansion of sine

=>
can prove a bound of 1.05 for the output
We wanted to illustrate that it works for tighter bounds

the vertical bar for y=0 is expected
(the inductive invariant must
include both the init state,
where y=0, and the end state, where
y is the approximate sine)
*/
```

Newton

```
init {
  x=[-1,1];
  out=[0,0];
}

body {
  y = x - x*x*x/6 + x*x*x*x*x/120 + x*x*x*x*x*x*x*x/5040;
  z = 1 - x*x/2 + x*x*x*x/24 + x*x*x*x*x*x*x/720;
  x = x - y / z;
  out = x;
}

goal {
  x=[-1,1];
  out=[-0.56,0.56];
}
/*
  Example file from Leopold Haller's benchmark
  Original code from
  http://www.cprover.org/cdfpl/
  Newton iterations: one step
  =>
  can prove an output bound of 0.56
*/
```

Newton2

```
init {
  x=[-1,1];
  out=[0,0];
}

body {
  y = x - x*x*x/6 + x*x*x*x*x/120 + x*x*x*x*x*x*x*x/5040;
  z = 1 - x*x/2 + x*x*x*x/24 + x*x*x*x*x*x*x/720;
  x = x - y / z;

  y = x - x*x*x/6 + x*x*x*x*x/120 + x*x*x*x*x*x*x*x/5040;
  z = 1 - x*x/2 + x*x*x*x/24 + x*x*x*x*x*x*x/720;
  x = x - y / z;

  out = x;
}

goal {
  x=[-1,1];
  out=[-0.1,0.1];
}
/*
Example file from Leopold Haller's benchmark
Original code from
http://www.cprover.org/cdfpl/
Newton iterations: two steps
=>
works for bound of 0.1
*/
```

Square root

```
init {
  x=[0,1];
  out=[0,0];
}

body {
out = 1 + 0.5*x - 0.125*x*x + 0.0625*x*x*x - 0.0390625*x*x*x*x;
}

goal {
  x=[0,1];
  out=[0,1.5];
}
/*
  Example file from Leopold Haller's benchmark
  Original code from
  http://www.cprover.org/cdfpl/
  Simple polynomial interpolation function for square root.
  =>
  we can prove that the output is <= 1.5
  we can prove tighter bound (such as 1.4 or 1.39844)
  using zonotopes but not using boxes and octagons
*/
```

Corner

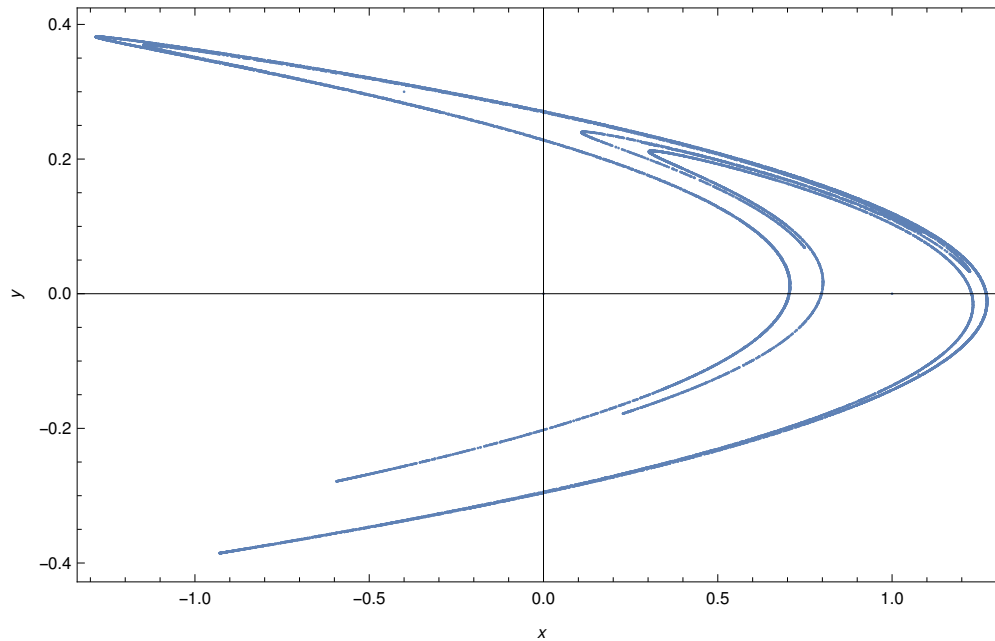
```
init {
  x=[0.9,1.1];
  y=[0.9,1.1];
}

body {
  d = (0.2 + x*x + y*y + 1.53*x*x*y*y)/2;
  x = x / d;
  y = y / d;
}

goal {
  x=[-2.1,2.1];
  y=[-2.1,2.1];
}
```

Henon

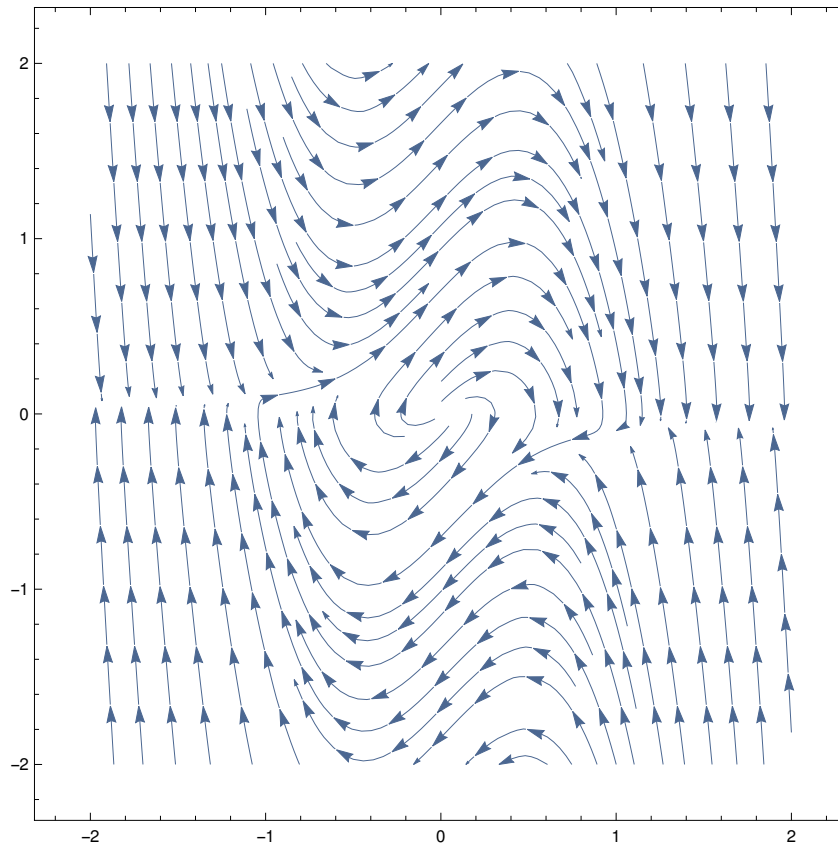
```
henon[alpha_, beta_] [{x_, y_}] := {y + 1 - alpha x^2, beta x}
list = NestList[henon[1.4, 0.3], {0, 0}, 20000];
ListPlot[list, Frame -> True, FrameLabel -> {x, y}]
```



```
(*Manipulate[ListPlot[NestList[henon[alpha, 0.3], {1, 1}, 20000]],
{alpha, 1.3, 1.4}, ContinuousAction -> False]*)
```

Van der Pol example from [HK13]

```
f[{x_, y_}] := -{-2 y, 0.8 x + 10 (x^2 - 0.21) y}
sp = StreamPlot[f[{x, y}], {x, -2, 2}, {y, -2, 2}]
```

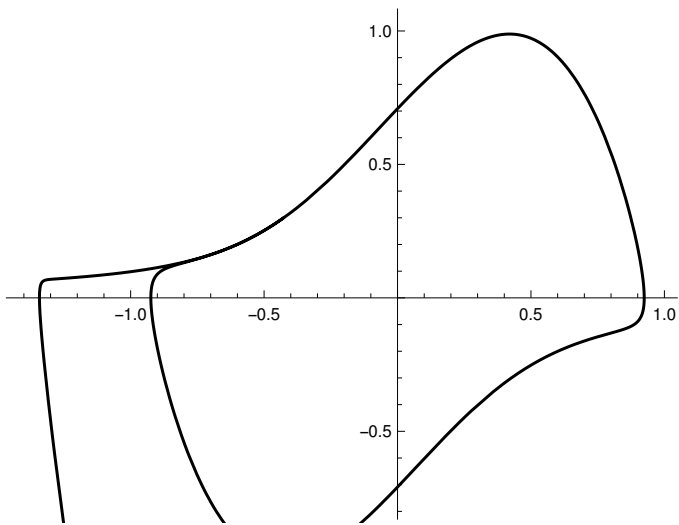


```
sol = NDSolve[{x'[t] == f[{x[t], y[t]}][[1]],
  y'[t] == f[{x[t], y[t]}][[2]], x[0] == -1.2, y[0] == -1.2}, {x, y}, {t, 0, 10}]
```

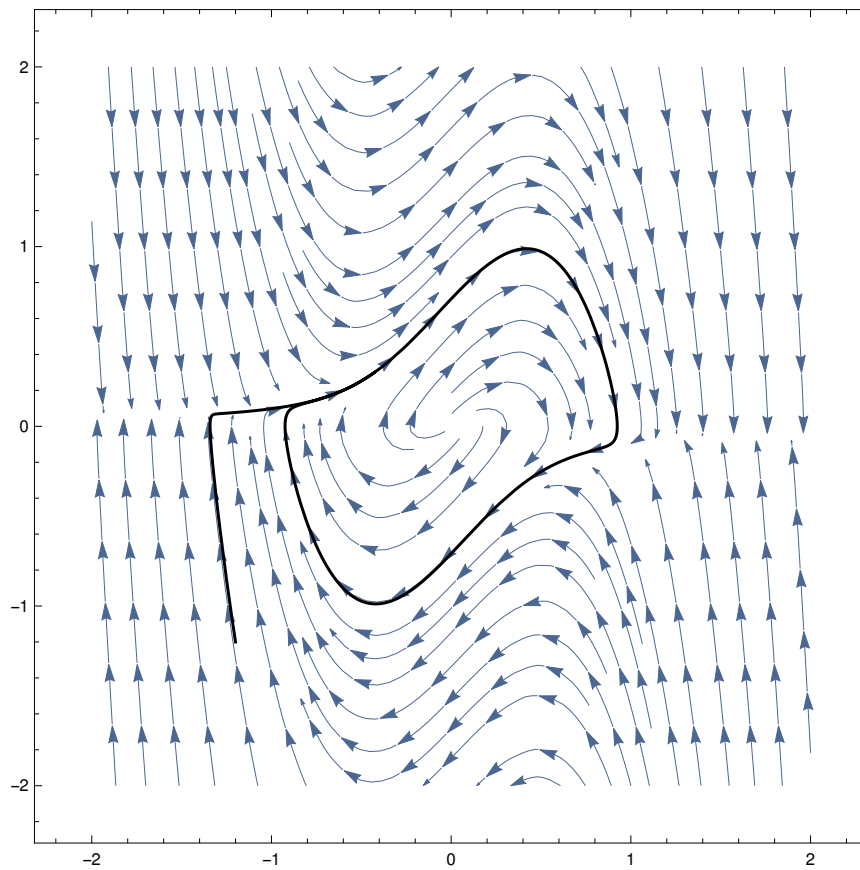
```
{x → InterpolatingFunction[  Domain: {{0., 10.}}
Output: scalar],
```

```
y → InterpolatingFunction[  Domain: {{0., 10.}}
Output: scalar]]]
```

```
rpp = ParametricPlot[{x[t], y[t]} /. sol, {t, 0, 10}, PlotStyle -> Black]
```

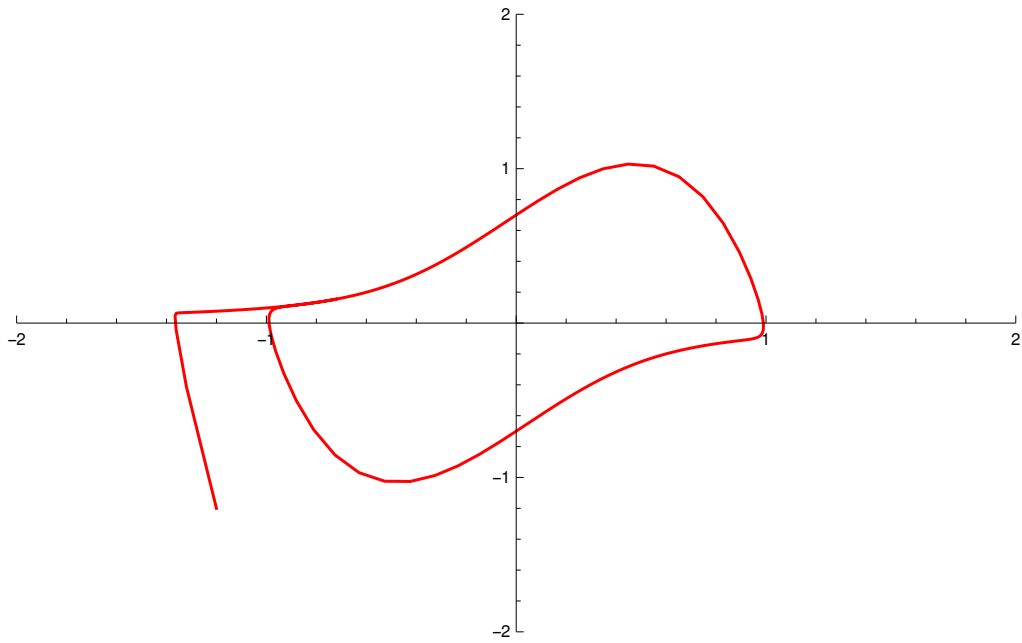


```
Show[sp, rpp]
```

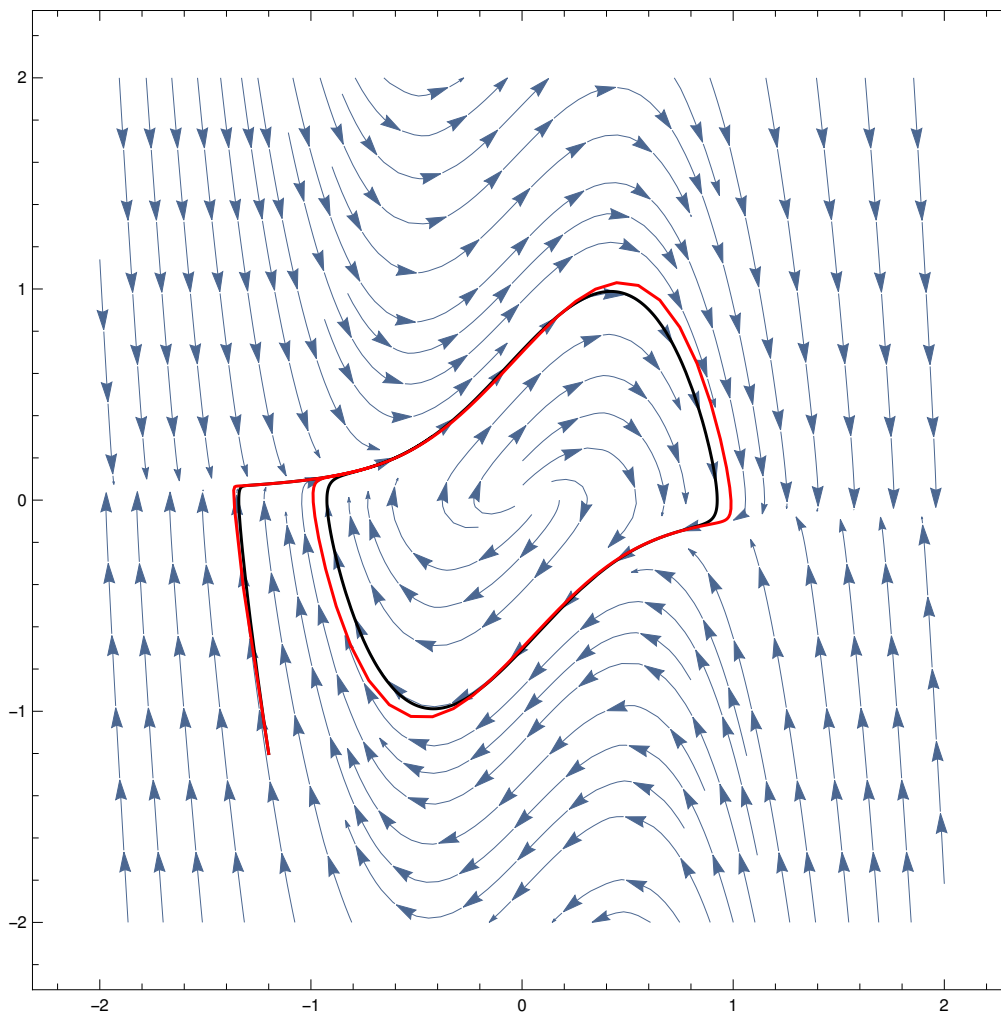


Euler approximation of Van der Pol

```
deltaT = 0.05;  
lp = NestList[# + f[#] deltaT &, {-1.2, -1.2}, Ceiling[10/deltaT]];  
onep = ListPlot[lp, Joined -> True, PlotStyle -> Red, PlotRange -> {{-2, 2}, {-2, 2}}]
```



```
Show[sp, rpp, onep]
```



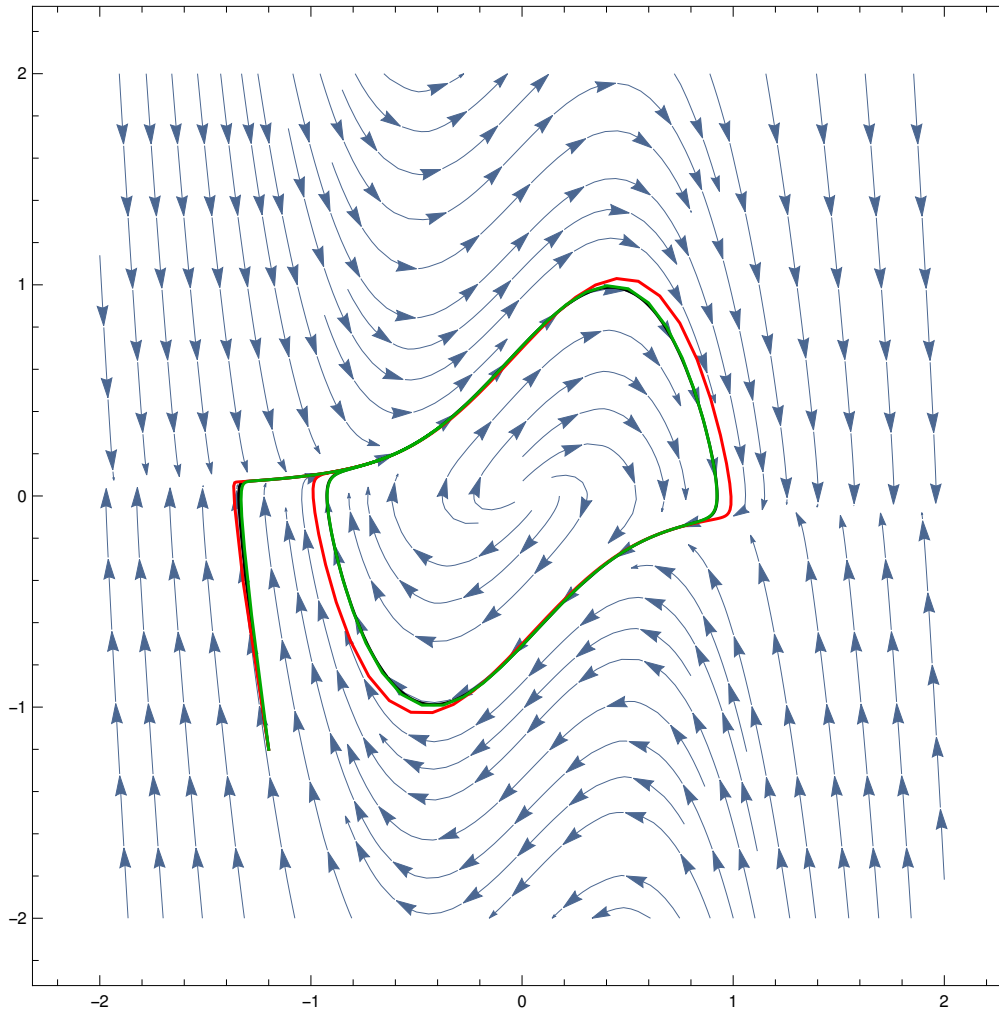
Taylor approximation of Van der Pol

$$f2[\{x_, y_\}] := (\text{Grad}[f[\{a, b\}], \{a, b\}] /. \{a \rightarrow x, b \rightarrow y\}) . f[\{x, y\}]$$

```

lpt =
  NestList[# + f[#] deltaT + f2[#] deltaT^2/2 &, {-1.2, -1.2}, Ceiling[10/deltaT]];
taylorp = ListPlot[lpt, Joined -> True, PlotStyle -> Darker[Green],
  PlotRange -> {{-2, 2}, {-2, 2}}];
(*taylorp1=ListPlot[lpt+0.1,lpt-0.1, Joined->True,PlotStyle->Darker[Green],
  PlotRange->{{-2,2},{-2,2}},,Filling->{1->{2}}];*)
Show[sp, rpp, onep, taylorp]

```



```

# + f[#] deltaT + f2[#] deltaT^2/2 &[{x, y}] // Simplify

```

```

{0.998 x + 0.10525 y - 0.025 x^2 y,
 0.01 x^3 + 1.10851 y - 0.5525 x^2 y + 0.125 x^4 y + x (-0.0421 - 0.05 y^2)}

```

```

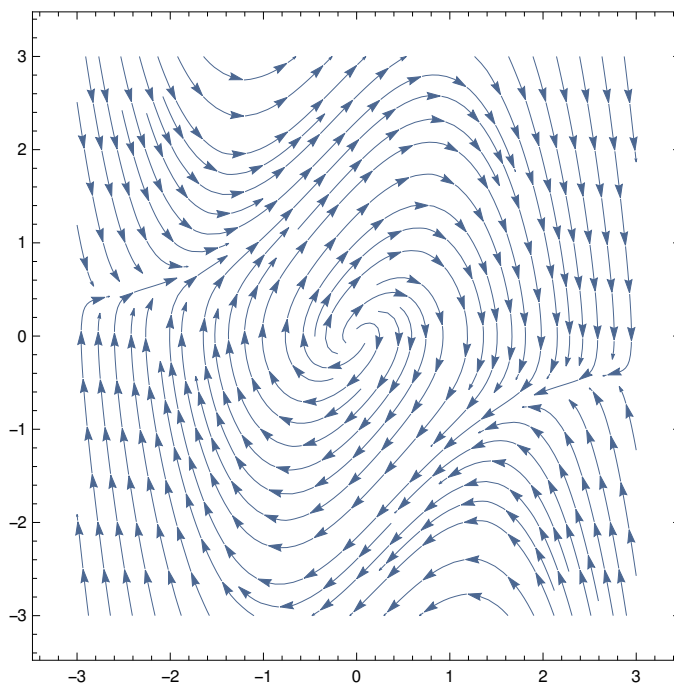
style[{0.9980000000000001` x + 0.1052500000000002` y - 0.0250000000000005` x^2 y,
  0.01000000000000002` x^3 + 1.1085125` y - 0.5525` x^2 y + 0.1250000000000003` x^4 y +
  x (-0.04210000000000005` - 0.0500000000000001` y^2)}, Bold, FontSize -> 20]
{0.998 x + 0.10525 y - 0.025 x^2 y, 0.01 x^3 + 1.10851 y -
  0.5525 x^2 y + 0.125 x^4 y + x (-0.0421 - 0.05 y^2)}

```

```

f[{x_, y_}] := -{-y, x + 1 (x^2 - 1) y}
sp1 = StreamPlot[f[{x, y}], {x, -3, 3}, {y, -3, 3}]

```



```

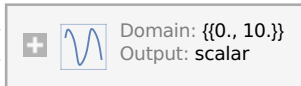
sol1 = NDSolve[{x'[t] == f[{x[t], y[t]}][[1]],
  y'[t] == f[{x[t], y[t]}][[2]], x[0] == 1.1, y[0] == 2.35}, {x, y}, {t, 0, 10}]

```

```

{{x -> InterpolatingFunction[

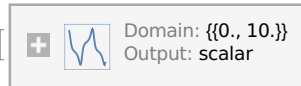
```



```

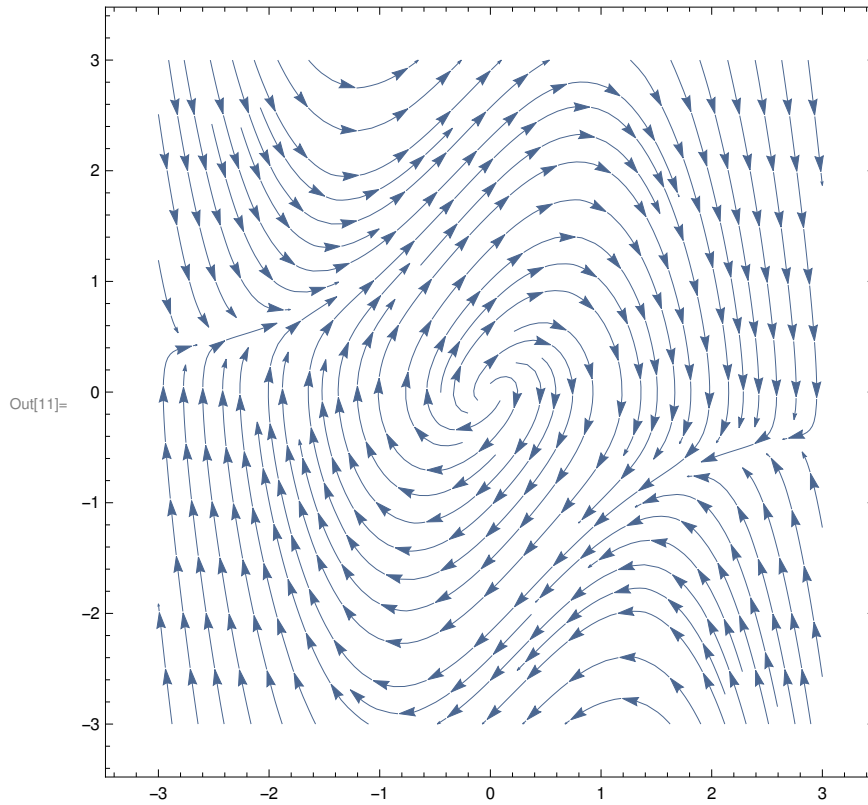
y -> InterpolatingFunction[

```





Van der pol example from [CAS12]

```
In[10]:= f[{x_, y_}] := -{-y, x + 1 (x^2 - 1) y}
sp1 = StreamPlot[f[{x, y}], {x, -3, 3}, {y, -3, 3}]
```

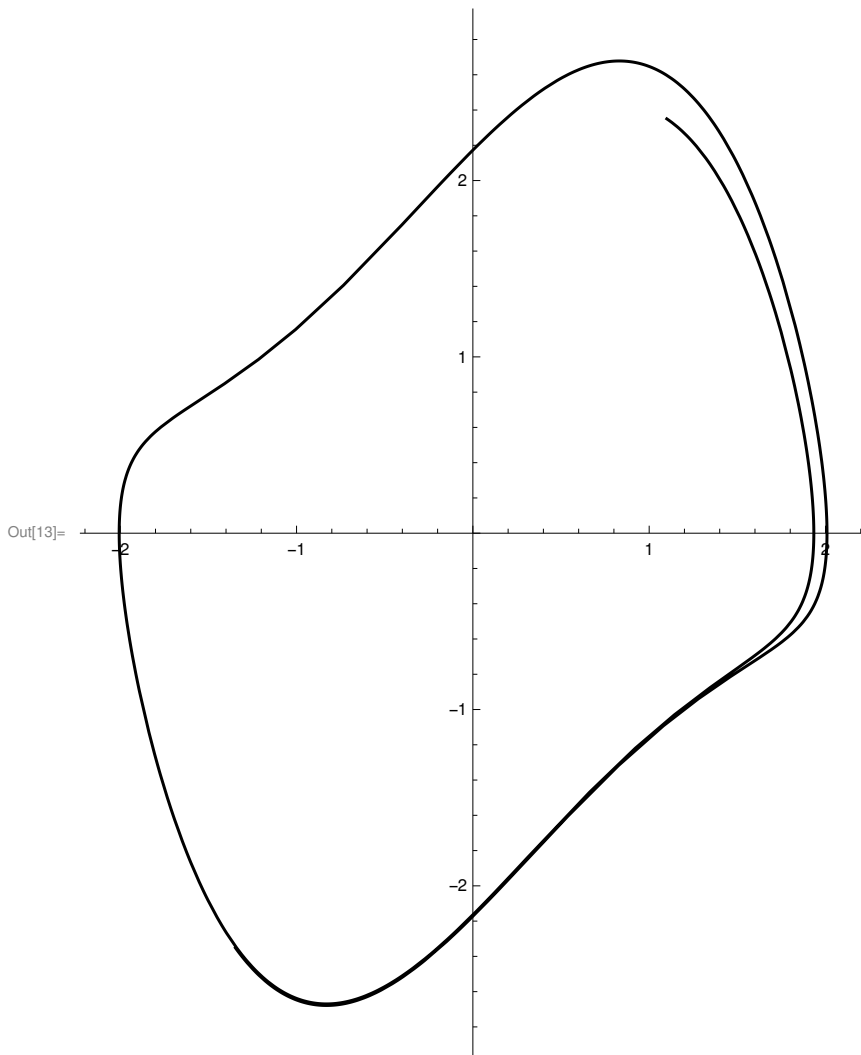


```
In[12]:= sol1 = NDSolve[{x'[t] == f[{x[t], y[t]}][[1]],
  y'[t] == f[{x[t], y[t]}][[2]], x[0] == 1.1, y[0] == 2.35}, {x, y}, {t, 0, 10}]
```

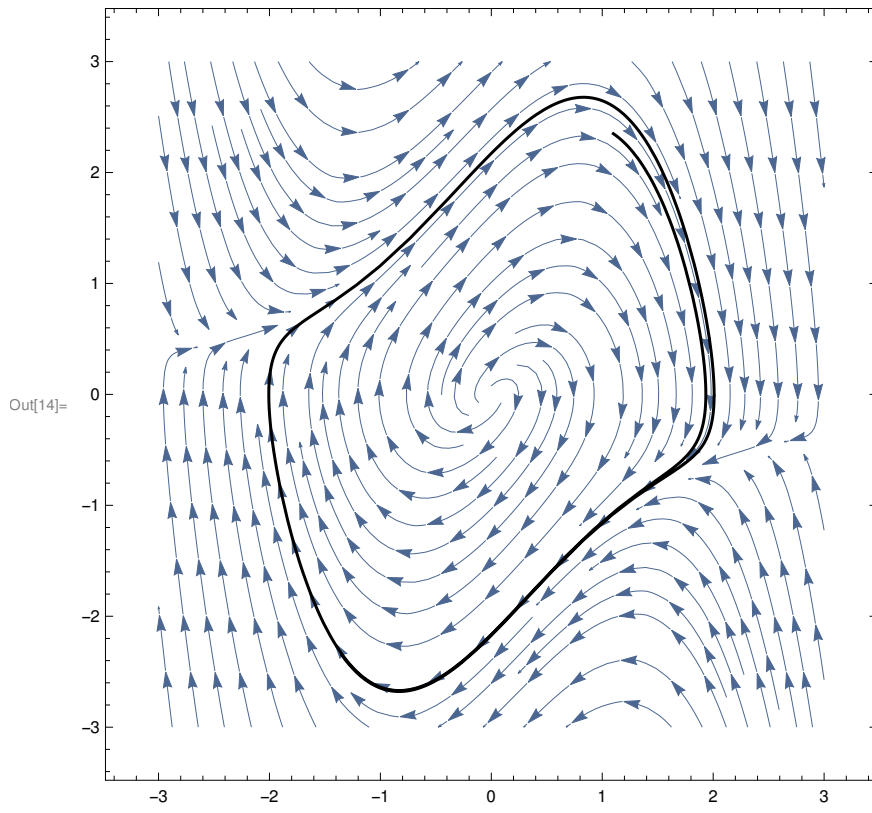
Out[12]=

```
{ {x -> InterpolatingFunction[  
  { +  Domain: {{0., 10.}}  
  Output: scalar  
  },  
  
  y -> InterpolatingFunction[  
  { +  Domain: {{0., 10.}}  
  Output: scalar  
  } ] }
```

```
In[13]:= rpp1 = ParametricPlot[{x[t], y[t]} /. sol1, {t, 0, 10}, PlotStyle -> Black]
```



```
In[14]:= Show[sp1, rpp1]
```



Taylor models

$$\begin{aligned}x = & 1.250000000000000 + 0.149999999999999 * x + 2.400000000000000 * 0.020000000000000 + \\ & 0.050000000000000 * 0.020000000000000 * y + -1.300000000000010 * 0.000400000000000 + \\ & -0.014062500000000 * 0.000400000000000 * y + \\ & -0.525000000000001 * 0.000400000000000 * x + -2.556250000000010 * \\ & 0.000080000000000 + -0.009375000000000 * 0.000400000000000 * x * y + \\ & -0.027000000000000 * 0.000400000000000 * x * x + -0.105696614583334 * \\ & 0.000080000000000 * y + -0.027062500000000 * 0.000080000000000 * x + \\ & 1.265805989583330 * 0.0000016000000 + [-0.00000187, 0.00000391];\end{aligned}$$
$$\begin{aligned}y = & 2.400000000000000 + 0.050000000000000 * y + -2.600000000000010 * 0.020000000000000 + \\ & -0.028125000000000 * 0.020000000000000 * y + -1.050000000000010 * \\ & 0.020000000000000 * x + -7.668750000000010 * 0.000400000000000 + \\ & -0.018750000000000 * 0.020000000000000 * x * y + -0.054000000000000 * \\ & 0.020000000000000 * x * x + -0.317089843750000 * 0.000400000000000 * y + \\ & -0.081187500000001 * 0.000400000000000 * x + 5.063223958333320 * 0.000080000000000 + \\ & -0.001125000000000 * 0.020000000000000 * x * x * y + \\ & -0.003125000000000 * 0.000400000000000 * y * y + \\ & -0.025453125000000 * 0.000400000000000 * x * y + 0.241312499999999 * \\ & 0.000400000000000 * x * x + 0.023016845703125 * 0.000080000000000 * y + \\ & 5.234816406249990 * 0.000080000000000 * x + 20.640046630859299 * 0.0000016000000 + \\ & -0.000375000000000 * 0.000400000000000 * x * y * y + \\ & 0.004148437500000 * 0.000400000000000 * x * x * y + \\ & 0.021937500000000 * 0.000400000000000 * x * x * x + \\ & -0.003656250000000 * 0.000080000000000 * y * y + \\ & 0.199033691406248 * 0.000080000000000 * x * y + \\ & 0.571417968749998 * 0.000080000000000 * x * x + 1.324195222218820 * \\ & 0.0000016000000 * y + 1.863067321777330 * 0.0000016000000 * x + \\ & -7.809814881388410 * 0.0000000320000 + [-0.00000497, 0.00000411];\end{aligned}$$
$$\begin{aligned}x = & 1.297460774828650 + 0.000993529427083 * y + 0.149789783499999 * x + \\ & 2.344975851362140 * 0.020000000000000 + -0.000003750000000 * x * y + \\ & -0.000010800000000 * x * x + 0.049311060068501 * 0.020000000000000 * y + \\ & -0.020990298377979 * 0.020000000000000 * x + -1.450013867722650 * \\ & 0.000400000000000 + -0.000001279250000 * 0.020000000000000 * y * y + \\ & -0.000383588980469 * 0.020000000000000 * x * y + -0.000978903656250 * \\ & 0.020000000000000 * x * x + -0.020369291332501 * 0.000400000000000 * y + \\ & -0.523460001038831 * 0.000400000000000 * x + -2.438721423199720 * \\ & 0.000080000000000 + -0.000000150000000 * 0.020000000000000 * x * y * y + \\ & -0.000020840625000 * 0.020000000000000 * x * x * y + 0.00008775000000 * \\ & 0.020000000000000 * x * x * x + -0.000064285425443 * 0.000400000000000 * y * y + \\ & -0.009760990561122 * 0.000400000000000 * x * y + -0.021854946267443 * \\ & 0.000400000000000 * x * x + -0.104173089989881 * 0.000080000000000 * y + \\ & 0.078628556796722 * 0.000080000000000 * x + 1.668768707932040 * 0.0000016000000 + \\ & 0.000000001649037 * 0.000400000000000 * y * y * y +\end{aligned}$$


```

0.000001043002865 * 0.0004000000000000 * x * y * y +
0.000084863493700 * 0.0004000000000000 * x * x * y +
0.000190747342518 * 0.0004000000000000 * x * x * x +
0.000002708388674 * 0.0000080000000000 * y * y +
0.000811266854204 * 0.0000080000000000 * x * y +
0.002070039621642 * 0.0000080000000000 * x * x +
0.000000000187136 * 0.0004000000000000 * x * y * y * y +
0.000000054116128 * 0.0004000000000000 * x * x * y * y +
0.000004028755362 * 0.0004000000000000 * x * x * x * y +
-0.000001719243105 * 0.0004000000000000 * x * x * x * x +
0.000000317575377 * 0.0000080000000000 * x * y * y +
0.000044123128946 * 0.0000080000000000 * x * x * y +
-0.000018578159556 * 0.0000080000000000 * x * x * x + [-0.000005, 0.000005];

```

```

y = 2.344975851362140 + 0.049311060068501 * y +
-0.020990298377979 * x + -2.900027735445300 * 0.0200000000000000 +
-0.000001279250000 * y * y + -0.000383588980469 * x * y +
-0.000978903656250 * x * x + -0.040738582665002 * 0.0200000000000000 * y +
-1.046920002077670 * 0.0200000000000000 * x +
-7.316164269599150 * 0.0004000000000000 + -0.000000150000000 * x * y * y +
-0.000020840625000 * x * x * y + 0.000008775000000 * x * x * x +
-0.000128570850887 * 0.0200000000000000 * y * y +
-0.019521981122245 * 0.0200000000000000 * x * y + -0.043709892534886 *
0.0200000000000000 * x * x + -0.312519269969641 * 0.0004000000000000 * y +
0.235885670390167 * 0.0004000000000000 * x + 6.675074831728170 * 0.0000080000000000 +
-0.000000045376909 * 0.0200000000000000 * y * y * y +
-0.000012570273660 * 0.0200000000000000 * x * y * y +
-0.000930416667882 * 0.0200000000000000 * x * x * y +
0.000852453574004 * 0.0200000000000000 * x * x * x +
-0.003278347717570 * 0.0004000000000000 * y * y +
-0.012955979927548 * 0.0004000000000000 * x * y + 0.271573835910568 *
0.0004000000000000 * x * x + 0.129357858163038 * 0.0000080000000000 * y +
5.309022682764400 * 0.0000080000000000 * x + 19.554460248285199 * 0.0000001600000000 +
0.000000000001263 * 0.0200000000000000 * y * y * y * y +
0.000000001501110 * 0.0200000000000000 * x * y * y * y +
0.000000308371906 * 0.0200000000000000 * x * x * y * y +
0.000017090963173 * 0.0200000000000000 * x * x * x * y +
0.000018457241547 * 0.0200000000000000 * x * x * x * x +
-0.000002044847252 * 0.0004000000000000 * y * y * y +
-0.000247796329925 * 0.0004000000000000 * x * y * y +
0.005601208918108 * 0.0004000000000000 * x * x * y +
0.020438597603958 * 0.0004000000000000 * x * x * x +
-0.001425966550607 * 0.0000080000000000 * y * y +
0.216066804280903 * 0.0000080000000000 * x * y + 0.433192114128132 *
0.0000080000000000 * x * x + 1.325571097710350 * 0.0000001600000000 * y +
-0.022250595729791 * 0.0000001600000000 * x + -13.857181294106701 *
0.000000003200000 + 0.000000000000139 * 0.0200000000000000 * x * y * y * y * y +
0.000000000058742 * 0.0200000000000000 * x * x * y * y * y +

```

0.000000008898612 * 0.020000000000000 * x * x * x * y * y +
 0.000000462124328 * 0.020000000000000 * x * x * x * x * y +
 -0.000000199803327 * 0.020000000000000 * x * x * x * x * x +
 0.00000000118108 * 0.000400000000000 * y * y * y * y +
 0.000000079252484 * 0.000400000000000 * x * y * y * y +
 0.000007510353115 * 0.000400000000000 * x * x * y * y +
 -0.000020500747434 * 0.000400000000000 * x * x * x * y +
 -0.000104248648700 * 0.000400000000000 * x * x * x * x +
 0.000000110874443 * 0.000080000000000 * y * y * y +
 0.000022112852103 * 0.000080000000000 * x * y * y +
 -0.001632329352903 * 0.000080000000000 * x * x * y +
 -0.004232503557940 * 0.000080000000000 * x * x * x + -0.000034295568214 *
 0.000000160000000 * y * y + -0.010297214428379 * 0.000000160000000 * x * y +
 -0.026282515936326 * 0.000000160000000 * x * x + [-0.00001, 0.00001];

$x = 1.343762120812580 + 0.001970769527200 * y +$
 0.149161222560479 * x + 2.284104820433750 * 0.020000000000000 +
 -0.000000051277503 * y * y + -0.000015319685699 * x * y +
 -0.000039103491315 * x * x + 0.048372527661454 * 0.020000000000000 * y +
 -0.041791875530009 * 0.020000000000000 * x +
 -1.592028846973500 * 0.000400000000000 + 0.000000000000660 * y * y * y +
 -0.000000002580258 * x * y * y + -0.000000382514117 * x * x * y +
 0.000000251650312 * x * x * x + -0.000005173419324 * 0.020000000000000 * y * y +
 -0.000777484108005 * 0.020000000000000 * x * y +
 -0.001741010640873 * 0.020000000000000 * x * x + -0.026521046375673 *
 0.000400000000000 * y + -0.515564400964093 * 0.000400000000000 * x +
 -2.289982762127380 * 0.000080000000000 + 0.000000000000075 * x * y * y * y +
 0.000000000021646 * x * x * y * y + 0.000000001611502 * x * x * x * y +
 -0.000000000687697 * x * x * x * x + -0.000000001724590 * 0.020000000000000 * y * y * y +
 -0.000000500347746 * 0.020000000000000 * x * y * y +
 -0.000037221622820 * 0.020000000000000 * x * x * y +
 0.000033965688133 * 0.020000000000000 * x * x * x +
 -0.000130270852546 * 0.000400000000000 * y * y +
 -0.009888531190765 * 0.000400000000000 * x * y + -0.016192022784668 *
 0.000400000000000 * x * x + -0.100535641042507 * 0.000080000000000 * y +
 0.184278970194921 * 0.000080000000000 * x + 2.044088614581010 * 0.000000160000000 +
 0.0000000000000073 * 0.020000000000000 * y * y * y * y +
 0.000000000061819 * 0.020000000000000 * x * y * y * y +
 0.000000009179844 * 0.020000000000000 * x * x * y * y +
 0.000000332895749 * 0.020000000000000 * x * x * x * y +
 0.000000327738620 * 0.020000000000000 * x * x * x * x +
 0.000000017956799 * 0.000400000000000 * y * y * y +
 0.000004386037167 * 0.000400000000000 * x * y * y +
 0.000183884216745 * 0.000400000000000 * x * x * y +
 0.000345508632357 * 0.000400000000000 * x * x * x +
 0.000010855234122 * 0.000080000000000 * y * y +
 0.001626455712536 * 0.000080000000000 * x * y + 0.003639003852049 *
 0.000080000000000 * x * x + 0.00000000004132 * 0.000400000000000 * y * y * y * y +

0.000000001662825 * 0.000400000000000 * x * y * y * y + 0.000000205735154 *
 0.000400000000000 * x * x * y * y + 0.000007244078502 * 0.000400000000000 * x * x * x * y +
 -0.000007102395848 * 0.000400000000000 * x * x * x * x +
 0.000000003630121 * 0.000008000000000 * y * y * y + 0.000001051401279 *
 0.000008000000000 * x * y * y + 0.000078092268443 * 0.000008000000000 * x * x * y +
 -0.000071324101806 * 0.000008000000000 * x * x * x + [-0.000009, 0.00001];

y = 2.284104820433750 + 0.048372527661454 * y +
 -0.041791875530009 * x + -3.184057693947000 * 0.020000000000000 +
 -0.000005173419324 * y * y + -0.000777484108005 * x * y +
 -0.001741010640873 * x * x + -0.053042092751346 * 0.020000000000000 * y +
 -1.031128801928190 * 0.020000000000000 * x +
 -6.869948286382120 * 0.000400000000000 + -0.000000001724590 * y * y * y +
 -0.000000500347746 * x * y * y + -0.000037221622820 * x * x * y +
 0.000033965688133 * x * x * x + -0.000260541705092 * 0.020000000000000 * y * y +
 -0.019777062381530 * 0.020000000000000 * x * y + -0.032384045569336 *
 0.020000000000000 * x * x + -0.301606923127520 * 0.000400000000000 * y +
 0.552836910584765 * 0.000400000000000 * x + 8.176354458324040 * 0.000008000000000 +
 0.0000000000000073 * y * y * y * y + 0.000000000061819 * x * y * y * y +
 0.000000009179844 * x * x * y * y + 0.000000332895749 * x * x * x * y +
 0.000000327738620 * x * x * x * x + -0.000000151962037 * 0.020000000000000 * y * y * y +
 -0.000019505016723 * 0.020000000000000 * x * y * y +
 -0.000683904856401 * 0.020000000000000 * x * x * y +
 0.001620847642007 * 0.020000000000000 * x * x * x +
 -0.003293760456925 * 0.000400000000000 * y * y +
 0.000289475752754 * 0.000400000000000 * x * y + 0.292868151442852 *
 0.000400000000000 * x * x + 0.233706537681607 * 0.000008000000000 * y +
 5.230244782780160 * 0.000008000000000 * x + 17.882827958977700 * 0.000000160000000 +
 0.000000000038134 * 0.020000000000000 * y * y * y * y +
 0.000000013039331 * 0.020000000000000 * x * y * y * y +
 0.000001215803377 * 0.020000000000000 * x * x * y * y +
 0.000033176887379 * 0.020000000000000 * x * x * x * y +
 0.000024043555475 * 0.020000000000000 * x * x * x * x +
 -0.000003019269423 * 0.000400000000000 * y * y * y +
 -0.000094570784590 * 0.000400000000000 * x * y * y +
 0.006724253313122 * 0.000400000000000 * x * x * y +
 0.017733161836865 * 0.000400000000000 * x * x * x +
 0.000918763969927 * 0.000008000000000 * y * y + 0.223822022223906 *
 0.000008000000000 * x * y + 0.273886467503252 * 0.000008000000000 * x * x +
 1.274395031891770 * 0.000000160000000 * y + -1.945143601429410 *
 0.000000160000000 * x + -19.478260107031701 * 0.000000003200000 +
 0.000000000000005 * 0.020000000000000 * y * y * y * y * y +
 0.000000000002273 * 0.020000000000000 * x * y * y * y * y +
 0.000000000378396 * 0.020000000000000 * x * x * y * y * y +
 0.000000026252562 * 0.020000000000000 * x * x * x * y * y +
 0.000000642664873 * 0.020000000000000 * x * x * x * x * y +
 -0.000000899012044 * 0.020000000000000 * x * x * x * x * x +
 0.000000001165744 * 0.000400000000000 * y * y * y * y +

```

0.000000269540031 * 0.0004000000000000 * x * y * y * y +
0.000011864752566 * 0.0004000000000000 * x * x * y * y +
-0.000077495924813 * 0.0004000000000000 * x * x * x * y +
-0.000221624901250 * 0.0004000000000000 * x * x * x * x +
0.000000035270696 * 0.0000080000000000 * y * y * y +
-0.000030618855085 * 0.0000080000000000 * x * y * y +
-0.003724434210715 * 0.0000080000000000 * x * x * y +
-0.007886401324901 * 0.0000080000000000 * x * x * x + -0.000137188369559 *
0.000000160000000 * y * y + -0.020574669671808 * 0.000000160000000 * x * y +
-0.046045758162329 * 0.000000160000000 * x * x + [-0.00002, 0.0002];

```

```

x = 1.388790528377040 + 0.002926807376751 * y +
0.148120633521254 * x + 2.217743353255480 * 0.0200000000000000 +
-0.000000206767389 * y * y + -0.000034811768690 * x * y +
-0.000080371401216 * x * x + 0.047193116592683 * 0.0200000000000000 * y +
-0.062151786069053 * 0.0200000000000000 * x +
-1.724247782317400 * 0.0004000000000000 + -0.000000000026620 * y * y * y +
-0.000000010824387 * x * y * y + -0.000001052768149 * x * x * y +
0.000001068596935 * x * x * x + -0.000011694429447 * 0.0200000000000000 * y * y +
-0.001171122281104 * 0.0200000000000000 * x * y +
-0.002269360567264 * 0.0200000000000000 * x * x + -0.032392961755710 *
0.0004000000000000 * y + -0.501407650374760 * 0.0004000000000000 * x +
-2.112700866032790 * 0.0000080000000000 + 0.0000000000000003 * y * y * y * y +
0.000000000001975 * x * y * y * y + 0.000000000287383 * x * x * y * y +
0.000000011161452 * x * x * x * y + 0.000000003020595 * x * x * x * x +
-0.000000005971264 * 0.0200000000000000 * y * y * y +
-0.000000928523470 * 0.0200000000000000 * x * y * y +
-0.000048239972011 * 0.0200000000000000 * x * x * y +
0.000073412958686 * 0.0200000000000000 * x * x * x +
-0.000195154974095 * 0.0004000000000000 * y * y +
-0.009748375354397 * 0.0004000000000000 * x * y + -0.010201633605169 *
0.0004000000000000 * x * x + -0.094865912898692 * 0.0000080000000000 * y +
0.286825336626227 * 0.0000080000000000 * x + 2.380573069100150 * 0.000000160000000 +
0.000000000001303 * 0.0200000000000000 * y * y * y * y +
0.000000000430397 * 0.0200000000000000 * x * y * y * y +
0.000000038224981 * 0.0200000000000000 * x * x * y * y +
0.000000964086594 * 0.0200000000000000 * x * x * x * y +
0.000000721195337 * 0.0200000000000000 * x * x * x * x +
0.000000065296705 * 0.0004000000000000 * y * y * y +
0.000010193504856 * 0.0004000000000000 * x * y * y +
0.000290524200886 * 0.0004000000000000 * x * x * y + 0.000448374696657 *
0.0004000000000000 * x * x * x + 0.000024060043898 * 0.0000080000000000 * y * y +
0.002394433871851 * 0.0000080000000000 * x * y + 0.004626094901613 *
0.0000080000000000 * x * x + 0.00000000022167 * 0.0004000000000000 * y * y * y * y +
0.000000004664106 * 0.0004000000000000 * x * y * y * y + 0.000000386805636 *
0.0004000000000000 * x * x * y * y + 0.000009068664284 * 0.0004000000000000 * x * x * x * y +
-0.000015966643929 * 0.0004000000000000 * x * x * x * x +
0.000000012369347 * 0.0000080000000000 * y * y * y + 0.000001916295773 *

```

0.000008000000000 * x * y * y + 0.000099035725391 * 0.000008000000000 * x * x * y +
 -0.000151283721521 * 0.000008000000000 * x * x * x + [-0.00001, 0.00001];

y = 2.217743353255480 + 0.047193116592683 * y +
 -0.062151786069053 * x + -3.448495564634790 * 0.020000000000000 +
 -0.000011694429447 * y * y + -0.001171122281104 * x * y +
 -0.002269360567264 * x * x + -0.064785923511421 * 0.020000000000000 * y +
 -1.002815300749520 * 0.020000000000000 * x +
 -6.338102598098360 * 0.000400000000000 + -0.00000005971264 * y * y * y +
 -0.000000928523470 * x * y * y + -0.000048239972011 * x * x * y +
 0.000073412958686 * x * x * x + -0.000390309948190 * 0.020000000000000 * y * y +
 -0.019496750708795 * 0.020000000000000 * x * y + -0.020403267210338 *
 0.020000000000000 * x * x + -0.284597738696077 * 0.000400000000000 * y +
 0.860476009878683 * 0.000400000000000 * x + 9.522292276400600 * 0.000008000000000 +
 0.00000000001303 * y * y * y * y + 0.000000000430397 * x * y * y * y +
 0.000000038224981 * x * x * y * y + 0.000000964086594 * x * x * x * y +
 0.000000721195337 * x * x * x * x + -0.000000273672332 * 0.020000000000000 * y * y * y +
 -0.000019998958506 * 0.020000000000000 * x * y * y +
 -0.000400467305555 * 0.020000000000000 * x * x * y +
 0.002260342306117 * 0.020000000000000 * x * x * x +
 -0.003168486782666 * 0.000400000000000 * y * y +
 0.013704598605890 * 0.000400000000000 * x * y + 0.304184312374466 *
 0.000400000000000 * x * x + 0.331906823692361 * 0.000008000000000 * y +
 4.999438502677970 * 0.000008000000000 * x + 15.682999698478900 * 0.000000160000000 +
 0.000000000201631 * 0.020000000000000 * y * y * y * y +
 0.000000041932093 * 0.020000000000000 * x * y * y * y +
 0.000002557453090 * 0.020000000000000 * x * x * y * y +
 0.000046252499394 * 0.020000000000000 * x * x * x * y +
 0.000016376060156 * 0.020000000000000 * x * x * x * x +
 -0.000002780370495 * 0.000400000000000 * y * y * y +
 0.000072403301582 * 0.000400000000000 * x * y * y +
 0.007431097002517 * 0.000400000000000 * x * x * y +
 0.014011038493892 * 0.000400000000000 * x * x * x +
 0.003239782903041 * 0.000008000000000 * y * y + 0.221679942965125 *
 0.000008000000000 * x * y + 0.102172960912431 * 0.000008000000000 * x * x +
 1.172444868656590 * 0.000000160000000 * y + -3.806384018159250 *
 0.000000160000000 * x + -24.374005375373901 * 0.000000032000000 +
 0.000000000000027 * 0.020000000000000 * y * y * y * y * y +
 0.0000000000005407 * 0.020000000000000 * x * y * y * y * y +
 0.000000000568726 * 0.020000000000000 * x * x * y * y * y +
 0.000000027950411 * 0.020000000000000 * x * x * x * y * y +
 0.000000493087093 * 0.020000000000000 * x * x * x * x * y +
 -0.000001919285337 * 0.020000000000000 * x * x * x * x * x +
 0.000000003516395 * 0.000400000000000 * y * y * y * y +
 0.000000479163501 * 0.000400000000000 * x * y * y * y +
 0.000011463007978 * 0.000400000000000 * x * x * y * y +
 -0.000165002122827 * 0.000400000000000 * x * x * x * y +
 -0.000323178151073 * 0.000400000000000 * x * x * x * x +

```
-0.000000857425884 * 0.000008000000000 * y * y * y +  
-0.000164589901815 * 0.000008000000000 * x * y * y +  
-0.006032666279263 * 0.000008000000000 * x * x * y +  
-0.010387488329342 * 0.000008000000000 * x * x * x + -0.000298249558236 *  
0.000000160000000 * y * y + -0.029657811832356 * 0.000000160000000 * x * y +  
-0.057277646086715 * 0.000000160000000 * x * x + [-0.00002, 0.00002];
```

Titre : Synthèse d'invariants : une approche programmation par contraintes basée sur l'abstraction zonotopique

Mots clés : Systèmes dynamiques, vérification de programme, génération d'invariants, interprétation abstraite, programmation par contraintes, zonotopes

Résumé : *Les systèmes dynamiques* sont des modèles mathématiques pour décrire l'évolution temporelle de l'état d'un système. Il y a deux classes de systèmes dynamiques pertinentes à cette thèse : les systèmes discrets et les systèmes continus. Dans *les systèmes dynamiques discrets* (ou les programmes informatiques classiques), l'état évolue avec un pas de temps discrets. Dans *les systèmes dynamiques continus*, l'état du système est fonction du temps continu, et son évolution caractérisée par des équations différentielles. Étant donné que ces systèmes peuvent prendre des décisions critiques, il est important de pouvoir vérifier des propriétés garantissant leur sûreté. Par exemple, sur un programme, l'absence de débordement arithmétique. Dans cette thèse, nous développons un cadre pour la vérification automatique des propriétés de sûreté des programmes. Un élément clé de cette vérification est la preuve de propriétés invariantes. Nous développons ici un algorithme pour synthétiser des *invariants inductifs* (des propriétés vraies pour l'état initial, qui sont stables dans l'évolution des états du programme, donc sont toujours vraies par récurrence) pour des programmes numériques. *L'interprétation abstraite* (IA) est une approche traditionnelle pour la recherche d'invariants inductifs des programmes numériques. L'IA interprète les instructions du programme dans un *domaine abstrait* (par exemple intervalles, octogones, polyèdres, zonotopes), domaine qui est choisi en fonction des propriétés à prouver. Un invariant inductif peut être calculé comme limite possiblement infinie des itérées d'une fonctionnelle croissante. L'analyse peut recourir aux *opérateurs d'élargissement* pour forcer la convergence, au détriment de la précision. Si l'invariant n'est pas prouvé, une solution standard est de remplacer le domaine par un nouveau domaine abstrait davantage susceptible de représenter précisément l'invariant. La *programma-*

tion par contraintes (PPC) est une approche alternative pour synthétiser des invariants, traduisant un programme en contraintes, et les résolvant en utilisant des solveurs de contraintes. Les contraintes peuvent opérer sur des domaines soit discrets, soit continus. La programmation classique par contraintes continues est basée sur un domaine d'intervalle, mais peut approximer une forme invariante complexe par une collection d'éléments abstraits. Une approche existante combine IA et PPC, raffinant de façon itérative, par *découpage* et contraction, une collection d'éléments abstraits, jusqu'à obtenir un invariant inductif. Celle-ci a été initialement présentée en combinaison avec intervalles et octogones. La nouveauté de notre travail est d'étendre ce cadre au domaine abstrait des zonotopes, un domaine sous-polyédrique qui présente un bon compromis en terme de précision et de coût. Cette extension demande de définir de nouveaux opérateurs sur les zonotopes, pour permettre le découpage et la contraction, ainsi que d'adapter l'algorithme générique. Nous introduisons notamment un nouvel algorithme de découpage de zonotopes basé sur un *pavage* par sous-zonotopes et parallélotopes. Nous proposons également des alternatives à certains opérateurs existants sur les zonotopes, mieux adaptés que les existants à la méthode. Nous avons implémenté ces opérations dans la bibliothèque APRON et avons testé l'approche sur des programmes présentant des invariants complexes, éventuellement non convexes. Les résultats démontrent un bon compromis par rapport à l'utilisation de domaines simples, comme les intervalles et les octogones, ou d'un domaine plus coûteux comme les polyèdres. Enfin, nous discutons de l'extension de l'approche pour trouver des ensembles d'invariants positifs pour des systèmes dynamiques continus.

Title : Synthesizing invariants : a constraint programming approach based on zonotopic abstraction

Keywords : Dynamical systems, program verification, invariant generation, abstract interpretation, constraint programming, zonotopes

Abstract : *Dynamical systems* are mathematical models for describing temporal evolution of the state of a system. There are two classes of dynamical systems relevant to this thesis : discrete and continuous. In *discrete dynamical systems* (or classical computer programs), the state evolves in discrete time steps, as described by difference equations. In *continuous dynamical systems*, the state of the system is a function of continuous time, characterized by differential equations. When we analyse the behaviour of a dynamical system, we usually want to make sure that it satisfies a *safety property* expressing that nothing bad happens. An example of a safety property of programs is the *absence of arithmetic overflows*. In this thesis, we design a framework related to the automatic verification of the *safety properties* of programs. Proving that a program satisfies a safety property of interest involves an invariance argument. We develop an algorithm for inferring invariants more precisely *inductive invariants* (properties which hold during the initial state, remains stable under the program evolution, and hence hold always due to induction) for numerical programs. A traditional approach for finding inductive invariants in programs is *abstract interpretation* (AI) that interprets the states of a program in an *abstract domain* (intervals, polyhedra, octagon, zonotopes) of choice. This choice is made based on the property of interest to be inferred. Using the AI framework, inductive invariant can be computed as limits of iterations of functions. However, for abstract domains which feature infinite increasing chain, for instance, interval, these computations may fail to converge. Then, the classical solution would be to withdraw that particular domain and in its place redesign a new abstract domain which can represent the shape of the invariant. One may also use convergence techniques like *widening* to enforce convergence, but

this may come at the cost of precision. Another approach called *constraint programming* (CP), can be used to find invariants by translating a program into constraints and solving them by using constraint solvers. Constraints in CP primarily operate on domains that are either discrete or continuous. Classical *continuous constraint programming* corresponds to interval domain and can approximate a complex shape invariant by a set of boxes, for instance, upto a precision criterion. An existing framework combines AI and continuous CP inspired by iterative refinement, *splitting* and tightening a collection of abstract elements. This was initially presented in combination with simple underlying abstract elements, boxes and octagons. The novelty of our work is to extend this framework by using zonotopes, a sub-polyhedral domain that shows a good compromise between cost and precision. However, zonotopes are not closed under intersection, and we had to extend the existing framework, in addition to designing new operations on zonotopes. We introduce a novel splitting algorithm based on *tiling* zonotopes by sub-zonotopes and parallelotopes. We also propose few alternative operators to the existing ones for a better efficiency of the method. We implemented these operations on top of the APRON library, and tested it on programs with non-linear loops that present complex, possibly non-convex, invariants. We present some results demonstrating the interest of this splitting-based algorithm to synthesize invariants on such programs. This algorithm also shows a good compromise by its use in combination with zonotopes as regards to its use with both simpler domains such as boxes and octagons, and more expressive domains like polyhedra. Finally, we discuss the extension of the approach to infer positive invariant sets for dynamical systems.