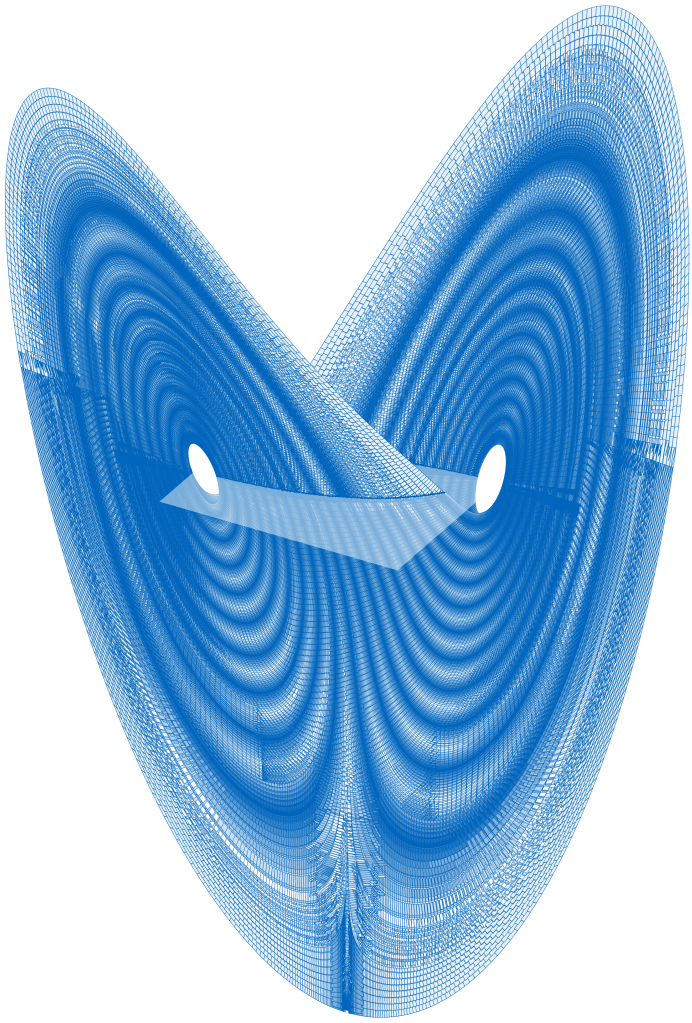
FAKULTÄT FÜR INFORMATIK

LEHRSTUHL FÜR LOGIK UND VERIFIKATION

A Verified ODE Solver and Smale's 14th Problem

Fabian Immler



FAKULTÄT FÜR INFORMATIK

LEHRSTUHL FÜR LOGIK UND VERIFIKATION

A Verified ODE Solver and Smale's 14th Problem

Fabian Immler

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Dr. Thomas Huckle
Prüfer der Dissertation:	
1.	Univ.-Prof. Tobias Nipkow, Ph.D.
2.	Prof. Warwick Tucker, Ph.D. Uppsala University, Schweden

Die Dissertation wurde am 22.12.2017 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 04.04.2018 angenommen.

Abstract: A Verified ODE Solver and Smale's 14th Problem

Ordinary differential equations (ODEs) arise in many mathematical models for e.g., physics, biology, or economics. This thesis enables reasoning about ODEs in the interactive theorem prover Isabelle/HOL with a formalization of abstract theory as well as concrete numerical algorithms.

The first contribution, a library of formalized mathematics for ODEs, encompasses the fundamental results for reasoning about ODEs and their solutions. This is in particular the notion of flow and its continuous and differentiable dependence on initial conditions. The library furthermore includes an important technique for the analysis of dynamical systems, the Poincaré map.

The second contribution is the implementation and verification of a rigorous ODE solver: a numerical algorithm that computes safe enclosures for the aforementioned solutions of ODEs, Poincaré maps and their derivatives. For the implementation, I chose data structures and algorithms that are amenable to formal verification while at the same time exhibiting reasonable performance: affine arithmetic for rigorous numerics and Runge-Kutta methods as approximation scheme for solutions of ODEs. A geometric algorithm for zonotope/hyperplane intersection computes enclosures for Poincaré maps. The ODE solver can also be applied to solve the variational equation in order to enclose the derivative of the flow.

As final contribution, we target one particular dynamical system where a verified algorithm is highly relevant, the Lorenz attractor. The Lorenz attractor is the subject of Smale's 14th problem, an important conjecture from the field of dynamical systems. Tucker settled this conjecture with a proof that relies on numerically computed bounds on a Poincaré map and its derivative. Our verified algorithms are capable of certifying those bounds and therefore lift the numerical part of Tucker's proof onto solid formal foundations.

Acknowledgment

First of all, I would like to thank Tobias Nipkow for supervising this thesis. You gave me the trust and freedom that allowed me to identify and pursue my own research interests and explore whatever problems that appeared along the way. Your door was always open and it was extremely pleasant that we could discuss almost everything in a relaxed atmosphere over a cup of coffee. You always, at the right moments, found the right words of criticism or encouragement and gave invaluable strategic advice.

It is a great honor that Warwick Tucker agreed to act as examiner of this thesis. Thank you and the whole CAPA group for hosting my stays in Uppsala.

Jeremy Avigad pointed me towards Warwick Tucker's proof.

A special 'thanks' goes to Johannes Hölzl, who taught me most of what I know about formalization of mathematics in Isabelle/HOL.

Much of my work would not have been possible without Peter Lammich, his Autoref framework and his helpful feedback and support on it.

I feel very privileged that I could work at the chair of logic and verification and thank all of the past and present members and everyone who is still visiting the group on a regular basis: Mohammad Abdulaziz, Jasmin Blanchette, Sascha Böhme, Julian Brunner, Lukas Bulwahn, Manuel Eberl, Florian Haftmann, Maximilian Haslbeck (der Dienstältere), Johannes Hölzl, Lars Hupel, Alexander Krauss, Ondřej Kunčar, Peter Lammich, Lars Noschinski, Andrei Popescu, Dmitriy Traytel, Makarius Wenzel, Simon Wimmer, and Bohua Zhan. I learned a lot from every single one of you, your projects, and the way you work.

My sincerest apologies go to our office managers Eleni Nikolaou-Weiß and Helma Piller. Too often I bothered them with very last minute requests.

Matthias Althoff and Albert Rizaldi gave me the opportunity to work with them on some more "applied" applications of formal verification.

I would like to thank Helmut Seidl as head of the PUMA research training group as well as all of its members. The regular PUMA workshops and events were fun and always a good opportunity to expand my horizon to other areas of research.

I really enjoyed supervising and working with all of my students who worked on (rigorous or verified) numerics and formalized mathematics: Yutaka Nagashima, Markus Westerlind, Christoph Traut, Maximilian Haslbeck (der Dienstjüngere), Fabian Hellauer, Jana Kuzmanova, Jochen Günther.

Brandon Bohrer was the first external user of my ODE formalization and triggered several useful developments.

Assia Mahboubi and Walid Taha invited me to their respective groups in Paris and Halmstad, thank you for giving me the opportunity to present preliminary versions of my work and discuss it with you and your colleagues.

It was an honor to be invited to the Dagstuhl seminar "Reliable Computation and Complexity on the Reals", the CIRM conference "Effective Analysis: Foundations, Implementations, Certification" and the "Big Proof" programme at the Isaac Newton Institute. Those were inspiring events, I could make many new contacts and feel very welcome in those communities.

I gratefully acknowledge financial support from DFG RTG 1480 (PUMA) and DFG Kosseleck grant NI 491/16-1.

All of my friends: your support means a lot to me. Every word of encouragement, your sympathetic coping with stressed-out versions of mine, and every small distraction contributed to this thesis in immeasurable ways. Julia, your love and patience, especially in the most chaotic phases of this work, is beyond words.

Final thanks and big hugs go to my parents: you made it possible that I could work on one of my hobbies and passions for a living. I remember the quote "Wenn sie klein sind, gib ihnen Wurzeln. Wenn sie groß sind, gib ihnen Flügel" pinned to our cupboard. It is hard to describe how grateful I am for the roots and wings that you gave me. Some people say the Lorenz attractor is shaped like a butterfly's wings, so I dedicate the "wings" on the cover page to you!

Contents

1 Introduction

This thesis describes a rigorous numerical ODE (ordinary differential equation) solver and its formal verification in the interactive theorem prover Isabelle/HOL. The main application is the Lorenz attractor: the verified ODE solver certifies the numerical bounds of Tucker's proof for Smale's 14th problem.

1.1 Motivation

Ordinary differential equations (ODEs) play an important role in many scientific disciplines. An ODE relates the temporal evolution of a quantity with its rate of change, which is a useful concept in models from many scientific and engineering disciplines. ODEs model e.g., the motion of particles, cars, trains, airplanes, or planets. Further examples of models are chemical reactions, spread of diseases, or the evolution of populations.

1.1.1 Rigorous ODE Solvers

Scientists produce models and analyze them in order to gain a deeper understanding of the real behavior. Assuming that the model is valid, i.e., it adequately represents real phenomena, it is essential that the analysis is carried out correctly. Drawing wrong conclusions from a valid model can lead to disastrous failures when e.g., safety-critical systems are subject of the analysis.

How can one verify that a model involving ODEs has been analyzed correctly? It depends on the kind of analysis, which can be broadly categorized as either symbolic or numeric. A symbolic analysis is easy to verify, for example checking that a given function solves an ODE. The fact that the function $t \mapsto e^t$ solves the ODE $\dot{x}(t) = x(t)$ is readily checked by symbolic differentiation. Unfortunately, many ODEs do not admit closed form expressions as solutions, which is why one often resorts to numerical simulations. Numerical simulations do, however, incur approximation errors and therefore uncertainty. This uncertainty is hard to quantify, the fact that a numerical simulation is a meaningful approximation of the modeled behavior is therefore hard to verify.

To mitigate this problem, rigorous ODE solvers have been developed. A rigorous ODE solver is a program that computes an approximation together with a safe bound on the approximation error. The output of a rigorous ODE solver consists of a set of enclosures that are guaranteed to contain the true solution. This is more reliable than mere simulations and finds its applications in the analysis of safety-critical systems and even mathematical proofs. The very idea of rigorous ODE solving is that the computation of a bound can be seen as a mathematically rigorous proof that the solution is contained in that bound.

1.1.2 A Verified ODE Solver

But rigorous ODE solvers give rise to a new concern. The fact that the solution is enclosed by the computed bounds depends on the fact that the rigorous ODE solver implements the underlying mathematical ideas faithfully. This is a real concern, given the numerous examples of software bugs, even in safety-critical systems. It is therefore not surprising that rigorous numerical tools have been affected by bugs, as well. Mahboubi *et al.* [94] point out bugs in the implementation of supposedly rigorous solvers. There are even examples of flaws in the underlying mathematical ideas and not just in the implementation, as Bünger [29] pointed out for an optimization that is used in a rigorous ODE solver.

A comprehensive approach to ensure correctness of an implementation is formal verification. That is, reasoning about computer programs in a rigorous, machine checkable calculus. We focus on formal verification in interactive theorem provers. The "interactive" part, formal proofs guided by human intuition, is necessary because specifications and correctness arguments can be too involved for purely automatic verification. Interactive formal verification induces large efforts. These efforts are worthwhile when high correctness guarantees are required, in particular for fundamental, potentially safety-critical software. Notable examples are the verified compiler CompCert [90] or the verified operating system kernel seL4 [81].

I claim that ODEs are of similar fundamental importance, because of their ubiquity in all sorts of scientific and engineering disciplines. Despite their importance, formal verification of rigorous ODE solvers has not been undertaken. There is ample work on (rigorous) numerics, from IEEE floating point numbers [57, 17], to formally verified libraries for rigorous numerical computations [104, 96, 33, 146, 26] and nonlinear optimization [128]. Libraries for mathematics, in particular real analysis, are well developed in many proof assistants (as surveyed e.g., by Boldo *et al.* [21]), but no comprehensive formalization of the theory of ODEs.

The main goal of this thesis is the formal verification of a rigorous ODE solver, a *verified* ODE solver. This is a challenging undertaking, because it requires the formalization of the underlying mathematics as well as intricate algorithms. Moreover, we have the ambition to devise an implementation that is not only formally verified, but also reasonably efficient to work on realistic example applications.

We accept these challenges, because a formally verified ODE solver rewards us with unprecedented guarantees about the reliability of the computed bounds.

1.1.3 Application to Smale's 14th Problem

Apart from safety-critical systems, another area where such high guarantees are desired is in mathematics, in the context of computer-assisted proofs. Computer-assisted proofs are proofs that depend on the output of a computer program and therefore crucially on a correct implementation. Computer-assisted proofs have therefore been the target of formal verification in outstanding examples like the Flyspeck project for a formal proof of the Kepler conjecture [55] and well as the formal verification of the four-color theorem [44].

With the verified ODE solver, we target one particular application: Tucker's proof of Smale's 14th problem. This was an important problem, its solution earned Tucker e.g., the

price of the European Mathematical Society [87]. The problem is part of a list of problems for the 21st century, composed by Fields medalist Stephen Smale, among prominent conjectures like the Riemann hypothesis or the question if $P = NP$. Tucker's proof [137, 138, 131] is based on numerical bounds produced by a rigorous ODE solver. On modern hardware, it computes for about 30 hours, so the steps carried out by the program are clearly not surveyable for human reviewers. There have been bugs in the first versions of Tucker's program [140, 136], which illustrates the additional value of formal verification: now we can be certain that the computed bounds are not compromised by potentially remaining bugs.

A short disclaimer might be in order here: it is only the computer-assisted part of Tucker's proof that is verified. That is to say, we formally proved that the numerical bounds used in Tucker's proof are valid, but not that they imply the solution of Smale's 14th problem.

1.2 Contributions

The central contribution of this thesis is the formal verification of a rigorous ODE solver. Fundamental to the verification is the formalization of a significant body of mathematics for ODEs, in particular for the flow of ODEs and the Poincaré map. The main application is the Lorenz attractor and the verification of the computer-assisted part of Tucker's proof.

The verified ODE solver is modular, fundamental for managing its verification. Therefore all parts can be seen as an independent contribution, each useful on its own. Nevertheless, all of the individual parts can be motivated along the lines of Tucker's proof on the Lorenz equations: The Lorenz equations induce a dynamical system, and therefore a notion of flow. Tucker uses a standard technique for the analysis of dynamical systems, a so-called Poincaré map, in his proof. Tucker proves chaos, i.e., sensitive dependence on initial conditions. This is quantified with the derivative of flow and Poincaré map. We verify rigorous numerical algorithms that compute enclosures for all of those entities and apply them to the numerical part of Tucker's proof.

- formalization of flow and Poincaré map
  - continuous and differentiable dependence on initial conditions
  - justification of numerical approximation schemes
- implementation and verification of rigorous numerical algorithms
  - a library for affine arithmetic
  - a rigorous ODE solver,
    based on Runge-Kutta methods and affine arithmetic
  - rigorous computation of the derivative of the flow,
    based on the variational equation
  - rigorous computation of Poincaré maps,
    based on geometric intersection of zonotopes and hyperplanes
- certification of the computer-assisted part of Tucker's proof

Every theorem or lemma that is displayed explicitly as such in this thesis possesses a formal counterpart checked by Isabelle/HOL and is available in the Archive of Formal Proof [76, 73]. We do, however, take the liberty to deviate from the precise formulations for the sake of readability.

1.3 Structure of This Thesis

This thesis is structured in a bottom-up fashion, which resembles the formalization, where everything needs to be built from solid foundations. We therefore start with abstract mathematical foundations and successively build more and more layers on top, until we finally arrive at a verified algorithm that certifies Tucker's proof. The topics of the individual chapters are so diverse that related work is discussed in each chapter individually. It makes sense to read the chapters in order. For a quick overview, one can attempt to start with chapter 7 and look up the necessary background on demand.

The constructions in chapter 3 depend very much on the way mathematics is formalized as described in chapter 2. But later on, we use the results of chapter 3 in a way that is agnostic of the concrete constructions. Similarly for chapter 5, where the internal constructions and proofs are not relevant outside the chapter.

- Chapter 2 presents background on Isabelle/HOL and its library of formalized mathematics, in particular analysis.
- Chapter 3 presents the formalized theory of ODEs and the Poincaré map for the analysis of dynamical systems and also contains the abstract numerical analysis of Runge-Kutta methods.
- Chapter 4 introduces how the abstract numerical scheme is turned into a rigorous numerical method, in particular using affine arithmetic.
- Chapter 5 contains a geometric algorithm for the intersection of zonotopes and hyperplanes.
- Chapter 6 describes the framework used for specifying and verifying high level algorithms operating on safe enclosures.
- Chapter 7 assembles the ingredients of the previous chapters to a verified ODE solver.
- Chapter 8 details on applying the verified ODE solver to the Lorenz attractor.

1.4 Publications

This thesis contains text that has appeared before in scientific papers. These are listed here, joint work is presented with the coauthors' permission.

[75] Fabian Immler and Johannes Hölzl. Numerical analysis of ordinary differential equations in Isabelle/HOL. In Lennart Beringer and Amy Felty, editors, *Interactive Theorem*

Proving: Third International Conference, ITP 2012, Princeton, NJ, USA, August 13-15, 2012. Proceedings, pages 377–392, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN 978-3-642-32347-8. URL https://doi.org/10.1007/978-3-642-32347-8_26

[67] Johannes Hölzl, Fabian Immler, and Brian Huffman. Type classes and filters for mathematical analysis in Isabelle/HOL. In Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, editors, *Interactive Theorem Proving: 4th International Conference, ITP 2013, Rennes, France, July 22-26, 2013. Proceedings*, pages 279–294, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-39634-2. URL https://doi.org/10.1007/978-3-642-39634-2_21

[69] Fabian Immler. Formally verified computation of enclosures of solutions of ordinary differential equations. In Julia M. Badger and Kristin Yvonne Rozier, editors, *NASA Formal Methods: 6th International Symposium, NFM 2014, Houston, TX, USA, April 29 – May 1, 2014. Proceedings*, pages 113–127, Cham, 2014. Springer International Publishing. ISBN 978-3-319-06200-6. URL https://doi.org/10.1007/978-3-319-06200-6_9

[72] Fabian Immler. A verified algorithm for geometric zonotope/hyperplane intersection. In *Proceedings of the 2015 Conference on Certified Programs and Proofs*, CPP '15, pages 129–136, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3296-5. URL http://doi.acm.org/10.1145/2676724.2693164

[71] Fabian Immler. Verified reachability analysis of continuous systems. In Christel Baier and Cesare Tinelli, editors, *Tools and Algorithms for the Construction and Analysis of Systems: 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015, Proceedings*, pages 37–51, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg. ISBN 978-3-662-46681-0. URL https://doi.org/10.1007/978-3-662-46681-0_3

[70] Fabian Immler. A verified enclosure for the Lorenz attractor (rough diamond). In Christian Urban and Xingyuan Zhang, editors, *Interactive Theorem Proving: 6th International Conference, ITP 2015, Nanjing, China, August 24-27, 2015, Proceedings*, pages 221–226, Cham, 2015. Springer International Publishing. ISBN 978-3-319-22102-1. URL https://doi.org/10.1007/978-3-319-22102-1_14

[78] Fabian Immler and Christoph Traut. The flow of ODEs. In Jasmin Christian Blanchette and Stephan Merz, editors, *Interactive Theorem Proving: 7th International Conference, ITP 2016, Nancy, France, August 22-25, 2016, Proceedings*, pages 184–199, Cham, 2016. Springer International Publishing. ISBN 978-3-319-43144-4. URL https://doi.org/10.1007/978-3-319-43144-4_12

[79] Fabian Immler and Christoph Traut. The flow of ODEs: Formalization of variational equation and Poincaré map. *Journal of Automated Reasoning*, 2018. ISSN 1573-0670. URL https://doi.org/10.1007/s10817-018-9449-5

[74] Fabian Immler. A verified ODE solver and the Lorenz attractor. *Journal of Automated Reasoning*, 61(1):73–111, 2018. ISSN 1573-0670. URL https://doi.org/10.1007/s10817-017-9448-y

The source code corresponding to this thesis is available in the Archive of Formal Proof:

[73] Fabian Immler. Affine arithmetic. *Archive of Formal Proofs*, September 2017. ISSN 2150-914x. `http://isa-afp.org/entries/Affine_Arithmetic.shtml`, Formal proof development

[76] Fabian Immler and Johannes Hölzl. Ordinary differential equations. *Archive of Formal Proofs*, September 2017. ISSN 2150-914x. `http://isa-afp.org/entries/Ordinary_Differential_Equations.shtml`, Formal proof development

The paper [75] evolved from a student project of mine, supervised by Johannes Hölzl. The contents corresponding to that project are in section 3.2, but the formalization as it is part of this thesis is much more streamlined. Chapter 2 is largely based on the joint paper [67], notable extensions by me are bounded linear functions (section 2.8.2) and uniform limits (section 2.4.2). The paper [69] presented an ODE solver based on affine arithmetic (section 4.3) and the Euler method. Chapter 5 is based on [72]. Aspects of [71] can be found in chapter 7, but very much streamlined and extended to enclosures for derivatives and a proper notion of Poincaré map. [70] is morally the basis for chapter 8, but has been extended significantly to formal notions of what is being computed, namely derivatives, cone fields, and expansion estimates. The developments of [78] are found in section 3.3. Christoph Traut contributed the proof of lemma 3.32 and theorem 3.33 as part of a student project. The article [79] extends the paper [78] with a formalization of Poincaré map (section 3.4). The article [74] can be seen as a synoptic overview of the results presented in this thesis.

Outside of the scope of this dissertation are the following publications.

[121] Albert Rizaldi, Fabian Immler, and Matthias Althoff. A formally verified checker of the safe distance traffic rules for autonomous vehicles. In Sanjai Rayadurgam and Oksana Tkachuk, editors, *NASA Formal Methods: 8th International Symposium, NFM 2016, Minneapolis, MN, USA, June 7-9, 2016, Proceedings*, pages 175–190, Cham, 2016. Springer International Publishing. ISBN 978-3-319-40648-0. URL `https://doi.org/10.1007/978-3-319-40648-0_14`

[122] Albert Rizaldi, Jonas Keinholz, Monika Huber, Jochen Feldle, Fabian Immler, Matthias Althoff, Eric Hilgendorf, and Tobias Nipkow. Formalising and monitoring traffic rules for autonomous vehicles in Isabelle/HOL. In Nadia Polikarpova and Steve Schneider, editors, *Integrated Formal Methods: 13th International Conference, IFM 2017, Turin, Italy, September 20-22, 2017, Proceedings*, pages 50–66, Cham, 2017. Springer International Publishing. ISBN 978-3-319-66845-1. URL `https://doi.org/10.1007/978-3-319-66845-1_4`

[77] Fabian Immler and Alexander Maletzky. Gröbner bases theory. *Archive of Formal Proofs*, May 2016. ISSN 2150-914x. `http://isa-afp.org/entries/Groebner_Bases.shtml`, Formal proof development

2
Mathematics in Isabelle/HOL

All of the results described in this thesis are formalized, that is, written in a formal language and proved in a machine-checkable calculus. We use the Isabelle theorem prover [116]. Isabelle is a logical framework and Isabelle/HOL [111] the most popular instantiation.

Isabelle follows the tradition of Edinburgh LCF [46] of having a small, trusted kernel. This kernel provides an abstract interface to what we call *formal theorems* and implements primitive logical inference rules. The system ensures that formal theorems can only be constructed by applying the primitive inference rules to existing formal theorems. A formal theorem therefore provides highest levels of mathematical rigor: its validity can be traced back to the very axioms of the underlying logic.

In this chapter we present Isabelle/HOL viewing it as a logic for formalizing mathematics—in the tradition of Whitehead and Russel's Principia Mathematica [143] and as alternative to e.g., Zermelo-Fraenkel set theory. A different view of Isabelle/HOL as a functional programming language with logic is provided by Part I of Concrete Semantics [110].

We will start with basic concepts and our syntactic conventions in section 2.1. A unique feature of Isabelle/HOL among other HOL based theorem provers is its axiomatic type classes. We elaborate on the library for analysis and how it is structured along a hierarchy of type classes (section 2.3). A result of this design is the fact that function spaces are best formalized as types (section 2.8). A central concept in mathematical analysis is that of limits, of which there exist many different notions: limits of sequences, limits when approaching a point, one-sided limits, uniform limits. In Isabelle/HOL, we work with a unified view (as promoted e.g., by Bourbaki [25]): limits of filters, where the filter is used to abstract over the various kinds of limits. This is presented in section 2.4. Limits are necessary for talking about continuity (section 2.5) and, particularly important for differential equations, differentiability and derivatives (section 2.6) as well as integration (section 2.7).

2.1 Isabelle/HOL

The logic on which the developments of this thesis are built on is Isabelle/HOL. This is an extension of Higher Order Logic (HOL)—a logic which has its roots in Church's Simple Theory of Types [32] and distinguishes between *types* and *terms*. Types must not depend on terms (in contrast to Martin-Löf type theory [97]).

Gordon's original HOL system [47] extended Church's Simple Theory of Types with top-level (or Hindley-Milner) polymorphism and type definitions. Isabelle/HOL extends the original HOL with axiomatic type classes [52, 142], which play a crucial role in structuring the library of formalized mathematics in Isabelle/HOL.

Types. Type variables are usually denoted by lower case Greek letters α, β, γ, and so on. Further types are constructed with type constructors like, e.g., the type of functions $\alpha \to \beta$, the type of sets $\alpha\ \mathit{set}$ of elements of type α, or the product type $\alpha \times \beta$.

Terms. For terms, we write $t :: \tau$ to indicate that a term t is of type τ. Terms are made up of constants $\mathit{constant}$, variables $x, y, \ldots$, function abstractions $(\lambda x.\ t)$, or function application.

Function application is written—as is common in functional programming and the λ-calculus—juxtaposition: $f\ t$ denotes the function $f :: \alpha \to \beta$ applied to an argument $t :: \alpha$.

Function abstraction $(\lambda x.\ t) :: \alpha \to \beta$ denotes the (anonymous) function that maps an argument $x :: \alpha$ to some term $t :: \beta$ (which may depend on x). A mathematician would write $(x \mapsto t)$.

Currying. Elements of the product type, pairs, are written $(a, b) :: \alpha \times \beta$. For binary functions, a mathematician usually writes $f\ (a, b)$, for a f applied to two arguments a, b. In Isabelle/HOL, such a notation means that $f :: (\alpha \times \beta) \to \gamma$ maps a product to some value. In functional programming (and Isabelle/HOL), it is more common to use a *Curried* (named after Haskell Curry) representation for functions with several arguments: one would use $g :: \alpha \to \beta \to \gamma$, such that g, when (partially) applied to one argument, yields a function in the second argument: $(g\ a) :: \beta \to \gamma$.

Top-Level Polymorphism. Constants are called polymorphic if their type contains type variables, e.g., the first projection function $\mathit{fst} : (\alpha \times \beta) \to \alpha$ for pairs. Type variables can be instantiated with concrete types, e.g., fst can be applied to a pair of natural numbers such that $\mathit{fst}\,(2 :: \mathbb{N}, 3 :: \mathbb{N}) :: \mathbb{N}$. Type variables are implicitly quantified at the outermost level (being very explicit, we would write $\mathit{fst} :: \forall \alpha\beta.\ (\alpha \times \beta) \to \alpha$). This is a restriction compared to System F [43], where types may be quantified at any level[1].

Type Definition. New types can be introduced with a type definition mechanism. Given a subset $S :: \sigma\ \mathit{set}$ of an existing type σ and a name $\bar{\alpha}\ \mathit{type}$ for the new type constructor depending on a list of types $\bar{\alpha}$, **typedef** declares a new type that is isomorphic to the set S.

$$\textbf{typedef}\ \bar{\alpha}\ \mathit{ty} := S :: \sigma\ \mathit{set}$$

Concepts involving the newly defined type are usually introduced using the package for lifting and transfer by Huffman and Kuncar [68]. With lifting, constants involving the new type are defined in terms of the old type, provided the invariant S is respected. With transfer, proof obligations involving the new type are transferred to obligations on the old type.

Type Classes. The feature that distinguishes Isabelle/HOL from other HOL-based theorem provers is the concept of (axiomatic) type classes [109, 52]. Type classes help to organize polymorphic specifications. A type class C specifies assumptions $P_1, \ldots, P_k$ for constants

[1] The term $\lambda x.\ x\ x$ can be typed in System F with $(\forall \alpha.\ \alpha \to \alpha) \to (\forall \alpha.\ \alpha \to \alpha)$, but is not top-level polymorphic.

$c_1, \ldots, c_m$ (that are to be overloaded) and may be based on other type classes $B_1, \ldots, B_n$. We use the following syntax to describe a type class in Isabelle/HOL:

$$\begin{array}{l}\textsf{class } C = B_1 + B_2 + \ldots + B_n + \\ \quad \textsf{fixes } c_1 :: \alpha\ \kappa_1 \textsf{ and } c_2 :: \alpha\ \kappa_2 \textsf{ and } \quad \cdots \quad \textsf{and } c_m :: \alpha\ \kappa_m \\ \quad \textsf{assumes } P_1 \textsf{ and } P_2 \textsf{ and } \quad \cdots \quad \textsf{and } P_k\end{array}$$

In the type class specification only one type variable, α, is allowed to occur. $\kappa_1, \ldots, \kappa_m$ are (fixed) type constructors. Variables in $P_1, \ldots, P_k$ are implicitly universally quantified. A type α is said to be an instance of the type class C if it provides definitions for the respective constants and respects the required assumptions. In this case we write $\alpha :: C$.

2.2 Base Types

Booleans, numbers, sets and lists are the base types that are most important for the purposes of this thesis.

Booleans. The type $\mathbb{B}$ consist of the two values $\mathit{True} :: \mathbb{B}$ and $\mathit{False} :: \mathbb{B}$. We use the usual logical connectives $\neg, \wedge, \vee, \longrightarrow, \forall, \exists$ for negation, conjunction, disjunction, implication, all-quantification, exists-quantification. Implication binds stronger than universal quantification, i.e., $\forall x.\ P\ x \longrightarrow Q\ x$ is equal to $\forall x.\ (P\ x \longrightarrow Q\ x)$. Bi-implication $\longleftrightarrow$ denotes equality of Booleans.

Sets. A set of elements of type α is of type $\alpha\ \mathit{set}$ The universe of a type α, i.e. the set of all elements of type α, is written $\mathit{UNIV}_\alpha :: \alpha\ \mathit{set}$. We use the usual notation $\in, \subseteq, \setminus$ for set membership, subset relation, set difference. We write set comprehension $\{x \mid P\ x\}$ for the set of all Elements $x :: \alpha$ for which the predicate $P :: \alpha \to \mathbb{B}$ holds. We write the image of a function $f :: \alpha \to \beta$ applied to a set $X :: \alpha\ \mathit{set}$ as $f(X) :: \beta\ \mathit{set}$. This can be ambiguous with regular function application but should be clear from the context.

Lists. A finite list of elements of type α is of type $\alpha\ \mathit{list}$. Lists are constructed inductively from the empty list $[] :: \alpha\ \mathit{list}$ and a constructor $\# :: \alpha \to \alpha\ \mathit{list} \to \alpha\ \mathit{list}$ which prepends an Element to a list. The list of the first three prime numbers is $2\#3\#5\#[]$, and we also use the notation $[2,3,5]$ for better readability. We mostly use the convention of using the suffix s for variables of a list type. E.g., when we use variables x, y, z, or X, Y, Z for elements of type α, we use variables xs, ys, zs, or XS, YS, ZS for elements of type $\alpha\ \mathit{list}$.

Lists have a length $\mathit{length} :: \alpha\ \mathit{list} \to \mathbb{N}$ and the set of elements in a list is denoted by $\mathit{set} :: \alpha\ \mathit{list} \to \alpha\ \mathit{set}$. For an index $i \leq \mathit{length}\ xs$, we denote the selection of the i-th element of the list xs with a subscript xs_i. $\mathit{map} :: (\alpha \to \beta) \to \alpha\ \mathit{list} \to \beta\ \mathit{list}$ takes a function $f :: \alpha \to \beta$ and a list $xs :: \alpha\ \mathit{list}$ and applies the function to every element of the list.

Functional programmers tend to take a recursive view on lists and functions on lists, e.g., $\mathit{map}\ f\ (x\#xs) = (f\ x)\#\mathit{map}\ f\ xs$ and $\mathit{map}\ f\ [] = []$. We will sometimes take a more declarative, extensional view by avoiding the constructors and referring to the elements at the valid indices of the list, e.g, $\forall i < \mathit{length}\ xs.\ (\mathit{map}\ f\ xs)_i = f\ (xs_i)$.

Numbers. In Isabelle/HOL different types are used for different sets of numbers (e.g., $\mathbb{N}$ is not a subset of $\mathbb{R}$), the connections between the different types are maintained with explicit injections.

Natural numbers are the type $\mathbb{N}$, which can be seen as constructed inductively from $0 :: \mathbb{N}$ and $Suc :: \mathbb{N} \to \mathbb{N}$, but we use numeral notations as e.g., $3 = Suc\,(Suc\,(Suc\ 0))$. Integer numbers are the type $\mathbb{Z}$, and real numbers the type $\mathbb{R}$. The injections $int\text{-}of\text{-}nat :: \mathbb{N} \to \mathbb{Z}$, $real\text{-}of\text{-}nat :: \mathbb{N} \to \mathbb{R}$, and $real\text{-}of\text{-}int :: \mathbb{Z} \to \mathbb{R}$ may seem to render formalization cluttered, in practice this is not a big problem because Isabelle uses coercive subtyping [134], which automatically inserts such injections in appropriate places. For example, one could write $\pi + Suc\ 0$ and the system would insert coercions for an internal representation as $\pi + real\text{-}of\text{-}nat\ (Suc\ 0)$.

Operations like addition $+$, subtraction $-$, multiplication $\cdot$ are polymorphic, i.e., they can be used for all number types (and even more), these operations are organized along a hierarchy of type classes as described in the following section 2.3.

2.3 Type Classes for Mathematical Analysis

Many structures in mathematics are organized hierarchically and share operations. For example, a ring $(R, +, \cdot)$ is based on an abelian group $(R, +)$ and a monoid $(R, \cdot)$. Every group shares the symbol $+$ (in additive notation): one uses the same symbol $+$ to denote addition of natural, integer, and real numbers.

In Isabelle, we talk about $+ : \mathbb{N} \to \mathbb{N} \to \mathbb{N}, + : \mathbb{Z} \to \mathbb{Z} \to \mathbb{Z}, + : \mathbb{R} \to \mathbb{R} \to \mathbb{R}$, because of the different types, $+$ denotes different functions and we call such an operation polymorphic. We write $+ :: \alpha \to \alpha \to \alpha$ and assume that α is constrained to type class that provides an addition operation.

Type classes are well suited to exhibit the hierarchical structure of spaces within mathematical analysis and organize polymorphic specifications. In this section, we present the hierarchy of type classes that are useful for mathematical analysis. Figure 2.1 shows the type class hierarchy. We group the type classes into topological, metric, vector and algebraic type classes.

2.3.1 Topological Spaces

A topology captures the notion of *nearness* of elements in a space with the help of so called *open* sets. An open set contains for each element also all elements which are in some sense *near* it. This structure is sufficient to express limits and continuity of functions on topological spaces. For a introduction into topology the reader may look into standard textbooks like [80].

A *topological space* is defined by its predicate of open sets, $open :: \alpha\ set \to \mathbb{B}$. In mathematics the support space, the union of all open sets, is usually explicitly given, whereas in Isabelle/HOL, the support space is the set of all elements of the type, which needs to be open: $open\ UNIV_\alpha$. Open sets are stable under binary intersection $(open\ U \longrightarrow open\ V \longrightarrow open\ (U \cap V))$ and arbitrary union $((\forall U \in S.\ open\ U) \longrightarrow open\ \bigcup S)$. Combining all of

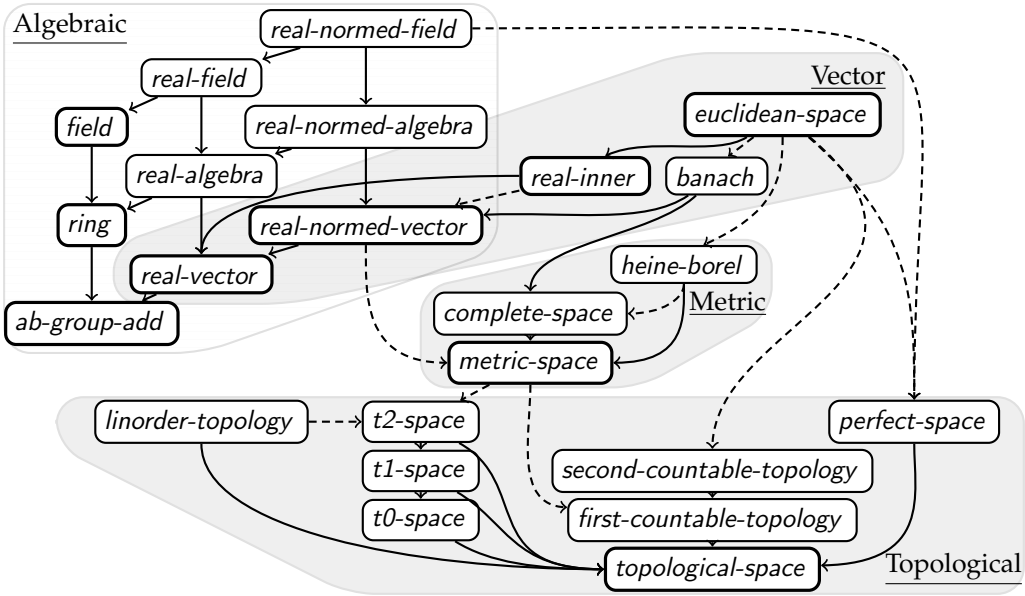


Figure 2.1: Hierarchy of type classes for mathematics [67]. Full lines are inheritance relations and dashed lines are proved subclass relations.

this, a topological space is a type in the following type class:

$$\begin{aligned}&\textbf{class } \textit{topological-space} = \\ &\quad \textbf{fixes } \textit{open} :: \alpha\ \textit{set} \to \mathbb{B} \\ &\quad \textbf{assumes } \textit{open}\ \textit{UNIV}_\alpha \\ &\qquad \textbf{and } \textit{open}\ U \longrightarrow \textit{open}\ V \longrightarrow \textit{open}\ (U \cap V) \\ &\qquad \textbf{and } (\forall U \in S.\ \textit{open}\ U) \longrightarrow \textit{open}\ \bigcup S\end{aligned}$$

In the context of topological spaces, one can define the notion of *closed* sets as the ones with open complement:

$$\begin{aligned}&\textit{closed} :: \alpha\ \textit{set} \to \mathbb{B} \\ &\textit{closed}\ U \longleftrightarrow \textit{open}\ (\textit{UNIV}_\alpha \setminus U)\end{aligned}$$

Refined notions of topological spaces with separation properties are separation spaces *t0-space* , *t1-space* , *t2-space* , and the notion of perfect spaces *perfect-space* . They are discussed in more detail in section 2.3 of the paper [67].

2.3.2 Metric Spaces

Metric spaces are specializations of topological spaces: while topological spaces talk about *nearness*, metric spaces require to explicitly give a *distance* between elements. This distance then induces a notion of *nearness*: a set is open iff for every element in that set, one can give a distance within which every element is *near*, i.e. in the open set. The following type class formalizes open sets induced by a distance:

$$\begin{aligned}&\textbf{class } \textit{open-dist} = \textbf{fixes } \textit{open} :: \alpha\ \textit{set} \to \mathbb{B} \textbf{ and } \textit{dist} :: \alpha \to \alpha \to \mathbb{R} \\ &\quad \textbf{assumes } \textit{open}\ U \longleftrightarrow (\forall x \in U.\ \exists e > 0.\ \forall y.\ \textit{dist}\ x\ y < e \longrightarrow y \in U)\end{aligned}$$

If the distance is a metric, it induces a metric space, and with the *open* sets defined as prescribed by the type class *open-dist*, it is a topological space.

$$\begin{array}{l}\textsf{class } \textit{metric-space} = \textit{open-dist} + \\ \quad \textsf{assumes } \textit{dist}\ x\ y = 0 \longleftrightarrow x = y \textsf{ and } \textit{dist}\ x\ y \leq \textit{dist}\ x\ z + \textit{dist}\ y\ z \\ \textsf{instance } \textit{metric-space} \subseteq \textit{topological-space}\end{array}$$

One aspect that makes real numbers an interesting metric space is the fact that they are *complete*, which means that every sequence where the elements get arbitrarily close converges. Such a sequence is called *Cauchy sequence*, and a metric space is complete iff every Cauchy sequence converges.

$$\begin{array}{l}\textit{Cauchy} :: (\mathbb{N} \to \alpha :: \textit{metric-space}) \to \mathbb{B} \\ \textit{Cauchy}\ X \longleftrightarrow (\forall e > 0.\ \exists M.\ \forall m, n \geq M.\ \textit{dist}\ (X\ m)\ (X\ n) < e) \\ \\ \textit{complete} :: (\alpha :: \textit{metric-space})\ \textit{set} \to \mathbb{B} \\ \textit{complete}\ U \longleftrightarrow (\forall X.\ (\forall i.\ X\ i \in U) \wedge \textit{Cauchy}\ X \longrightarrow \exists x \in U.\ X \longrightarrow x) \\ \\ \textsf{class } \textit{complete-space} = \textit{metric-space} + \textsf{assumes } \textit{complete}\ \textit{UNIV}_\alpha\end{array}$$

2.3.3 Vector Spaces

One aspect that is often abstracted away from products of real numbers is their property of being a vector space, i.e. a space where addition and scaling can be performed. Let us present in this section the definition of vector spaces, normed vector spaces, and how derivatives are generalized for normed vector spaces.

Usually, a vector space is defined on an Abelian group of vectors V, which can be scaled with elements of a field F, and where distributive and compatibility laws need to be satisfied by scaling and addition. The type class based approach restricts the number of type variables to one. Therefore locales (Isabelle's module system for dealing with parametric theories [53]) are used to abstractly reason about vector spaces with arbitrary combinations of F and V (which may be of different types). The instantiation of the field F with the type of real numbers is used as the type class *real-vector* of real vector spaces. The type class *ab-group-add*, which formalizes an Abelian group, provides the operations for addition and additive inverse for the type of vectors α (subtraction is defined in terms of these operations).

$$\begin{array}{l}\textsf{class } \textit{real-vector} = \textit{ab-group-add} + \textsf{fixes } \cdot_R :: \mathbb{R} \to \alpha \to \alpha \\ \quad \textsf{assumes} \quad r \cdot_R (a + b) = r \cdot_R a + r \cdot_R b \quad \textsf{and} \quad (r + q) \cdot_R a = r \cdot_R a + q \cdot_R a \\ \qquad \textsf{and} \quad r \cdot_R (q \cdot_R a) = (r \cdot q) \cdot_R a \qquad \textsf{and} \quad 1 \cdot_R a = a\end{array}$$

We write $\cdot_R$ for scalar multiplication in the vector space to distinguish it from multiplication $\cdot$ on real numbers. For $r, s :: \mathbb{R}$ and $x :: \alpha :: \textit{real-vector}$, we take the liberty to write (scalar) multiplication also as rs, rx, or $r \cdot x$, wherever the precise meaning should be clear from the context.

A generalization of the length of a vector of real numbers is given by the norm in a vector space. The norm induces a distance in a vector space. Similar to *open-dist*, which describes how *dist* induces *open* sets, *dist-norm* formalizes how the norm induces a distance.

$$\begin{array}{l}\textsf{class } \textit{dist-norm} = \textsf{fixes } \textit{norm} :: \alpha \to \mathbb{R} \textsf{ and } - :: \alpha \to \alpha \to \alpha \\ \quad \textsf{assumes } \textit{dist}\ x\ y = \textit{norm}\ (x - y)\end{array}$$

A *normed vector space* is a vector space *real-vector* with a separating and positively scalable norm, for which the triangle equality holds. The distance for the instantiation as metric space and open sets for the topology are induced by *dist-norm* and *open-dist*, respectively. This makes every normed vector space a metric space.

$$\begin{aligned}&\textbf{class } \textit{real-normed-vector} = \textit{real-vector} + \textit{dist-norm} + \textit{open-dist} + \\ &\quad \textbf{assumes} \quad \textit{norm } x = 0 \longleftrightarrow x = 0 \textbf{ and } \textit{norm } (r \cdot_R x) = |r| \cdot \textit{norm } x \\ &\qquad \textbf{and} \quad \textit{norm } (x + y) \leq \textit{norm } x + \textit{norm } y \\ &\textbf{instance } \textit{real-normed-vector} \subseteq \textit{metric-space}\end{aligned}$$

About syntactic conventions, we will also write $\|x\| = \textit{norm } x$ for the norm of x.
Complete normed vector spaces are called Banach spaces.

$$\textbf{class } \textit{banach} = \textit{complete-space} + \textit{real-normed-vector}$$

Further specializations of (normed) vector spaces are available by including multiplication $*$ from a ring *ring* that is compatible with scaling vectors, which yields *real-algebra*, an algebra over the field of real numbers.

$$\begin{aligned}&\textbf{class } \textit{real-algebra} = \textit{real-vector} + \textit{ring} \\ &\quad \textbf{assumes } r \cdot_R x * y = r \cdot_R (x * y)\end{aligned}$$

A *real-algebra* endowed with a sub-multiplicative norm is a *real-normed-algebra*.

$$\begin{aligned}&\textbf{class } \textit{real-normed-algebra} = \textit{real-normed-vector} + \textit{real-algebra} \\ &\quad \textbf{assumes } \|x * y\| \leq \|x\| \cdot \|y\|\end{aligned}$$

2.3.4 Euclidean Space

We consider spaces with Euclidean structure also as a type class. Elements of a space with Euclidean structure consist of finitely many real valued coordinates.

A first abstraction towards this notion is given by a normed vector space with an *inner product*. While the norm can be interpreted as the length of a vector, the inner product can be used to describe the angle between two vectors together with their lengths (the cosine of the angle is the inner product divided by the product of the lengths). *dist-norm* and *open-dist* specify the induced metric and topology. The inner product is used to induce a norm $\|x\| = \sqrt{x \bullet x}$. An inner product is a commutative bilinear operation $\bullet$ on vectors, for which $0 \leq x \bullet x$ holds with equality iff $x = 0$.

$$\begin{aligned}&\textbf{class } \textit{real-inner} = \textit{real-vector} + \textit{dist-norm} + \textit{open-dist} + \\ &\quad \textbf{fixes } \bullet :: \alpha \to \alpha \to \mathbb{R} \\ &\quad \textbf{assumes} \quad \|x\| = \sqrt{x \bullet x} && \textbf{and } x \bullet y = y \bullet x \\ &\qquad \textbf{and} \quad (x + y) \bullet z = x \bullet z + y \bullet z && \textbf{and } (r \cdot_R x) \bullet y = r \cdot_R (x \bullet y) \\ &\qquad \textbf{and} \quad 0 \leq x \bullet x && \textbf{and } x \bullet x = 0 \longleftrightarrow x = 0 \\ &\textbf{instance } \textit{real-inner} \subseteq \textit{real-normed-vector}\end{aligned}$$

The structure of Euclidean space can be captured with an inner product and a finite coordinate basis. A coordinate Basis is a finite set of orthogonal (i.e., their inner product

equals zero) vectors, each of length 1. Moreover, the zero vector is characterized by zero "coordinates" with respect to the basis.

class *euclidean-space* = *real-inner* +
  fixes *Basis* :: α *set*
  assumes *finite Basis* **and** *Basis* $\neq \emptyset$ **and** $(\forall u \in \mathit{Basis}.\ x \bullet u = 0) \longleftrightarrow x = 0$
    and $u \in \mathit{Basis} \longrightarrow v \in \mathit{Basis} \longrightarrow u \bullet v =$ **if** $u = v$ **then** 1 **else** 0
instance *euclidean-space* $\subseteq$ *perfect-space*, *second-countable-topology*,
  banach, *heine-borel*

Any Euclidean space is a Banach space with a perfect second countable topology and satisfies the Heine-Borel property (closed bounded sets are exactly the compact sets).

Concrete instances are real numbers $\mathbb{R}$, Complex numbers $\mathbb{C}$, and any (binary or finite) product of Euclidean spaces is a Euclidean space.

When $b \in \mathit{Basis}$, we also write x_b for the projection of x to b, i.e., $x_b := x \bullet b$. We write $[x; z] := \{y \mid \forall b \in \mathit{Basis}.\ x_b \leq y_b \wedge y_b \leq z_b\}$ :: α :: *euclidean-space set* for hyperrectangles (or boxes) in Euclidean spaces (for the instance $\mathbb{R}$, those are the closed intervals). Beware a potential confusion with two-element lists $[a, b]$. The infimum inf :: $\alpha \to \alpha \to \alpha$ is defined componentwise $(\inf\ x\ y) \bullet i = \min(x \bullet i)(y \bullet i)$, analogously the supremum sup. An order $x \leq y \longleftrightarrow (\forall b \in \mathit{Basis}.\ x_b \leq y_b)$ is also defined componentwise.

For computability, it is important that elements of *euclidean-space* can be represented unambiguously. One can fix an order on the basis vectors in a class *executable-euclidean-space*

class *executable-euclidean-space* = *euclidean-space* +
  fixes *Basis-list* :: α *list*
  assumes *set Basis-list* = *Basis*
    and *distinct Basis-list*

Elements x :: α :: *executable-euclidean-space* can be interpreted as lists of real numbers with the help of *eucl-of-list* : $\mathbb{R}$ *list* $\to \alpha$ and *list-of-eucl* : $\alpha \to \mathbb{R}$ *list*. We therefore have

$$\mathit{eucl\text{-}of\text{-}list}\ (\mathit{list\text{-}of\text{-}eucl}\ x) = x$$

and if the dimension of the spaces (i.e., the cardinality of the set *Basis*) match the lengths of the involved lists, we have the inverse:

$$\mathit{length}\ xs = |\mathit{Basis} :: \alpha\ \mathit{set}| \longrightarrow \mathit{list\text{-}of\text{-}eucl}\,(\mathit{eucl\text{-}of\text{-}list}\,xs :: \alpha)$$

In the context of *executable-euclidean-space*, we also write x_i if $i \in \mathbb{N}$ and $i <$ *length Basis-list* to denote the i-th coordinate of x.

$$x_i := x_{\mathit{Basis\text{-}list}_i} = x \bullet (\mathit{Basis\text{-}list}_i)$$

2.4 Filters and Limits

Filters are a useful concept in topology—promoted e.g., by Bourbaki [25]—because they help to unify various kinds of limits and convergence, including limits of sequences, limits

of functions at a point, one-sided, asymptotic, or uniform limits. A *filter* is a set of sets (or equivalently a predicate on predicates) with a certain order structure.

The intuition is that many varieties of logical quantification are filters, such as "for all x in set A"; "for sufficiently large n"; "for all but finitely many x"; "for x sufficiently close to y". These quantifiers are similar to the ordinary universal quantifier ($\forall$) in many ways. In particular, each holds for the always-true predicate, preserves conjunction, and is monotonic:

$$(\square x.\ \mathit{True})$$
$$(\square x.\ P\ x) \longrightarrow (\square x.\ Q\ x) \longrightarrow (\square x.\ P\ x \wedge Q\ x)$$
$$(\forall x.\ P\ x \longrightarrow Q\ x) \longrightarrow (\square x.\ P\ x) \longrightarrow (\square x.\ Q\ x)$$

In Isabelle/HOL, a filter $\mathcal{F}$ is defined as a predicate on predicates that satisfies all three of the above rules. (Filters are not required to be *proper*; that is, we admit the trivial filter "for all x in $\{\}$" which holds for all predicates, including $\lambda x.\ \mathit{False}$.)

$$\begin{aligned}
&\mathit{is\text{-}filter} :: ((\alpha \to \mathbb{B}) \to \mathbb{B}) \to \mathbb{B}\\
&\mathit{is\text{-}filter}\ \mathcal{F} =\\
&\quad \mathcal{F}\ (\lambda x.\mathit{True}) \wedge\\
&\quad (\forall P, Q.\ \mathcal{F}\ (\lambda x.\ P\ x) \longrightarrow \mathcal{F}\ (\lambda x.\ Q\ x) \longrightarrow \mathcal{F}\ (\lambda x.\ P\ x \wedge Q\ x)) \wedge\\
&\quad (\forall P, Q.\ (\forall x.\ P\ x \longrightarrow Q\ x) \longrightarrow \mathcal{F}\ (\lambda x.\ P\ x) \longrightarrow \mathcal{F}\ (\lambda x.\ Q\ x))
\end{aligned}$$

This notion is abstracted in a type $\alpha\ \mathit{filter}$ comprising all filters over the type α.

$$\textbf{typedef}\ \alpha\ \mathit{filter} = \{\mathcal{F} \mid \mathit{is\text{-}filter}\ \mathcal{F}\}$$

We use filter quantifiers $\forall_F x.\ P\ x$ to express that a predicate P holds under the filter $F :: \alpha\ \mathit{filter}$.

For a filter $F :: \alpha\ \mathit{filter}$ and its corresponding raw representation $\mathcal{F} :: (\alpha \to \mathbb{B}) \to \mathbb{B}$, we will usually show only its characteristic equation $\forall_F x.\ P\ x \longleftrightarrow \mathcal{F}\ P$ and leave the raw definition (in terms of the isomorphism between $\alpha\ \mathit{filter}$ and $(\alpha \to \mathbb{B}) \to \mathbb{B}$) and the proof obligation $\mathit{is\text{-}filter}\ \mathcal{F}$ implicit.

Filters are equipped with a *finer than* ordering, which is defined accordingly:

$$F_1 \leq F_2 \longleftrightarrow \forall P.\ (\forall_{F_2} x.\ P\ x) \longrightarrow (\forall_{F_1} x.\ P\ x)$$

With this ordering, $\alpha\ \mathit{filter}$ is a complete lattice, where the bottom element is the trivial filter and the top element the filter corresponding to $\forall$-quantification.

2.4.1 Basic filters.

The *principal filter* of a set B represents a bounded quantifier, i.e. "for all x in B".

$$\forall_{\mathit{principal}\ S} x.\ P\ x \longleftrightarrow (\forall x \in S.\ P\ x)$$

On any linearly ordered type $\alpha :: \mathit{linorder}$, the filter $\mathit{at\text{-}top} :: \alpha\ \mathit{filter}$ means "for sufficiently large y" or "as $y \longrightarrow +\infty$", and $\mathit{at\text{-}bot} :: \alpha\ \mathit{filter}$ as "for sufficiently small y" or "as $y \longrightarrow -\infty$". $\mathit{sequentially} :: \mathbb{N}\ \mathit{filter}$ as an abbreviation for $\mathit{at\text{-}top}$ as a filter on the natural numbers.

$$(\forall_{\mathit{at\text{-}top}} x.\ P\ x) \longleftrightarrow (\exists x.\ \forall y \geq x.\ P\ y)$$
$$(\forall_{\mathit{sequentially}} n.\ P\ n) \longleftrightarrow (\exists n_0.\ \forall n \geq n_0.\ P\ y)$$
$$(\forall_{\mathit{at\text{-}bot}} x.\ P\ x) \longleftrightarrow (\exists x.\ \forall y \leq x.\ P\ y)$$

In the context of a topological space, the neighborhood filter *nhds* x is the filter that means "for all y in some open neighborhood of x".

$$(\forall_{\textit{nhds } x}.\ P\ x) \longleftrightarrow (\exists U.\ \textit{open}\ x \wedge x \in U \wedge (\forall y \in U.\ P\ y))$$

The principal filter can be used to construct refinements of the neighborhood filter. The *punctured* neighborhood filter *at* x *within* U means "for all $y \in U$ and $y \neq x$ in some neighborhood of x". We also use one-sided filters *at-left* and *at-right*. *at* x is an abbreviation for *at* x *within* $\textit{UNIV}_\alpha$. $F_1 \sqcap F_2$ is the infimum of the filters F_1 and F_2. For functions, one can define a filter that expresses that functions are uniformly close:

$$\begin{array}{lcl}
\textit{at } \square \textit{ within } \square & :: & (\alpha :: \textit{topological-space}) \to \alpha\ \textit{set} \to \alpha\ \textit{filter} \\
\textit{at-left}, \textit{at-right} & :: & (\alpha :: \textit{linorder-topology}) \to \alpha\ \textit{filter} \\
\textit{uniformly-on} & :: & \alpha\ \textit{set} \to (\alpha \to \beta :: \textit{metric-space}) \to \alpha \to \beta\ \textit{filter} \\
\textit{at } x \textit{ within } U & = & \textit{nhds } x \sqcap \textit{principal } (U \setminus \{x\}) \\
\textit{at-left } x & = & \textit{at } x \textit{ within }]\infty, x[\\
\textit{at-right } x & = & \textit{at } x \textit{ within }]x, \infty[\\
\textit{uniformly-on } S\ l & = & \bigsqcap_{0<\delta} \textit{principal}\{f \mid \forall x \in S.\ \textit{dist}\,(f\ x)\ (l\ x) < \delta\}
\end{array}$$

2.4.2 Limits.

Filters can be used to express a general notion of limits. To illustrate this, we start with the usual epsilon-delta definitions of limits of functions and sequences on reals, and then incrementally generalize the definitions. Finally we end up with a single definition, parameterized over two filters, that can express diverse kinds of limits in arbitrary topological spaces. Here are the usual epsilon-delta definitions of limits for sequences y_n and for a function f at a point a.

$$\begin{aligned}
(y_n \longrightarrow L) &= (\forall \epsilon > 0.\ \exists n_0.\ \forall n \geq n_0.\ \|y_n - L\| < \epsilon) \\
(\lim_{x \to a} f(x) = L) &= (\forall \epsilon > 0.\ \exists \delta > 0.\ \forall x.\ 0 < \|x - a\| < \delta \longrightarrow \|f(x) - L\| < \epsilon)
\end{aligned}$$

The reader may recognize "$\exists n_0.\ \forall n \geq n_0$" as the filter *sequentially*. Also note that "$\exists \delta > 0.\ \forall x.\ 0 < \|x - a\| < \delta$" is equivalent to the punctured neighborhood filter $(\textit{at } a)$. Therefore we can rewrite the above definitions as follows.

$$\begin{aligned}
(y_n \longrightarrow L) &= (\forall \epsilon > 0.\ \forall_{\textit{sequentially}}\ n.\ \|y_n - L\| < \epsilon) \\
(\lim_{x \to a} f(x) = L) &= (\forall \epsilon > 0.\ \forall_{\textit{at } a}\ x.\ \|f(x) - L\| < \epsilon)
\end{aligned}$$

Here we can generalize those two definitions by parameterizing over the filter. We write $(f \longrightarrow L)\ F$ to denote that f tends to L under the filter F:

$$(f \longrightarrow L)\ F = (\forall \epsilon > 0.\ \forall_F\ x.\ |f(x) - L| < \epsilon) \tag{2.1}$$

This yields a general notion of limit: by instantiating F with various filters, we can express many different kinds of limits: *sequentially* for sequences, *at* a for a function at a point,

at-top or *at-bot* for a function at $\pm\infty$, *at-left* a or *at-right* a for one-sided limits.

$$(x_n \longrightarrow L) = (x \longrightarrow L)\ \textit{sequentially}$$
$$(\lim_{x\to a} f(x) = L) = (f \longrightarrow L)\ (\textit{at}\ a)$$
$$(\lim_{x\to a^+} f(x) = L) = (f \longrightarrow L)\ (\textit{at-right}\ a)$$
$$(\lim_{x\to -\infty} f(x) = L) = (f \longrightarrow L)\ \textit{at-bot}$$

Generalized Limit. All of the previous notions talk about limits approaching a point L. Even this can be generalized to, as we will see, e.g., infinities or functions (in a uniform limit). First of all, note that when applying a function to the argument of each predicate in a filter, one gets a filter again. This is expressed by *filtermap* $f\ F$, which transforms the filter F with a function f.

$$\forall_{(\textit{filtermap}\ f\ F)}\ x.\ P\ x \longleftrightarrow \forall_F\ x.\ P\ (f\ x)$$

filtermap can be used to rewrite (2.1):

$$(f \longrightarrow L)\ F = (\forall \epsilon > 0.\ \forall_{(\textit{filtermap}\ f\ F)}\ y.\ |y - L| < \epsilon)$$

But this is saying that *filtermap* $f\ F$ is eventually in every open neighborhood of L, which is equivalent to the following:

$$(f \longrightarrow L)\ F = (\textit{filtermap}\ f\ F \leq \textit{nhds}\ L)$$

Finally, we can generalize *nhds* L to an arbitrary filter G and obtain a generalized limit *filterlim* $f\ F\ G$, which is used to define the *tendsto* relation and sequential limit:

$$\begin{array}{lcl}
\textit{filterlim} & :: & (\alpha \to \beta) \to \alpha\ \textit{filter} \to \beta\ \textit{filter} \to \mathbb{B} \\
\textit{filterlim}\ f\ F\ G & \longleftrightarrow & \textit{filtermap}\ f\ F \leq G \\
(\square \longrightarrow \square)\ \square & :: & (\alpha \to \beta) \to (\beta :: \textit{topological-space}) \to \alpha\ \textit{filter} \to \mathbb{B} \\
(f \longrightarrow L)\ F & \longleftrightarrow & \textit{filterlim}\ f\ F\ (\textit{nhds}\ L) \\
\square \longrightarrow \square & :: & (\mathbb{N} \to \alpha) \to (\alpha :: \textit{topological-space}) \to \mathbb{B} \\
X \longrightarrow L & \longleftrightarrow & (X \longrightarrow L)\ \textit{sequentially}
\end{array}$$

This generalized notion of limit is only based on filters and does not even require topologies. This can be used in e.g., *filterlim* $(\lambda x.\ -x)$ *at-bot at-top* to express that that $-x$ goes to positive infinity as x approaches negative infinity.

For *filterlim* one can provide a composition rule for convergence. Further rules about e.g. elementary functions are available for normed vector spaces.

$$\textit{filterlim}\ f\ F_1\ F_2 \longrightarrow \textit{filterlim}\ g\ F_2\ F_3 \longrightarrow \textit{filterlim}\ (g \circ f)\ F_1\ F_3$$

A small example to illustrate where such a formalization enables compositional reasoning is to prove

$$((\lambda x.\ \textit{exp}\ (-1/x)) \longrightarrow 0)\ (\textit{at-right}\ 0)$$

by composing the rules

- *filterlim* $(\lambda x.\ \frac{1}{x})$ (*at-right* 0) *at-top*.
- *filterlim* $(\lambda x.\ -x)$ *at-top* *at-bot*, and
- $(exp \longrightarrow 0)$ *at-bot*,

Uniform Limit The generalized limit *filterlim* is sufficiently general to capture the notion of uniform limit, as well. For this, we use the *uniformly-on X l* filter, a filter that quantifies over functions $f :: \alpha \to \beta ::$ *metric-space* that are (on a domain X) uniformly close to a function l:

$$(\forall_{(\textit{uniformly-on } X\ l)} f.\ P\ f) \longleftrightarrow (\exists \varepsilon > 0.\ \forall f.\ (\forall t \in T.\ \textit{dist}\ (f\ x)\ (l\ x) < \varepsilon) \longrightarrow P\ f)$$

With the help of this filter, we can describe uniform convergence:

$$\textit{uniform-limit}\ X\ f\ l\ F \longleftrightarrow \textit{filterlim}\ f\ F\ (\textit{uniformly-on}\ X\ l)$$

which is usually defined as follows for a sequence of functions f_n.

$$(f_n \xrightarrow{\text{uniformly}} l) \longleftrightarrow (\forall \varepsilon > 0.\ \exists N.\ \forall n \geq N.\ \forall x \in X.\ |f_n\ x - l\ x| < \varepsilon)$$

When instantiating F with the *sequentially* filter, one gets the usual definition of uniform convergence for a sequence of functions $f_n : \mathbb{N} \to \alpha$ or a family of functions f_y as y approaches z:

$$\begin{aligned}&\textit{uniform-limit}\ X\ f\ l\ \textit{sequentially} :=\\ &\quad \forall \varepsilon > 0.\ \exists N.\ \forall x \in X.\ \forall n \geq N.\ \|f_n\ x - l\ x\| < \varepsilon\end{aligned}$$

$$\begin{aligned}&\textit{uniform-limit}\ X\ f\ l\ (\textit{at}\ z) :=\\ &\quad \forall \varepsilon > 0.\ \exists \delta > 0.\quad \forall y.\ \|y - z\| < \delta \longrightarrow (\forall x \in X.\ \textit{dist}\ (f_y\ x)\ (l\ x) < \varepsilon)\end{aligned}$$

The advantage of the filter approach is that many important lemmas can be expressed for arbitrary filters, for example the uniform limit theorem, which states that the uniform limit l of a family of continuous functions f_y is continuous.

Theorem 2.1 (Uniform Limit Theorem)**.**

$$(\forall_F y.\ \textit{continuous-on}\ X\ f_y) \longrightarrow \textit{uniform-limit}\ X\ f\ l\ F \longrightarrow \textit{continuous-on}\ X\ l$$

A frequently used criterion to show that a series of functions converges uniformly is the Weierstrass M-test. Assuming majorants M_n for the functions f_n and assuming that the series of majorants converges, it allows to deduce uniform convergence of the partial sums towards the series.

Lemma 2.2 (Weierstrass M-Test)**.**

$$\begin{aligned}&\forall n.\ \forall x \in X.\ \|f_n\ x\| \leq M_n \longrightarrow \sum_{n \in \mathbb{N}} M_n < \infty \longrightarrow\\ &\textit{uniform-limit}\ X\ (\lambda n\ x.\ \sum_{i \leq n} f_i\ x)\ (\lambda x.\ \sum_{i \in \mathbb{N}} f_i\ x)\ \textit{sequentially}\end{aligned}$$

2.5 Continuity

Filters are also used to generalize notions of continuity, e.g., continuity at a point, continuity from the left, or continuity from the right. *Continuity* of a function f at a filter F says that the function converges on F towards its value $f\ a$ while F converges to a.

$$\begin{array}{lcl} \mathit{continuous} & :: & \alpha\ \mathit{filter} \to (\alpha \to \beta) \to \mathbb{B} \\ \mathit{continuous}\ F\ f & \longleftrightarrow & \exists a.\ ((\lambda x.\ x) \longrightarrow a)\ F \wedge (f \longrightarrow f\ a)\ F \end{array}$$

Often times, a function needs to be continuous not only at a point, but on a set. For this one uses *continuous-on*: for every point x in the set S, f converges to $f\ x$ when approaching the limit from within S.

$$\begin{array}{l} \mathit{continuous\text{-}on}\ ::\ \alpha\ \mathit{set} \to (\alpha :: \mathit{topological\text{-}space} \to \beta :: \mathit{topological\text{-}space}) \to \mathbb{B} \\ \mathit{continuous\text{-}on}\ S\ f \longleftrightarrow \forall x \in S.\ (f \longrightarrow f\ x)\ (\mathit{at}\ x\ \mathit{within}\ S) \end{array}$$

2.6 Derivatives

The main topic of this work is ordinary differential equations, it is therefore important to formally reason about derivatives. We use two different (but related) concepts: the ordinary (vector) derivative and the total (or Fréchet) derivative.

Total Derivative. The total derivative (or Fréchet derivative) is a generalization of the ordinary derivative (of functions $\mathbb{R} \to \mathbb{R}$) for arbitrary normed vector spaces. To illustrate this generalization, recall that the ordinary derivative denotes the slope of a function: the derivative of f at x equals m, iff

$$\lim_{h \to 0} \frac{f(x+h) - f(x)}{h} = m \tag{2.2}$$

Moving the m under the limit, one sees that the function $\lambda h.\ h \cdot m$ is a linear approximation for the difference of the function value at nearby points x and $x+h$:

$$\lim_{h \to 0} \frac{f(x+h) - f(x) - h \cdot m}{h} = 0$$

This concept can be generalized by replacing $\lambda h.\ h \cdot m$ with an arbitrary (bounded) linear function A. In the following equation, A is a linear function.

$$\lim_{h \to 0} \frac{f(x+h) - f(x) - A\ h}{\|h\|} = 0 \tag{2.3}$$

Note that in the previous equation, we can drop many of the restrictions on the type of f. We started with $f : \mathbb{R} \to \mathbb{R}$ in equation 2.2, but the last equation still makes sense for $f : \alpha \to \beta$ for normed vector spaces α, β. We call $A : \alpha \to \beta$ the total derivative $\mathrm{D}f$ of f at a point x. It is formalized with the predicate *has-derivative*, which is formalized with a filter to parametrize the way h tends to zero in equation 2.3.

Definition 2.3 (Total Derivative)**.** *For*
$f :: \alpha :: \textit{real-normed-vector} \to \beta :: \textit{real-normed-vector}$,
$A :: \alpha \to \beta$, $x :: \alpha$, *and* $S :: \alpha\ \textit{set}$

$$(f\ \textit{has-derivative}\ A)\ (\textit{at}\ x\ \textit{within}\ X) \longleftrightarrow \left(\left(\lambda y.\ \frac{\|f\ y - f\ x - A\ (y - x)\|}{\|y - x\|}\right) \longrightarrow 0\right)\ (\textit{at}\ x\ \textit{within}\ X)$$

For ease of notation, whenever there is an assumption $(f\ \textit{has-derivative}\ A)\ F$ in the formalization, we write, close to common mathematical notation, $\mathsf{D}f|_F$, or even $\mathsf{D}f|_x$ instead of $\mathsf{D}f|_{\textit{at}\ x}$ or $\mathsf{D}f|_{\textit{at}\ x\ \textit{within}\ X}$.

Ordinary Derivative For functions $x :: \mathbb{R} \to \alpha$ in one real variable, the total derivative is a linear *function* $\mathbb{R} \to \alpha$ that approximates x at a given point. In contrast to that, the ordinary derivative gives the rate of change (as a value of type α) at the given point. The ordinary derivative of a vector valued function $x :: \mathbb{R} \to \alpha$ is formalized as *has-vderivative* (the *v* stands for "vector") based on the total derivative *has-derivative*:

Definition 2.4 (Ordinary Derivative)**.** *For*
$x :: \mathbb{R} \to \alpha :: \textit{real-normed-vector}$,
$x' :: \alpha$, $t : \mathbb{R}$, *and* $T : \mathbb{R}\ \textit{set}$

$$(x\ \textit{has-vderivative}\ x')\ (\textit{at}\ t\ \textit{within}\ T) \longleftrightarrow (x\ \textit{has-derivative}\ (\lambda d.\ d \cdot x'))\ (\textit{at}\ t\ \textit{within}\ T)$$

Similarly to the total derivative, we also adopt a special notation whenever there is an assumption $(x\ \textit{has-vderivative}\ x')\ (\textit{at}\ t\ \textit{within}\ T)$. We then write the following (and leave the domain T of approaching the limit implicit):

$$\dot{x}(t) = x'$$

Similar to *continuous* at a point and *continuous-on* on a set, we also provide a notion *has-vderiv-on* for the derivative on a set. In contrast to definition 2.4, the derivative x' here is a function of the point in time where the derivative is taken.

Definition 2.5 (Ordinary Derivative on a Set)**.** *For*
$x :: \mathbb{R} \to \alpha :: \textit{real-normed-vector}$,
$x' :: \mathbb{R} \to \alpha$ *and* $T : \mathbb{R}\ \textit{set}$

$$(x\ \textit{has-vderiv-on}\ x')\ T \longleftrightarrow (\forall t \in T.\ (x\ \textit{has-vderivative}\ x'\ t)\ (\textit{at}\ t\ \textit{within}\ T))$$

2.7 Integrals

Integrals are defined for functions from Euclidean space to Banach space. A Banach space is a complete normed vector space.

$$\textbf{class}\ \textit{banach} = \textit{real-normed-vector} + \textit{complete-space}$$

The formalization includes properties like linearity, monotone and dominated convergence, as well as the fundamental theorem of calculus. For an integrand $f :: \alpha :: \textit{euclidean-space} \to \beta :: \textit{banach}$, we denote with $\int_a^b f\ x\,\mathrm{d}x$ the integral of f over the box $[a; b]$.

2.8 Function Spaces as Types

Type classes make specifications and reasoning about common spaces generic: The original motivation was to generalize from $\mathbb{R}^n$ to isomorphic spaces without explicitly reasoning about the isomorphism. Some examples include $\mathbb{R}$ (which is isomorphic to $\mathbb{R}^1$), $\mathbb{C}$ (which is isomorphic to $\mathbb{R}^2$), or tuples of e.g., type $\mathbb{R} \times \mathbb{R} \times \mathbb{R}$ (which is isomorphic to $\mathbb{R}^3$).

It turned out, however, that the fine-grained hierarchy of type classes also supports functional analysis, i.e., analysis of spaces whose elements are functions. The approach is therefore to define types for function spaces and instantiate type classes accordingly. A concrete example where this is necessary is the case of bounded continuous functions and its role in the proof of the Picard-Lindelöf theorem.

2.8.1 Bounded Continuous Function

We motivate bounded continuous functions with the Picard-Lindelöf theorem, which guarantees the existence of a unique solution to an initial value problem. For an ODE f with initial value x_0 at time t_0, a unique solution on the time interval $[t_0; t_1]$ is constructed by considering iterations of the following operator for continuous functions $\phi : [t_0; t_1] \to \mathbb{R}^n$:

$$P(\phi) := \left(\lambda t.\ x_0 + \int_{t_0}^{t} f(\tau, \phi(t))\, \mathrm{d}\tau\right)$$

From a mathematician's point of view, P operates on the Banach space of continuous functions on the compact domain $[t_0; t_1]$ and therefore the Banach fixed point theorem guarantees the existence of a unique fixed point (which is by construction the unique solution).

In order to formalize this in Isabelle/HOL, there are two obstructions to overcome: First, the concept of Banach space is a type class in Isabelle/HOL, we therefore need to introduce a type for the mappings $\phi : [t_0; t_1] \to \mathbb{R}^n$ from above. But this poses the second problem: functions in Isabelle/HOL are total and types must not depend on term parameters like t_0 and t_1.

We work around these restrictions by introducing a type of *bounded* continuous functions, which is a Banach space and comprises (with a suitable choice of representations) every continuous functions on all compact domains.

$$\textbf{typedef}\ \alpha \to_{bc} \beta := \{f :: \alpha \to \beta \mid f \text{ continuous on } \alpha \wedge (\exists B.\ \forall t.\ \|f\ t\| \leq B)\}$$

Functions on a given compact domain are then encoded (see *ext-cont* in section 3.2.1) with suitable representatives.

In order to define operations on type $\alpha \to_{bc} \beta$, the Lifting and Transfer package [68] is an essential tool: operations on the plain function type $\alpha \to \beta$ are automatically lifted to definitions on the type $\alpha \to_{bc} \beta$ when supplied with a proof that functions in the result are bounded continuous under the assumption that argument functions are bounded continuous. We write application of a bounded continuous function $f : \alpha \to_{bc} \beta$ with an element $x : \alpha$ as follows.

Definition 2.6 (Application of Bounded Continuous Functions)**.**

$$(f \cdot_{bc} x) :: \beta$$

The norm on $\alpha \to_{bc} \beta$ is the supremum of the range and the vector space operations $+, \cdot$ are defined pointwise.

Definition 2.7 (Normed Vector Space of Bounded Continuous Functions)**.**

$$\begin{array}{rcl} \|f\| & := & \sup \{\|f \cdot_{bc} x\| \mid x \in \mathit{UNIV}_\alpha\} \\ (f+g) \cdot_{bc} x & := & f \cdot_{bc} x + g \cdot_{bc} x \\ (a \cdot f) \cdot_{bc} x & := & a \cdot (f \cdot_{bc} x)) \end{array}$$

The type $\to_{bc}$ with the above operations forms a complete normed vector space (a Banach space). This allows us to use the Banach fixed point theorem for operators on this type.

2.8.2 Bounded Linear Functions

Similar to the type of bounded continuous functions, we also introduce a type of bounded *linear* functions (also known as continuous linear functions), but for a different notion of "bounded".

For vector spaces α and β, a linear function is a function $f : \alpha \to \beta$ that is compatible with addition and scalar multiplication.

$$\mathit{linear}\ f := \forall x\, y\, c.\ f(c \cdot x + y) = c \cdot f(x) + f(y)$$

Let us assume normed vector spaces, i.e., α,β :: *normed-vector-space*. Linear functions are continuous if the norm of the result is linearly bounded by the norm of the argument. We cast bounded linear functions $\alpha \to \beta$ as a type $\alpha \to_{bl} \beta$ in order to make it an instance of Banach space.

$$\textbf{typedef}\ \alpha \to_{bl} \beta := \{f :: \alpha \to \beta \mid \mathit{linear}\ f \wedge \exists K.\ \forall x.\ \|f(x)\| \leq K\|x\|\}$$

The construction is very similar to bounded continuous functions. We write bounded linear function application $(f \cdot_{bl} x)$. For simplicity of notation, will often write f instead of $(\lambda x.\ f \cdot_{bl} x)$, which is done in Isabelle/HOL as well with the help of coercions [134]. Vector space operations are also analogous to $\to_{bc}$ defined pointwise. The usual choice of a norm for bounded linear functions is the operator norm: the maximum of the image of the bounded linear function on the unit ball. With this norm, $\alpha \to_{bl} \beta$ forms a normed vector space and we prove that it is Banach if α and β are Banach.

Definition 2.8 (Norm in Banach Space $\to_{bl}$)**.** *For $f :: \alpha \to_{bl} \beta$,*

$$\|f\| := \max \{\|f \cdot_{bl} y\| \mid \|y\| \leq 1\}$$

Having (bounded) linear functions as a separate type makes many formulations easier. For example, consider Harrison's formalization of multivariate analysis in HOL Light

(from which Isabelle/HOL's analysis descended). In Harrison's formalization, continuity is formalized for functions f of type $\mathbb{R}^n \to \mathbb{R}^m$.

$$(\mathit{continuous}\ f\ (\mathit{at}\ x)) = (\forall e > 0.\ \exists d > 0.\ \forall y.\ \|x - y\| < d \longrightarrow \|f\ x - f\ y\| < e)$$

Most of Harrison's formalization is geared towards viewing derivatives as linear functions of type $\mathbb{R}^n \to \mathbb{R}^m$. For continuously differentiable functions, one therefore needs to reason about functions $f' : \mathbb{R}^n \to (\mathbb{R}^n \to \mathbb{R}^m)$, where $f'\ x$ is the derivative of f at a point x. Continuity of f' is written in an explicit ε-δ form and involves the operator norm $\mathit{onorm} : (\mathbb{R}^n \to \mathbb{R}^m) \to \mathbb{R}$, which is quite verbose:

$$(\forall e > 0.\ \exists d > 0.\ \forall y.\ \|x - y\| < d \longrightarrow \mathit{onorm}\,(\lambda v.\ f'\ x\ v - f'\ y\ v) < e))$$

The ε-δ form could of course be captured in a separate definition, but this would be very similar to the definition of continuity and would introduce redundancy.

In the Isabelle/HOL formalization, *continuous* is defined for functions $f :: \alpha \to \beta$ for topological spaces α and β. If α and β are normed vector spaces, the above equality for *continuous* holds in Isabelle/HOL, too. And indeed, the norm of bounded linear functions is defined using *onorm* such that $\mathit{onorm}\,(\lambda v.\ (f'\ x) \cdot_{bl} v - (f'\ y) \cdot_{bl} v) = \|f'x - f'y\|$ holds. Then, continuity of a derivative $f' : \alpha \to (\alpha \to_{bl} \beta)$ can simply be written as $(\mathit{continuous}\ f'\ (\mathit{at}\ x))$, which is a better abstraction to work with and also avoids redundant formalizations for different kinds of continuity.

(Bounded) Linear Functions between Euclidean Spaces. In hindsight, instead of bounded linear functions between normed vector spaces with the operator norm, a type for linear functions between Euclidean spaces could have been more helpful in the formalization. This type is also (bounded) linear, but with e.g., the Frobenius norm (interpreting the linear function $\mathbb{R}^n \to \mathbb{R}^m$ as matrix $\mathbb{R}^{n\times m}$ and taking the norm of this matrix), the resulting type would be a Euclidean space. In one place of our formalization (section 7.4), we make such a distinction and explicitly convert between $\mathbb{R}^n \to_{bl} \mathbb{R}^n$ and $\mathbb{R}^{n\times n}$. This is in order to evaluate the variational equation, where we want to re-use the verified algorithm for Euclidean space and therefore need an explicit representation type $\mathbb{R}^{n\times n}$, which is an instance of Euclidean space. With a tentative type of linear functions between Euclidean spaces, this conversion would not be necessary.

Bounded Linear Operators. Bounded linear operators are a type copy of bounded linear functions $\alpha \to_{bl} \alpha$ from one type α into itself. One can also compose bounded linear functions according to $(f \circ_{bl} g) \cdot_{bl} x = f \cdot_{bl} (g \cdot_{bl} x)$. Bounded linear operators form a Banach algebra with composition as multiplication:

Definition 2.9 (Banach Algebra of Bounded Linear Operators)**.** *For* $f, g : \alpha \to_{bl} \alpha$,

$$(f * g) \cdot_{bl} x := (f \circ_{bl} g) \cdot_{bl} x$$

2.9 Related Work

We first concentrate on related work that has directly influenced the emergence of the mathematics library in Isabelle before surveying formalizations of real analysis in other proof assistants. An extensive survey is given by Boldo *et al.* [21].

2.9.1 Origins of Isabelle's Mathematics Library

Isabelle's mathematics library has its origins in Fleuriot and Paulson's [37] theory of real analysis which covered sequences, series, limits, continuity, transcendental functions, nth roots, and derivatives. These notions were all specific to $\mathbb{R}$, although much was also duplicated at type $\mathbb{C}$. This material has since been adapted to the new type class hierarchy. The non-standard analysis part with ${}^*\mathbb{R}$ and ${}^*\mathbb{C}$ is not adapted.

Much of the formalization of analysis has been ported to Isabelle/HOL from Harrison's multivariate analysis library for HOL Light [58, 61]. In addition to limits, convergence, continuity, and derivatives, Harrison's library also covers topology, linear algebra, and Henstock-Kurzweil integration, which has all been ported to Isabelle/HOL, mostly by Amine Chaieb and Robert Himmelmann. The formalization in HOL Light library is mostly specific to $\mathbb{R}^n$, its generalization to type classes in Isabelle/HOL is mostly due to Brian Huffman and Johannes Hölzl.

Instead of formalizing limits with filters, Harrison invented a variant of nets which also bore some similarities to filter bases. His library provided a tends-to relation parameterized by a single net, but did not have an equivalent of the general limit operator *filterlim* which is parameterized by two filters (see Section 2.4).

In contrast to HOL Light, Isabelle/HOL supports coercive subtyping [134]. In particular, the automatic insertion of coercions of the embeddings of integers and natural numbers to the real numbers makes statements much easier to read than the corresponding ones from HOL Light.

2.9.2 Analysis in Proof Assistants

Here we describe the formalized mathematics (in particular real analysis) in other proof assistants and highlight some of the most prominent results in order to give an impression of the state of the art of formalized mathematics.

HOL Light HOL Light's library has mostly been developed by John Harrison. A landmark result achieved with this library was the formal verification of the Kepler conjecture in the Flyspeck project [55]. Harrison's library also covers complex analysis [60], including results like, e.g., Cauchy's integral theorem or Cauchy's integral formula. Maggesi developed a general theory of metric space (as a predicate on sets, instead of type classes) in HOL Light [93] and also formalizes a complete metric space of continuous functions.

Isabelle/HOL Hölzl formalized an extensive library of measure and probability theory [67]. Advanced results in Isabelle/HOL include e.g., the central limit theorem [8], Green's theorem [1], and parts of the Flyspeck project.

HOL4 The library for real analysis in HOL4 provides basic real analysis as well as measure theory [100], and Henstock-Kurzweil integration [126].

Coq Coq's standard library contains an axiomatization of real numbers. It's conservative extension Coquelicot [20] aims at presenting a user-friendly library of real analysis. In Coquelicot, (general) limits are also formalized with filters. Coquelicot provides a notion of limit that may take values in the extended real numbers. Derivatives and integrals are formalized as total functions with explicit assumptions on differentiability and integrability. This makes the formalization similar the the Isabelle/HOL formalization, in particular because dependent types are not used to encode differentiable or integrable functions, which could be a possible design choice.

A different approach is taken by Spitters and van der Weegen [130], who formalize a type class hierarchy for algebraic types in Coq. The motivation for their hierarchy of type classes is slightly different compared to Isabelle/HOL: their goal is efficient computation, hence they support different implementations for isomorphic types. In contrast, the goal in Isabelle/HOL is to share definitions and proofs for types which share the same mathematical structure. They also introduce type classes in category theory which is not possible in Isabelle as type classes are restricted to one type variable.

Outstanding formalizations in Coq are e.g., the formalizations of the Four Color Theorem or the Odd Order Theorem by Gonthier *et al.* [44, 45].

Mizar. Mizar, invented by Andrzej Trybulec, is based on (Tarski Grothendieck) set theory [135]. In the Mizar mathematical library, many proofs of advanced topological theorems are formalized, the overview by Naumowicz and Kornilowicz [105] lists e.g., the Jordan Curve Theorem, the Brouwer Fixed Point Theorem, Urysohn's Lemma, the Tichonov Theorem, or the Tietze Extension Theorem.

PVS. Lester [91] formalizes topology and constructive real numbers (as streams of digits that yield Cauchy sequences) in PVS. He formalizes topological spaces, T_2-spaces, second countable space, and metric spaces. He does not provide vector spaces *above* metric spaces and he does not use filters or nets to express limits.

3
Ordinary Differential Equations

ODEs describe how a system evolves in time. A solution is one possible evolution, formally defined in section 3.1. An important class of problems is initial value problems (IVPs). An IVP is an ODE together with an initial value. The problem is to characterize solutions evolving from the given initial value. Under mild assumptions, such a solution exists and is unique. The formalization thereof is discussed in section 3.2.

A natural question is to ask how the choice of initial value affects the solution. The solution to an ODE depending on the initial value is captured by the notion of flow. We formalize the flow and prove conditions for analytical properties like continuity of differentiability. Most of these properties seem very "natural", as Hirsch, Smale and Devaney call them in their textbook [65]. However, despite being "natural" properties and fairly standard results, they are delicate to prove: In the textbook, the authors present these properties rather early, but

> "postpone all of the technicalities [...], primarily because understanding this material demands a firm and extensive background in the principles of real analysis." [65]

In section 3.3, we show how to cope with these technicalities in a formal setting. This confirms that Isabelle/HOL supplies a sufficient background of real analysis.

Moreover we present the formalization of the so-called Poincaré map (section 3.4), an important notion for analyzing dynamical systems induced by ODEs. A dynamical system is a time-dependent process which is homogeneous in time, i.e., its evolution depends only on the initial state, but not on time. The Poincaré map plays a central role in Tucker's proof about the Lorenz attractor, which motivated its formalization.

3.1 Solutions

We assume ODEs as given by their right hand side $f :: \mathbb{R} \times \alpha \to \alpha :: \mathit{banach}$:

$$\dot{x}\, t = f(t, x\, t) \tag{3.1}$$

A solution to this ODE is any function $\phi :: \mathbb{R} \to \alpha$, which evolves according to the ODE. We formalize this with a predicate $(\phi \ \mathit{solves\text{-}ode}\ f)\ T\ X$ for explicit time domain T and space domain X.

Definition 3.1 (Solution of ODE)**.**

$$(\phi \ \mathit{solves\text{-}ode}\ f)\ T\ X := (\phi \ \mathit{has\text{-}vderiv\text{-}on}\ (\lambda t.\ f(t, \phi\, t)))\ T \wedge (\forall t \in T.\ \phi\, t \in X)$$

The left conjunct prescribes that the derivative of ϕ satisfies the ODE, whereas the right conjunct ensures that the solutions remains in the explicitly given domain $X :: \alpha\ set$ for time arguments in $T :: \mathbb{R}\ set$.

The predicate $(\phi\ \textit{uniquely-solves-ode}\ f\ \textit{from}\ t_0)\ T\ X$ encodes that the solution ϕ is unique on a time interval T w.r.t. solutions ψ on shorter time intervals T' and for the same initial value at initial time t_0.

Definition 3.2 (Unique Solution of Initial Value Problem)**.**

$$
\begin{aligned}
&(\phi\ \textit{uniquely-solves-ode}\ f\ \textit{from}\ t_0)\ T\ X := \\
&\quad (\phi\ \textit{solves-ode}\ f)\ T\ X \wedge \\
&\quad t_0 \in T \wedge \\
&\quad \textit{is-interval}\ T \wedge \\
&\quad (\forall \psi.\ \forall T' \subseteq T.\ (t_0 \in T' \wedge \textit{is-interval}\ T' \wedge (\psi\ \textit{solves-ode}\ f)\ T'\ X \wedge \psi\ t_0 = \phi\ t_0) \longrightarrow \\
&\qquad (\forall t \in T'.\ \psi\ t = \phi\ t))
\end{aligned}
$$

3.2 Initial Value Problems: Existence and Uniqueness

An initial value problem is an ODE together with an initial condition $x\ t_0 = x_0$. In this section, we formalize conditions on f, T, and X under which we can construct a unique solution ϕ such that $(\phi\ \textit{uniquely-solves-ode}\ f\ \textit{from}\ t_0)\ T\ X$ holds. The Picard-Lindelöf theorem establishes the existence of a local unique solution (section 3.2.1) via the Banach fixed point theorem. This can be extended to a globally unique solution, which we formalize in section 3.2.2.

3.2.1 Local Existence and Uniqueness

Lipschitz continuity is a basic assumption for the Picard-Lindelöf theorem. It is stronger than continuity: it limits how fast a function can change. A function g is (globally) Lipschitz continuous on a set S if the slope of the line connecting any two points on the graph of g is bounded by a Lipschitz constant L:

Definition 3.3 (Lipschitz Continuity)**.**

$$\textit{lipschitz}\ S\ g\ L := (\forall x, y \in S.\ \textit{dist}\ (g\ x)\ (g\ y) \leq L \cdot \textit{dist}\ x\ y)$$

The Banach fixed point theorem guarantees the existence of a unique fixed point of a Lipschitz continuous map g on a complete subset of a metric space α. This theorem is part of the mathematics library ported from HOL-Light. In order to present how this theorem is being used, we give its precise formulation in Isabelle/HOL as follows:

Theorem 3.4 (Banach Fixed Point Theorem)**.**
For $\beta :: \textit{metric-space}$, $S :: \beta\ set$, and $g :: \beta \to \beta$:

$$
\begin{aligned}
&(\textit{complete}\ S \wedge S \neq \emptyset \wedge 0 \leq L \wedge L < 1 \wedge \textit{lipschitz}\ S\ g\ L \wedge g(S) \subseteq S) \longrightarrow \\
&(\exists_1 x.\ x \in S \wedge g\ x = x)
\end{aligned}
$$

For the Picard-Lindelöf theorem in its most basic form, we instantiate the Banach fixed point theorem with the Picard operator on the space of bounded continuous functions. The following list details on how to instantiate β, S, and g in the Banach fixed point theorem 3.4 in order to obtain the Picard-Lindelöf theorem.

- $\beta := \mathbb{R} \to_{bc} \alpha$. The type $\to_{bc}$ of bounded continuous functions is a (complete) metric space.
- $S := [t_0; t_1] \to_{bc} X$. Here, the bounded continuous function space $[t_0; t_1] \to_{bc} X$ is the *set* of all bounded continuous functions with values in X for arguments in $[t_0; t_1]$. To be precise, $[t_0; t_1] \to_{bc} X := \{x :: \mathbb{R} \to_{bc} \alpha \mid \forall t \in [t_0; t_1].\ x \cdot_{bc} t \in X\}$.
- $g := P :: (\mathbb{R} \to_{bc} \alpha) \to (\mathbb{R} \to_{bc} \alpha)$ where

  $$P\ \phi_{bc} := \textit{ext-cont}_{[t_0;t_1]} \left(\lambda t.\ x_0 + \int_{t_0}^{t} f(\tau, \phi_{bc} \cdot_{bc} \tau) \mathrm{d}\tau \right)$$

  is the Picard operator $\phi \mapsto \lambda t.\ x_0 + \int_{t_0}^{t} f\,(\tau, \phi\ \tau)\ \mathrm{d}\tau$, but embedded in the type of bounded continuous functions with the help of $\textit{ext-cont}_{[t_0,t_1]}$, which is defined to continuously and constantly extend the argument outside the given interval.

  $$\textit{ext-cont}_{[t_0;t_1]}\ x \cdot_{bc} t := \begin{cases} x\ t_0, & \text{if } t < t_0 \\ x\ t_1, & \text{if } t > t_1 \\ x\ t, & \text{if } t \in [t_0; t_1] \end{cases}$$

If X is *closed* in the topology of α, then $[t_0, t_1] \to_{bc} X$ is closed (and complete) in the space $\mathbb{R} \to_{bc} \alpha$, and thus satisfies the first two assumptions of the Banach fixed point theorem 3.4. The third, fourth and fifth assumptions (on a suitable Lipschitz condition on g respectively P) can be proved from a continuous (in t) and uniformly (w.r.t. t) Lipschitz continuous (in x) right hand side f:

$$(\forall t \in [t_0; t_1].\ \textit{lipschitz}\ \ \textit{UNIV}_{\alpha}\ (\lambda x.\ f(t, x))\ L)$$

The last assumption ($g(S) \subseteq S$) is satisfied by assuming that $[t_0, t_1] \to X$ is closed under the Picard operator for continuous functions ϕ that satisfy the initial condition $\phi\ t_0 = x_0$.

$$(\forall t \in [t_0; t_1].\ \forall \phi.\ \phi\ t_0 = x_0 \wedge \phi \in [t_0; t] \to X \wedge \textit{continuous-on}\ [t_0; t]\ \phi) \longrightarrow \\ \phi\ t_0 + \int_{t_0}^{t} f(\tau, \phi\ \tau)\ \mathrm{d}\tau \in X$$

We summarize the aforementioned assumptions under the name *unique-on-closed*:

$$\begin{aligned} &\textit{unique-on-closed}\ X\ L := \\ &\quad \textit{closed}\ X \wedge \\ &\quad \textit{continuous-on}\ f\ ([t_0; t_1] \times X) \wedge \\ &\quad (\forall t \in [t_0; t_1].\ \textit{lipschitz}\ X\ (\lambda x.\ f(t, x))\ L) \wedge \\ &\quad (\forall t \in [t_0; t_1].\ \forall \phi.\ \phi\ t_0 = x_0 \wedge \phi \in [t_0; t] \to X \wedge \textit{continuous-on}\ [t_0; t]\ \phi \\ &\qquad \longrightarrow \phi\ t_0 + \int_{t_0}^{t} f(\tau, \phi\ \tau)\ \mathrm{d}\tau \in X) \end{aligned}$$

The assumptions *unique-on-closed* and small enough t_1 (i.e. $(t_1 - t_0) \cdot L < 1$, which makes the Lipschitz constant of P smaller than 1) make it possible to apply the Banach fixed point theorem. It guarantees the existence of a unique fixed point ϕ_{bc} for the mapping P. Together with the fundamental theorem of calculus, it follows that the fixed point ϕ_{bc} of P is a solution to the IVP.

$$((\lambda t.\ \phi_{bc} \cdot_{bc} t)\ \textit{solves-ode}\ f)\ [t_0; t_1]\ \textit{UNIV}_\alpha \wedge \phi_{bc} \cdot_{bc} t_0 = x_0$$

Moreover every (continuously extended) solution ψ is a fixed point of P

$$\forall \psi.\ ((\psi\ \textit{solves-ode}\ f)\ [t_0; t_1]\ \textit{UNIV}_\alpha \wedge \psi\ t_0 = x_0) \longrightarrow \\ P(\textit{ext-cont}_{[t_0;t_1]}\ \psi) = \textit{ext-cont}_{[t_0,t_1]}\ \psi$$

from which we conclude the existence of a unique solution $\phi\ t := \phi_{bc} \cdot_{bc} t$:

Lemma 3.5.

$$(\textit{unique-on-closed}\ X\ L \wedge (t_1 - t_0) \cdot L < 1) \longrightarrow \\ (\phi\ \textit{uniquely-solves-ode}\ f\ \textit{from}\ t_0)\ [t_0; t_1]\ X$$

This result can be strengthened by dropping the assumption $(t_1 - t_0) \cdot L < 1$: We can subdivide the interval $[t_0; t_1]$ into n intervals $[s_0, s_1] \cup \cdots \cup [s_{n-1}; s_n] = [t_0; t_1]$ such that every individual interval is small enough to ensure a contraction: $(s_{i+1} - s_i) \cdot L < 1$ Then we can invoke lemma 3.5 for a solution ϕ_{i+1} on the interval $[s_i; s_{i+1}]$. With induction on n, we may assume a solution $\phi_{0,i}$ on $[s_0; s_i]$ and combine them to a solution according to the following lemma:

Lemma 3.6.

$$((\phi_{0,i}\ \textit{uniquely-solves-ode}\ f\ \textit{from}\ s_0)\ [s_0; s_i]\ X \wedge \\ (\phi_{i+1}\ \textit{uniquely-solves-ode}\ f\ \textit{from}\ s_i)\ [s_i; s_{i+1}]\ X) \longrightarrow \\ ((\lambda t.\ \textbf{if}\ t \leq s_i\ \textbf{then}\ \phi_{0,i}\ t\ \textbf{else}\ \phi_{i+1}\ t)\ \textit{uniquely-solves-ode}\ f\ \textit{from}\ s_0)\ [s_0; s_{i+1}]\ X$$

It then follows a first version of the Picard-Lindelöf theorem:

Theorem 3.7 (Picard-Lindelöf)**.**

$$\textit{unique-on-closed}\ X\ L \longrightarrow (\phi_{0,n}\ \textit{uniquely-solves-ode}\ f\ \textit{from}\ t_0)\ [t_0; t_1]\ X$$

To summarize the proof of the Picard-Lindelöf theorem, the overall approach is similar to textbooks (e.g., the one by Walter [141]), but we have seen some particularities of the formalization that are worth pointing out: Moving explicitly between functions $\mathbb{R} \to \alpha$ and the dedicated type of bounded continuous functions $\mathbb{R} \to_{bc} \alpha$ is the most prominent difference to textbook proofs. In textbook proofs, the induction on n (after the with $(t_1 - t_0) \cdot L < 1$ restricted lemma 3.5) can be avoided by using a different metric (depending on t_0, t_1), which is not possible in our setting, as the metric is fixed once and for all in the type class instantiation of type $\to_{bc}$ (section 2.8.1).

Note that we left the co-domain X generic, suitable specializations are presented in the following. An easy instantiation is to take $X = \textit{UNIV}_\alpha$, if f satisfies a global Lipschitz condition on the whole type universe:

Theorem 3.8 (Picard-Lindelöf on a Strip)**.**

$$\begin{aligned}&(\textit{continuous-on}\ f\ ([t_0;t_1]\times \textit{UNIV}_\alpha)\ \wedge\\ &(\forall t\in[t_0;t_1].\ \textit{lipschitz}\ \textit{UNIV}_\alpha\ (\lambda x.\ f(t,x))\ L))\ \longrightarrow\\ &\exists\phi.\ (\phi\ \textit{uniquely-solves-ode}\ f\ \textit{from}\ t_0)\ [t_0;t_1]\ X\end{aligned}$$

Often times, the assumption on a global Lipschitz condition is too restrictive, we therefore give (as is also standard in textbook expositions) an explicit variant where X is a closed ball $\mathcal{B}_b(x_0)$ of radius b around the initial value x_0. Then t_1 needs to be chosen small enough such that the solution does not leave $\mathcal{B}_b(x_0)$: $C := \textit{norm}\,(f([t_0;t_1]\times\mathcal{B}_b(x_0))$ is the maximum slope of a solution confined to $\mathcal{B}_b(x_0)$ for time in $[t_0;t_1]$. Therefore, at time t_1, the distance of the solution to the initial value can be at most $(t_1-t_0)C$, if we assume $t_1-t_0\leq b/C$. The solution is guaranteed to remain in the closed ball $\mathcal{B}_b(x_0)$. This can be proved with an application of the mean value theorem. This suffices to show the last conjunct of *unique-on-closed*, i.e., that $[t_0;t_1]\to_{bc} X$ is closed under the Picard operator. In summary:

Theorem 3.9 (Picard-Lindelöf on a Ball)**.**

$$\begin{aligned}&(\textit{continuous-on}\ f\ ([t_0;t_1]\times\mathcal{B}_b(x_0))\ \wedge\\ &(\forall t\in[t_0;t_1].\ \textit{lipschitz}\ \mathcal{B}_b(x_0)\ (\lambda x.\ f(t,x))\ L)\ \wedge\\ &t_1-t_0\leq b/C\ \wedge\\ &(\forall t\in[t_0;t_1].\ \forall x\in\mathcal{B}_b(x_0).\ \textit{norm}\,(f(t,x))\leq C))\ \longrightarrow\\ &\exists\phi.\ (\phi\ \textit{uniquely-solves-ode}\ f\ \textit{from}\ t_0)\ [t_0;t_1]\ (\mathcal{B}_b(x_0))\end{aligned}$$

Many functions do not have a global Lipschitz constant (e.g. $\lambda x.\ x^2$ on $\mathbb{R}$). The weaker assumption of *local Lipschitz continuity* allows one to prove the existence of a solution in a neighborhood of the initial value. We consider binary functions that are locally Lipschitz continuous in their second argument (uniformly w.r.t. the first): A function f is locally Lipschitz continuous in its second variable if for every point (t,x) of the domain, there exists a neighborhood $\mathcal{B}_\epsilon(t)\times\mathcal{B}_\epsilon(x)$ inside which there exists a Lipschitz constant L:

Definition 3.10 (Local Lipschitz Continuity)**.**

$$\begin{aligned}\textit{local-lipschitz}\ f := \forall(t,x)\in T\times X.\ \exists\epsilon>0.\ \exists L.\\ \forall u\in\mathcal{B}_\epsilon(t)\cap T.\ \textit{lipschitz}\ (\lambda x.\ f(u,x))\ (\mathcal{B}_\epsilon(x)\cap X)\ L\end{aligned}$$

The only assumptions that are needed to show the existence of a unique solution are open sets for time T and space X and a locally Lipschitz continuous right-hand side f that is continuous in t:

Definition 3.11 (Conditions for Local Unique Solution)**.**

$$\begin{aligned}&\textit{ll-on-open}\ T\ X\ f :=\\ &\quad\textit{open}\ T\ \wedge\\ &\quad\textit{open}\ X\ \wedge\\ &\quad\textit{local-lipschitz}\ T\ X\ f\ \wedge\\ &\quad(\forall x\in X.\ \textit{continuous-on}\ T\ (\lambda t.\ f(t,x)))\end{aligned}$$

From these assumptions, we can choose for every $t \in T$ and $x \in X$ suitable closed balls with radius ε_t for time and ε_x for space such that there is a local unique solution ϕ according to theorem 3.9.

Theorem 3.12 (Local Unique Solution)**.**

$$\exists \varepsilon_t > 0.\ \exists \varepsilon_x > 0.\ \exists \phi.\ (\phi\ \textit{uniquely-solves-ode}\ f\ \textit{from}\ t_0)\ (\mathcal{B}_{\varepsilon_t}(t_0))\ (\mathcal{B}_{\varepsilon_x}(x_0))$$

3.2.2 Global Existence and Uniqueness

The construction of a global unique solution goes along the lines of Walter's textbook [141]. First, we define the set of all solutions with initial condition x_0 at time t_0 on compact time intervals and together with the bound of their respective intervals.

$$\Phi\ t_0\ x_0 := \{(\phi, t_1) \mid [t_0; t_1] \subseteq T \wedge \phi\ t_0 = x_0 \wedge (\phi\ \textit{solves-ode}\ f)\ [t_0; t_1]\ X\}$$

From this set of solutions, the maximal existence interval is defined as the union of all closed intervals with solutions:

Definition 3.13 (Maximal Existence Interval)**.**

$$\textit{ex-ivl}\ t_0\ x_0 := \bigcup_{(\phi, t_1) \in \Phi\ t_0\ x_0} [t_0; t_1]$$

By definition, the *maximal* existence interval deserves its name, because as soon as any solution from t_0 exists until some time t_1, this is contained in *ex-ivl* $t_0\ x_0$. It is also straightforward to see that *ex-ivl* is an interval.

Because of theorem 3.12, for every time $t_1 \in$ *ex-ivl* $t_0\ x_0$, there is a ball in T that contains t_1 and on which a solution exists. It follows that the existence interval is an open set.

Theorem 3.14 (Open Existence Interval)**.**

$$(t_0 \in T \wedge x_0 \in X) \longrightarrow \textit{open}\ (\textit{ex-ivl}\ t_0\ x_0)$$

All solutions $(\phi_1, t_1), (\phi_2, t_2) \in \Phi$ take the same values on the intersection of their existence intervals (if not, there is a maximal time t^* up to which they are equal, but then theorem 3.12 can be invoked from that point and yields a contradiction). Therefore *sol* is well defined as follows.

Definition 3.15 (Global Solution)**.**

$$\textit{sol}\ t_0\ x_0\ t := \phi\ t, \qquad \textit{for some}\ (\phi, t) \in \Phi\ t_0\ x_0$$

It follows that *sol* is the unique solution on the maximal existence interval *ex-ivl*.

Theorem 3.16 (Global Unique Solution)**.**

$$\begin{aligned}&\textit{ll-on-open}\ T\ X\ f \wedge t_0 \in T \wedge x_0 \in X \longrightarrow \\ &(\textit{sol}\ t_0\ x_0\ \textit{uniquely-solves-ode}\ f\ \textit{from}\ t_0)\ (\textit{ex-ivl}\ t_0\ x_0)\ X\end{aligned}$$

3.3 Flow

In this section, we focus on how the global unique solution *sol* t_0 x_0 as constructed in the previous section depends on changes in the initial conditions t_0, x_0. But first of all, we simplify matters (and notation) a little bit. We consider open sets T, X and restrict ourselves to autonomous (this means that f does not depend on t) ODEs[1]. More precisely, we consider an ODE

$$\dot{x}\, t = f\ (x\, t) \tag{3.2}$$

with locally Lipschitz continuous right hand side $f :: \alpha :: \mathit{banach} \to \alpha$:

$$\mathit{ll\text{-}on\text{-}open}\ \ \mathit{UNIV}_{\mathbb{R}}\ X\ (\lambda t.\ \lambda x.\ f\ x)$$

To emphasize joint dependence on initial condition x_0 and flow time t, we write $\phi(x_0, t)$ for the solution of equation (3.2) for an initial condition $x\ 0 = x_0$. This solution depending on initial conditions is called the *flow* of the differential equation and we define it in terms of the global solution *sol* from the previous section.

Definition 3.17 (Flow)**.**

$$\phi(x_0, t) := \mathit{sol}\ 0\ x_0$$

In the context of autonomous ODEs, we write *ex-ivl* x_0 instead of *ex-ivl* t_0 x_0.

The flow ϕ (together with *ex-ivl*, which guarantees the flow to be well-defined) is a convenient way to talk about solutions, because after guaranteeing that they are well-defined, these constants have many nice properties, which can be stated without further assumptions.

3.3.1 Properties of the Flow

For a first overview, we sill start listing the main results that we formalized about ϕ and *ex-ivl*. Their more technical constructions and sketches of proofs are deferred to section 3.3.2.

3.3.1.1 Composition of solutions

A first nice property is the abstract property of the generic notion of flow. The flow allows one to easily state composition of solutions and to algebraically reason about them. As illustrated in figure 3.1, flowing from x_0 for time $s + t$ is equivalent to first flowing for time s, and from there flowing for time t.

This only works if the flow is defined also for the intermediate times (the theorem can not be true for $\phi(x_0, t + (-t))$ if $t \notin \mathit{ex\text{-}ivl}$).

Theorem 3.18 (Flow property)**.**

$$s \in \mathit{ex\text{-}ivl}\,(x_0) \longrightarrow t \in \mathit{ex\text{-}ivl}\,(\phi(x_0, s)) \longrightarrow \phi(x_0, s + t) = \phi(\phi(x_0, s), t)$$

[1] Many of our results are also formalized for non-autonomous ODEs, but the notation is clearer, and if the right-hand side is locally Lipschitz continuous in time t, the non-autonomous ODE can directly be encoded as an autonomous one.

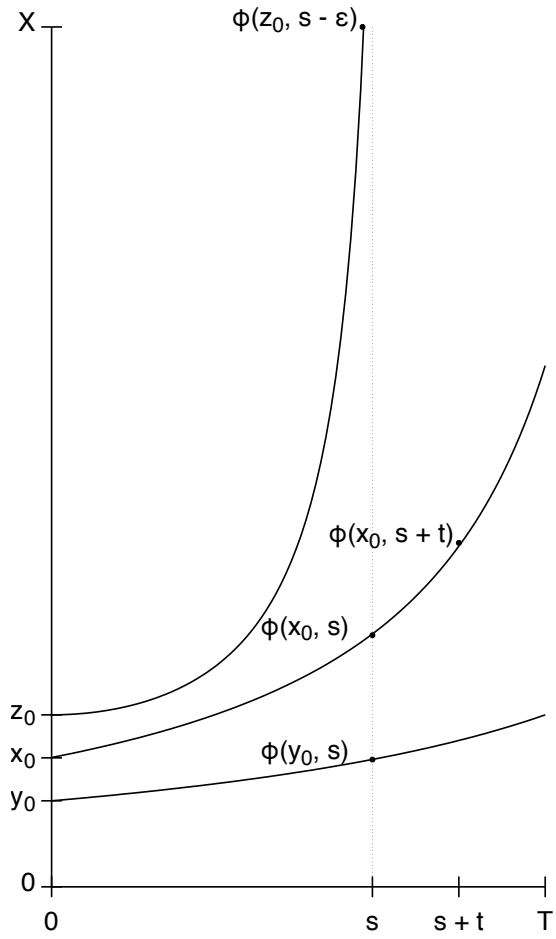


Figure 3.1: The flow for different initial values

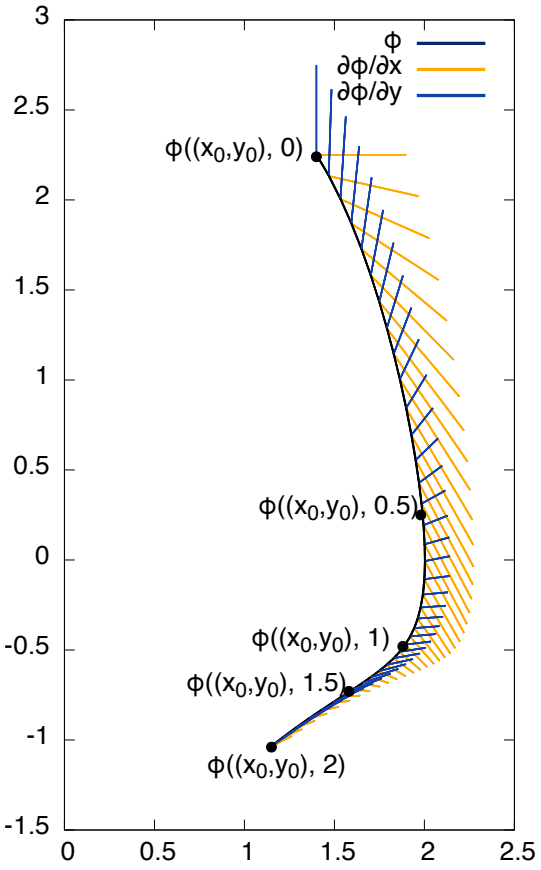


Figure 3.2: Flow ϕ of the van der Pol system $(\dot{x}, \dot{y}) = (y, (1 - x^2)y - x)$ and its partial derivatives $\frac{\partial \phi}{\partial x}$, $\frac{\partial \phi}{\partial y}$ for initial condition $(x_0, y_0) = (1.4, 2.25)$.

3.3.1.2 Continuity of the Flow

In the previous lemma, the assumption that the flow is defined (i.e., that the time is contained in the existence interval) was important. Let us now look at the domain $\{(x,t).\ t \in \mathit{ex\text{-}ivl}\,(x)\} \subseteq T \times X$ of the flow in more detail. It is called the *state space* and denoted by Ω.

Definition 3.19 (State Space)**.**

$$\Omega := \{(x,t).\ t \in \mathit{ex\text{-}ivl}\,(x)\}$$

Consider an element in the state space. $(x,t) \in \Omega$ means that we can follow a solution starting at x for time t. It is natural to expect that solutions starting close to x can be followed for times that are close to t. In topological parlance, the state space is open.

Theorem 3.20 (Open State Space)**.** $\mathit{open}\ \Omega$

In the previous theorem, the state space allows us to reason about the fact that solutions are *defined* for close times and initial values. For quantifying how deviations in the initial values are propagated by the flow, Grönwall's lemma is an important tool that is used in several proofs. Because of its importance in the theory of dynamical systems, we list it here as well, despite it being a rather technical result. Starting from an *implicit* inequality $g\ t \leq C + K \cdot \int_0^t g(s)\,\mathrm{d}s$ involving a continuous, nonnegative function $g : \mathbb{R} \to \mathbb{R}$, it allows to deduce an *explicit* bound for g:

Lemma 3.21 (Grönwall)**.**

$$\begin{aligned}
&0 < C \longrightarrow 0 < K \longrightarrow \mathit{continuous\text{-}on}\ [0;a]\ g \longrightarrow \\
&\forall t.\ 0 \leq g\ t \leq C + K \cdot \int_0^t (g\ s)\,\mathrm{d}s \longrightarrow \\
&\forall t \in [0;a].\ g\ t \leq C \cdot e^{K \cdot t}
\end{aligned}$$

Grönwall's lemma can be used to show that solutions deviate *at most* exponentially fast: $\exists K.\ \|\phi(x,t) - \phi(y,t)\| < \|x - y\| \cdot e^{K \cdot |t|}$ (see also Lemma 3.27). Therefore, by choosing x and y close enough, one can make the distance of the solutions arbitrarily small. This implies that the flow is continuous in its first argument and can be used to show that the flow is continuous on the state space:

Theorem 3.22 (Continuity of Flow)**.** $\mathit{continuous\text{-}on}\ \Omega\ \phi$

3.3.1.3 Differentiability of the Flow

Continuity states that small deviations in the initial values result in small deviations of the flow. But one can be more precise on how initial deviations propagate. Let us assume that f is a continuously differentiable function on Euclidean space, namely:

Definition 3.23.

$$\begin{aligned}
&\mathit{c1\text{-}on\text{-}open}\ X\ (f :: \alpha :: \mathit{euclidean\text{-}space} \to \alpha) := \\
&\mathit{open}\ X \wedge (f\ \mathit{has\text{-}derivative}\ (\lambda x.\ \mathrm{D}f|_{at\,x}))\ (\mathit{at}\ x) \wedge \mathit{continuous\text{-}on}\ X\ (\lambda x.\ \mathrm{D}f|_{at\,x})
\end{aligned}$$

Then the flow is also continuously differentiable: the way initial deviations propagate can be approximated by a (bounded) linear function. So instead of solving the ODE for perturbed initial values, one can approximate the resulting perturbation with this linear function: We write $\phi_t := \lambda x.\ \phi(x,t)$, then one has $\mathsf{D}\phi_t|_{\mathit{at}\ x} \cdot_{\mathit{bl}} v \approx \phi_t(x+v) - \phi_t(x)$. By using a basis vector for v, one gets the corresponding partial derivative of the flow. $\mathsf{D}\phi_t|_{\mathit{at}\ x} :: \alpha \to_{\mathit{bl}} \alpha$ is called the space derivative of the flow.

As an example, figure 3.2 depicts a two-dimensional flow ϕ starting at (x_0, y_0) and its evolution (in black) up to time $t = 2$ in black. Along with the flow, it shows the evolution of the partial derivatives $\frac{\partial\phi((x_0,y_0),t)}{\partial x} = \mathsf{D}\phi_t|_{\mathit{at}\ (x_0,y_0)} \cdot_{\mathit{bl}} (1,0)$ and $\frac{\partial\phi(x_0,y_0)}{\partial y} = \mathsf{D}\phi_t|_{\mathit{at}\ (x_0,y_0)} \cdot_{\mathit{bl}} (0,1)$.

The derivative of ϕ with respect to time is given by the ODE f, one can therefore write the total derivative of the flow as $\mathsf{D}\phi|_{\mathit{at}\ (x,t)} \cdot_{\mathit{bl}} (d_x, d_t) := \mathsf{D}\phi_t|_x \cdot_{\mathit{bl}} d_x + d_t f(\phi(x,t))$. Putting everything together, the total derivative W of the flow exists and is continuous on the state space.

Theorem 3.24 (Differentiability of the Flow)**.**

$$(\phi\ \mathit{has\text{-}derivative}\ \mathsf{D}\phi|_{\mathit{at}\ (x,t)})\ (\mathit{at}\ (x,t)) \wedge \mathit{continuous\text{-}on}\ \Omega\ (\lambda(x,t).\ \mathsf{D}\phi|_{\mathit{at}\ (x,t)})$$

3.3.2 Proofs about the Flow

Let us now turn to some more technical lemmas that are required to prove the main properties from the previous section. If the results are also formalized for non-autonomous ODEs, we will list those more general versions. The flow of a non-autonomous ODE is then denoted with $\phi(t_0, x_0, t)$.

3.3.2.1 The Frontier of the State Space

It is important to study the behavior of the flow at the frontier of the state space (e.g., as time or the solution tend to infinity), because it allows one to deduce conditions under which solutions can be continued further and yields techniques to gain more precise information on the existence interval *ex-ivl*.

If the solution only exists for finite time, it necessarily explodes (i.e., the solution leaves every compact set):

Lemma 3.25 (Explosion for Finite Existence Interval)**.**

$$\begin{aligned}&\sup(\mathit{ex\text{-}ivl}\,(t_0,x_0)) < \infty \longrightarrow \mathit{compact}\ K \longrightarrow\\ &\exists t \geq t_0.\ t \in \mathit{ex\text{-}ivl}\,(t_0,x_0) \wedge \phi(t_0,x_0,t) \notin K\end{aligned}$$

This lemma can be used to prove a condition on the right hand side f of the ODE, to certify that the solution exists for an arbitrary time in T.

Lemma 3.26 (Condition for Global Existence of Solution)**.**

$$\begin{aligned}&(\forall s \in T.\ \forall u \in T.\ \exists L.\ \exists M.\ \forall t \in [s,u].\ \forall x \in X.\ \|f\ t\ x\| \leq M + L \cdot \|x\|) \longrightarrow\\ &\mathit{ex\text{-}ivl}\,(t_0,x_0) = T\end{aligned}$$

3.3.2.2 Continuity of the Flow

The following lemmas are all related to continuity of the flow. With the help of Grönwall's lemma 3.21, one can show that whenever two solutions (starting from different initial conditions x_0 and y_0) both exist for a time t and are restricted to some set Y on which the right hand side f satisfies a (global) Lipschitz condition K, then the distance between the solutions grows at most exponentially with increasing time:

Lemma 3.27 (Exponential Initial Condition for Two Solutions)**.**

$$\begin{aligned}
&t \in \textit{ex-ivl}\,(t_0, x_0) \longrightarrow t \in \textit{ex-ivl}\,(t_0, y_0) \longrightarrow \\
&Y \subseteq X \longrightarrow \forall s \in [t_0; t].\ \phi(t_0, x_0, s) \in Y \longrightarrow \\
&Y \subseteq X \longrightarrow \forall s \in [t_0; t].\ \phi(t_0, y_0, s) \in Y \longrightarrow \\
&s \in [t_0; t].\ \textit{lipschitz}\ Y\ (f\ s)\ K \longrightarrow \\
&\|\phi(t_0, x_0, t) - \phi(t_0, y_0, t)\| \leq \|x_0 - y_0\| \cdot e^{K \cdot (t - t_0)}
\end{aligned}$$

Note that it can be hard to establish the assumptions of this lemma, in particular the first two assumptions that both solutions from x_0 and y_0 exist for the same time t. Consider figure 3.1: not all solutions (e.g., from z_0) do necessarily exist for the same time s. One can choose, however, a neighborhood of x_0, such that all solutions starting from within this neighborhood exist for at least the same time, and with the help of the previous lemma, one can show that the distance of these solutions increases at most exponentially:

Lemma 3.28 (Exponential Initial Condition of Close Solutions)**.**

$$\begin{aligned}
&a \in \textit{ex-ivl}\,(t_0, x_0) \longrightarrow b \in \textit{ex-ivl}\,(t_0, x_0) \longrightarrow a \leq b \\
&\exists \delta > 0.\ \exists K > 0.\ U_\delta(x_0) \subseteq X \wedge \\
&\quad (\forall y \in U_\delta(x_0).\ \forall t \in [a; b]. \\
&\qquad t \in \textit{ex-ivl}\,(t_0, x_0) \wedge |\phi(t_0, x_0, t) - \phi(t_0, y, t)| \leq \|x_0 - y\| \cdot e^{K \cdot |t - t_0|})
\end{aligned}$$

Using this lemma is the key to showing continuity of the flow (theorem 3.22).

A different kind of continuity is not with respect to the initial condition, but with respect to the right-hand side of the ODE.

Lemma 3.29 (Continuity with respect to ODE)**.** *Assume two right-hand sides f, g defined on X and uniformly close ($\forall x \in X.\ \|f\ x - g\ x\| < \varepsilon$). Furthermore, assume a global Lipschitz constant K for f on X. Then the deviation of the flows ϕ_f and ϕ_g can be bounded:*

$$\|\phi_f(x_0, t) - \phi_g(x_0, t)\| \leq \frac{\varepsilon}{K} \cdot e^{K \cdot t}$$

3.3.2.3 Differentiability of the Flow

The proof for the differentiability of the flow incorparates many of the tools that we have presented up to now, we will therefore sketch some of the details of this proof (which follows the presentation of Hirsch *et al.* [65]). This proof has been formalized by Christoph Traut as part of a student's project.

We denote the derivative *along* the flow from x_0 with $A_{x_0} : \mathbb{R} \to (\alpha :: \textit{euclidean-space} \to_{bl} \alpha)$:

Definition 3.30 (Derivative along the Flow)**.** $A_{x_0}(t) := \mathsf{D}f|_{\mathsf{at}\ \phi(x_0,t)}$

The derivative of the flow is the solution to the so so-called variational equation, a non-autonomous linear ODE. The initial condition $\xi :: \alpha$ is supposed to be a perturbation of the initial value (like v_x and v_y in figure 3.2) and in what follows we will prove that the solution to this ODE is a good (linear) approximation of the propagation of this perturbation.

$$\begin{cases} \dot{u}(t) = A_{x_0}(t) \cdot_{bl} u(t) \\ u(0) = \xi \end{cases} \tag{3.3}$$

We will write $u_{x_0}(\xi, t)$ for the flow of this ODE and omit the parameter x_0 and/or the initial value ξ if they can be inferred from the context.

As a prerequisite for the next proof, we begin by proving that $u_{x_0}(\xi, t)$ is linear in ξ, a property that holds because u is the solution of a linear ODE (this is often also called the "superposition principle").

Lemma 3.31 (Linearity of $u_{x_0}(\xi, t)$ in ξ)**.**

$$\alpha \cdot u_{x_0,a}(t) + \beta \cdot u_{x_0,b}(t) = u_{x_0,\alpha\cdot a+\beta\cdot b}(t).$$

Because $\lambda\xi.\ u_{x_0}(\xi, t) : \alpha \to \alpha$ is linear on Euclidean space $\alpha :: \mathit{euclidean\text{-}space}$, it is also bounded linear, so we will identify this function with the corresponding element of type $\alpha \to_{bl} \alpha$. The main efforts for proving differentiability of the flow go into proving the following lemma, showing that the aforementioned function is the derivative with respect to space of the flow $\phi_t = (x_0 \mapsto \phi(x_0, t))$ for fixed time t.

Lemma 3.32 (Space Derivative of the Flow)**.** *For $t \in \mathit{ex\text{-}ivl}\,(x_0)$,*

$$(\mathsf{D}\phi_t|_{\mathsf{at}\ x_0}) \cdot_{bl} \xi = u_{x_0}(\xi, t)$$

Proof. The proof starts out with the integral identities of the flow, the perturbed flow, and the linearized propagation of the perturbation:

$$\begin{aligned} \phi(x_0, t) &= x_0 + \int_0^t f(\phi(x_0, s))\,\mathrm{d}s \\ \phi(x_0 + \xi, t) &= x_0 + \xi + \int_0^t f(\phi(x_0 + \xi, s))\,\mathrm{d}s \\ u_{x_0}(\xi, t) &= \xi + \int_0^t A_{x_0}(s) \cdot u_{x_0}(\xi, s)\,\mathrm{d}s \\ &= \xi + \int_0^t f'(\phi(x_0, s)) \cdot u_{x_0}(\xi, s)\,\mathrm{d}s \end{aligned}$$

Then, for any fixed ε, after a sequence of estimations (3 pages in the textbook proof) involving e.g., uniform convergence (section 2.4.2) of the first-order remainder term of the Taylor expansion of f, continuity of the flow (theorem 3.22), and linearity of u (lemma 3.31) one can prove the following inequality.

$$\frac{\|\phi(x_0 + \xi, t) - \phi(x_0, t) - u_{x_0}(\xi, t)\|}{\|\xi\|} \leq \varepsilon$$

This shows that $u_{x_0}(\xi, t)$ is indeed a linear approximation for the propagation of the initial perturbation ξ and exactly the definition for the space derivative of the flow. □

Note that $u_{x_0}(\xi, t)$ yields the space derivative in direction of the vector ξ. The total space derivative of the flow is then the linear function $\lambda\xi.\ u_{x_0,\xi}(t)$. This derivative can equivalently be described as the solution of the following "matrix-valued" variational equation:

$$\begin{cases} \dot{W}_{x_0}(t) = A_{x_0}(t) \circ_{bl} W_{x_0}(t) \\ W_{x_0}(0) = 1_{bl} \end{cases} \tag{3.4}$$

This IVP is defined for linear operators of type $\alpha \to_{bl} \alpha$. Thanks to lemma 3.26, one can show that it is defined on the same existence interval as the flow ϕ. The solution $W_{x_0} :: \mathbb{R} \to \alpha \to_{bl} \alpha$ is related to solutions $u_{x_0} :: \mathbb{R} \to \alpha$ of the "vector-valued" variational IVP (3.3) as follows:

$$u_{x_0}(\xi, t) = W_{x_0}(t) \cdot_{bl} \xi$$

The derivative of the flow ϕ at (x_0, t) with respect to t is given directly by the ODE, namely $f(\phi(x_0, t))$. Therefore the total derivative of the flow is characterized as follows:

Theorem 3.33 (Derivative of the Flow)**.**

$$\mathsf{D}\phi|_{\mathsf{at}\ (x_0,t)} \cdot_{bl} (\xi, \tau) = W_{x_0}(t) \cdot_{bl} \xi + \tau \cdot f\ (\phi(x_0, t))$$

This result is also based on deducing the total derivative of a function g by looking at its (continuous!) partial derivatives g_1 and g_2 (that is, the derivatives w.r.t. one variable while fixing the other).

Lemma 3.34 (Total Derivative via Continuous Partial Derivatives)**.** *For* $g : \alpha \to \beta \to \gamma$, $g_1 : \alpha \to \beta \to (\alpha \to_{bl} \gamma)$, $g_2 : \alpha \to \beta \to (\beta \to_{bl} \gamma)$

$$\begin{aligned} &\forall x.\ \forall y.\ \mathsf{D}(\lambda x.\ g\ x\ y)|_{\mathsf{at}\ x} = g_1\ x\ y \longrightarrow \\ &\forall x.\ \forall y.\ \mathsf{D}(\lambda y.\ g\ x\ y)|_{\mathsf{at}\ y} = g_2\ x\ y \longrightarrow \\ &\mathit{continuous}\ (\mathsf{at}\ (x,y))\ (\lambda(x,y).\ g_1\ x\ y) \longrightarrow \\ &\mathit{continuous}\ (\mathsf{at}\ (x,y))\ (\lambda(x,y).\ g_2\ x\ y) \longrightarrow \\ &\mathsf{D}(\lambda(x,y).\ g\ x\ y)|_{\mathsf{at}\ (x,y)} \cdot_{bl} (t_1, t_2) = (g_1\ x\ y) \cdot_{bl} t_1 + (g_2\ x\ y) \cdot_{bl} t_2 \end{aligned}$$

3.3.2.4 Continuity of Derivative

We can also show that the derivative $\mathsf{D}\phi|_{\mathsf{at}\ (x_0,t)} \cdot_{bl} (\xi, \tau)$ is continuous with respect to (x_0, t). First of all, $\tau \cdot f\ (\phi(x_0, t))$ is continuous because of definition 3.11 and theorem 3.22. Second, $W_{x_0}(t)$ is continuous with respect to t. What remains to show is continuity of the space derivative regarding x_0. The proof of this relies on theorem 3.29, because for different values of x_0, W_{x_0} is the solution to ODEs with slightly different right-hand sides. A technical difficulty here is to establish the assumption of *global* Lipschitz continuity for theorem 3.29.

3.4 Poincaré Map

The Poincaré map is an important tool for analyzing dynamical systems. Whereas the flow describes the evolution of a continuous system with respect to *time*, it is the Poincaré map

that allows to describe the evolution with respect to some *space* variables. A Poincaré section is a subset Σ of the state space. Consider the illustration in figure 3.3. For a point $x \in \Sigma$, the Poincaré map is defined as the point $P(x)$ where the flow starting from x, i.e., $\lambda t.\ \phi(x,t)$, first hits the Poincaré section Σ.

Studying the dynamics of a system using a Poincaré map has two advantages: first, the Poincaré map is a map on a lower-dimensional space and second, instead of analyzing the continuous dynamics of the original flow, one can confine the analysis to discrete iterations of the Poincaré map.

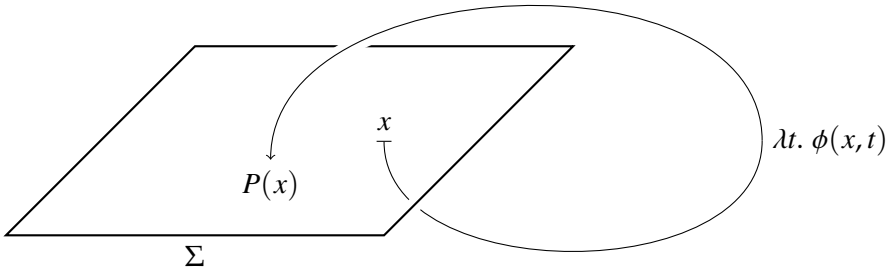


Figure 3.3: Illustration of Poincaré map P on Poincaré section Σ.

3.4.1 Properties of the Poincaré Map

A Poincaré section is usually given as an implicit surface $\Sigma = \{x \mid s(x) = c\}$ with continuously differentiable s. The Poincaré map is defined with the help of the first return time $\tau(x)$.

Definition 3.35 (First Return Time)**.** *$\tau(x)$ is the least $t > 0$ such that $\phi(x,t) \in \Sigma$.*

Obviously, τ is only well-defined for values that actually return to Σ, which we encode in the predicate *returns-to*.

Definition 3.36.

$$\textit{returns-to}\,(\Sigma, x) := \exists t > 0.\ \phi(x,t) \in \Sigma$$

The return time can then be used to define the Poincaré map as follows.

Definition 3.37 (Poincaré map)**.**

$$P(x) := \phi(x, \tau(x))$$

For a sensitivity analysis of the flow, one can study the derivative of the Poincaré map. Its derivative can be given in terms of the derivative of the flow ϕ_t for fixed time t ($\phi_t := \lambda x.\ \phi(x,t)$) and the function s defining the implicit surface for $\Sigma = \{x \mid s(x) = c\}$.

Theorem 3.38 (Derivative of Poincaré map)**.**

$$\mathsf{D}P|_x \cdot_{bl} h = \mathsf{D}\phi_{\tau(x)}|_x \cdot_{bl} h - \frac{\mathsf{D}s|_{P(x)} \cdot_{bl} (\mathsf{D}\phi_{\tau(x)}|_x \cdot_{bl} h)}{\mathsf{D}s|_{P(x)} \cdot_{bl} (f(P(x)))} f(P(x))$$

For a rough intuition, the derivative $\mathsf{D}P|_x \cdot_{bl} h$ of the Poincaré map is related to the space derivative of the flow at the return time $\mathsf{D}\phi_{\tau(x)}|_x \cdot_{bl} h$. But it needs to corrected in the direction $f(P(x))$ in which the flow passes through Σ, because P varies only *on* Σ and not through it. This correction factor also depends on the tangent space $\mathsf{D}s|_{P(x))}$ of the section Σ at $P(x)$.

In section 3.4.2, we precise the assumptions under which τ (and therefore P) is continuous and differentiable.

3.4.2 Proofs about the Poincaré Map

Here we sketch how to prove continuity and differentiability of the Poincaré map. We assume a Poincaré section to be a subset (via S) of an implicit surface with continuously differentiable s:

$$\Sigma = \{x \in S \mid s(x) = 0\}$$

Key techniques are the implicit function theorem and results about inverses of linear functions.

3.4.2.1 Inverse Functions

In the Banach space of bounded linear functions, the set of invertible functions is open:

Theorem 3.39.

$$\mathit{open}\,\{f :: \alpha \to_{bl} \beta \mid \exists f^{-1}.\ f \circ_{bl} f^{-1} = 1_{bl} \wedge f^{-1} \circ_{bl} f = 1_{bl}\}$$

The proof of this theorem is based on the fact that the inverse of the disturbed identity function $1_{bl} + w$ with $\|w\| < 1$ is the convergent series $\sum_i (-1)^i w^n$:

Lemma 3.40. *For $1_{bl}, w :: \alpha \to_{bl} \alpha$ with $\|w\| < 1$, $(\sum_i (-1)^i w^n)$ is convergent and the left and right inverse of $1_{bl} + w$:*

$$(\sum_i (-1)^i w^n) * (1_{bl} + w) = 1_{bl}$$
$$(1_{bl} + w) * (\sum_i (-1)^i w^n) = 1_{bl}$$

Moreover, one can bound the norm of the inverse,

Lemma 3.41. $\|(1_{bl} + w)^{-1} - (1_{bl} + w)\| \leq \frac{\|w\|^2}{1-\|w\|}$

which is necessary to prove the set of invertible linear functions open for theorem 3.39.

3.4.2.2 Implicit Function Theorem

The implicit function theorem is a powerful tool to construct (differentiable) functions satisfying a given (implicit) equation. Given an "equation" $F :: \alpha \times \beta \to \gamma$ with a root $F(x, y) = 0$, the theorem allows to "extend" the root (x, y) in an ε-neighborhood $U_\varepsilon(x)$ to a solution function u. This means that the image of the graph of u under F remains constant, namely $F(x, u(x)) = 0$.

For a precise formulation, F needs to be continuously differentiable with invertible derivative:

Theorem 3.42 (Implicit Function Theorem)**.** *Assume a zero $F(x,y) = 0$ of a continuously differentiable function F. We use the following notation for*
the derivative of F w.r.t. the 1st argument $f_1 \cdot d := \mathsf{D}F|_{(x,y)}(d,0)$ and
the derivative of F w.r.t. the 2nd argument $f_2 \cdot d := \mathsf{D}F|_{(x,y)}(0,d)$.
Assume that f_2 is invertible, i.e., f_2^{-1} exists with $f_2^{-1} \circ f_2 = 1_{bl}$ and $f_2 \circ f_2^{-1} = 1_{bl}$.

Then there exist $u :: \alpha \to \beta$ and $\varepsilon > 0$ such that:

- $u(x) = y$
- $F(x, u(x)) = 0$
- $\forall s \in U_\varepsilon(x).\ F(s, u(s)) = 0$
- *continuous-on* $U_\varepsilon(x)$ u
- $\mathsf{D}u|_x = -f_2^{-1} \circ f_1$
- *u is unique: for every v, V where $V \subseteq U_\varepsilon(x)$ is open and connected, v with continuous-on V v, $v(x) = y$, and $(\forall s \in V.\ F(s, v(s)) = 0)$, it holds that* $\forall s \in V.\ v(s) = u(s)$

Existence of such a function u on a neighborhood $U_\varepsilon(x)$ can be reduced to the *inverse function theorem*, which already exists in Isabelle's library. We therefore perform this reduction, which yields the expression for the derivative $\mathsf{D}u|_x = -f_2^{-1} \circ f_1$. Openness of invertible linear maps (theorem 3.39) is required for this construction.

3.4.2.3 Continuity and Differentiability of the Poincaré Map

The idea for proving differentiablity of the Poincaré map is to apply the implicit function theorem 3.42 to find a differentiable function u that solves $s(\phi(x, u(x))) = 0$ in a neighborhood $U_\varepsilon(x)$. The implicit function u is unique (w.r.t. continuous functions). Since $s(\phi(x, \tau(x))) = 0$, all one has to do is prove that τ is continuous, because then τ is equal to u according to theorem 3.42 and therefore $\mathsf{D}\tau|_{at\ x} = \mathsf{D}u|_{at\ x}$.

But first, one needs more assumptions on the Poincaré section Σ in order to carry out the construction of the implicit function u: we assume S closed and s continuously differentiable, moreover the flow needs to be transversal at the return map ($\mathsf{D}s|_{at\ \phi(x,\tau(x))} \cdot f(P(x)) \neq 0$). Furthermore, $P(x)$ needs to be in the relative (with respect to the implicit surface) interior of the Poincaré section ($\exists \delta.\ U_\delta(P(x)) \cap \{x \mid s(x) = 0\} \subseteq S$). We summarize these assumptions in the following definition.

Definition 3.43 (Assumptions for Poincaré Section)**.**

$$\mathsf{D}s|_{at\ \phi(x,\tau(x))} \cdot f(P(x)) \neq 0 \wedge (\exists \delta.\ U_\delta(P(x)) \cap \{x \mid s(x) = 0\} \subseteq S)$$

There are two cases for which we prove $\tau(x)$ continuous: First, if $x \notin \Sigma$, then τ is continuous in any sufficiently small neighborhood around x. The limit can therefore be approached from within an arbitrary set X, i.e., *at x within X*. Second, if $x \in \Sigma$, then τ is continuous only on the side of the surface Σ to which the vector field points to, i.e.,

(*at* x *within* $\{x \mid s(x) \leq 0\}$). That is because if y is taken (arbitrarily) close to x, but on the other side of Σ, then $\tau(y)$ is (arbitrarily) close to zero, but $\tau(x) > 0$. More formally, the two cases are given by theorems 3.44 and 3.45:

Theorem 3.44 (Continuity of τ outside Σ)**.**
continuous τ (*at* x *within* X)*, if the assumptions from definition 3.43 and the following conditions hold:*

- *returns-to* (Σ, x)
- $x \notin \Sigma$

Theorem 3.45 (Continuity of τ on Σ)**.**
continuous τ (*at* x *within* $\{x \mid s(x) \leq 0\}$)*, if the assumptions from definition 3.43 and the following conditions hold:*

- *returns-to* (Σ, x)
- $x \in \Sigma$
- $\mathrm{D}s|_{\mathit{at}\ x} \cdot f(x) < 0$

When τ is continuous (e.g., under the same same assumptions as those of theorem 3.44), τ is equal to the solution of the implicit function theorem. This yields an explicit expression for the derivative $\mathrm{D}\tau$.

Theorem 3.46 (Derivative of τ)**.**
Under the assumptions of theorem 3.44:

$$\mathrm{D}\tau|_x \cdot h = -\frac{\mathrm{D}s|_{P(x)} \cdot (\mathrm{D}\phi_{\tau(x)}|_x \cdot h)}{\mathrm{D}s|_{P(x)} \cdot (f(P(x)))} f(P(x))$$

From continuity and differentiability of the flow (theorems 3.22 and 3.24) and τ (theorems 3.45, 3.44, and 3.46), it follows that the Poincaré map is continuous and differentiable.

It is interesting to note that our definition of Poincaré map slightly differs from the approach that many textbooks (e.g., [117, chapter 3], [123, chapter 5.8], [65, chapter 10.3]) take. The original application of Poincaré maps is the study of periodic orbits, and textbooks usually define the return time implicitly for a periodic point as follows: For a point x with period $\tau(x)$ and a choice of Σ transversal to the flow, these textbooks invoke the implicit function theorem to obtain a continuous function $\tau_i(x)$ and declare the obtained function as the return time. This way, the definition implicitly depends on x and is valid only locally.

In contrast, our construction (as first return time) is a notion that is given globally (on the whole state space and not just on Σ) and a-priori (without any implicit constructions). We use the uniqueness condition of the implicit function theorem 3.42 and the continuity results of theorem 3.44 and 3.45 to show equality of τ and τ_i.

3.5 Related Work

To conclude this chapter, the formalization of the flow (section 3.3) contains essentially all lemmas and proofs of at least 22 pages (Chapter 17) of the textbook by Hirsch *et al.* [65]. This corresponds to section 2.1 to 2.5 on "Nonlinear Systems: Local Theory" (about 30 pages) of Perko's textbook [117]. The formalization moreover comprises a notion of Poincaré map, which is more flexible than what is usually presented in textbooks.

I am not aware of any other formalization that covers this foundational part of the theory of ODEs and dynamical systems in similar detail.

Boldo *et al.* [18] formalize partial differential equations stemming from acoustic wave equations in Coq. This particular problem admits an analytical solution and they simply assume that the solution is unique. Another important result, fundamental for numerical approximations of partial differential equations is the formalization of the Lax-Milgram theorem in Coq by Boldo *et al.* [22]. This formalization is similar to ours in the sense that it is also based on functional analysis and formalization of bounded linear functions.

Platzer's proof assistant KeYmaera X [40] is based on Differential Dynamic Logic [119]. Based on my formalization of ODEs, Bohrer *et al.* [16, 15] formalized Differential Dynamic Logic in Isabelle/HOL.

Gouëzel [48] formalized ergodic theory, i.e., dynamical systems with an invariant measure in Isabelle/HOL, his formalization includes Kac's Formula and Birkhoff and Kingman theorems. Gouëzel also formalized aspects of functional analysis, in particular L^p-spaces [49].

Maggesi [93] formalized a theory of metric spaces (as a predicate instead of type classes) for a formalization of a local version of the Picard-Lindelöf theorem (similar to theorem 3.9) in HOL Light.

4

Rigorous Numerics in Affine Arithmetic

Rigorous (or *guaranteed*) numerics means computing with sets that are guaranteed to enclose the quantities of interest, for example using intervals [102]. But in principle, enclosures can be any data structure that represent sets of real values, popular alternative choices are intervals, affine forms (they represent a class of sets called zonotopes), or Taylor models. A central idea of the approach that we are following here is the deep embedding of arithmetic expressions (section 4.1), which allows one to stay agnostic about the representation of concrete enclosures.

For concrete computations, we use software floating point numbers (section 4.2) for approximations of real numbers and affine arithmetic (section 4.3) for rigorous evaluation of expressions.

4.1 Expressions

For formalizations, it is useful to have a deep embedding of arithmetic expressions like e.g., done by Dorel and Melquiond [96] as well as Hölzl [66]. This work builds on Hölzl's language of arithmetic expressions given in the left of figure 4.1. We write $[\![e]\!]_{vs} :: \mathbb{R}$ for the interpretation of an expression e over an environment of variables $vs :: \mathbb{R}$ *list*. Expressions are interpreted recursively according to the right of figure 4.1. The role of the environment vs gets clear from the interpretation of variables. *Var i* is interpreted as the i-th element of the list vs. More operations can be derived from this language of expressions, e.g., subtraction $x - y = x + (-y)$, division $\frac{x}{y} = x \cdot (\frac{1}{y})$, and $\sin(x) = \cos(x - \frac{pi}{2})$.

Independent of the choice of representation of enclosures, a rigorous approximation scheme *approx* needs to satisfy that for every environment xs in an enclosure XS, the interpretation $[\![e]\!]_{xs} :: \mathbb{R}$ of the expression e is contained in the result of approximation scheme evaluated for the enclosure XS. We could call this the fundamental property of rigorous numerics.

$$xs \in XS \longrightarrow [\![e]\!]_{xs} \in \textit{approx } e \; XS$$

4.1.1 Expressions for Euclidean Space

Expressions of type *aexp* are interpreted as real numbers. But we would like to do rigorous numerics for vectors of real numbers $\mathbb{R}^n$ or more general, elements of an arbitrary type $\alpha :: \textit{euclidean-space}$. We work with the type class *executable-euclidean-space*, which extends *euclidean-space* by fixing an order on the Basis elements and therefore enables operations *eucl-of-list* $: \mathbb{R}$ *list* $\to \alpha$ and *list-of-eucl* $: \alpha \to \mathbb{R}$ *list*. Then we can simply interpret a list of

aexp =	*Add aexp aexp*	$[\![\mathit{Add}\ a\ b]\!]_{vs} =$	$[\![a]\!]_{vs} + [\![b]\!]_{vs}$
|	*Mult aexp aexp*	$[\![\mathit{Mult}\ a\ b]\!]_{vs} =$	$[\![a]\!]_{vs} \cdot [\![b]\!]_{vs}$
|	*Minus aexp*	$[\![\mathit{Minus}\ a]\!]_{vs} =$	$-[\![a]\!]_{vs}$
|	*Inverse aexp*	$[\![\mathit{Inverse}\ a]\!]_{vs} =$	$1/[\![a]\!]_{vs}$
|	*Cos aexp*	$[\![\mathit{Cos}\ a]\!]_{vs} =$	$\cos([\![a]\!]_{vs})$
|	*Arctan aexp*	$[\![\mathit{Arctan}\ a]\!]_{vs} =$	$\arctan([\![a]\!]_{vs})$
|	*Abs aexp*	$[\![\mathit{Abs}\ a]\!]_{vs} =$	$\lvert[\![a]\!]_{vs}\rvert$
|	*Max aexp aexp*	$[\![\mathit{Max}\ a\ b]\!]_{vs} =$	$\max([\![a]\!]_{vs}, [\![b]\!]_{vs})$
|	*Min aexp aexp*	$[\![\mathit{Min}\ a\ b]\!]_{vs} =$	$\min([\![a]\!]_{vs}, [\![b]\!]_{vs})$
|	*Pi*	$[\![\mathit{Pi}]\!]_{vs} =$	π
|	*Sqrt aexp*	$[\![\mathit{Sqrt}\ a]\!]_{vs} =$	$\sqrt{[\![a]\!]_{vs}}$
|	*Exp aexp*	$[\![\mathit{Exp}\ a]\!]_{vs} =$	$e^{[\![a]\!]_{vs}}$
|	*Powr aexp aexp*	$[\![\mathit{Powr}\ a\ b]\!]_{vs} =$	$([\![a]\!]_{vs})^{[\![b]\!]_{vs}}$
|	*Ln aexp*	$[\![\mathit{Ln}\ a]\!]_{vs} =$	$\ln([\![a]\!]_{vs})$
|	*Power aexp* $\mathbb{N}$	$[\![\mathit{Power}\ a\ n]\!]_{vs} =$	$([\![a]\!]_{vs})^{n}$
|	*Floor aexp*	$[\![\mathit{Floor}\ a]\!]_{vs} =$	$\lfloor[\![a]\!]_{vs}\rfloor$
|	*Num* $\mathbb{R}$	$[\![\mathit{Num}\ r]\!]_{vs} =$	r
|	*Var* $\mathbb{N}$	$[\![\mathit{Var}\ i]\!]_{vs} =$	$vs\ !\ i$

Figure 4.1: Data type of arithmetic expressions and interpretation

aexp expressions es : *aexp list* componentwise to obtain an element of Euclidean space with *eucl-of-list*, i.e., $[\![es]\!]_{vs} :: \alpha$.

$$[\![es]\!]_{vs} = \mathit{eucl\text{-}of\text{-}list}\,(\mathit{map}\ (\lambda e.\ [\![e]\!]_{vs})\ es)$$

In contrast to the componentwise interpretation, the approximation of a list of expressions should not be componentwise: an approximation function *approx* for lists of expressions should be typed *approx* :: *aexp list* $\to$ $\mathbb{R}$ *list set*, which allows *approx* to keep track of dependencies between the components of the result.

4.1.2 Derivatives

The derivatives with respect to one variable can be computed symbolically from the structure of the expression. That is, for an expression e :: *aexp* one can compute (recursively) an expression $\frac{\partial e}{\partial i}$ :: *aexp* which is interpreted as the derivative with respect to the i-th variable (if this derivative exists).

Lemma 4.1 (Symbolic Partial Derivative)**.**

$$[\![\frac{\partial e}{\partial i}]\!]_{xs} = \frac{\partial\,[\![e]\!]_{xs}}{\partial xs_i}$$

If we consider a list of expressions es :: *aexp list* as a function on Euclidean space, i.e., according to $f_{es} := \lambda x.\ [\![es]\!]_{\mathit{list\text{-}of\text{-}eucl}\ x}$, then the total derivative of f_{es} can be represented by the Jacobian matrix of the partial derivatives of f_{es}. We can represent matrices as a

flat list (according to *eucl-of-list / list-of-eucl*, because matrices are also elements of the type class *euclidean-space*). For computing derivatives, however, we directly produce a list of expressions $\mathsf{D}(es, vs)$ :: *aexp list* that is interpreted as the product of the derivative matrix with a vector.

Definition 4.2 (Total Derivative of Expressions)**.**

$$(\mathsf{D}(es, vs))_i = \mathit{Sum}_{[0..<\mathit{length}\ vs]}(\lambda j.\ \mathit{Mult}\ (\frac{\partial\, es_i}{\partial j})\ vs_j)$$

In this definition, $[0..<n]$ denotes the (sorted) list of natural numbers less than n. $\mathit{Sum}_{xs}\ f$ is the *aexp* expression corresponding to the sum of all elements in xs after application of f, i.e., $\mathit{Sum}_{[]}\ f = \mathit{Num}\ 0$ and $\mathit{Sum}_{x\#xs}\ f = \mathit{Add}\ (f\ x)\ (\mathit{Sum}_{xs}\ f)$. The definition is chosen such that the interpretation of the resulting list of expressions $\mathsf{D}(es, vs)$ corresponds to the total derivative of the function f_{es} encoded by the list of expressions es multiplied with the vector encoded by vs.

Lemma 4.3 (Total Derivative of Expressions)**.**

$$[\![\mathsf{D}(es, vs)]\!]_{xs} = \mathsf{D}f_{es}|_{\mathit{at}\ (\mathit{eucl\text{-}of\text{-}list}\ xs)} \cdot [\![vs]\!]_{xs}$$

This way, we can produce expressions for higher derivatives as e.g., used in the multivariate Taylor series expansion (section 7.2.1):

$$\begin{aligned}\mathsf{D}^0(es, vs) &= es\\ \mathsf{D}^{i+1}(es, vs) &= \mathsf{D}(\mathsf{D}^i(es, vs), vs)\end{aligned}$$

Note that the proper interpretation can only be written down in Isabelle's type system for fixed values of i: the resulting object is an i-linear function, so the resulting type depends on a term argument. This could also be encoded as functions taking lists as arguments, but fixed values of i suffice for our purposes. As an example, this is the interpretation of the derivatives up to second order.

$$\begin{aligned}[\![\mathsf{D}^0(es, vs)]\!]_{xs} &= [\![es]\!]_{xs}\\ [\![\mathsf{D}^1(es, vs)]\!]_{xs} &= \mathsf{D}f_{es}|_{\mathit{at}\ (\mathit{eucl\text{-}of\text{-}list}\ xs)} \cdot_{bl} [\![vs]\!]_{xs}\\ [\![\mathsf{D}^2(es, vs)]\!]_{xs} &= \mathsf{D}(\lambda y.\ \mathsf{D}f|_{\mathit{at}\ y})|_{\mathit{at}\ (\mathit{eucl\text{-}of\text{-}list}\ xs)} \cdot_{bl} [\![vs]\!]_{xs} \cdot_{bl} [\![vs]\!]_{xs}\end{aligned}$$

4.1.3 Straight Line Programs

Often times, expression contain common subexpressions. This is not desirable because one would need to perform redundant computations. We therefore follow Dorel and Melquiond's [96] approach and compile plain *aexp* expressions to straight-line programs with static single assignment.

For us, a straight line program is just a list of arithmetic expressions, which is interpreted according to function $\mathit{slp} : \mathit{aexp\ list} \to \mathbb{R}\ \mathit{list} \to \mathbb{R}\ \mathit{list}$:

$$\begin{aligned}\mathit{slp}\ []\ \ \ \ \ \ vs &= vs\\ \mathit{slp}\ (e\#es)\ vs &= \mathit{slp}\ es\ ([\![e]\!]_{vs}\#vs)\end{aligned}$$

The idea is that a straight line program only contains unary or binary operations which put the result on top of the evaluation stack. The following example illustrates sharing the term $x + y$ in a computation of $(x + y) \cdot (x + y)$:

$$\mathit{slp}\ [\mathit{Add}\,(\mathit{Var}\ 0)(\mathit{Var}\ 1), \mathit{Mult}\,(\mathit{Var}\ 0)(\mathit{Var}\ 0)]\ [x, y] = [(x + y)(x + y), x + y, x, y]$$

We provide a function *slp-of*, which eliminates common subexpressions by traversing an expression bottom-up and saving subexpressions in a map that gives the index of the subexpression in the resulting straight line program. It is proved to be correct in the sense that the interpretation of the first element of the resulting list equals the interpretations of the original expression.

Lemma 4.4 (Correctness of *slp-of*)**.**

$$(\mathit{slp}\ (\mathit{slp\text{-}of}\ e)\ vs)_0 = [\![e]\!]_{vs}$$

4.2 Numerics

We begin with discussing the representation of real numbers in calculations with finite precision. Our numerical algorithms are specified to operate on real numbers $\mathbb{R}$, which makes verification convenient. Concrete computations for type $\mathbb{R}$ are carried out only on the subset that is representable with software floating point numbers $m \cdot 2^e$ for (unbounded) integers $m, e \in \mathbb{Z}$, which is cast as a subtype of the real numbers:

$$\textbf{typedef}\ \mathbb{F} = \{m \cdot 2^e :: \mathbb{R} \mid m, e \in \mathbb{Z}\}$$

This yields an injective function $(\cdot)_{\mathbb{R}} :: \mathbb{F} \to \mathbb{R}$ and its partially specified inverse $(\cdot)_{\mathbb{F}} :: \mathbb{R} \to \mathbb{F}$. Arithmetic constants 0, 1, $+$, $-$, $\cdot$, $<$, $\leq$ are lifted from the respective ones on the real numbers, so are type class instantiations (in particular as linearly ordered ring).

For an executable view on $\mathbb{F}$, we introduce a (non-injective) constructor $\mathit{Float}\ m\ e = (m \cdot 2^e)_{\mathbb{F}}$ and declare it as a datatype constructor for code generation [51]. This is a standard approach of light-weight data refinement [54]. Operations on type $\mathbb{F}$ are then implemented with concrete operations on arbitrary precision integers, for example multiplication:

$$(\mathit{Float}\ m_1\ e_1) \cdot (\mathit{Float}\ m_2\ e_2) = \mathit{Float}\ (m_1 \cdot m_2)\ (e_1 + e_2).$$

All further operations, including explicit round-off operations are defined in terms of real numbers and are mapped to the corresponding executable functions on floating point numbers by the code generator.

This also includes round-off operations. They are introduced as explicit operations in our formalization. We have functions $\mathit{trunc}^+ :: \mathbb{N} \to \mathbb{R} \to \mathbb{R}$ and $\mathit{trunc}^- :: \mathbb{N} \to \mathbb{R} \to \mathbb{R}$, which return the floating point number rounded with a precision of p bits either upwards or downwards:

Definition 4.5.

$$\mathit{trunc}^-\ p\ x := \lfloor x \cdot 2^{p - \lfloor \log_2 |x| \rfloor} \rfloor \cdot 2^{p - \lfloor \log_2 |x| \rfloor}$$
$$\mathit{trunc}^+\ p\ x := \lceil x \cdot 2^{p - \lfloor \log_2 |x| \rfloor} \rceil \cdot 2^{p - \lfloor \log_2 |x| \rfloor}$$

These operations can be implemented with software floating point numbers, e.g., downwards rounding:

Lemma 4.6 (Implementation of trunc^-)**.**

$$\begin{aligned}&\mathit{trunc}^-\ p\ (\mathit{Float}\ m\ e) = \\ &\quad \textbf{\textit{let}}\ d = \log_2 |m| - p - 1\ \textbf{\textit{in}} \\ &\quad \textbf{\textit{if}}\ 0 < d \\ &\quad \textbf{\textit{then}}\ \mathit{Float}\ (m \div 2^d)\ (e + d) \\ &\quad \textbf{\textit{else}}\ \mathit{Float}\ m\ e\end{aligned}$$

For our purposes of computing safe enclosures, however, the precise definition or precision of the rounding operations is not important. All that matters is that trunc^+ and trunc^- return a safe enclosure:

Theorem 4.7 (Safe Rounding)**.**

$$\mathit{trunc}^-\ p\ x \leq x \leq \mathit{trunc}^+\ p\ x$$

4.3 Affine Arithmetic

Up to now, we kept the discussion on the level of expressions, let us now motivate affine arithmetic as a concrete approximation scheme.

The most basic approximation scheme is interval arithmetic [102] where the real quantities are enclosed in intervals. A problem of interval arithmetic is that dependencies between variables are lost, e.g. for an enclosure $x \in [0;2]$, the expression $x - x$ evaluates to $[-2;2]$ in interval arithmetic whereas the exact result would be representable as the interval $[0;0]$.

Another problem, the wrapping effect, occurs when computing with multidimensional intervals – which are Cartesian products of intervals. It can be visualized already in the two-dimensional case. Assume a rectangle $[p;q] = [p_x;q_x] \times [p_y;q_y]$ as in figure 4.2 and a function rotating the rectangle. When using interval arithmetic, the rotated rectangle needs to be enclosed in a Cartesian product $[r_x;s_x] \times [r_y;s_y]$ of intervals again, leading to large overapproximations, in particular for iterated applications.

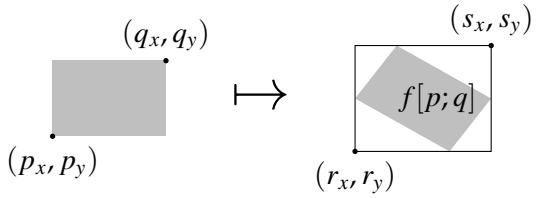


Figure 4.2: Wrapping effect

4.3.1 Tracking Linear Dependencies

Affine arithmetic [34, 125] improves over interval arithmetic by tracking linear dependencies. The data structure that underlies affine arithmetic is a formal sum

$$A_0 + \sum_i \varepsilon_i A_i$$

where the ε_i are treated as formal parameters. Such a formal sum is called affine form. Abstractly, an affine form A is a function where only finitely many arguments map to nonzero values. We consider them as a separate type:

$$\textbf{typedef } \mathit{affine\text{-}form} := \{a :: \mathbb{N} \to \mathbb{R} \mid \mathit{finite}\ \{i \mid a\ i \neq 0\}\}$$

We write the i-th element of an affine form $A :: \mathit{affine\text{-}form}$ as A_i. An affine form is interpreted for a valuation $\varepsilon : \mathbb{N} \to \mathbb{R}$ of the formal parameters with the function *affine*.

$$\mathit{affine}\ \varepsilon\ A := A_0 + \sum_i \varepsilon_i \cdot A_i$$

Looking at the interpretation, one often calls the terms ε_i noise symbols, A_0 the center, and the remaining A_i generators. The idea is that noise symbols are shared between affine forms and that they are treated symbolically: the sum of two affine forms is given by the pointwise sum of their generators, and scalar multiplication with a constant factor is also done componentwise.

Definition 4.8 (Addition and Scalar Multiplication of Affine Forms)**.**

$$(A + B)_i := A_i + B_i$$
$$(c \cdot A)_i := c \cdot A_i$$

With these definitions, the sum and scalar multiplication are interpreted as the sum and scalar multiplication of the respective interpretations:

Lemma 4.9.

$$\mathit{affine}\ \varepsilon\ (A + B) = (A_0 + B_0) + \sum_i \varepsilon_i \cdot (A_i + B_i) = \mathit{affine}\ \varepsilon\ A + \mathit{affine}\ \varepsilon\ B$$
$$\mathit{affine}\ \varepsilon\ (c \cdot A) = c \cdot A_0 + \sum_i \varepsilon_i \cdot (c \cdot A_i) = c \cdot \mathit{affine}\ \varepsilon\ A$$

Affine forms are used to represent sets. By convention, the *range* of an affine form is the set of all *affine* evaluations where the noise symbols range in $[-1; 1]$. For the range of a list of affine forms, those are evaluated jointly for the same valuation of the noise symbols, reflecting the intuition that those are shared.

$$\mathit{range}\ A := \{\mathit{affine}\ \varepsilon\ A \mid -1 \leq \varepsilon_i \leq 1\}$$
$$\mathit{joint\text{-}range}\ AS := \{\mathit{map}\ (\mathit{affine}\ \varepsilon)\ AS \mid -1 \leq \varepsilon_i \leq 1\}$$

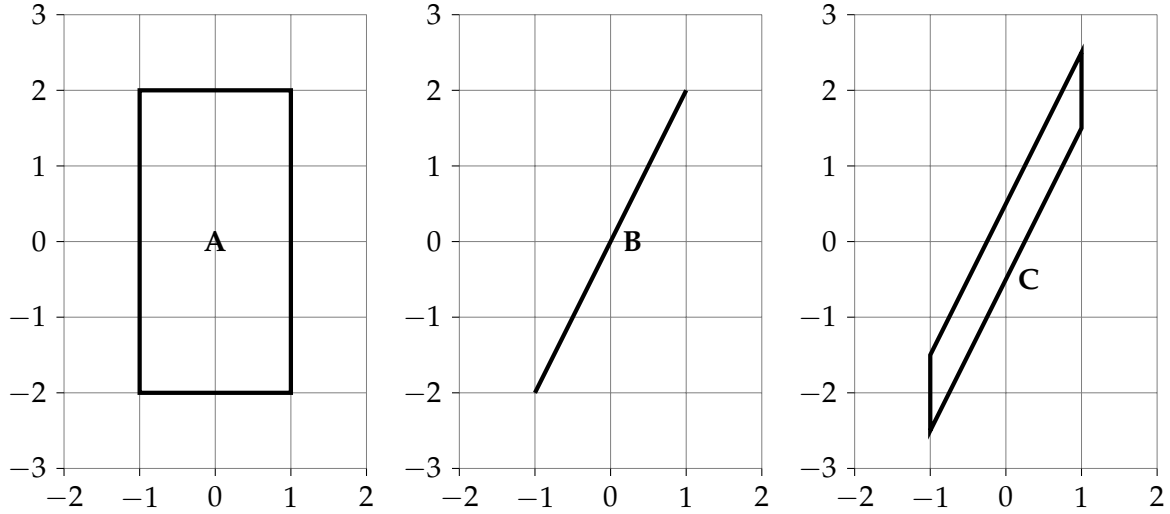


Figure 4.3: Illustration of *joint-range* for (left) $A = [\varepsilon_1, 2\varepsilon_2]$, (middle) $B = [\varepsilon_1, 2\varepsilon_1]$, (right) $C = [\varepsilon_1, 2\varepsilon_1 + 0.5\varepsilon_2]$

The joint range is used to represent multi-dimensional sets with shared dependencies. For example, $2\varepsilon_1$ and $2\varepsilon_2$ both represent the same set $\mathit{affine}\ (2\varepsilon_1) = \mathit{affine}\ (2\varepsilon_2) = [-2; 2]$, but for example the joint range of $[\varepsilon_1, 2\varepsilon_2]$ is the rectangle A in figure 4.3 whereas the joint range of $[\varepsilon_1, 2\varepsilon_1]$ is the line segment B from $[-1, -2]$ to $[1, 2]$. A third example is the parallelogram C represented by the joint range of $[\varepsilon_1, 2\varepsilon_1 + 0.5\varepsilon_2]$

As a concrete example, let us examine how affine arithmetic handles the dependency problem in the introductory example $x - x$ for $x \in [0; 2]$. The interval $[0; 2]$ is represented by the affine form $1 + 1 \cdot \varepsilon_1$. This is the affine form represented by the function $X := (\lambda i.\ \textbf{if}\ i = 0 \vee i = 1\ \textbf{then}\ 1\ \textbf{else}\ 0)$. For this function, $\mathit{range}\ X = [0, 2]$ holds. Then, in affine arithmetic, $(1 + 1 \cdot \varepsilon_1) - (1 + 1 \cdot \varepsilon_1) = 0 + 0 \cdot \varepsilon_1$, which corresponds to the constant zero function. Therefore $\mathit{range}\ (X - X) = \{0\}$.

4.3.2 Multiplication

Multiplication is not a linear operation. The idea for an affine arithmetic definition of multiplication is to return a linear approximation and account for the linearization error with a fresh noise symbol. Consider the multiplication of two affine forms with formal parameters ε.

$$(\mathit{affine}\ \varepsilon\ A) \cdot (\mathit{affine}\ \varepsilon\ B) = A_0 B_0 + \sum_i \varepsilon_i \cdot (A_0 B_i + A_i B_0) + (\sum_{i>0} \varepsilon_i A_i) \cdot (\sum_{i>0} \varepsilon_i B_i)$$

The resulting term is not an affine form, because the last summand on the right contains non-linear terms (i.e., products of ε_i). But if we assume a proper valuation with $\varepsilon_i \in [-1; 1]$, the last summand on the right can be bounded by $(\sum_{i>0} |A_i|)(\sum_{i>0} |B_i|)$. Therefore, if k is fresh in A and B, we can define $A \cdot B$ such that

$$\mathit{affine}\ \varepsilon\ (A \cdot B) := A_0 B_0 + \sum_i \varepsilon_i \cdot (A_0 B_i + A_i B_0) + \varepsilon_k \cdot ((\sum_{i>0} |A_i|) \cdot (\sum_{i>0} |B_i|))$$

the k-th generator bounds the linearization error. As a result, multiplication of affine forms is a safe overapproximation:

Theorem 4.10 (Multiplication of Affine Forms)**.**

$$[a, b] \in \textit{joint-range}\ [A, B] \longrightarrow a \cdot b \in \textit{range}\ (A \cdot B)$$

4.3.3 Conversions to and from Intervals

The sum of the absolute values of all generators $\sum_i |A_i|$ is called the total deviation of an affine form. With it, we can define lower and upper bounds inf, sup of an affine form $A :: \textit{affine-form}$ and elementwise of a list of affine forms $AS :: \textit{affine-form list}$.

Definition 4.11 (Interval Bounds of Affine Form)**.**

$$\inf\ A := A_0 - \sum_i |A_i|$$

$$\sup\ A := A_0 + \sum_i |A_i|$$

$$\inf\ AS := \textit{map}\ \inf\ AS$$

$$\sup\ AS := \textit{map}\ \sup\ AS$$

Intervals with bounds given by inf and sup do indeed enclose the *range* respectively *joint-range*. In the latter case the interval is to be interpreted as a Cartesian product of intervals and represents a hypercube.

Lemma 4.12 (Correctness of Interval Bounds)**.**

$$\textit{range}\ A \subseteq [\inf\ A; \sup\ A]$$

$$\textit{joint-range}\ AS \subseteq [\inf\ AS; \sup\ AS]$$

We can also go in the reverse direction, defining an affine form that represents an interval $[l; u] :: \mathbb{R}\ \textit{set}$.

Definition 4.13 (Affine Form of Interval)**.**

$$\textit{affine}\ \varepsilon\ (\textit{affine-of-ivl}\ i\ l\ u) := \frac{l+u}{2} + \frac{u-l}{2}\varepsilon_i$$

This yields a range equal to the given interval.

Lemma 4.14 (Correctness of Affine Form of Interval)**.**

$$\textit{range}\ (\textit{affine-of-ivl}\ i\ l\ u) = [l; u]$$

This works for any index i for the formal parameter. This index becomes relevant for the joint range. For a hyperrectangle $[ls; us]$ for $ls, us :: \mathbb{R}\ \textit{list}$, different formal parameters need to be chosen for every dimension.

Definition 4.15. *For* $i < \textit{length}\ ls = \textit{length}\ us$*:*

$$(\textit{affines-of-ivls}\ ls\ us)_i := \textit{affine-of-ivl}\ i\ (ls_i)\ (us_i)$$

This ensures that the corresponding *joint-range* equals the hyperrectangle $[ls; us]$.

Lemma 4.16.

$$\textit{joint-range}\ (\textit{affines-of-ivls}\ ls\ us) = [ls; us]$$

4.3.4 Minkowski Sum

The Minkowski sum $X \oplus Y = \{x + y.\ x \in X \wedge y \in Y\}$ can easily be implemented for affine forms. Given $A :: \textit{affine-form}$ and $B :: \textit{affine-form}$, we first choose n such that $\forall i \geq n.\ A_i = 0$. That is, every noise symbol with index $\geq n$ is fresh in A. We then define C as an affine form representing the same set as B, but with noise symbols independent from A:

$$C_{i<n} = 0$$
$$C_{i\geq n} = B_{n-i}$$

This yields $\textit{range}\ C = \textit{range}\ B$. Because the generators in C are independent from A, the sum $A + C$ represents the Minkowski sum:

Lemma 4.17 (Minkowski Sum of A and B)**.**

$$\textit{range}\ (A + C) = \textit{range}\ A \oplus \textit{range}\ B$$

Note that because of possibly shared noise symbols, $\textit{range}\ (A + B)$ does not necessarily equal $\textit{range}\ A \oplus \textit{range}\ B$.

4.3.5 Round-off Operations

All the operations presented up to now are in exact arithmetic. For efficiency reasons, we would like to work with a finite precision p, which means that we have to take round-off errors into account. Our approach is to provide an explicit round-off operation *trunc-err-affine* to round all generators of the affine form to some given precision p with $\textit{trunc}^-\ p$ (from section 4.2).

Definition 4.18.

$$(\textit{truncate-affine}\ p\ A)_i := \textit{trunc}^-\ p\ A_i \tag{4.1}$$

The variant *trunc-err-affine* returns an overapproximation of the incurred round-off errors, which we need to take into account if we want to retain safe enclosures.

Definition 4.19.

$$\textit{trunc-err-affine}\ p\ A := (\textit{truncate-affine}\ p\ A, \textit{trunc}^+\ p\ (\sum_i |A_i - \textit{trunc}^-\ p\ A_i|)$$

Correctness of *trunc-err-affine* states that the resulting affine form B encloses the original one with some uncertainty e.

Theorem 4.20 (Rounding Generators of an Affine Form)**.**

$$\textit{trunc-err-affine}\ p\ A = (B, e) \longrightarrow \textit{range}\ A \subseteq \textit{range}\ B \oplus [-e, e]$$

4.3.6 Extended Affine Forms

One optimization, that is usually only hinted at in descriptions of how to efficiently implement affine arithmetic is the following: when approximating a complicated expression using affine arithmetic, more and more noise symbols will appear to account for linearization and round-off errors. Since all of them are independent, it does not make sense to keep distinct noise symbols for each of them. One can rather accumulate them in a special error symbol. For this, we work with affine forms A with an associated error e, which we simply write as a pair $(A, e) :: \mathit{affine\text{-}form} \times \mathbb{R}$ and call it *extended affine form*.

We will use extended affine forms as main data structure to approximate expressions. An extended affine form is interpreted with $\mathit{affine\text{-}err}$ for a valuation ε as the interval that stems from the exact evaluation of the affine form with the uncertainty from the additional error term:

Definition 4.21 (Interpretation of Extended Affine Form)**.**

$$\mathit{affine\text{-}err}\ \varepsilon\ (A, e) := [\mathit{affine}\ \varepsilon\ A - e, \mathit{affine}\ \varepsilon\ A + e]$$

4.3.7 Approximation of Elementary Operations

For extended affine forms, we provide the arithmetic operations addition, multiplication, combined with an immediate round-off operation. Consider e.g., addition, where argument affine forms A with associated error e_1 and B with error e_2 are first added exactly $(A + B)$ and then truncated elementwise to C with associated error e_3. For the result, all errors are accumulated in the second component.

Definition 4.22.

$$\begin{aligned}&\mathit{add\text{-}affine}\ p\ (A, e_1)\ (B, e_2) = \\ &\quad \textbf{\textit{let}}\ (C, e_3) = \mathit{trunc\text{-}err\text{-}affine}\ p\ (A + B); \\ &\quad \textbf{\textit{in}}\ (C, \mathit{trunc}^+\ p\ (e_1 + e_2 + e_3))\end{aligned}$$

Correctness of operations on extended affine forms states that if the arguments are in the range of the arguments, then the result of the "ideal" operation is in the range resulting from the operation on affine forms. Moreover the dependencies of the formal variables stay intact, i.e., the valuation ε remains the same. In the example of addition:

Theorem 4.23 (Correctness of Addition)**.**

$$x \in \mathit{affine\text{-}err}\ \varepsilon\ (X, e_X) \longrightarrow y \in \mathit{affine\text{-}err}\ \varepsilon\ (Y, e_Y) \longrightarrow x + y \in \mathit{affine\text{-}err}\ \varepsilon\ (\mathit{add\text{-}affine}\ X\ Y)$$

We proved similar correctness theorems for multiplication $\mathit{mult\text{-}affine}$ and unary minus. We guided our implementation by the descriptions of de Figueiredo and Stolfi [34].

4.3.8 Approximation of Standard Functions

We also provide affine arithmetic approximations for standard functions. Most of them can be approximated with so-called *min-range approximations*. We describe the general setup, how it is used for monotone standard functions like division, $\exp, \ln, \arctan, \sqrt{}$. With a bit of further efforts, this can also be used for periodic functions like $\sin, \cos$.

4.3.8.1 Generic Linear Operation With Rounding

As basis for all operations that follow, we use a generic linear operation that involves round-off operations and also adds a noise symbol for further uncertainties (this is also what Stolfi and de Figueredo [34] discuss in detail):

We assume a generic affine operation given by parameters a, b and some uncertainty d. The idea is that $a \cdot x + b$ approximates some (possibly nonlinear) function f up to an error d. This idea is captured in the following theorem. We give the implementation of *affine-unop* immediately afterwards.

Theorem 4.24 (Correctness of *affine-unop*)**.**

$$(x \in \textit{affine-err}\ \varepsilon\ (X, e_X) \wedge |f(x) - (a \cdot x + b)| \leq d) \longrightarrow$$
$$f(x) \in \textit{affine-err}\ \varepsilon\ (\textit{affine-unop}\ \ p\ a\ b\ d\ (X, e_X))$$

affine-unop is defined by performing exact operations $a \cdot X$, $A + b$ and rounding in between, while accumulating all errors into the additional error term of the resulting extended affine form (also respecting the initial uncertainty e_X and linearization error d):

$$\begin{aligned}\textit{affine-unop}\ \ p\ a\ b\ d\ (X, e_X) :=& \textbf{let}\\ &(A, e_A) = \textit{trunc-err-affine}\,(a \cdot X);\\ &(B, e_B) = \textit{add-affine}\ A\ b\\ &\textbf{in}\ (B, \textit{trunc}^+\ p\ (|a|e_X + e_A + e_B + d))\end{aligned}$$

4.3.8.2 Min-Range Approximation

There are various degrees of freedom for linearizing a non-linear function f on an interval $[l; u]$ using an affine form with a generic linear operation *affine-unop*: a, b, d must be chosen such that $\|f(x) - (\alpha \cdot x + b)\| \leq d)$ holds. Three approaches are commonly used in the literature: One, setting $a = 0$, which results in an interval approximation the function f. Two, minimizing the error d, which results in the first-order Tchebychev approximation. Three, the so-called *min-range approximation*, which is in some sense a good compromise between the two other options:

The error of the Tchebychev approximation is quadratic in the size of the interval X, (which is desirable, because one usually works with sufficiently small uncertainties), for the interval approximation the error is only linear. For monotone functions, the Tchebychev approximation has one drawback: the range covered by the approximation can be larger than that of the original function (which is not the case for the interval approximation). The min-range approximation combines the advantages of both other approximation schemes: its error is quadratic in the size of the argument interval and its range is the same as that of the interval approximation.

The idea behind the min-range approximation (for monotone functions) is to maximize the slope of the enclosure while fixing the range of the approximation. Consider figure 4.4, which depicts a min-range approximation of sin on the interval $[0; \frac{\pi}{3}]$: It does not exceed the interval $[\sin(0); \sin(\frac{\pi}{3})]$. Any smaller slope would be just as safe, but the slope could not be chosen larger (at $x = \frac{\pi}{3}$ the upper bound is tangential to $\sin(x)$).

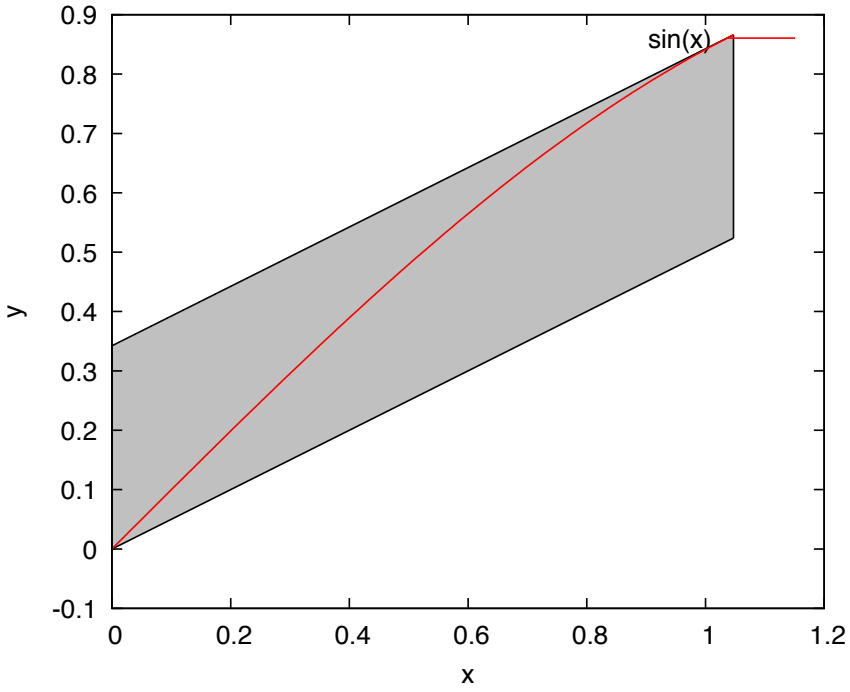


Figure 4.4: Min-range approximation of $\sin[0; \frac{\pi}{3}]$

The abstract justification of the min-range approximation is given by the following lemma for a differentiable function f (with derivative f'). a needs to be a lower bound on the derivative and b and d need to be chosen such that it accounts for the error of the linear function centered between $f(u)$ and $f(l)$ as well as for the error that b makes with respect to the center (the second summand on the right of the inequality in condition 2 of lemma 4.25.

Lemma 4.25 (Justification for Min-Range Approximation)**.**

$$\forall x \in [l; u].\ |f(x) - (a \cdot x + b)| \leq d$$

if the following conditions are satisfied:

1. $\forall y \in [l; u].\ a \leq f'(y)$
2. $d \geq \frac{f(u) - f(l) - a \cdot (u - l)}{2} + |(f(l) + f(u) - a \cdot (l + u))/2 - b|$

For a concrete implementation of a min-range approximation for a function f on an interval $[l; u]$, we rely on interval arithmetic extensions F, F' of f and its derivative f'. To remain safe, we need to choose for a the lower bound of the interval arithmetic evaluation of the derivative (in lemma 4.25, a needs to be a lower bound). For d, we need to choose the upper bound for a safe over-approximation. For b it does not really matter (we choose the mid-point) as long as we respect the error e_b.

$$\begin{aligned}\textit{min-range-aform}\,(F, F', (X, e_X)) := \textbf{let}\\ [l;u] &= [\inf(X, e_X); \sup(X, e_X)]\\ [a;\_] &= \min(F'(l), F'(u));\\ [b - e_b; b + e_b] &= \frac{F(l) + F(u) - \alpha \cdot (l + u)}{2}\\ [\_;d] &= \frac{F(u) - F(l) - a \cdot (u - l) + e_\beta}{2}\\ \textbf{in}\ \textit{affine-unop}\,(a, b, d, (X, e_X))\end{aligned}$$

For the following correctness theorem, we assume that the extended affine form (X, e_X) is bounded by an interval $[l;u]$. The derivative f' attains its maximum at one of the endpoints of the interval. This is a slight generalization to what is demanded in the literature [34, 125] where one assumes a convex function f. This is particularly useful for sin and cos, since they are not convex, but satisfy the former assumption. Moreover we assume valid interval extensions F, F' of f, f'. Then the resulting extended affine form is a safe enclosure for f:

Lemma 4.26 (Correctness of Min-Range Approximation)**.**

$$\forall x \in \textit{affine-err}\ \varepsilon\ (X, e_X).\ f(x) \in \textit{affine-err}\ \varepsilon\ (\textit{min-range-aform}\,(F, F', (X, e_X)))$$

if the following conditions are satisfied:

1. $\textit{ivl}\,(X, e_X) = [l;u]$
2. $\forall x \in [l;u].\ \min(f'(l), f'(u)) \leq f'(x)$
3. $\forall x \in [l;u].\ f(x) \subseteq F([l;u])$
4. $\forall x \in [l;u].\ f'(x) \subseteq F'([l;u])$

Something that is not mentioned in the theorems here should be noted: *min-range-aform* only produces an approximation that is close to the min-range approximation if the function f is monotone on $[l;u]$. But a similar version of *min-range-aform* is easily provided for the antitonic case.

With this setup (we rely on Hölzl's [66] library of interval arithmetic in Isabelle/HOL), we can provide Min-Range approximations for arctan, $\sqrt{\ }$, exp, and the natural logarithm ln.

4.3.8.3 Trigonometric Functions

The trigonometric functions cos and sin pose the problem that they are not monotonic. This can be alleviated in two steps:

The first step, range reduction, exploits periodicity to reduce the argument to the range $[0; 2\pi]$. Range reduction (shifting the argument x by $-2\pi \cdot \lfloor \frac{x}{2\pi} \rfloor$) is computed using interval arithmetic. If the argument after reduction is not contained in $[0; 2\pi]$ (which might be due to imprecise interval arithmetic or an argument interval that is larger than 2π), we simply return the interval approximation of cos on the reduced argument.

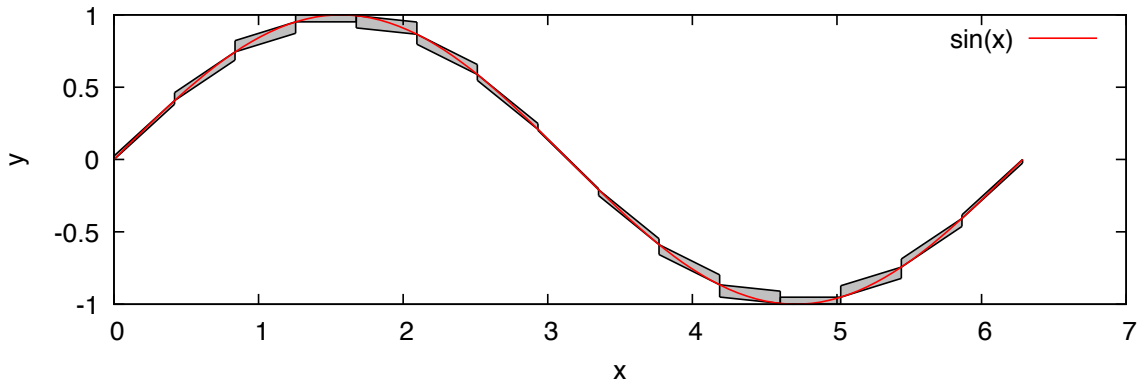


Figure 4.5: Min-range approximations (fifteen subdivisions) of $\sin[0;2\pi]$

The second step is a case distinction if the argument is contained in the antitone part $[0;\pi]$ or the monotone part $[\pi;2\pi]$. It is possible that this distinction cannot be decided (if e.g., the argument interval straddles π), but then again, the only valid min-range approximation is the interval approximation (with 1 as upper bound). A series of such computed min-range approximations for $\sin(x)$ is shown in figure 4.5.

4.3.9 Approximation of Expressions

Arithmetic expressions *aexp* as given in section 4.1 can be approximated with affine arithmetic by recursively following the structure of the expression. Below we give addition and lookup of a variable as example but refrain from a presentation of further cases. Approximation is performed inside the option monad: *approx* either returns *Some* result or *None* to indicate failures like e.g., approximating the inverse of an affine form that contains zero or looking up an undefined variable.

$$
\begin{aligned}
&\mathit{approx}\ p\ (\mathit{Add}\ a\ b)\ \mathit{VS}\ l = \\
&\quad \textbf{do}\ A \leftarrow \mathit{approx}\ p\ a\ \mathit{VS} \\
&\qquad B \leftarrow \mathit{approx}\ p\ b\ \mathit{VS} \\
&\qquad \mathit{Some}\ (\mathit{add\text{-}affine}\ p\ A\ B)
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{approx}\ p\ (\mathit{Var}\ i)\ \mathit{VS}\ l = \\
&\quad \textbf{if}\ i < \mathit{length}\ \mathit{VS}\ \textbf{then}\ \mathit{Some}\ (\mathit{VS}_i)\ \textbf{else}\ \mathit{None}
\end{aligned}
$$

Variables in *approx* are looked up from a list of extended affine forms. This is interpreted for a valuation ε as a set of lists of real numbers. Correctness follows from an induction over the structure of the expression and states that the interpreted expression is enclosed in the resulting extended affine form for the same valuation of noise symbols.

Theorem 4.27 (Affine Arithmetic Approximation of Expressions)**.**

$$\begin{aligned}
&\mathit{approx}\ p\ \mathit{expr}\ \mathit{VS} = \mathit{Some}\ X \longrightarrow \\
&\varepsilon \in \mathit{UNIV}_{\mathbb{N}} \to [-1;1] \longrightarrow \\
&\mathit{length}\ \mathit{vs} = \mathit{length}\ \mathit{VS} \longrightarrow \\
&(\forall i < \mathit{length}\, \mathit{vs}.\ \mathit{vs}_i \in \mathit{affine\text{-}err}\ \varepsilon\ (\mathit{VS}_i)) \longrightarrow \\
&[\![\mathit{expr}]\!]_{\mathit{vs}} \in \mathit{affine\text{-}err}\ \varepsilon\ X
\end{aligned}$$

The usage of extended affine forms is a sensible choice for evaluating a single expression, but for iterative computations, the associated errors grow monotonically and do not share any dependencies. This is alleviated by incorporating the error terms into the affine form, therefore increasing the number of generators. Then those can e.g., be contracted in further iterations.

4.3.10 Summarizing Noise Symbols

During longer computations, more and more noise symbols will accumulate in an affine form, which impairs performance. The number of noise symbols can be reduced by summarizing (or condensing) them. There are several methods described e.g., by Althoff *et al.* [5], de Figueiredo and Stolfi [34], or Girard [41]. We implemented the latter two approaches, because they fit into the same framework and are easier to formalize (Althoff's approaches require to invert matrices).

The approaches that we formalize are both based on selecting a subset of the generators and enclosing them with a box (thereby discarding all the dependencies).

Definition 4.28.

$$\begin{aligned}
&(\mathit{select}\ P\ \mathit{XS})_i := \textbf{\textit{if}}\ P\ i\ \textbf{\textit{then}}\ (\mathit{XS})_i\ \textbf{\textit{else}}\ 0 \\
&(\mathit{summarize}\ P\ \mathit{XS}) := \mathit{select}\ P\ \mathit{XS} \oplus \mathit{box}(\mathit{select}\ (\neg P)\ \mathit{XS})
\end{aligned}$$

The approach of de Figueiredo and Stolfi is to choose P such that all generators with an absolute value smaller than a given fraction of the maximum deviation of the affine form are selected. For Girard's method, one selects a given number of generators which are most perpendicular.

For correctness, the concrete choice of P is irrelevant, the correctness theorem states that summarization returns a safe overapproximation:

Theorem 4.29 (Summarizing Noise Symbols)**.**

$$\mathit{range}\ X \subseteq \mathit{range}\ (\mathit{summarize}\ P\ X)$$

4.3.11 Splitting

Errors are introduced due to linearization of operations on affine forms. These errors are quadratic in the size of the generators, acceptable precision can therefore be maintained if the size of the generators is kept small. One means to achieve this is to split an affine form X into two smaller affine forms such that the union of the resulting ranges covers the original range. X can be split by halving the k-th generator X_k and moving the center X_0 accordingly:

$$\begin{aligned}
&\mathit{split}\ k\ X := (Y, Z)\\
&\quad \text{where}\\
&\quad (Y)_0 := a_0 - a_k/2\\
&\quad (Z)_0 := a_0 + a_k/2\\
&\quad (Y)_k := \tfrac{X_k}{2}\\
&\quad (Z)_k := \tfrac{X_k}{2}\\
&\quad (Y)_i := X_i \qquad (\forall 0 < i \neq k)\\
&\quad (Z)_i := X_i \qquad (\forall 0 < i \neq k)
\end{aligned}$$

The range of the resulting affine forms encloses the range of the argument, which follows from the definition of *range*.

Theorem 4.30 (Splitting)**.** $\mathit{split}\ k\ X = (Y, Z) \longrightarrow \mathit{range}(X) \subseteq \mathit{range}(Y) \cup \mathit{range}(Z)$

Except for trivial cases, after a split of X, the resulting ranges will overlap (that is, $\mathit{joint\text{-}range}\ Y \cap \mathit{joint\text{-}range}\ Z \neq \emptyset$). For concrete applications and performance reasons, one wants a split to be as effective as possible (i.e., as little overlap as possible). This depends on the choice of k. Althoff [5] discusses that a summarization of noise symbols (see section 4.3.10) before splitting yields more effective splits, however at the cost of overapproximation incurred by summarization. Althoff also discusses a performance index depending on the choice of the generator k along which the split is performed. We use a less sophisticated method and always split the longest generator after the sum of the lengths of all generators exceeds a configurable threshold, which worked well for our purposes.

4.3.12 Code Generation

We use data refinement via the code generator [54] to implement the type *affine-form* with association lists $(\mathbb{N} \times \mathbb{R})$ *list* that are (reverse) strictly sorted (with respect to the keys). This sparse representation is useful because the largest index of a non-zero generator can be directly read off by inspecting only the first element. Adding a fresh generator can be done by simply prepending the new element. Binary operations are efficiently and easily implemented by merging the two lists of generators.

4.4 Counterexample Generation

Based on the deep embedding of arithmetic expressions, I implemented an extension of Isabelle's quickcheck [108, 28] command. Quickcheck is used to test tentative lemmas by searching for counterexamples. This works for lemma statements involving executable

constructs (in particular data types). Such a method is also useful when proving facts about real valued functions, and works well for arithmetic involving $+, -, *, /$, because those can be executed on a representation of real numbers with rational numbers.

As soon as transcendental functions are involved, the standard setup does not work anymore. I extended quickcheck with an additional *generator* that can be invoked with `quickcheck[approximation]`, it is available after loading the theory `Approximation.thy`.

`quickcheck[approximation]` evaluates conjectures first with approximate floating point arithmetic (i.e., compiling real valued functions in HOL to the corresponding IEEE floating point implementations of PolyML). Once a potential counterexample is found in floating point arithmetic, `quickcheck[approximation]` tries to certify the counterexample with interval arithmetic (this avoids spurious counterexamples for formulae like $sin^2(x) + cos^2(x) = 1$). Another means to avoid spurious counterexamples is to declare the option `quickcheck_approximation_epsilon` $= \varepsilon$ such that left- and right-hand sides of counterexamples for equalities (or inequalities) have to differ by at least the given ε.

4.5 Related Work

Floating Point Numbers in ITPs. The formalization of arbitrary-precision floating-point numbers originates from Obua's work [112] and has been used for an implementation of interval arithmetic by Hölzl [66]. A specification of floating point numbers according to the IEEE-754 standard was formalized in Isabelle/HOL by Yu [145]. This was inspired by Harrison's [59] extensive formalization in HOL Light. In Coq, there is a comprehensive formalization of floating point numbers by Boldo and Melquiond [17], as well as some efforts in ACL2 [101].

Rigorous Numerics in ITPs. Muñoz and Lester [104] use rational interval arithmetic in PVS to efficiently approximate real valued functions. In addition to basic arithmetic operations they also support trigonometric functions. Dorel and Melquiond [96] implement a similar method in Coq. Similar to the presentation here, they use a representation for expressions and straight line programs and provide approximation functions using intervals, centered forms, and Taylor models.

Other formalizations of Taylor models in theorem provers have been in the context of Ariadne [33] and as a means for rigorous numerics [26, 146].

Solovyev [128] implements a global optimization method that is evaluated directly in HOL Lights kernel. He uses first and second derivatives to exploit monotonicity and convexity of the target function in order to reduce the search space for the global optimization.

Exact Real Arithmetic in ITPs. An alternative to rigorous numerics (i.e., explicit enclosures) is to perform arithmetic on a computable representation of real numbers. One can for example represent real numbers by a sequence of rational numbers. Each element in this sequence has a defined distance to the exact result. Harrison [56] uses this approach to compute the logarithm. O'Connor [114] approximates real numbers by organizing their completion of rational numbers in a monad. O'Connor and Spitters [113] use this monadic construction

in order to implement arbitrary approximations of the Riemann integral. Krebbers and Spitters [84, 83] extend this work to use arbitrary-precision floating-point numbers.

5 Computational Geometry

In the previous chapter 4, we discussed affine forms as our main data structure for representing enclosures. The geometric interpretation is that affine forms represent zonotopes (section 5.2.1). Intersection of zonotopes with hyperplanes is an important operation, e.g., for computing enclosures of Poincaré maps (where the Poincaré section is given as hyperplane). This chapter is about the verification of an approximative algorithm due to le Guernic and Girard [42] to compute enclosures for the intersection of zonotopes with hyperplanes.

Formal verification of geometric algorithms is a challenging and interesting topic: such algorithms are easily presented with the geometric intuition, but this intuition needs to be made precise formally. The core of the intersection algorithm that we will verify in this chapter is similar to convex hull algorithms for points in the two-dimensional plane. For such algorithms, Knuth [82] has developed a theory that axiomatizes the notion of orientation of points. The idea is that for three points p, q, r in the plane, visiting them in order requires either a counterclockwise (ccw) turn (written pqr) or clockwise ($\neg pqr$) turn. Knuth observed that already few of the properties fulfilled by the ccw predicate pqr suffice to define a theory rich enough to formalize many concepts in algorithmic geometry.

In contrast to Knuth's theory, which is targeted towards a finite, discrete set of points, our application is about sets in continuous vector spaces. We therefore extend Knuth's theory to continuous vector spaces.

The first part of this chapter is devoted to the presentation of Knuth's system of axioms (section 5.1), the standard instantiation for points in the plane (section 5.1.1), and the additional constraints that are needed to talk about ccw systems on vector spaces instead of discrete sets (section 5.1.2). In the second part (section 5.2), we use this theory to reason about the correctness of le Guernic and Girard's algorithm.

5.1 CCW System

Knuth introduces the notion of a *ccw system* as a set of points together with a ccw predicate written pqr for points p, q, r. In this section, we write (scalar) multiplication always explicitly as $x \cdot y$ to avoid potential confusion of a product $x \cdot y \cdot z$ a ccw predicate xyz. A ccw system has to satisfy the following properties (we call them axioms, following Knuth's terminology), inspired by the relations satisfied by points in the plane. For all axioms in the following, there is the additional implicit assumption that the involved points are pairwise distinct. For three points, only simple axioms need to be fulfilled:

Axiom 5.1 (Cyclic Symmetry)**.** $pqr \longrightarrow qrp$

Axiom 5.2 (Antisymmetry)**.** $pqr \longrightarrow \neg prq$

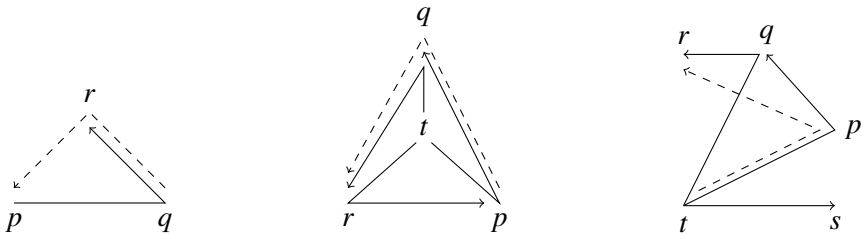


Figure 5.1: Cyclic symmetry (left), interiority (middle), transitivity (right); dashed predicates are implied by solid ones

Axiom 5.3 (Nondegeneracy)**.** $pqr \vee prq$

Cyclic symmetry and the more interesting case of interiority, which involves four points, are illustrated in figure 5.1. Interiority states that if one point t is left of three lines pq qr rp, then the three other points are oriented in a triangle according to pqr.

Axiom 5.4 (Interiority)**.** $tpq \wedge tqr \wedge trp \longrightarrow pqr$

The most important tool for reasoning is transitivity, which involves five points and works if three points p, q, r lie in the half-plane left of the line ts, i.e., $tsp \wedge tsq \wedge tsr$. Then, fixing t as first element for the ccw relation, we have transitivity in the second and third element: $tpq \wedge tqr \longrightarrow tpr$ (see figure 5.1).

Axiom 5.5 (Transitivity)**.**

$$tsp \wedge tsq \wedge tsr \wedge tpq \wedge tqr \longrightarrow tpr$$

The same intuition also holds for the other side of the half-plane:

Axiom 5.6 (Dual Transitivity)**.**

$$stp \wedge stq \wedge str \wedge tpq \wedge tqr \longrightarrow tpr$$

Knuth shows that under the assumptions of Cyclic Symmetry, Antisymmetry, and Non-degeracy, Transitivity holds if and only if Dual Transitivity holds. Knuth requires more than half a page of low level reasoning, but as this reasoning is carried out abstractly in a small first order theory, sledgehammer (Isabelle's interface to various automatic theorem provers) is able to find a proof that consists of just one single invocation of an automated prover.

5.1.1 Instantiation for Points in the Plane

Up to now, our reasoning was based abstractly on ccw systems, but of course we also want to use the results for a concrete ccw predicate. Well known from analytic geometry is the fact that ccw orientation is given by the sign of the following determinant $|pqr|$:

$$|pqr| := \begin{vmatrix} p_x & p_y & 1 \\ q_x & q_y & 1 \\ r_x & r_y & 1 \end{vmatrix} = \begin{array}{l} p_x * q_y + p_y * r_x + q_x * r_y - \\ (r_x * q_y + r_y * p_x + q_x * p_y) \end{array}$$

Points are collinear iff $|pqr| = 0$. Under the assumption that one works with a finite set of points where no three points are collinear, the following predicate $pqr^>$ satisfies the axioms of a ccw system.

$$pqr^> := |pqr| > 0$$

Most axioms can easily be proved using Isabelle/HOL's rewriting for algebraic structures. Transitivity is slightly more complicated, but can also be solved automatically after a proper instantiation of Cramer's rule, which is easily proved automatically:

$$|tpr| = \frac{|tqr||stp| + |tpq||str|}{|stq|} \text{ , if } ||stq| \neq 0$$

5.1.2 CCW on a Vector Space

Knuth presented his axioms with a finite set of discrete points in mind, in our case we need to talk about orientation of arbitrary points in a continuous set. We therefore require consistency of the orientation predicate when vector space operations are involved.

We stick to the the predicate $pqr^>$ because we can rule out degenerate cases in pre- and postprocessing phases (sections 5.2.4.6 and 5.2.4.5). As vector space, we consider points $p, q, r \in \mathbb{R}^2$.

One obvious requirement is that orientation is invariant under translation (figure 5.2, left):

Theorem 5.7 (Translation)**.**

$$(p + s)(q + s)(r + s)^> = pqr^>$$

With translation invariance, we can reduce every ccw triple to a triple with 0 as origin, and from there it is easy to state consistency with respect to scaling: If at q, there is a ccw turn to r, then every point on the ray from 0 through q will induce a ccw turn to r (figure 5.2, right).

Theorem 5.8 (Scaling)**.**

$$\alpha > 0 \longrightarrow 0(\alpha \cdot q)r^> = 0qr^>$$

Negative scalars can be treated by requiring that reflecting one point at the origin inverts the ccw predicate:

Theorem 5.9 (Reflection)**.**

$$0(-p)q = 0qp$$

Furthermore, the addition of vectors q and r, which are both ccw of a line p needs to be ccw of p as well:

Theorem 5.10 (Addition)**.**

$$0pq \longrightarrow 0pr \longrightarrow 0p(q + r)$$

Equipped with these theorems, we can simplify many of the ccw predicates that can occur. For example, one can get rid of all parts of the third component which are collinear with the second:

$$\gamma > 0 \longrightarrow 0a(\gamma \cdot a + b)^> = 0ab^>$$

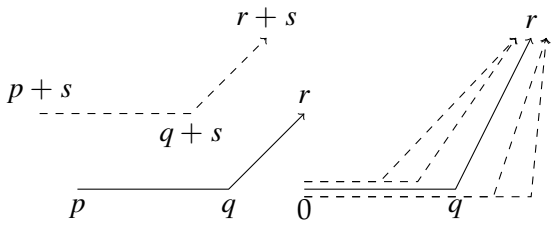


Figure 5.2: Invariance under translation (left), invariance under scaling (right)

Some ccw predicates involving a sum can be reduced to showing the ccw predicate for every summand:

$$\forall i \leq k.\ pq(r_i)^{>} \longrightarrow pq\left(\sum_{i\leq k} r_i\right)^{>}$$

5.2 Verification of le Guernic and Girard's Algorithm

The geometric objects that are represented by affine forms are called zonotopes. Computing the intersection of zonotopes with hyperplanes is an important operation and can be performed geometrically. Unfortunately, the complexity for computing the exact intersection grows exponentially with the number of generators. An overapproximation of the zonotope (reducing the number of generators as in section 4.3.10) before computing the intersection is possible but often leads to too coarse overapproximations. Therefore le Guernic and Girard [42] proposed a way to directly compute overapproximations to the intersection.

5.2.1 Zonotopes

Zonotopes are the sets represented by affine forms, they are particular centrally symmetric, convex polytopes. A zonotope can be visualized as the Minkowski sum of line segments defined by the generators. The Minkowski sum $X \oplus Y$ operates on two sets and returns the set containing all possible sums between elements of the first and second set:

$$X \oplus Y = \{x + y.\ x \in X \wedge y \in Y\}$$

For a generator a_i, the corresponding line segment is $l_i = \{\varepsilon \cdot a_i.\ -1 \leq \varepsilon \leq 1\}$. Figure 5.3 illustrates how a zonotope is built by incrementally taking the Minkowski sum of the three line segments l_1, l_2, l_3 corresponding to generators a_1, a_2, a_3. In the two-dimensional case, we can speak of corners $(c_0, c_1, \dots)$ and edges $(c_0c_1, c_1c_2, \dots)$ of the zonotope (compare figure 5.4). Corners are of the form $a_0 + \sum_i \varepsilon_i \cdot a_i$ for $\varepsilon_i \in \{-1, 1\}$, edges are of the form $c_i(c_i + 2 \cdot a_j)$ for corners c_i and generators a_j. We call the set of all edges the hull of the zonotope.

5.2.2 Reduction to a Two-Dimensional Problem

The first idea of le Guernic and Girard's algorithm is to overapproximate a given set X tightly from a set D of directions. D can be chosen arbitrarily. For every direction

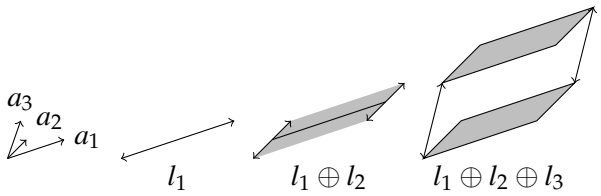


Figure 5.3: Construction of a zonotope

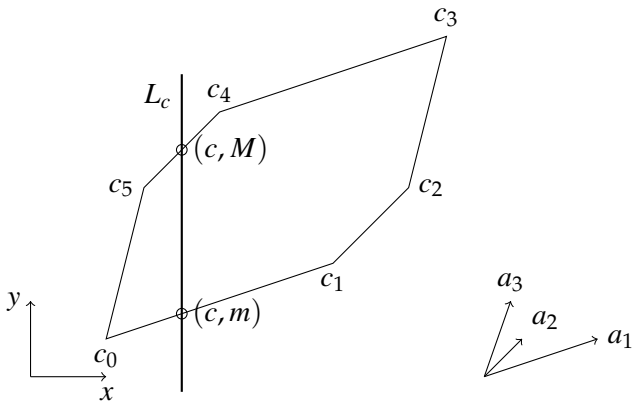


Figure 5.4: Corners and edges of a zonotope, intersecting line L_c

$d \in D \subseteq \mathbb{R}^n$, the infimum m_d and supremum M_d of the sets $\{\langle x, d\rangle.\ x \in X\}$ needs to be determined. Geometrically speaking, m_d and M_d give the translation of two hyperplanes with normal vector d. The two hyperplanes bound X from below and above, respectively. An overapproximation P is then given by the points between all of these hyperplanes:

$$X \subseteq P = \{x \in \mathbb{R}^n.\ \forall d \in D.\ m_d \leq \langle x, d\rangle \leq M_d\}$$

The second observation of le Guernic and Girard is that when the set X is the intersection of some set Z with a hyperplane $G = \{x.\ \langle x, g\rangle = c\}$, then the computation of the overapproximation P can be reduced to a set of two-dimensional problems with the linear transformation $\Pi_{g,d} : \mathbb{R}^n \to \mathbb{R}^2$, $\Pi_{g,d}(x) = (\langle x, g\rangle, \langle x, d\rangle)$ for every $d \in D$.

Theorem 5.11 (Two-Dimensional Reduction)**.**

$$\{\langle x, d\rangle.\ x \in Z \cap G\} = \{y.\ (c, y) \in \Pi_{g,d}(Z)\}$$

The theorem is an easy consequence of the definitions of G and $\Pi_{g,d}$. For every direction d, the theorem allows to reduce the computation of the intersection $Z \cap G$ on the left-hand side to the intersection of the projected two-dimensional zonotope $\Pi_{g,d}(Z)$ with the vertical line $L_c = \{(x, y).\ x = c\}$.

We therefore need an algorithm *bound-intersect-2D* that computes the intersection of a two-dimensional zonotope S with a vertical line L_c, returning the minimal m and maximal M second coordinate of the intersection, as illustrated in figure 5.4. Correctness of the algorithm is specified as follows:

Lemma 5.12 (Correctness of 2D-Zonotope/Line Intersection)**.**

$$\textit{bound-intersect-2D}\,(S, L_c) = (m, M) \longrightarrow S \cap L_c \subseteq \{(x, y).\ x = c \wedge m \leq y \leq M\}$$

With this specification and theorem 5.11 in mind, we can easily define the algorithm *bound-intersect* (Z, g, c, d), which returns the lower and upper translation (m, M) of the hyperplanes with normal vector d that bound the intersection $Z \cap \{x.\ \langle x, g\rangle = c\}$ of a zonotope Z with the hyperplane normal to g at c.

Definition 5.13 (Bound for Intersection)**.**

$$\textit{bound-intersect}\,(Z, g, c, d) := \textit{bound-intersect-2D}\,(\Pi_{g,d}(Z), L_c)$$

The idea to implement the algorithm *bound-intersect-2D* for two-dimensional zonotopes is to first compute the edges of the zonotope with an algorithm *hull-of-zonotope* S. We compute bounds on the intersection of the vertical line L_c with every edge ab as follows: for the line segment ab we assume $a_x \leq g \leq b_x$, otherwise we just flip the corners. If the segment is vertical, i.e., $a_x = b_x$, we return $m = \min(a_y, b_y)$ and $M = \max(a_y, b_y)$ as bounds on the intersection. If $a_x < b_x$, we use approximate operations with fixed precision to compute bounds on the exact point of intersection $m \leq \frac{b_y - a_y}{b_x - a_x}(c - a_x) + a_y \leq M$. The zonotope then intersects the line between the minimum and maximum bounds m, M of all edges. The only part missing is how to compute *hull-of-zonotope*, which we sketch in the following.

5.2.3 Computation of Two-Dimensional Hulls

To formally reason about the computed intersection, some guarantees concerning the edges computed by *hull-of-zonotope* are required: in particular that they actually enclose the zonotope. To see how this can be ensured, we first take a look at how they are actually computed. Intuitively, assuming that all generators point upwards, one starts at the lowest corner c_0 in figure 5.4 and appends to it the "rightmost" generator a_1 (twice) to reach c_1. One then continues with the "rightmost" of the remaining generators, a_2 and is in the process essentially "wrapping up" the hull of the zonotope.

We therefore need a way to reason about "rightmost" vectors. Similar ideas of "wrapping up" a set of points also occur for convex hull algorithms, they have been studied extensively in the literature with Knuth's axiomatic theory of counterclockwise (ccw) systems (section 5.1). The algorithm *hull-of-zonotope* is easier to implement than convex hull algorithms, because it basically only needs to sort the generators. The verification, however, poses additional challenges as we do not deal with discrete set of points, but rather with the continuous set given by the zonotope which needs to be enclosed by the computed segments. This is why we make use of our extended theory of ccw systems on vector spaces (section 5.1.2).

5.2.4 Verification of Two-Dimensional Hulls

Equipped with the formalisms to reason about orientation in the plane, we now detail on the algorithm *hull-of-zonotope* to compute the hull of a two-dimensional zonotope $a_0 + \sum_i \varepsilon_i \cdot a_i$ and how we verified it.

1. input: an affine form $a_0 + \sum_i \varepsilon_i \cdot a_i$, given by the list of its generators $a_0, \ldots, a_n$, all pointing upwards
2. find the lowest corner c_0 of the zonotope, i.e., $c_0 = -\sum_i a_i$
3. sort the generators, i.e., assume $i < j \longrightarrow 0(a_i)(a_j)^>$
4. double the generators, i.e., $b_i := 2 \cdot a_i$
5. append generators in order, i.e., $c_{i+1} = c_i + b_{i+1}$
6. reflect the corners $c_0, \ldots, c_n$, i.e., $c_{n+i+1} = -c_{i+1}$
7. output: a list of edges $c_0c_1, c_1c_2 \ldots, c_nc_{n+1}, \ldots, c_{2n-1}c_{2n}$

Figure 5.5: Algorithm *hull-of-zonotope*

The verification of the algorithm is simpler when we assume that the generators a_i are not collinear and that all of them point upwards, i.e., $(a_i)_y > 0$. In fact, the instantiation of the ccw predicate $pqr^>$ requires this. We present a suitable preprocessing in subsection 5.2.4.6, which ensures that these conditions are always met when computing the hull. We can also assume that the zonotope is centered around the origin, i.e., $a_0 = 0$.

The aim is to compute a list of corners c_i of the zonotope generated by the a_i. We first compute the corners on the right side (c_1, c_2, c_3 in figure 5.3) by appending the generators in sorted order. For sorting, we need the notion of a total order induced by the ccw predicate (section 5.2.4.1). Then we reflect the obtained corners according to the central symmetry of zonotopes. In a bit more detail, the algorithm can be described as in figure 5.5.

Implementing this algorithm in a functional language is straightforward. The specification that we aim to verify is that the returned edges enclose the interior of the zonotope, i.e., every point x in the zonotope is left of all the edges c_ic_{i+1}. According to the definition of $pqr^>$, the ccw predicate can be used to describe half-planes: $X_{pq} :: \mathbb{R}^2\ set$ with $X_{pq} := \{x \mid pqx^>\}$ is the half-plane left of the line pq. The interior of any given convex polygon can therefore be described as the intersection of half-planes defined by the edges of the polygon: for corners $c_0, \ldots, c_n$ of a polygon, the interior of the polygon is the set $P = \bigcap_{i=0}^{n} X_{c_jc_{j+1}}$.

The verification can be outlined as follows: First, appending sorted vectors in order keeps certain linear combinations ccw oriented with respect to the line segments. Second, we establish that these linear combinations represent the interior of the zonotope, therefore the interior of the zonotope is ccw of the line segments after step 5. Then, because of symmetries, the same holds for the reflected corners on the left, i.e., the ones after step 6. Finally, we give a relaxed notion of ccw that allows to include not only the interior but also the edges of the zonotope.

5.2.4.1 Total Order from CCW

As sketched in section 5.2.3, we need to be able to select a "rightmost" element of a set of vectors. With transitivity of the ccw predicate, we can obtain a total order on vectors which

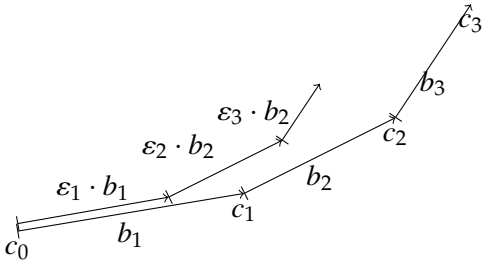


Figure 5.6: $c_0 + \mathit{polychain}\,([b_1, b_2, b_3])$ and illustration of lemma 5.15

allows to do just this: Given a center t and another point s, the ccw predicate tpq can be used to define a total order on points p, q in the half-plane left of ts, i.e., $p < q$ iff tpq. Axioms 5.2 and 5.3 directly provide antisymmetry and totality. Transitivity of the order follows from axiom 5.5 and the assumption that all points are in the half-plane left of ts.

This order is used to specify a *ccw-sorted* list of points R, with respect to a center p:

Definition 5.14.

$$\mathit{ccw\text{-}sorted}\ p\ R := (\forall i\ \forall j.\ i < j \longrightarrow pr_i r_j)$$

A list of points can only be sorted if all points are in one half-plane through the center, because the first element r_0 of a *ccw-sorted* list restricts all subsequent points to the half-plane left of pr_0.

5.2.4.2 Appending Sorted Vectors

We write $\mathit{polychain}\ B$ for the list of points obtained from a list B consisting of vectors b_i by appending the vectors b_i in order.

$$(\mathit{polychain}\ B)_{i+1} = (\mathit{polychain}\ B)_i + b_{i+1}$$

A crucial property is that whenever a list of vectors B is sorted, then linear combinations (with coefficients between 0 and 1) of the elements b_i of B are ccw of $\mathit{polychain}\ B$. Compare also figure 5.6.

Lemma 5.15. *Assume* $\mathit{ccw\text{-}sorted}\ B$ *and* $\forall i.\ 0 < \varepsilon_i < 1$ *and* $C = \mathit{polychain}\ B$. *Then:*

$$c_j c_{j+1} \left(\sum_i \varepsilon_i \cdot b_i \right)^{>}$$

The proof goes by induction the length of B resp. C and makes use of ccw vector space theorems like the ones given in subsection 5.1.2.

5.2.4.3 The Interior of the Zonotope

The interior of the zonotope, i.e., the points constructed as linear combinations $a_0 + \sum_i \varepsilon_i \cdot a_i$ for $-1 < \varepsilon_i < 1$, can also be represented as linear combinations $c_0 + \sum_i \varepsilon_i \cdot b_i$ for a different set of $0 < \varepsilon_i < 1$: the linear combinations are just translated to the lowest point c_0, and doubling the vectors in step 3 makes up for the smaller range for the ε_i.

Now assume that after step 4, we have computed n corners $c_0, \ldots, c_n$. Since the c_j are sorted, we have from lemma 5.15 that all linear combinations $x = c_0 + \sum_i \varepsilon_i \cdot b_i$ are left of the line segments $c_j c_{j+1}$. But according to the previous considerations, this means that the interior of the zonotope is left of the computed line segments after step 4:

Lemma 5.16. *After step 4,*
$c_j c_{j+1}(\sum_i \varepsilon_i \cdot a_i)^>$ *holds for* $-1 < \varepsilon_i < 1$

Note that at this step it is important to consider only the strict interior (i.e., $-1 < \varepsilon_i < 1$) of the zonotope, because points on the edges do not satisfy the (strict) ccw predictate $pqr^>$ and can only be reached with $-1 \leq \varepsilon \leq 1$.

5.2.4.4 Reflected Corners

Step 5 of the algorithm simply reflects the already computed corners at the origin, an operation under which the orientation predicate remains invariant. Because in addition to that, every zonotope is centrally symmetric, we can deduce from lemma 5.16 that the reflected line segments enclose the interior of the zonotope as well. We also have $c_n = -c_0$.

Lemma 5.17. *After step 5,*
$c_j c_{j+1}(\sum_i \varepsilon_i \cdot a_i)^>$ *holds for* $-1 < \varepsilon_i < 1$

5.2.4.5 "Postprocessing": Continuously Relaxing CCW

In order to also include the edges into our reasoning, we define the slightly relaxed ccw predicate $pqr^\geq$, which holds for all points on the line through pq and for all points on the half-plane left of pq.

$$pqr^\geq := |pqr| \geq 0$$

For every segment $c_j c_{j+1}$, the half-plane $X_j = \{x.\ c_j c_{j+1} x^\geq\}$ is topologically closed. We know that all points from the interior are contained in this half-plane and show that points from the edges of the zonotope are also contained: Assume some $x = \sum_i \varepsilon_i \cdot a_i$ with $-1 \leq \varepsilon_i \leq 1$, i.e., x may be on the edges or in the interior. Then define $x_m = \sum_i (\varepsilon_i \cdot (1 - \frac{1}{m})) \cdot a_i$ for $m = 1, 2, \ldots$, we therefore have strict inequalities $-1 < \varepsilon_i \cdot (1 - \frac{1}{m}) < 1$, which imply according to lemma 5.17 that $x_m \in X_j$. Moreover, as m goes to infinity, x_m tends to x, and since X_j is closed, we can conclude $x \in X_j$.

In summary, the line segments output by *hull-of-zonotope* define a polygon (as intersection of half-planes) that encloses the zonotope:

Theorem 5.18. *After step 5,*
$c_j c_{j+1}(\sum_i \varepsilon_i \cdot a_i)^\geq$ *holds for* $-1 \leq \varepsilon_i \leq 1$

Theorem 5.18 proves that *hull-of-zonotope* encloses all points of the zonotope, but not that all enclosed points actually belong to the zonotope. We have also proved that theorem, but do not make use of it, as we are only interested in an overapproximation of the intersection.

5.2.4.6 Preprocessing for Generators

Recall that at the beginning of section 5.2.4, we assumed the generators a_i of the zonotope to be nonaligned and pointing upwards. Given a zonotope with arbitrary generators a'_i, it is easy to compute a new set of generators a_i that meet the above conditions and represent the same zonotope.

Consider an element $x = \sum_i \varepsilon_i \cdot a'_i$ with generators a'_i which point downwards, i.e., $(a'_i)_x < 0$. Set $a_i = -a'_i$ and also negate ε_i, then $x = \sum_i \varepsilon_i \cdot a_i$ and all generators a_i point upwards.

Concerning collinear generators, one can start with the first generator a'_1 and find the indices C of collinear generators, i.e., $|0a'_1a'_i| = 0$ for $1 < i \in C$. Then we can set $a_1 = a'_1 + \sum_{i \in C} a'_i$ and find an appropriate ε_1 such that $x = \sum_i \varepsilon_i \cdot a_i$. Then recurse on the list of remaining generators a_i with $i \notin C$ and finally obtain a list of generators which are pairwise not collinear, i.e., for $i \neq j$, we have $|0a_ia_j| \neq 0$.

5.2.5 The Final Intersection Algorithm

Recall that the algorithm *bound-intersect* returns hyperplanes that bound the intersection from above and below. The final result of our verification can then be summarized by a short formal statement: The intersection of a zonotope (the range of the affine form given by generators $A = (a_i)_i$) with a hyperplane is bounded by the computed half-planes:

Theorem 5.19 (Bounding the Zonotope/Hyperplane Intersection)**.**

$$\begin{aligned}&\textit{bound-intersect}\,(A, g, c, d) = (m, M) \longrightarrow \\ &\{a_0 + \sum_i \varepsilon_i \cdot a_i.\ -1 \leq \varepsilon_i \leq 1\} \cap \{x.\ \langle x, g\rangle = c\} \subseteq \\ &\quad \{x.\ m \leq \langle x, d\rangle \leq M\}\end{aligned}$$

The intersection is represented by half-planes, if however one wants to continue calculating with zonotopes, it is necessary to choose several directions d_i in a way that the polytope resulting from the intersection of all the half-planes can be represented as a zonotope. For this purpose, one can choose for example hyperrectangles or parallelotopes.

5.2.6 Experiments

All of the algorithms we presented are given as functional programs in Isabelle/HOL. We can therefore make use of Isabelle's code generator [51] to conduct some experiments and demonstrate that our formalization can actually be used to compute intersections of zonotopes with hyperplanes. Here, numbers are represented by floating point numbers $m \cdot 2^e$ for unbounded integers m, e (see also section 4.2).

We give a short example where the naive approach of projecting the set to the hyperplane G would result in a very large overestimation, but the intersection (in this case from Basis-parallel directions) is tight, i.e. it touches the original set in every direction. The zonotope Z in the example is given by $[(0,0,0);(2,1,0)] \oplus (5,10,20)$. The hyperplane is given by $G = \{x.\ \langle(0,0,1), x\rangle = 3\}$. Computing the intersection in this case takes negligible time. For a more complex scenario, we created random ten-dimensional zonotopes with fifty

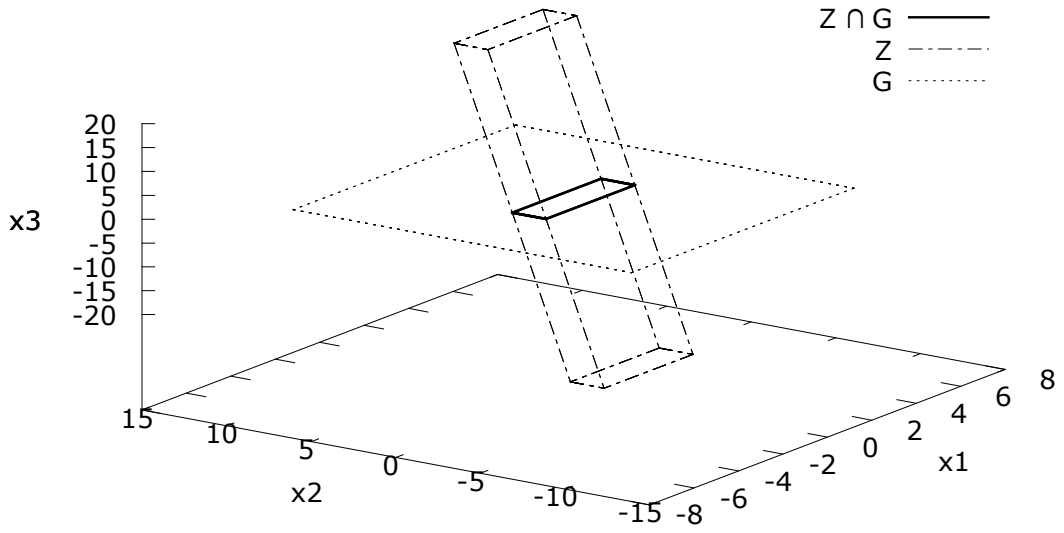


Figure 5.7: Intersection as it might occur in reachability analysis

generators, computing the intersection with a random hyperplane takes 6-7 milliseconds on a 2.9 GHz laptop. Twice the number of generators requires twice as much time, doubling the dimension also increases the amount of time needed by a factor of two. For our purposes, the code exhibits reasonable performance and scaling behavior.

5.3 A Consistent CCW Predicate for Degenerate Cases

This section is not directly related to the verification of zonotope/hyperplane intersection, but it is interesting as an example of formalizing geometry.

Recall that the instantiation of a ccw system with $pqr^{>}$ only worked with the additional assumption that all involved points are not collinear. In some cases it is possible to get rid of "degenerate" situations with some sort of preprocessing, which is what we did in section 5.2.4.6. Here we give an alternative instantiation of ccw systems for arbitrary points in the plane, as demonstrated in chapter 14 of Knuth's monograph.

Knuth proposes to refine the ccw predicate $pqr^{>}$ in degenerate cases by including the lexicographic order $\prec$ on points.

$$p \prec q := p_x < q_x \vee (p_x = q_x \wedge p_y < q_y)$$

The refined ccw predicate pqr^{*} can then be defined as follows:

$$pqr^{*} := pqr^{>} \vee (|pqr| = 0 \wedge p \prec q \prec r \vee q \prec r \prec p \vee r \prec p \prec q)$$

This predicate can be shown to form a ccw system for arbitrary points in the plane. As elaborated by Knuth as well, an important part of the reasoning is that the convex combination of two points lies lexicographically between them:

$$p \prec q \wedge 0 \leq \alpha \leq 1 \longrightarrow p \prec \alpha \cdot p + (1-\alpha) \cdot q \prec q$$

Knuth does not explicitly mention it, but a similar rule for triangles is needed as well.

$$\begin{aligned} &p \prec q \prec r \wedge 0 \leq \alpha \wedge 0 \leq \beta \wedge 0 \leq \gamma \wedge \alpha + \beta + \gamma = 1 \longrightarrow \\ &\quad p \prec \alpha \cdot p + \beta \cdot q + \gamma \cdot r \prec r \end{aligned}$$

This rule is necessary to establish the following rules, all of which establish some sort of consistency between the lexicographic order and the orientation predicate. Like Knuth, we abbreviate $s \in \Delta pqr = spq^* \wedge sqr^* \wedge srp^*$ to express s lying inside the triangle given by p, q, r and $\square pqrs = pqr^* \wedge qrs^* \wedge rsp^* \wedge spq^*$ to describe an oriented tetragon p, q, r, s. Then each of the following configurations is impossible (for pairwise distinct points): first (or second), the rightmost (or leftmost) point s lies in the triangle given by the points p, q, r on the left (or right). Third, if p and q are left of r and s, then the points cannot form an oriented tetragon p, r, q, s, because two of the lines would have to cross.

$$p \prec q \prec r \prec s \wedge s \in \Delta pqr$$

$$s \prec p \prec q \prec r \wedge s \in \Delta spqr$$

$$p \prec r \wedge p \prec s \wedge q \prec r \wedge q \prec s \wedge \square prqs$$

As in the previous sections, transitivity of pqr^* can be established relatively easily algebraically, using Cramer's rule. However, when collinear points are involved, case distinctions are needed, some need to derive new collinearities, e.g., if q lies on a line with tp and on a line with tr, then r and p are on the same line: $|tpq| = |tqr| = 0 \longrightarrow |trp|$.

In addition it is necessary to rule out configurations which are according to Knuth's presentation without further elaboration just "impossible", but require careful formal proof. Consider e.g. t, s, r, p aligned and q not aligned according to $tpq^>$ and $tqr^>$. This configuration can be realized, and is only impossible by investigating subtle contradictions given by the possible lexicographic orderings of t, s, r, p in the context of the proof.

Other important lemmas that are not mentioned by Knuth are needed for translating the corner of a ccw turn, which only works when translating in one direction, which is why the lexicographic order is needed:

$$|trs| = 0 \wedge t \prec r \prec s \wedge trp^> \longrightarrow tsp^>$$

5.4 Related Work

Noting that the main contribution of our work is the verification of a geometric algorithm, we can compare this work with several other formalizations, especially with verifications of convex hull algorithms: They all have in common that they base their reasoning on Knuth's notion of ccw system.

Pichardie and Bertot [118] were the first to formalize Knuth's ccw system and verify a functional convex hull algorithm in Coq. Meikle and Fleuriot [99] formalized an imperative algorithm and verified it using Hoare logic in Isabelle/HOL. Brun *et al.* [27] verify an algorithm based on hypermaps to compute the convex hull.

The basic notion of a ccw system is straightforward to formalize, however it excludes in its pure form "degenerate" configurations of points, that is, three points lying on the same line. To cope with these special cases, different approaches have been used: Pichardie and Bertot give two possible solutions, one is to extend the ccw theory with an additional predicate pqr', which is true whenever q lies on the line between p and r. This requires nine additional axioms and therefore makes the abstract theory more cumbersome to use. The second approach they formalized (and which has been elaborated by Knuth) perturbs the points of the system in a continuous manner to get rid of degenerate configurations, continuity carries the results over to the degenerate case. Our approach from section 5.2.4.5 is similar in the sense that we extend results from the nondegenerate interior of a zonotope to the frontier, which contains degenerate points. Meikle and Fleuriot take a more pragmatic approach and modify their algorithms to explicitly check for collinear points.

In section 5.3, we digressed into yet another possibility (also already described by Knuth) to refine the ccw predicate $pqr^{>}$ in a consistent way for arbitrary points in the plane, and see how we can follow Knuth's reasoning with our formalization of ccw systems in Isabelle/HOL. Unfortunately this approach does not directly work for ccw systems on vector spaces.

It is worth mentioning that we restrict our attention to zonotopes and not the more general approach of using support functions as done by le Guernic and Girard [89], because many algorithms related to reachability analysis get more involved and require to solve e.g. linear programming and other optimization problems that are not formalized in Isabelle/HOL.

6

Specification and Verification of Rigorous Numerical Algorithms

Rigorous numerical algorithms can be characterized by the fact that they compute with enclosures of some real quantity. An important insight is that for the verification, anything that is an enclosure of the real quantity is sufficient for proving correctness.

This gives the opportunity to abstract over implementation details along two axes. One, the concrete representation of the enclosure, which could be e.g., ellipsoids, intervals, zonotopes, or Taylor models. Two, the actual algorithm that computes the enclosure, which might also be influenced by heuristics or optimizations.

We use Lammich's [85, 86] framework *Autoref* for (automatic) refinement of nondeterministic specifications. This framework provides the infrastructure to systematically refine abstract, nondeterministic specifications (e.g., an enclosure for the solution of an ODE) to concrete, executable implementations (e.g., a rigorous Runge-Kutta method) as well as abstract data structures (e.g., sets for enclosures) to concrete ones (e.g., intervals or zonotopes).

The advantage of a framework like Autoref is that one can conveniently verify the correctness of algorithms on an abstract level, not caring about implementation details. Moreover, data-structures and algorithms can be modularly modified while the abstract correctness proofs remain valid.

This chapter presents a framework (based on Autoref) for writing rigorous numerical algorithms in an abstract, high-level language. To this end, we first specify a set of abstract operations that is useful for rigorous numerical algorithms (section 6.1). With the help of Autoref, those abstract algorithms can be translated (mostly) automatically to concrete, executable implementations. To make this translation possible, Autoref needs to be provided with concrete data structures (section 6.2).

As introductory example to illustrate the level of abstraction on which we aim to specify algorithms, consider definition 6.1. This algorithm operates on a work-set $\mathcal{X}$, think of $\mathcal{X}$ as a union of intervals. The algorithm proves (when it terminates) that some function f is positive on every interval in $\mathcal{X}$. Inside the **while**-loop, an interval X is removed from the work-set, and the algorithm comes up with a rigorous enclosure F for the image of some function f on X and finds a lower bound ℓ on F. If this lower bound is positive, the **while** loop continues with the rest of the work-set $\mathcal{X}$. If ℓ is negative, the algorithm splits the current set X into two parts X_1 and X_2 and adds those to the work-set, in the hope that successive iterations will be successful in proving positivity for those smaller sets.

Definition 6.1 (Pseudo-Code for Global Optimization Work-Set Algorithm)**.**

while $(\mathcal{X} \neq \emptyset)$
  remove X *from* $\mathcal{X}$
  choose F *such that* $f(X) \subseteq F$
  choose ℓ *such that* $\forall x \in F.\ x \geq \ell$
  if $\ell \geq 0$ ***then continue***
  else $\{$
    split $X = X_1 \cup X_2$
    $\mathcal{X} := \{X_1, X_2\} \cup \mathcal{X}$
  $\}$

We aim for a framework that allows users to specify rigorous numerical algorithms at about the level of abstraction of definition 6.1. We will use this algorithm as the running example to illustrate the developments presented in this chapter. The precise formal definition of this algorithm in Isabelle/HOL is given later on in definition 6.4. There will, however, be one striking difference: reasoning about work-set algorithms involving (rigorous numerical) enclosures can be abstracted even more than in the view of the algorithm in definition 6.1. This is because $\mathcal{X}$ is of type α *set set*, the algorithm proves that $f(\bigcup_{X \in \mathcal{X}} X) \geq 0$. But this nesting of sets of sets can be avoided: we will specify work-set algorithm as operating on just some set $X :: \alpha$ *set*. Removing an element is choosing some Y and Z such that $X = Y \cup Z$, where Y morally represents one element of the work-set and Z the remaining work-set.

6.1 Nondeterministic Specifications

Autoref is based on refinement calculus [9]. Specifications are encoded as nondeterministic results (section 6.1.1). In this calculus, one can express and verify correctness of algorithms (section 6.1.2). We specify a set of operations geared towards the specification of rigorous numerical algorithms (section 6.1.3).

6.1.1 Nondeterministic Results

Autoref is based on a nondeterminism monad α *nres*, where programs can either fail or yield a set of values as result, which is captured in the data type *nres*.

$$\textsf{datatype}\ \alpha\ \mathit{nres} = \mathit{FAIL} \mid \mathit{RES}\ (\alpha\ \mathit{set})$$

The type *nres* is equipped with a refinement relation $\leq$. It is defined such that for results

$$\mathit{RES}\ S \leq \mathit{RES}\ T \longleftrightarrow S \subseteq T$$

and *FAIL* is the top element of this order: $\forall x.\ x \leq \mathit{FAIL}$.

Deterministic results are singleton nondeterministic results. We write this as *return*:

Definition 6.2 (Deterministic Results)**.**

$$\mathit{return}\ x := \mathit{RES}\ \{s\}$$

The type *nres* can also be used to provide specifications. The set of all results satisfying a given property P is written as specification *spec*.

Definition 6.3 (Nondeterministic Specification)**.**

$$\mathit{spec}\ P := \mathit{RES}\{x.\ P\ x\}$$

The type α *nres* is a monad for which we use **do** notation. We will only appeal to the intuition that $x \leftarrow X$ binds (nondeterministically) some Element $x :: \alpha$ in the set of nondeterministic results of $X :: \alpha$ *nres* to $f :: \alpha \to \beta$ *nres*, which yields a nondeterministic result of type β *nres*.

$$\begin{array}{l}\textbf{do}\ \{\\ \quad x \leftarrow X;\\ \quad f\ x\\ \}\end{array}$$

We can also use **while** and **if-then-else** statements. For $b :: \sigma \to \mathbb{B}$, $f :: \sigma \to \sigma$ and $X_0 :: \sigma$, the while loop **while** $b\ f\ X_0$ denotes iterated application of f on the initial state X_0, as long as the predicate b holds. The introductory example, specified in the *nres* monad should give an intuition on **while**, **do**, and the *nres* monad.

Definition 6.4 (Global Optimization Work-Set Algorithm in the *nres* Monad)**.**

$$\begin{array}{l}\mathit{global\text{-}optimization}\ f_e\ X_0 :=\\ \textbf{\textit{while}}\ (\lambda X.\ X \neq \emptyset)\\ (\lambda X.\ \textbf{\textit{do}}\ \{\\ \quad (Y,Z) \leftarrow \mathit{spec}\ (\lambda(Y,Z).\ X = Y \cup Z);\\ \quad F \leftarrow \mathit{spec}\ (\lambda F.\ \forall y \in Y.\ [\![f_e]\!]_{\mathit{list\text{-}of\text{-}eucl}\ y} \in F);\\ \quad \ell \leftarrow \mathit{spec}\,(\lambda \ell.\ \forall x \in F.\ \ell \leq x);\\ \quad \textbf{\textit{if}}\ \ell \geq 0\ \textbf{\textit{then}}\\ \quad \textbf{\textit{return}}\ Z\\ \quad \textbf{\textit{else do}}\ \{\\ \quad\quad (Y_1,Y_2) \leftarrow \mathit{spec}\,(\lambda(Y_1,Y_2).Y \subseteq Y_1 \cup Y_2);\\ \quad\quad \textbf{\textit{return}}\ (Y_1 \cup Y_2 \cup Z)\\ \quad \}\\ \})\ X_0\end{array}$$

It is worth pointing out some differences to the pseudo code from definition 6.1. The work-set $\mathcal{X} :: \alpha\ \mathit{set}\ \mathit{set}$ turned into one enclosure $X :: \alpha\ \mathit{set}$. The **while**-loop is now specified

in a functional way, i.e., without implicit state: The **while** loop takes (as last argument) some explicit initial state X_0 and repeatedly applies the function in the second argument to it (as long as the function on the first argument returns *True*). Instead of "removing" an element from $\mathcal{X}$, we now obtain some Y and Z such that $X \subseteq Y \cup Z$ and return either Z (instead of **continue**) or $Y_1 \cup Y_2 \cup Z$ as new state. Furthermore, instead of a function f, we assume the function given as interpretation $\lambda.\ [\![f_e]\!]_{\textit{list-of-eucl}\ y}$ of some *aexp*-expression f_e (according to section 4.1).

6.1.2 Verification Condition Generation

The *nres* monad allows one to specify correctness, e.g, of a program f whose inputs x satisfy the precondition P and every possible value y in its nondeterministic result satisfies the postcondition Q:

$$\forall x.\ P\ x \longrightarrow f\ x \leq \textit{spec}\,(\lambda y.\ Q\ y)$$

Proving that a program fulfills a specification therefore amounts to proving that the program refines a *spec* statement. For such statements, Autoref provides a verification condition generator, which follows the structure of the program. For illustration, we present the structural rules for *spec*, monadic bind, and the **while** loop.

A specification refines another specification if one predicate implies the other:

Lemma 6.5 (Verification Condition for Specification)**.**

$$(\forall x.\ P\ x \longrightarrow Q\ x) \longrightarrow \textit{spec}\ P \leq \textit{spec}\ Q$$

Binding an element x in the nondeterministic result of X to a function f refines a specification Q if X refines the specification that f applied to any nondeterministic result x in X refines the specification Q.

Lemma 6.6 (Verification Condition for Bind)**.**

$$X \leq \textit{spec}\ (\lambda x.\ f\ x \leq \textit{spec}\ Q) \longrightarrow \textbf{\textit{do}}\ \{x \leftarrow X;\ f\ x\} \leq \textit{spec}\ Q$$

A **while** loop requires an invariant I on the state. **while** $b\ f\ X_0$ refines a specification Q if the invariant holds for the initial state X_0, an application of the function body on a state X preserves the invariant, and the invariant after termination $\neg b\ X$ refines the specification Q.

Lemma 6.7 (Verification Condition for **while**)**.**

$$\begin{aligned}
&I\ X_0\ \longrightarrow\\
&(\forall X.\ I\ X \longrightarrow b\ X \longrightarrow f\ X \leq \textit{spec}\ I)\ \longrightarrow\\
&(\forall X.\ I\ X \longrightarrow \neg b\ X \longrightarrow Q\ X)\ \longrightarrow\\
&(\textbf{\textit{while}}\ b\ f\ X_0) \leq \textit{spec}\ Q
\end{aligned}$$

Ideally, the algorithm is specified in such a way, that after structural application of all verification conditions, the proof obligations expose the main algorithmic ideas that require interactive proof.

We return to the running example *global-optimization*. *global-optimization* $f_e\ X_0$ should satisfy that upon termination, $[\![f_e]\!]$ applied to any element in X_0 is greater or equal zero:

Lemma 6.8 (Specification of *global-optimization*)**.**

$$\textit{global-optimization}\ f_e\ X_0 \leq \textit{spec}\ (\lambda\_.\ \forall x \in X_0.\ [\![f_e]\!]_{\textit{list-of-eucl}\ x} \geq 0)$$

As loop invariant, we choose $I\ X := \forall x \in X_0 \setminus X.\ [\![f_e]\!]_{\textit{list-of-eucl}\ x} \geq 0$. After generating the verification conditions, Isabelle can automatically prove that *global-optimization* satisfies the specification. This was precisely our goal of using a high-level specification language: That for verification, only the main algorithmic ideas are exposed. In this example, the algorithmic ideas are so simple that they can be proved automatically.

6.1.3 Specifications for Enclosures

We use *spec* to specify a collection of operations that are useful for rigorous numerical algorithms. We will see that the specifications almost never demand precise results. The specifications nondeterministically allow for safe uncertainties in the result. This gives sufficient freedom for possible implementations, which may be limited to work e.g., only with finite precision and can therefore not represent the result exactly.

Subdivisions are a means to maintain precision, we therefore have abstract specifications for splitting a set (with and without the possibility to perform overapproximations):

$$\textit{split-spec}_{\subseteq}\ X := \textit{spec}\,(\lambda(A,B).\ X \subseteq A \cup B)$$
$$\textit{split-spec}_{=}\ X := \textit{spec}\,(\lambda(A,B).\ X \subseteq A = B)$$

The following specifications yield some lower/upper bound on the set. Note that the specification only demands *some* lower bound i (resp. upper bound s), not necessarily the precise greatest lower bound.

$$\textit{Inf-spec}\ X := \textit{spec}\,(\lambda i.\ \forall x \in X.\ i \leq x)$$
$$\textit{Sup-spec}\ X := \textit{spec}\,(\lambda s.\ \forall x \in X.\ x \leq s)$$

Depending on the concrete representation of sets, one might not be able to *decide* certain properties, but only give a positive answer if the precision is sufficient. An example is disjointness of sets. This depends on the concrete representation, but it might be that if two sets are very close, finite precision calculations are not sufficient to decide whether the sets are actually disjoint or not. An abstract specification that accounts for such uncertainties is the following. If the return value b is true, then one can be sure that the two sets X and Y are disjoint, otherwise one knows nothing.

$$\textit{disjoint-spec}\ X\ Y := \textit{spec}\,(\lambda b.\ b \longrightarrow X \cap Y = \emptyset)$$

As seen in the previous chapter 5, depending on the data structure, one can not (or does not want to) compute an exact representation for the intersection of sets. The following specifications allow one to overapproximate an intersection, while guaranteeing that the result is restricted to one of the arguments.

$$\textit{inter-spec}_1\ X\ Y := \textit{spec}\,(\lambda R.\ X \cap Y \subseteq R \wedge R \subseteq X)$$
$$\textit{inter-spec}_2\ X\ Y := \textit{spec}\,(\lambda R.\ X \cap Y \subseteq R \wedge R \subseteq Y)$$

To bridge the gap to concrete numerical computations and the results from chapter 4, we use a specification that satisfies the fundamental property of rigorous numerics.

$$\textit{approx-spec}\ f\ X := \textit{spec}\,(\lambda R.\ \forall x \in X.\ [\![f]\!]_x \in R)$$

6.2 Data Refinement

In the previous section 6.1, we have seen nondeterministic specifications for operations on enclosures (at that point, just arbitrary sets). In this section, we will present different representations for enclosures and how the framework is instrumented to refine the abstract operations to concrete ones using the given data structures.

Data refinement in Autoref is centered around a collection of so-called *transfer* rules. Transfer rules relate abstract with concrete operations. A transfer rule involves a transfer relation $R :: \gamma \times \alpha\ \textit{set}$, which relates a concrete implementation $c :: \gamma$ of an abstract element $a :: \alpha$ and is of the following form.

$$(c :: \gamma, a :: \alpha) \in R$$

Transfer rules are used to synthesize concrete algorithms from abstract ones by following the structure of the algorithm (similar to the generation of verification conditions). Relators and relations are used to express the relationship between concrete and abstract types.

6.2.1 Natural Relators

For the types of functions, products, sets, or data types like lists and *nres*, we can use the natural relators $A \to_r B, A \times_r B, \langle A\rangle \textit{set}_r, \langle A\rangle \textit{list}_r, \langle A\rangle \textit{nres}_r$ with relations A, B for the argument types:

$$\begin{aligned}
(f, f') \in A \to_r B &\longleftrightarrow \forall (x,y) \in A.\ (f\ x, f'\ x) \in B \\
((a,b),(a',b')) \in A \times_r B &\longleftrightarrow (a,a') \in A \wedge (b,b') \in B \\
(X, X') \in \langle A\rangle \textit{set}_r &\longleftrightarrow (\forall x \in X.\ \exists x' \in X'.\ (x,x') \in A) \wedge \\
&\qquad (\forall x' \in X'.\ \exists x \in X.\ (x,x') \in A) \\
(xs, xs') \in \langle A\rangle \textit{list}_r &\longleftrightarrow \textit{length}\ xs = \textit{length}\ xs' \wedge \\
&\qquad (\forall i < \textit{length}\ xs.\ (xs_i, xs'_i) \in A) \\
(\textit{RES}\ X, \textit{RES}\ X') \in \langle A\rangle \textit{nres}_r &\longleftrightarrow (X, X') \in \langle A\rangle \textit{set}_r
\end{aligned}$$

br is used to build a relation from an abstraction function $a :: (\gamma \to \alpha)$ and an invariant $I :: \gamma \to \mathbb{B}$ on the concrete type.

$$\textit{br}\ a\ I := \{(c, a\ c) \mid I\ c\}$$

6.2.2 Representing Vectors

We represent vectors (an arbitrary type α of class Euclidean space) as lists of real numbers where the length matches the dimension of the Euclidean space. This representation is expressed with the relation lv_r.

$$\mathit{lv}_r := \mathit{br}\ \mathit{eucl\text{-}of\text{-}list}\ (\lambda xs.\ \mathit{length}\ xs = DIM(\alpha))$$

With this representation, concrete algorithms are monomorphic (i.e., do not contain a type variable for Euclidean space α). This has the advantage that code can be generated once and for all and can therefore be used as a stand-alone tool. Operations on vectors are simply implemented componentwise. For example, the transfer rule for addition of vectors implemented with lv_r is as follows.

Lemma 6.9 (Transfer Rule for Addition of Vectors)**.**

$$((\lambda xs.\ \lambda ys.\ \mathit{map}2\ (+)\ xs\ ys), (+)) \in \mathit{lv}_r \to \mathit{lv}_r \to \mathit{lv}_r$$

$\mathit{map}2$ is defined such that $(\mathit{map}2\ f\ xs\ ys)_i = f\ (xs_i)\ (ys_i)$.

6.2.3 Representing Enclosures

We provide several implementations for the sets that can be used as enclosures. Intervals are represented by pairs of element types (which, in turn are implemented via some relation A) with the relator ivl_r and zonotopes are represented using the joint range *joint-range* of affine forms with the relation affine_r.

$$\langle A\rangle \mathit{ivl}_r := \{((a', b'), [a; b]) \mid (a', a) \in A \wedge (b', b)\}$$

$$\mathit{affine}_r := \mathit{br}\ (\lambda A.\ \mathit{eucl\text{-}of\text{-}list}\,(\mathit{joint\text{-}range}\ A))\ (\lambda\_.\ \mathit{True})$$

6.2.4 Symbolic Representations

There are situations where operations can not be carried out on the concrete representation, but one would still like to use them for the abstract algorithmic description. A solution is a symbolic, or lazy, representation. For example, zonotopes are not closed under intersection, but one can represent the intersection of two zonotopes simply by the two zonotopes. A similar idea was implemented e.g., by Althoff and Krogh [6] who represent sets as the intersection of a finite number of zonotopes.

We provide refinement relations for symbolic representations of hyperplanes and halfspaces. Moreover we provide relators for binary intersection and finite union.

Hyperplanes and Halfspaces. We symbolically represent hyperplanes using the constant *Sctn* that keeps normal vector n and translation c of a hyperplane. It is interpreted as the hyperplane itself or for the halfspace below the hyperplane ($\langle x, n\rangle$ is the inner product).

$$\mathit{plane\text{-}of}(\mathit{Sctn}\ n\ c) := \{x \mid \langle x, n\rangle = c\}$$

$$\mathit{halfspace}\,(\mathit{Sctn}\ n\ c) := \{x \mid \langle x, n\rangle \leq c\}$$

$\langle A\rangle \mathit{sctn}_r$ is the natural relator that allows one to change the representation of the normal vector. With this, we can give a concrete implementation relation for hyperplanes and half-spaces.

$$\langle A\rangle \mathit{plane}_r := \langle A\rangle \mathit{sctn}_r \circ \mathit{br}\ \mathit{plane\text{-}of}\ (\lambda\_.\ \mathit{True})$$

$$\langle A\rangle \mathit{halfspace}_r := \langle A\rangle \mathit{sctn}_r \circ \mathit{br}\ \mathit{halfspace}\ (\lambda\_.\ \mathit{True})$$

Operations are on the implementation side just identity functions, but construct useful abstract objects (namely, the hyperplane or halfspace) on the abstract level:

$$((\lambda x.\ x), \mathit{plane\text{-}of}) \in \langle A\rangle \mathit{sctn}_r \to_r \langle A\rangle \mathit{plane}_r$$
$$((\lambda x.\ x), \mathit{halfspace}) \in \langle A\rangle \mathit{sctn}_r \to_r \langle A\rangle \mathit{halfspace}_r$$

Binary Intersection. With the relation $\langle A, B\rangle \mathit{inter}_r$, we interpret pairs of representations of enclosures as the intersection.

$$((\lambda x\ y.\ (x, y)), \cap) \in A \to_r B \to_r \langle A, B\rangle \mathit{inter}_r$$

There are two operations to extract the original information. *unintersect* adds uncertainty by throwing away the information from the second part of the intersection. *get-inter* extracts both components.

$$\begin{aligned}
&\mathit{unintersect} :: \alpha\ \mathit{set} \to \alpha\ \mathit{set}\ \ \mathit{nres}\\
&\mathit{unintersect}\ \ X \leq \mathit{spec}\,(\lambda R.\ X \subseteq R)\\
&((\lambda(x, y).\ \mathit{return}\ \ x), \mathit{unintersect}) \in \langle A, B\rangle \mathit{inter}_r \to \langle A\rangle \mathit{nres}_r
\end{aligned}$$

$$\begin{aligned}
&\mathit{get\text{-}inter} :: \alpha\ \mathit{set} \to \alpha\ \mathit{set} \times \alpha\ \mathit{set}\ \ \mathit{nres}\\
&\mathit{get\text{-}inter}\ \ X \leq \mathit{spec}\,(\lambda(R, S).\ X = R \cup S)\\
&((\lambda(x, y).\ (x, y)), \mathit{get\text{-}inter}) \in \langle A, B\rangle \mathit{inter}_r \to \langle A \times_r B\rangle \mathit{nres}_r
\end{aligned}$$

Finite Union. For work-set algorithms like the running example (definition 6.4), we realized that it can be convenient to view a concrete set of enclosures abstractly as just one single enclosure. For a relation $A : (\beta \times \alpha\ \mathit{set})\ \mathit{set}$ that implements single enclosures for sets of type α with some concrete representation of type β, and a relation $S : (\sigma \times \beta\ \mathit{set})\ \mathit{set}$ that implements sets of concrete elements β, we define a relation that represents the union of all those elements as follows:

$$\begin{aligned}
&\langle S, A\rangle \mathit{Union}_r : (\sigma \times \alpha\ \mathit{set})\ \mathit{set}\\
&\langle S, A\rangle \mathit{Union}_r := S \circ \langle A\rangle \mathit{set}_r \circ \mathit{br}\ (\lambda X.\ \bigcup_{x \in X} x)(\lambda_.\ \mathit{True})
\end{aligned}$$

Currently, we only use lists to implement the set of concrete representations S, for which we write $\langle A\rangle \mathit{Union}_{lr} := \langle \mathit{list\text{-}set}_r, A\rangle \mathit{Union}_r$, and operations like union or extracting one element

(with the specification $\mathit{split\text{-}spec}_=$) can be implemented with the respective operations on lists/sets:

$$(\lambda xs\ ys.\ \mathit{return}\,(xs@ys), \cup) \in \langle A\rangle \mathit{Union}_{lr} \to_r \langle A\rangle \mathit{Union}_{lr} \to_r \langle A\rangle \mathit{Union}_{lr}$$
$$(\lambda x.\ \mathit{return}\,(\mathit{hd}\,x, \mathit{tl}\ x), \mathit{split\text{-}spec}_=) \in \langle A\rangle \mathit{Union}_{lr} \to_r \langle A \times_r \langle A\rangle \mathit{Union}_{lr}\rangle \mathit{nres}_r$$

6.2.5 Example

Autoref automatically (but sometimes needs guidance) identifies operations from specifications and chooses appropriate relations for their implementations. For the running example definition 6.4, Autoref can be instrumented to implement the work-set X with $\langle \mathit{affine}_r\rangle \mathit{Union}_{lr}$, identify $\mathit{spec}\ (\lambda(Y,Z).\ X = Y \cup Z)$ with $\mathit{split\text{-}spec}_=\ X$, and so on for all operations. In the end, this will yield an implementation on concrete (executable) data structures like affine forms *affine-form* and a deterministic monad of type $\alpha\ \mathit{dres}$ instead of the nondeterministic $\alpha\ \mathit{nres}$ monad.

$$\mathit{global\text{-}optimization}_{impl} :: \mathit{aexp\ list} \to \mathit{affine\text{-}form\ list\ list} \to \mathit{affine\text{-}form\ list\ list\ dres}$$

By the automatic construction, the implementation $\mathit{global\text{-}optimization}_{impl}$ is related to the abstract algorithm *global-optimization*.

Lemma 6.10 (Implementation of *global-optimization*)**.**

$$(\mathit{global\text{-}optimization}_{impl}, \mathit{global\text{-}optimization}) \in$$
$$\langle \mathit{aexp}_{rel}\rangle \mathit{list}_r \to_r \langle \mathit{affine}_r\rangle \mathit{Union}_{lr} \to_r \langle\langle \mathit{affine}_r\rangle \mathit{Union}_{lr}\rangle \mathit{nres}_r$$

For a correctness theorem stated outside the framework of Autoref, the implementation lemma 6.10 and the abstract specification (lemma 6.8) can be combined by unfolding the definitions of the implementing relations.

Theorem 6.11 (Correctness of $\mathit{global\text{-}optimization}_{impl}$)**.**

$$\mathit{global\text{-}optimization}_{impl} = \mathit{dRETURN}\ []$$
$$\forall x \in \bigcup(\mathit{eucl\text{-}of\text{-}list}\,(\mathit{joint\text{-}range}\,(X_0))).\ [\![f_e]\!]_{\mathit{list\text{-}of\text{-}eucl}\ x} \geq 0$$

To summarize, with the presented setup, we could specify an algorithm on an abstract level (definition 6.4) and obtain an implementation $\mathit{global\text{-}optimization}_{impl}$ as well as a corresponding correctness theorem mostly automatically.

6.2.6 Relations to Guide Heuristics

Often, in particular to guide heuristics, an algorithm needs to carry around information that does not influence correctness proofs. An ODE solver for example, modifies its step size, also based on previous values. An implementation needs to carry this information around, but for verifying the algorithm, this only introduces unnecessary clutter. We therefore introduce a relation info_r that carries more information (implemented via A) in the implementation, but keeps the abstract semantics (implemented via B):

$$\langle A, B\rangle \mathit{info}_r := \{((a', b'), b) \mid \exists a.\ (a', a) \in A \wedge (b', b) \in B\}$$

Adding information is simply done by using a pair in the implementation side, semantically, this information is simply discarded ($\mathit{put\text{-}info}\ a\ b := b$). Information can be extracted with *get-info*, which is semantically just an arbitrary element ($\mathit{get\text{-}info}\ b := \mathit{spec}(\lambda\_.\ \mathit{True})$). The implementations are straightforward:

$$(\lambda a\ b.\ (a,b), \mathit{put\text{-}info}) \in A \to_r B \to_r \langle A, B\rangle \mathit{info}_r$$
$$((\lambda(a,b).\ \mathit{return}\ a), \mathit{get\text{-}info}) \in \langle A, B\rangle \mathit{info}_r \to \langle A\rangle \mathit{nres}_r$$

7 A Verified ODE Solver

In this chapter, we assemble all of the results and techniques from the previous chapters: In the framework of chapter 6, we specify an algorithm of Bouissou *et al.*'s [24] for rigorous numerical (chapter 4) enclosures of ODEs (as formalized in chapter 3). Bouissou *et al.*'s [24] algorithm use affine arithmetic evaluations of Runge-Kutta methods.

This forms the basis for rigorous enclosures of Poincaré maps, which are computed geometrically according to chapter 5.

The main results of this chapter are verified algorithms for solutions of ODEs as well as Poincaré maps. These algorithms can be used to either generate code to obtain a highly trusted rigorous ODE solver or as part of tactics to formally prove bounds on ODEs or Poincaré maps in Isabelle/HOL.

The definitions are presented very much simplified and idealized compared to the real implementation in Isabelle/HOL. But the intention is that they still convey the main algorithmic ideas and the organization of the verification.

7.1 Generic Operations

The algorithms in this chapter are phrased in the framework of chapter 6. Most algorithms are generic in the concrete representation of enclosures, for which we assume a relation encl_r and implementations for the abstract operations that are needed for the reachability analysis algorithms: an approximation scheme for expressions $\mathit{approx\text{-}spec}$, enclosures from intervals using an implementation $\mathit{encl\text{-}of\text{-}ivl}$, lower and upper bounds with $\mathit{Inf\text{-}spec}$, $\mathit{Sup\text{-}spec}$, computation of intersections with a plane $\mathit{inter\text{-}spec}_2$ (note that the relation fixes the second argument to represent a plane, abstractly $\mathit{inter\text{-}spec}_2$ is just intersection on sets).

- $(\mathit{approx\text{-}encl}, \mathit{approx\text{-}spec}) \in \mathit{slp}_r \to_r \mathit{encl}_r \to_r \langle\langle \mathit{encl}_r \rangle \mathit{option}_r \rangle \mathit{nres}_r$
- $(\lambda x\ y.\ \mathit{encl\text{-}of\text{-}ivl}\ x\ y, \lambda x\ y.\ [x;y]) \in \mathit{lv}_r \to_r \mathit{lv}_r \to_r \mathit{encl}_r$
- $(\mathit{inf\text{-}encl}, \mathit{Inf\text{-}spec}) \in \mathit{encl}_r \to_r \langle \mathit{lv}_r \rangle \mathit{nres}_r$
- $(\mathit{sup\text{-}encl}, \mathit{Sup\text{-}spec}) \in \mathit{encl}_r \to_r \langle \mathit{lv}_r \rangle \mathit{nres}_r$
- $(\mathit{split\text{-}encl}, \mathit{split\text{-}spec}_{\subseteq}) \in \mathit{real}_r \to_r \mathit{nat}_r \to_r \mathit{encl}_r \to_r \langle \mathit{encl}_r \times_r \mathit{encl}_r \rangle \mathit{nres}_r$
- $(\mathit{inter\text{-}encl\text{-}plane}, \mathit{inter\text{-}spec}_2) \in \mathit{encl}_r \to_r \langle \mathit{lv}_r \rangle \mathit{plane}_r \to_r \langle \mathit{encl}_r \rangle \mathit{nres}_r$

Currently, the only instantiation of this scheme is with affine arithmetic (in this case we set encl_r to affine_r). Nevertheless, this structure keeps the formalization modular and one

can imagine to add further instantiations with e.g., Taylor models, or centered forms in the future.

7.2 Rigorous Runge-Kutta Methods

Runge-Kutta methods are a popular choice for numerically approximating the solution of ODEs. We will assume an autonomous ODE

$$\dot{x}\ t = f\ (x\ t)$$

with a right hand side $f :: \mathbb{R}^n \to \mathbb{R}^n$ that is smooth on some domain (which we often denote by X). We also assume that f is represented by some arithmetic expression (section 4.1) f_e :: *aexp list*, i.e., $f = (\lambda x.\ [\![f_e]\!]_{\textit{list-of-eucl}\ x}) =: [\![f_e]\!]$, such that f can be approximated with affine arithmetic (section 4.3.9) or generically with *approx-spec*.

Runge-Kutta methods are one-step methods, where the idea is to approximate the solution $\phi(x_0, h)$ with a line segment whose slope may depend on the initial value x_0 and the step size h. In this work, we consider Euler's method *euler* and a generic two-stage Runge-Kutta method $\textit{rk2}_p$. Compare also figure 7.1. Euler's method takes as slope for the line segment the slope of the solution at the initial value (given by the right-hand side f of the ODE). The two-stage Runge-Kutta method evaluates the right-hand side f for two arguments: once (like the Euler method) at the initial value x_0 and once (depending on a parameter p) on the line segment given by an Euler-like approximation $(x_0 + hpf(x_0))$.

Definition 7.1 (Euler's Method and Two-stage Runge-Kutta Method)**.**

$$\textit{euler}(x_0, h) := x_0 + h \cdot f(x_0)$$

$$\textit{rk2}_p(x_0, h) = x + h \cdot \psi_p(x_0, h)$$

where

$$\psi_p(x_0, h) = (1 - \frac{1}{2p}) f(x_0) + \frac{1}{2p} f(x_0 + hpf(x_0))$$

We will see that Euler's method approximates the flow with a quadratic error, whereas the two-stage Runge-Kutta method *rk2* yields an error that is cubic in the step size:

$$\|\phi(x_0, h) - \textit{euler}(x_0, h)\| \in \mathcal{O}(h^2)$$
$$\|\phi(x_0, h) - \textit{rk2}_p(x_0, h)\| \in \mathcal{O}(h^3)$$

The quality of these approximations is proved by comparing the Taylor series expansions of both the solution and the approximation scheme. This also yields explicit bounds on the constants hidden in the $\mathcal{O}$-notation, which we need for rigorous bounds enclosing the solution. We will come to a discussion on the explicit bounds in section 7.2.2, right after talking about multivariate Taylor series expansions and how they are formalized in Isabelle/HOL in section 7.2.1.

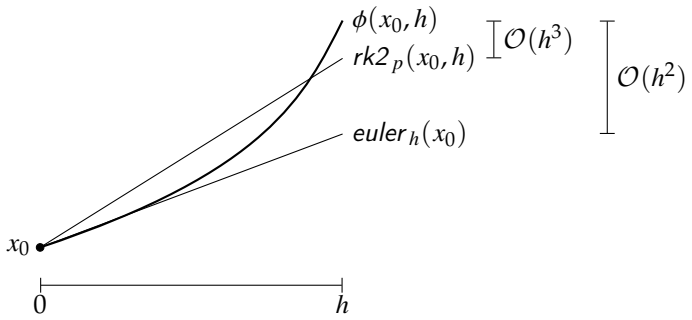


Figure 7.1: Illustration of flow ϕ, Euler's method $euler_h(x_0)$, and Runge-Kutta method $rk2_p(x_0, h)$.

7.2.1 Multivariate Taylor Series Expansion

Taylor series expansions for functions in several variables bear some complications, in particular for the formalization. A proper choice of concepts and notation is extremely important. It could e.g., be done with higher partial derivatives, but then one risks drowning in indices. The total derivative (section 2.6) summarizes all of the information of continuous partial derivatives, so this helps for an abstract and clear notation.

One complication is that higher (total) derivatives are multilinear mappings, and the type of the mapping would depend on the order: Consider e.g., a function $f : \alpha \to \beta$ and its derivative $f'x := \mathsf{D}f|_{at\,x}$. The derivative of $f' :: \alpha \to \alpha \to_{bl} \beta$ is the function $f''x := \mathsf{D}f'|_{at\,x}$ with $f'' :: \alpha \to \alpha \to_{bl} \alpha \to_{bl} \beta$. How do we encode this? The solution in the case of multivariate Taylor series expansions is easy: higher derivatives are repeatedly applied to the same argument. The n-th derivative of f at x, e.g., is used only as $(f^{(n)}\ x) \cdot_{bl} h \cdot_{bl} \ldots \cdot_{bl} h$ with n applications of h.

But this can be encoded in a single type: that is, we use a function $f^{(n)}\ x\ d_1 \cdot_{bl} d_2$, which is supposed to denote the n-th higher derivative of f, evaluated at x and applied n times to d_1. The multivariate Taylor series expansion can then be expressed for arbitrary vector spaces (Banach in the image) and an integral bounding the remainder term:

Theorem 7.2 (Multivariate Taylor Series Expansion)**.** *For* $f :: \alpha ::$ *real-normed-vector* $\to \beta ::$ *banach*

$$
\begin{aligned}
&n > 0 \longrightarrow \\
&(\forall x.\ f^{(0)}\ x\ h\ h = f\ x) \longrightarrow \\
&(\forall x.\ \forall d.\ \forall i < n.\ \mathsf{D}(\lambda x.\ f^{(i)}\ x\ h\ h)|_{at\ x} \cdot d = f^{(i+1)}\ x\ d) \longrightarrow \\
&f\ (x+h) = \sum_{i<n} \frac{1}{i!} f^{(i)}\ x\ h\ h + \int_0^1 \frac{(1-\xi)^{n-1}}{(n-1)!} f^{(n)}\ (x+\xi h)\ h\ h\,\mathrm{d}\xi
\end{aligned}
$$

7.2.2 Approximation Error

As a result of comparing the multivariate Taylor series expansion of the solution $\phi(x_0, h)$ and $rk2_p(x_0, h)$, we can enclose the error of the two-stage Runge-Kutta method with the help of

the first and second derivative of the right hand side f of the ODE (details can be found in textbooks on numerical approximations of ODEs, e.g., [35]). If we set $f' := \lambda x.\ \mathsf{D} f|_{\mathsf{at}\ x}$ and $f'' := \lambda x.\ \mathsf{D} f'|_{\mathsf{at}\ x}$, then the remainder is contained in the convex hull of any set that contains $\mathit{rk2\text{-}remainder}_h(x_0, p, s, X)$, where X is an enclosure for the evolution of ϕ.

Lemma 7.3 (Runge-Kutta Method with Remainder Term)**.** *For $0 < p \leq 1$ and a convex a-priori enclosure X for the flow $\phi(x_0, [0;h]) \subseteq X$:*

$$\phi(x_0, h) \in \mathit{rk2}_h(x_0) + \mathit{convex\text{-}hull}\,(\mathit{rk2\text{-}remainder}_h(x_0, p, [0;1], X))$$

where

$$\begin{aligned}\mathit{rk2\text{-}remainder}_h(x_0, p, s, X) := \frac{h^3}{2} \cdot \Big(& \\ \frac{1}{3}(f''(X)) \cdot_{\mathit{bl}} (f(X)) \cdot_{\mathit{bl}} (f(X)) + (f'(X)) \cdot_{\mathit{bl}} (f'(X)) \cdot_{\mathit{bl}} (f(X)) & \\ - \frac{p}{2} f''\big(x_0 + hps_2(f\ x_0)\big) \cdot_{\mathit{bl}} (f\ x_0) \cdot_{\mathit{bl}} (f\ x_0)\Big) &\end{aligned}$$

The convex hull and the evaluation of intervals for s stem from safe enclosures for the remainder terms of the Taylor series expansions. A similar result holds for the remainder of Euler's method, but we will not go into the details here.

7.2.3 Rigorous Enclosures

According to lemma 7.3, a safe bound for the Runge-Kutta method depends on an a-priori enclosure X for the solution $\phi(x_0, [0;h]) \subseteq X$. For the moment, let us assume that we dispose of such an enclosure. The Runge-Kutta method can then be used to tighten this enclosure: X is only used in $\mathit{rk2\text{-}remainder}_h$, which is scaled by h^3, and h is usually small.

Because we assumed that f is represented by an expression $f_e :: \mathit{aexp}\ \mathit{list}$, it is straightforward to define an *aexp* expression *rk2-aexp* that is interpreted as Runge-Kutta method with remainder term $\mathit{rk2}_h(x_0) + \mathit{rk2\text{-}remainder}_h(x_0, p, s, X)$. The expressions for the derivatives f' and f'' that occur in *rk2-remainder* are computed symbolically from f_e via $\mathsf{D}^{1,2}$ according to section 4.1.2.

Lemma 7.4 (Deeply Embedded Expression for Runge-Kutta Method)**.**

$$[\![\mathit{rk2\text{-}aexp}]\!]_{x_0,h,p,s,X} = \mathit{rk2}_h(x_0) + \mathit{rk2\text{-}remainder}_h(x_0, p, s, X)$$

Together with the fundamental property of rigorous numerics, this can be used to enclose the solution of an ODE, according to the following calculation ($\forall x_0 \in X_0$ and $\forall h \in H$):

$$\begin{aligned}\phi(x_0, h) &\in \mathit{rk2}_h(x_0) + \mathit{rk2\text{-}remainder}_h(x_0, p, [0;1], X) \\ &= [\![\mathit{rk2\text{-}aexp}]\!]_{x_0,h,p,[0;1],X} \\ &\subseteq \mathit{approx}\ \mathit{rk2\text{-}aexp}\ (X_0, h, p, [0;1], X)\end{aligned}$$

This says that, when we have an enclosure X_0 for the initial value x_0 (or a set of initial values), and an a-priori enclosure X for the solution ($\phi(X_0, [0;h]) \subseteq X$), we can evaluate *rk2-aexp* with a safe approximation scheme *approx* to obtain a (hopefully) tighter enclosure for the solution.

Lemma 7.5 (Runge-Kutta Enclosure)**.**

$$\phi(X_0, [0;h]) \subseteq X \longrightarrow \phi(X_0, h) \in \textit{approx rk2-aexp } (X_0, h, p, [0;1], X)$$

7.2.4 Certification of Step: A-Priori Enclosures

If we actually want to compute an enclosure for ϕ (which we do want to!), lemmas 7.3 and 7.5 leave us with a cyclic dependence: The Runge-Kutta method $\textit{rk2}_h$ can be used to compute an enclosure for ϕ, but only if we have an enclosure X for $\phi(x_0, [0;h])$ to estimate the approximation error $\textit{rk2-remainder}_h(x_0, p, s, X)$. Moreover, we do not even know if $\phi(x_0, [0;h])$ is well-defined, i.e., we do not know if $h \in \textit{ex-ivl}\,(x_0)$.

We resolve these problems in the same way Bouissou *et al.* [24] did: we construct a rough a-priori enclosure X for $\phi(x_0, [0;h])$. The construction will also guarantee $h \in \textit{ex-ivl}\,(x_0)$.

The idea is to certify the existence of a unique solution according to the Picard-Lindelöf theorem 3.7. We recall the assumptions (with the Picard operator $P(\phi) = (\lambda t.\ x_0 + \int_{t_0}^{t} f\,(\tau, \phi\ \tau)\,\mathrm{d}\tau)$) of the theorem (simplified for the special case of autonomous right-hand side f) here:

$$\begin{aligned}
&\textit{closed } X \wedge \\
&\textit{lipschitz } X\ f\ L \wedge \\
&(\forall \phi \in ([0;h] \to X).\ \phi\ 0 = x_0 \wedge \textit{continuous-on}\,[0;h]\ \phi \longrightarrow P(\phi) \in ([0;h] \to X))
\end{aligned}$$

In our construction, candidates for X will be (compact, closed) intervals, they are therefore *closed* and satisfy the first assumption. Since f is smooth (and in particular $\mathcal{C}^1$), there exists a Lipschitz constant L on the compact (!) interval X.

It remains to be shown that P is an endomorphism on the space of continuous functions from $[0;h]$ to X. Like Bouissou [24], we use the set-based overapproximation

$$Q(X) := x_0 + [0;h] \cdot (f\ X)$$

of the operator P. Q is an overapproximation of P in the following sense: for a set X with $\phi(x_0, [0;h]) \subseteq X$, $t \in [0;h]$ and continuous ϕ, the following holds.

$$\begin{aligned}
P(\phi)\ t &= x_0 + \int_0^t f\,(\tau, \phi\ \tau)\,\mathrm{d}\tau \\
&\subseteq x_0 + \int_0^t f(X)\mathrm{d}\tau \\
&\subseteq x_0 + t \cdot f(X) \\
&\subseteq x_0 + [0;h] \cdot f(X) \\
&\subseteq Q(X)
\end{aligned}$$

A post-fixed point X of Q, i.e., $Q(X) \subseteq X$, implies that P is an endomorphism on $([0;h] \to X)$, i.e., $P(\phi) \in ([0;h] \to X)$ for continuous $\phi \in ([0;h] \to X)$. Therefore, a post fixed-point of Q certifies the existence of a unique solution (according to theorem 3.7). Moreover the solution is bounded by X.

$$Q(X) \subseteq X \longrightarrow \exists \psi.\ (\psi\ \textit{uniquely-solves-ode}\ f\ \textit{from}\ 0)\ [0;h]\ X$$

From the existence and uniqueness of one solution, we can deduce that the flow ϕ is defined and bounded by X.

Lemma 7.6 (Condition for A-Priori Bound)**.**

$$Q(X) \subseteq X \longrightarrow h \in \textit{ex-ivl}\,(x_0) \wedge \phi(x_0, [0;h]) \subseteq X$$

The remaining question is how to come up with a post-fixed point of Q. This can be achieved by looking at iterated applications of Q on an initial enclosure $X_0 \ni x_0$ until a post fixed point $Q^{i+1}(X_0) \subseteq Q^i(X_0)$ is reached after the i-th iteration.

Note that it is possible that the iteration of Q does not reach a fixed point if the step size is too large – one can then repeat the phase with a smaller step size. It is also possible to accelerate the iteration by including some sort of widening into the operator Q. The definition *cert-stepsize* sketches one possible implementation (which restarts with half the step size after a maximum of k iterations).

Definition 7.7 (Certification of Step)**.**

$$\begin{aligned}
&\textit{cert-stepsize}\ X_0\ h := \\
&\quad \textbf{\textit{if}}\ \exists i \leq k.\ Q^{i+1}(X_0) \subseteq Q^i(X_0) \\
&\quad \textbf{\textit{then return}}\ (h, Q^{i+1}(X_0)) \\
&\quad \textbf{\textit{else}}\ \textit{cert-stepsize}\ X_0\ (\frac{h}{2})
\end{aligned}$$

cert-stepsize satisfies the specification that it returns a step size h and an enclosure X such that a time step h is well-defined ($h \in$ *ex-ivl*) and X bounds the evolution of ϕ.

Lemma 7.8 (Certification of Step)**.**

$$\textit{cert-stepsize}\ X_0\ h_0 \leq \textit{spec}\ (\lambda(h, X).\ \forall x \in X_0.\ h \in \textit{ex-ivl}\,(x_0) \wedge \phi(x_0, [0;h]) \subseteq X)$$

7.2.5 One-Step Method with Adaptive Step Size Control

We will now use *cert-stepsize* and the approximation scheme *rk2-aexp* to implement and verify a rigorous one-step method. That is, a method that encloses the flow of an ODE in a series of discrete steps in time. Consider figure 7.2. Starting from an initial set X_0, we use *cert-stepsize* for an a-priori bound C_0 that certifies the existence of a solution for time step h_0. With *rk2-aexp*, we obtain a tighter enclosure X_1 at time h_0. We can also obtain a tighter enclosure on the interval $[0;h_0]$ by applying *rk2-aexp* to the time interval $[0;h_0]$ (dark gray). This process is repeated for successive steps of size $h_1, h_2, h_3, \ldots$ producing enclosures $X_1, X_2, X_3, \ldots$ and so on.

A single step is implemented as follows. We use *rk2-remainder-aexp* to obtain a bound on the approximation error, which we will use later on for an adaptive step size control.

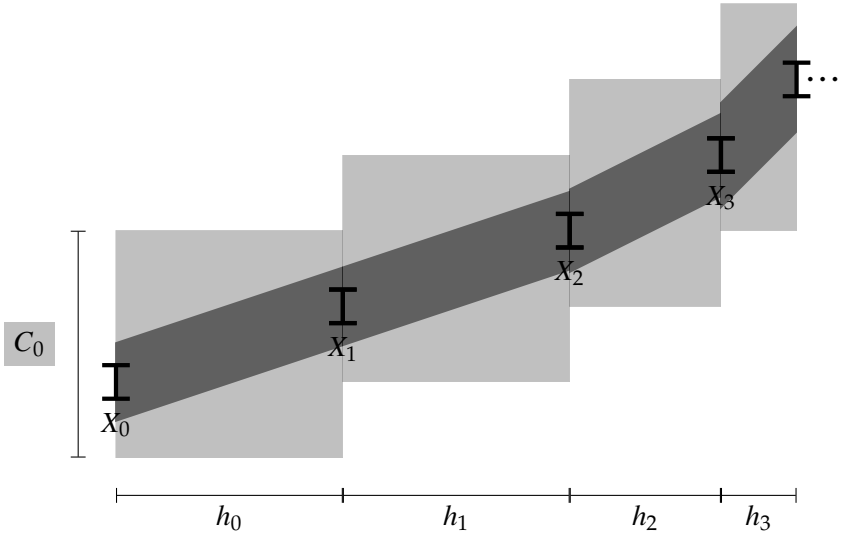


Figure 7.2: Illustration of rigorous one-step method.

Definition 7.9.

$$
\begin{aligned}
&\mathit{rkstep}\ X_0\ h_0 := \\
&\quad \textbf{\textit{do}}\ \{ \\
&\qquad (h, C_0) \leftarrow \mathit{cert\text{-}stepsize}\ X_0\ h_0 \\
&\qquad e \leftarrow \mathit{approx\text{-}spec}\ \mathit{rk2\text{-}remainder\text{-}aexp}\ (X_0, h, p, [0;1], C_0) \\
&\qquad X_1 \leftarrow \mathit{approx\text{-}spec}\ \mathit{rk2\text{-}aexp}\ (X_0, h, p, [0;1], C_0) \\
&\qquad C \leftarrow \mathit{approx\text{-}spec}\ \mathit{rk2\text{-}aexp}\ (X_0, [0;h], p, [0;1], C_0) \\
&\qquad \textbf{\textit{return}}(h, C, X_1, e) \\
&\quad \}
\end{aligned}
$$

Combining *cert-stepsize* with a rigorous Runge-Kutta step *rk2-aexp*, *rkstep* satisfies the specification that results from combining lemma 7.8 with the specification of Runge-Kutta enclosure (lemma 7.5).

Lemma 7.10 (Certified Runge-Kutta Step)**.**

$$\mathit{rkstep}\ X_0\ h_0 \leq \mathit{spec}\,(\lambda(h, C, X_1, e).\ \forall x_0 \in X_0.\ h \in \mathit{ex\text{-}ivl}\,(x_0) \wedge \phi(x_0, h) \in X_1 \wedge \phi(x_0, [0;h]) \subseteq C$$

The following definition *one-step* iterates single steps of *rkstep* in a simple loop. In the loop (over (t, X, h)), it keeps track of the total time t, the enclosure X at time t and the next step size h.

Definition 7.11.

$$
\begin{array}{l}
\textit{one-step}\ X_0\ t_{end} := \\
\quad \textbf{\textit{do}}\ \{ \\
\quad\quad (X, \_, \_) \leftarrow \\
\quad\quad\quad \textbf{\textit{while}}\ (\lambda(X,t,h).\ t < t_{end})\ (\lambda(X,t,h).\ \textbf{\textit{do}}\ \{ \\
\quad\quad\quad\quad (h, \_, Y, e) \leftarrow \textit{rkstep}\ X\ (\max(h, t_{end} - t)) \\
\quad\quad\quad\quad t \leftarrow t + h \\
\quad\quad\quad\quad h \leftarrow \textit{adapt-stepsize}\ h\ e \qquad\qquad //\ \ldots = \textit{spec}\ (\lambda\_.\ \textit{True}) \\
\quad\quad\quad\quad \textbf{\textit{return}}(Y,t,h) \\
\quad\quad\quad \}) \\
\quad\quad\quad (X_0, 0, h_{start}) \\
\quad\quad \textbf{\textit{return}}\ X \\
\quad \}
\end{array}
$$

The loop invariant of $\textit{one-step}\ X_0\ t_{end}$ is $\lambda(X,t,h).\ \forall x_0 \in X_0.\ t \in \textit{ex-ivl}(x_0) \wedge \phi(x_0,t) \in X$. It directly yields the correctness theorem for *one-step*:

Theorem 7.12 (Correctness of One-Step Method)**.**

$$\textit{one-step}\ X_0\ t_{end} \leq \textit{spec}\ (\lambda X.\ t_{end} \in \textit{ex-ivl}(x_0) \wedge \phi(x_0, t_{end}) \in X)$$

The correctness theorem follows from combining the loop invariant and the specification lemma 7.10 of *rkstep* with the flow property theorem 3.18. The next stepsize h is not relevant for the correctness and neither is the actual algorithm for adapting the step size. *adapt-stepsize* only needs to fulfill the trivial specification $\textit{spec}\ (\lambda\_.\ \textit{True})$, one is therefore free to choose any implementation for *adapt-stepsize* (We use a method similar to the one described by Bouissou *et al.* [24]) and neither the particular implementation nor h induce unnecessary clutter in the verification of *one-step*.

7.3 Poincaré Maps

The Poincaré map simplifies reasoning about the dynamics of an ODE, because one need not consider temporal dependencies. Recall section 3.4, the Poincaré map is defined as the flow at the return time $P(x) := \phi(x, \tau(x))$. In a sense, the definition of P hides the dependency of ϕ on the time variable.

We do exploit this simplification for our verification of rigorous numerical algorithms enclosing the Poincaré map: We formalize the enclosure of an evolution from an initial set X to some other set Y with the ternary predicate $\curvearrowright$, where $X \curvearrowright_C Y$ holds if the evolution flows every point of $X \subseteq \mathbb{R}^n$ to some point in $Y \subseteq \mathbb{R}^n$ and does not leave the set C in the meantime. We call C the *flowpipe* from X to Y. Note that $\curvearrowright$ does not involve any parameters for the dependency of ϕ on time.

Definition 7.13 (Flows-to Predicate)**.**

$$X \curvearrowright_C Y := \forall x \in X.\ \exists t \geq 0.\ \phi(x_0, t) \in Y \wedge (\forall s \in [0;t].\ \phi(x_0, s) \in C)$$

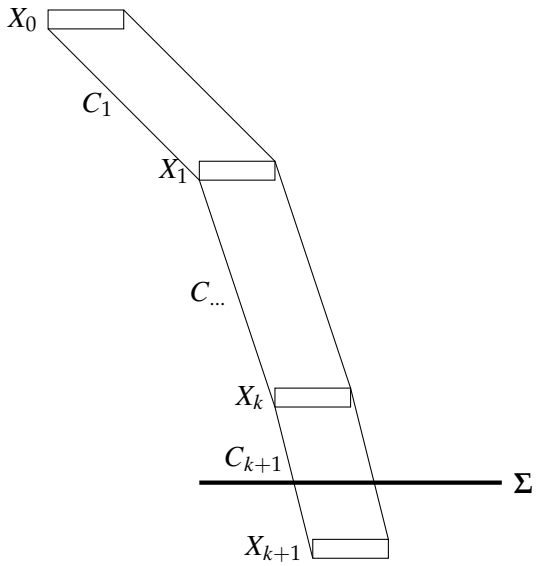


Figure 7.3: Computation of enclosure for Poincaré map.

We write $X \curvearrowright Y$ if C is irrelevant or the domain of f.

The main goal of this section is to compute a Poincaré map from an initial set X_0 onto a Poincaré section Σ (we restrict ourselves to hyperplanes, i.e., $\Sigma = \{x \mid \langle x, n \rangle = c\}$ for some normal vector n and translation c). The basic idea (compare figure 7.3) is as follows: starting from a set X_0, perform a series of steps $X_1, X_2, \ldots$ in the style of *one-step*. With the flows-to predicate, this yields $X_0 \curvearrowright_{C_1} X_1 \curvearrowright_{C_2} \ldots \curvearrowright_{C_k} X_k$, which composes to $X_0 \curvearrowright X_k$.

Stop (at X_k) before the evolution would cross the Poincaré section in the subsequent step. Then perform another step X_{k+1} to cross Σ. This step $X_k \curvearrowright_{C_{k+1}} X_{k+1}$ implies $P(X_k) \subseteq (C_{k+1} \cap \Sigma)$. With $X_0 \curvearrowright X_k$, this composes to $P(X_0) \subseteq (C_{k+1} \cap \Sigma)$. Therefore $C_{k+1} \cap \Sigma$ is the desired overapproximation of the Poincaré map $P(X_0)$.

We need to, however, add some refinements to the basic idea: If reachable sets grow above a given threshold, we perform subdivisions to increase precision (section 7.3.1). Furthermore, one cannot ensure that C_{k+1} will always end up on the other side of Σ. Therefore the step from C_k to C_{k+1} in the basic idea is actually another sequence of steps, until the currently reachable set is on the other side of Σ. This results in a collection of enclosures for the intersection, which typically overlap. This in turn results in redundant computations when continuing from there. To avoid this, we include a method to remove such overlaps (section 7.3.3). Furthermore, we want to compose a series of intermediate Poincaré maps (section 7.3.4).

7.3.1 Reachability Loop

single-step is defined to perform one step in the reachability analysis up to the Poincaré section. The implementing relation of *single-step* reads as follows. We define it to either split the argument set or perform one Runge-Kutta step.

Definition 7.14.

$$
\begin{array}{l}
\mathit{single\text{-}step}\ X := \\
\quad \textbf{\textit{do}}\ \{ \\
\quad\quad w \leftarrow \mathit{width\text{-}spec}\ X \\
\quad\quad h \leftarrow \mathit{get\text{-}stepsize}\ X \\
\quad\quad \textbf{\textit{if}}\ w \leq \mathit{max\text{-}width} \\
\quad\quad \textbf{\textit{then do}}\ \{ \\
\quad\quad\quad (h, \_, Y, C) \leftarrow \mathit{rkstep}\ X\ h \\
\quad\quad\quad h' \leftarrow \mathit{adapt\text{-}stepsize}\ h\ e \\
\quad\quad\quad \textbf{\textit{return}}(\mathit{put\text{-}info}\ h'\ Y, C) \\
\quad\quad \}\ \textbf{\textit{else do}}\ \{ \\
\quad\quad\quad (Y, Z) \leftarrow \mathit{split\text{-}spec}_{\subseteq}\ X \\
\quad\quad\quad \textbf{\textit{return}}(\mathit{put\text{-}info}\ h\ (Y \cup Z), (Y \cup Z)) \\
\quad\quad \} \\
\quad \}
\end{array}
$$

single-step is an example of where we hide implementation details in the implementing relations. In order to make sense of this definition, it is instructive to look at the actual implementing relation.

$$
\begin{array}{l}
(\mathit{single\text{-}step}_{\mathit{impl}}, \mathit{single\text{-}step}) \in \\
\quad \langle \mathit{real}_r, \mathit{encl}_r \rangle \mathit{info}_r \rightarrow \langle \langle \langle \mathit{real}_r, \mathit{encl}_r \rangle \mathit{info}_r \rangle \mathit{Union}_{lr}, \langle \mathit{encl}_r \rangle \mathit{Union}_{lr} \rangle \mathit{nres}_r
\end{array}
$$

The information on the last (and next) step size is reflected in the refinement relation $\langle \mathit{real}_r, \mathit{encl}_r \rangle \mathit{info}_r$. The fact that *single-step* either returns one set (after performing the Runge-Kutta step) or two sets (after splitting) is hidden behind Union_{lr}. In contrast to the implementation relation, the type of $\mathit{single\text{-}step} :: \mathbb{R}^n\ \mathit{set} \rightarrow (\mathbb{R}^n\ \mathit{set} \times \mathbb{R}^n\ \mathit{set})$ is as abstract as one would like it to be, it does not clutter the verification. The correctness theorem can therefore be stated as simple as follows:

Theorem 7.15. $\mathit{single\text{-}step}\ X\ h \leq \mathit{spec}\,(\lambda(C, Y).\ X \curvearrowright_C Y)$

For the verification, $\mathit{width\text{-}spec}\ X = \mathit{spec}\ (\lambda\_.\ \mathit{True})$ can be ignored, because it only implements a heuristic value for the width of the argument set X. The specification follows in the if-branch from the correctness of a Runge-Kutta step (lemme 7.10), in the else branch from the specification of $\mathit{split\text{-}spec}_{\subseteq}$ and the fact that $X \curvearrowright Y$ holds whenever $X \subseteq Y$ (choose time $t = 0$ in the definition of the flows-to predicate $\curvearrowright$).

Note that *single-step* returns (from an implementation point of view) a collection of enclosures implemented using $\langle \mathit{encl}_r \rangle \mathit{Union}_{lr}$, so we need some sort of work-set algorithm to resolve all currently reachable sets. The continuous reachability loop *reach-cont* does so:

Definition 7.16 (Continuous Reachability Loop)**.**

$$
\begin{array}{l}
\mathit{reach\text{-}cont}\ \Sigma\ X_0 := \\
\quad \textbf{\textit{do}}\ \{ \\
\quad\quad (\_, C, I) \leftarrow \\
\quad\quad\quad \textbf{\textit{while}}\ (\lambda(X, C, I).\ X \neq \emptyset)\ (\lambda(X, C, I).\ \textbf{\textit{do}}\ \{ \\
\quad\quad\quad\quad (X_1, X_2) \leftarrow \mathit{split\text{-}spec}_{=} X \\
\quad\quad\quad\quad (Y_1, C_1) \leftarrow \mathit{single\text{-}step}\ X_1 \\
\quad\quad\quad\quad d \leftarrow \mathit{disjoint\text{-}spec}\ C_1\ \mathit{sctns} \\
\quad\quad\quad\quad \textbf{\textit{if}}\ d \\
\quad\quad\quad\quad \textbf{\textit{then return}}\ (X_2 \cup Y_1, C \cup C_1, I) \\
\quad\quad\quad\quad \textbf{\textit{else return}}\ (X_2, C, I \cup X_1) \\
\quad\quad\quad \})) \\
\quad\quad\quad (X_0, \emptyset, \emptyset) \\
\quad\quad \textbf{\textit{return}}\ (C, I) \\
\quad \}
\end{array}
$$

The loop in *reach-cont* maintains three kinds of sets (see also figure 7.4): X is the collection of sets whose future reachable sets still need to be explored. C is the collection of all flowpipes explored so far. I is the collection of sets where reachability analysis has stopped because of an intersection with the Poincaré section Σ. The algorithm takes one element out of the work-set X by splitting the collection of enclosures using $\mathit{split\text{-}spec}_{=}$, performs a single step, and checks for an intersection with one of the Poincaré sections and updates X, C, and I accordingly.

The loop invariant of *reach-cont* is roughly the following: Elements from X_0 flow via C to X and I, while avoiding Σ.

$$X_0 \curvearrowright_C (X \cup I) \wedge C \cap \Sigma = \emptyset$$

The specification of *reach-cont* is therefore relatively simple:

Theorem 7.17. $\mathit{reach\text{-}cont}\ \Sigma\ X_0 \leq \mathit{spec}(\lambda(C, I).\ X_0 \curvearrowright_C I \wedge C \cap \Sigma = \emptyset)$

It is worth noticing that this simplicity is due to the fact that the work-list and heuristic info on individual step sizes is hidden via refinement relations in the implementation.

$$
\begin{array}{l}
(\mathit{reach\text{-}cont}_{\mathit{impl}}, \mathit{reach\text{-}cont}) \in \\
\quad \langle \mathit{lv}_r \rangle \mathit{plane}_r \rightarrow \langle \langle \mathit{real}_r, \mathit{encl}_r \rangle \mathit{info}_r \rangle \mathit{Union}_{lr} \rightarrow \langle \langle \langle \mathit{real}_r, \mathit{encl}_r \rangle \mathit{info}_r \rangle \mathit{Union}_{lr} \rangle \mathit{nres}_r
\end{array}
$$

If this were represented on the specification level (e.g., by using sets of enclosures paired with their current step size), the specification would have to be much more cluttered:

$$(\bigcup_{(h,x)\in X_0} x) \curvearrowright_C ((\bigcup_{(h,x)\in X} x) \cup (\bigcup I))$$

Such a specification distracts the user and also automatic proof tools, and we are therefore happy to hide this in the abstraction.

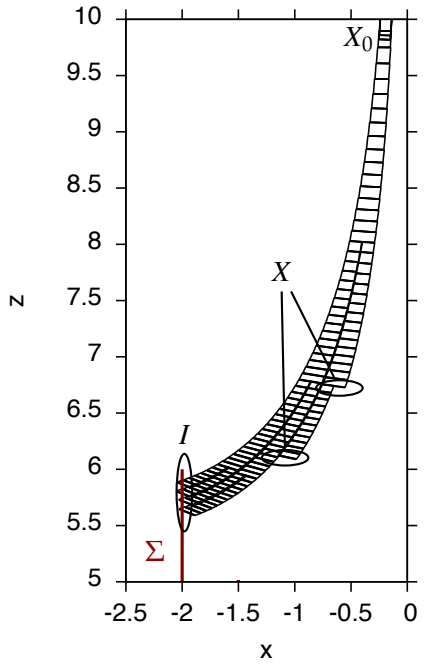


Figure 7.4: Continuous reachability loop

7.3.2 Resolve Intersection

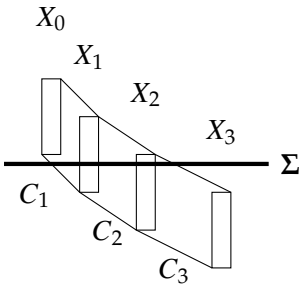


Figure 7.5: Iteration for computation of Poincaré map.

The algorithm *reach-cont* performs reachability analysis until each enclosure in I is just about to intersect Σ. Hidden in the implementing relation, I is actually a union of individual enclosures. Starting from one such enclosure X_0, we compute the Poincaré map in an iteration of reachable sets $X_0, X_1, \ldots$ until the reachable set (here X_3) is below the hyperplane Σ, compare figure 7.5. During the iteration, we collect the intersection of the flowpipes $(C_1 \cup C_2 \cup C_3)$ with Σ. This is implemented in the definition of *intersect-flow*.

Definition 7.18.

$$
\begin{array}{l}
\mathit{intersect\text{-}flow}\ \Sigma\ X_0 := \\
\quad \textbf{\textit{do}}\ \{ \\
\quad\quad (X, J) \leftarrow \\
\quad\quad\quad \textbf{\textit{while}}\ (\lambda(X, J).\ \neg X \subseteq \Sigma_{\leq})\ (\lambda(X, J).\ \textbf{\textit{do}}\ \{ \\
\quad\quad\quad\quad (\_, C, Y, \_) \leftarrow \mathit{rkstep}\ X \\
\quad\quad\quad\quad C' \leftarrow \mathit{inter\text{-}spec}_2\ C\ \Sigma \\
\quad\quad\quad\quad \textbf{\textit{return}}\ (Y, C') \\
\quad\quad\quad \}) \\
\quad\quad\quad (X_0, \emptyset) \\
\quad\quad \textbf{\textit{return}}\ J \\
\quad \}
\end{array}
$$

Recall that we assumed $\Sigma = \{x \mid \langle x, n\rangle = c\}$. We denote the halfspace (strictly) above Σ with $\Sigma_{>} = \{x \mid \langle x, n\rangle > c\}$, similar for $\Sigma_{\leq}$. The loop invariant in *intersect-flow* is that X_0 can be split in two sets $X_0 = A \cup B$ such that A flows (above Σ) to the part of X which is above Σ and the Poincaré map of B is enclosed by J:

$$\exists A.\ \exists B.\ X_0 = A \cup B \wedge A \curvearrowright_{\Sigma_{>}} (X \cap \Sigma_{>}) \wedge P(B) \subseteq J$$

Runge-Kutta step and intersection of the flowpipe with Σ as in the body of the loop maintain this invariant (this is a relatively straightforward consequence of the definitions of flow and Poincaré map).

Upon termination, X is below Σ, i.e., $X \subseteq \Sigma_{\leq}$. From the loop invariant, we have that $A \curvearrowright_{\Sigma_{>}} (X \cap \Sigma_{>})$. But this implies $(X \cap \Sigma_{>} = \emptyset)$ that $A \curvearrowright \emptyset$ and therefore $A = \emptyset$ and $B = X_0$. This yields the specification of *intersect-flow*:

Lemma 7.19 (Correct Resolution of Poincaré Map)**.**

$$\mathit{intersect\text{-}flow}\ \Sigma\ X_0 \leq \mathit{spec}\ (\lambda J.\ P(X_0) \subseteq J)$$

The implementation of *intersect-flow* is guided by this implementing relation:

$$(\mathit{intersect\text{-}flow}_{\mathit{impl}}, \mathit{intersect\text{-}flow}) \in \langle \mathit{lv}_r \rangle \mathit{plane}_r \rightarrow \langle \langle \mathit{encl}_r \rangle \mathit{Union}_{lr} \rangle \mathit{nres}_r$$

7.3.3 Summarization of Intersections

When the intersection is computed by flowing the reachable set through the hyperplane step by step, the set J is implemented as consisting of individual intersections I_i. Many of the sets I_i usually overlap, in order to avoid redundant enclosures, it is desirable to remove the overlaps. Summarizing at a Poincaré section is also a means to counteract the (due to splitting) increasing number of reachable sets.[1]

[1] An example of reduction of number of reachable sets can be seen in the plot of figure 8.5, in particular at $x = \pm 0.1$ and $x = \pm 0.75$.

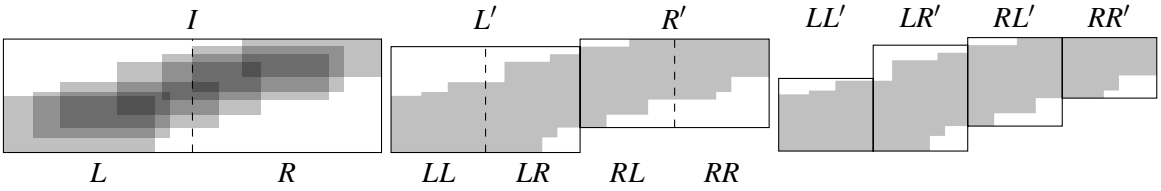


Figure 7.6: Summarization of intersections: overlapping sets J (gray) and interval enclosure I. Split $I = L \cup R$, reduction $L \rightsquigarrow L'$, $R \rightsquigarrow R'$, split $L' = LL \cup LR$, split $R' = RL \cup RR$, and reduction $LL \rightsquigarrow LL'$, $LR \rightsquigarrow LR'$, $RL \rightsquigarrow RL'$, $RR \rightsquigarrow RR'$.

The (very naive) overapproximation that we use successively refines a union of interval enclosures. Compare also figure 7.6: start with an interval enclosure I of the union of overlapping sets $J \subseteq I$. Split the interval I such that $I = L \cup R$. Then the size of L and R can be reduced to L' and R' by restricting L respectively R to the part which has a non-empty intersection with J. That is, $L \rightsquigarrow L'$ if $L \cap J \subseteq L'$, similar for $R \rightsquigarrow R'$. Then proceed recursively until the widths of the resulting enclosures are below a given threshold.

We call the resulting algorithm *reduce*, the only specification it needs to fulfill is that the result is a safe overapproximation:

Lemma 7.20. *Correct Summarization*

$$\mathit{reduce}\ X \leq \mathit{spec}\ (\lambda Y.\ X \subseteq Y)$$

It is implemented to operate on a union of enclosures encl_r, and internally converts back and forth between enclosures encl_r and intervals $\langle \mathit{lv}_r \rangle \mathit{ivl}_r$.

$$(\mathit{reduce}_{\mathit{impl}}, \mathit{reduce}) \in \langle \mathit{encl}_r \rangle \mathit{Union}_{\mathit{lr}} \to \langle \langle \mathit{encl}_r \rangle \mathit{Union}_{\mathit{lr}} \rangle \mathit{nres}_r$$

7.3.4 Intermediate Poincaré Maps

The overall algorithm *poincare* takes a list of Poincaré sections and resolves them in order by alternating continuous reachability *reach-cont* and resolution of intersections with Poincaré sections *intersect-flow*, followed by summarization of intersections *reduce*.

Definition 7.21.

$$\begin{aligned}
&\mathit{poincare}\ []\ X_0 := \textbf{\textit{return}}\ X_0\\
&\mathit{poincare}\ [\Sigma_1, \ldots, \Sigma_n]\ X_0 :=\\
&\quad \textbf{\textit{do}}\ \{\\
&\qquad X_1 \leftarrow \mathit{reach\text{-}cont}\ (\bigcup_{i \leq n} \Sigma_i)\ X_0\\
&\qquad X_2 \leftarrow \mathit{intersect\text{-}flow}\ \Sigma_1\ X_1\\
&\qquad X_3 \leftarrow \mathit{reduce}\ X_2\\
&\qquad \mathit{poincare}\ [\Sigma_2, \ldots, \Sigma_n]\ X_3\\
&\quad \}
\end{aligned}$$

We take the liberty to write *reach-cont* $(\bigcup_{i\leq n}\Sigma_i)\ X_0$, which means that reachability analysis stops upon reaching any of the Σ_i. This yields $P_{\Sigma_1}(X_0) \subseteq X_3$ and X_3 is reachable without touching $\bigcup_{2\leq i\leq n}\Sigma_i$. This is important to be able to compose $P_{\Sigma_1}(X_0)$ with the Poincaré map P_{Σ_n} of the recursive call. The overall specification is that *poincare* encloses the Poincaré map to the last section in the argument.

Theorem 7.22 (Correctness of Enclosure with intermediate Poincaré Maps)**.**

$$\mathit{poincare}\,[\Sigma_1,\ldots,\Sigma_n]\ X_0 \leq \mathit{spec}\ (\lambda R.\ P_{\Sigma_n}(X_0) \subseteq R)$$

7.4 Derivatives

For Tucker's proof, it is necessary to compute not only the Poincaré map, but also its derivative. In general, the derivative of the flow or Poincaré map provides quantitative information on how the dynamics depend on initial conditions.

The derivative of the flow can be encoded as a higher dimensional ODE according to the variational equation (theorem 3.33). For an ODE with right hand side $f : \mathbb{R}^n \to \mathbb{R}^n$, we construct a new ODE of type $\mathbb{R}^n \times \mathbb{R}^{n\times n}$ with right hand side $\lambda(x, W).\ (f\ x, \mathsf{D}f|_{\mathit{at}\ x} \cdot W)$. Here the first component contains the solution, and the second component its matrix of partial derivatives.

With a modified predicate $\curvearrowright'$ for reachability with derivative,

$$\begin{aligned} X \curvearrowright'_C Y := \forall (x,d) \in X.\ \exists t \geq 0.\ (\phi(x_0,t), \mathsf{D}\phi_t|_{\mathit{at}\ x_0} \cdot_{\mathit{bl}} d) \in Y \wedge \\ (\forall 0 \leq s \leq t.\ (\phi(x_0,s), \mathsf{D}\phi_s|_{\mathit{at}\ x_0} \cdot_{\mathit{bl}} d) \in C_X) \end{aligned}$$

we can show that *reach-cont'*, i.e., the instantiation of *reach-cont* for the modified ODE satisfies the specification

Theorem 7.23 (Continuous Reachability with Derivatives)**.**

$$\mathit{reach\text{-}cont'}\ \mathit{sctns}\ X'_0 \leq (\lambda(C', Y').\ X'_0 \curvearrowright'_{C'} Y')$$

The Poincaré map, however requires extra care, because we cannot simply intersect the derivative of the flow with the Poincaré section: the derivative of the Poincaré map is given according to the expression in theorem 3.38. For a hyperplane $H = \{x \mid \langle x, n\rangle = c\}$, the derivative is given as follows (for $x \in \{x \mid \langle x, n\rangle = c\}$):

$$\mathsf{D}P|_x \cdot_{\mathit{bl}} h = \mathsf{D}\phi_{\tau(x)}|_x \cdot_{\mathit{bl}} h - \frac{\mathsf{D}s|_{P(x)} \cdot_{\mathit{bl}} (\mathsf{D}\phi_{\tau(x)}|_x \cdot_{\mathit{bl}} h)}{\mathsf{D}s|_{P(x)} \cdot_{\mathit{bl}} (f(P(x)))} f(P(x)) \tag{7.1}$$

We can evaluate this expression using affine arithmetic. But we need to be able to enclose all quantities that occur on the right hand side, in particular $P(x) = \phi(x, \tau(x))$ and $\mathsf{D}\phi_{\tau(x)}|_x \cdot d$. But we can enclose those: assume a step in computing an intersection, i.e., $X \curvearrowright_C Y$. Let us assume for simplicity that $(X \cup Y) \cap H = \emptyset$ and X and Y are on opposite sides of the hyperplane. Then the intersection of the flowpipe C with the section H encloses the Poincaré map: $P(X) = \{\phi(x, \tau(x)) \mid x \in X\} \subseteq C \cap H$. For an extended flow $X' \curvearrowright_{C'} Y'$, this means $\{(\phi(x, \tau(x)), \mathsf{D}\phi_{\tau(x)}|_x \cdot_{\mathit{bl}} d) \mid (x, d) \in X'\} \subseteq C' \cap H \times \mathbb{R}^{n*n}$. Therefore both $P(x) = \phi(x, \tau(x))$

and $\mathsf{D}\phi_{\tau(x)}|_{\mathit{at}\ x} \cdot d$ are enclosed by the result of the intersection $C' \cap H \times \mathbb{R}^{n\times n}$ for which we can use the regular intersection algorithm from chapter 5.

This results in a modified algorithm *poincare′*, which is defined like *poincare*, but uses *reach-cont′* instead of *reach-cont* and combines the computation of *intersect-flow* with the formula in equation 7.1.

The resulting correctness theorem reads as follows. *poincare′* computes an enclosure for the Poincaré map of X_0 and how the derivative $\mathsf{D}P$ propagates a matrix (or bounded linear function) DX_0.

Theorem 7.24 (Poincaré map with Derivative)**.**

$$\mathit{poincare}'\ [\Sigma_1, \ldots, \Sigma_n]\ (X_0, DX_0) \leq \mathit{spec}\ (\lambda(R, DR).\ P_{\Sigma_n}(X_0) \subseteq R \wedge \mathsf{D}P|_{X_0} \circ_{bl} DX_0 \subseteq DR)$$

7.5 Tactics

We provide automated tactics with which one can prove bounds on ODEs, Poincaré maps, and their derivatives by reducing them to a computation with *one-step*, *poincare*, or *poincare′*. We illustrate the approach for *one-step*, the other methods are treated similarly.

We assume a goal statement in a fixed form, namely

$$t \in [t^-; t^+] \to x \in [x^-; x^+] \to \phi_f(x, t) \in [l; u]$$

ϕ_f stands for the flow of an ODE with right-hand side f. This goal statement is then *reified* in an automated procedure, i.e., translated to a form where every function and bound in the statement is expressed as the interpretation of an explicit *aexp* expression. That is, we are left with a goal of the following form.

$$t \in [[\![t_e^-]\!]_{[]}; [\![t_e^+]\!]_{[]}] \to x \in [[\![x_e^-]\!]_{[]}; [\![x_e^+]\!]_{[]}] \to \phi_{[\![f_e]\!]}(x, t) \in [[\![l_e]\!]_{[]}; [\![u_e]\!]_{[]}]$$

Intervals with bounds given by *aexp* expressions can be approximated with standard interval arithmetic. These intervals can then be represented as encl_r using *encl-of-ivl*. We therefore get T implemented as encl_r such that $[[\![t_e^-]\!]_{[]}; [\![t_e^+]\!]_{[]}] \subseteq T$, similar for an X with $[[\![x_e^-]\!]_{[]}; [\![x_e^+]\!]_{[]}] \subseteq X$. We can also come up with an *inner* approximation for the resulting bound, namely $R \subseteq [[\![l_e]\!]_{[]}; [\![u_e]\!]_{[]}]$.

The above statement can be proved with the correctness theorem 7.12 of *one-step*, when computing a result S and checking that $S \subseteq R$.

$$\mathit{one\text{-}step}\ X\ T = \mathit{return}\ S \wedge S \subseteq R$$

one-step X T is then evaluated with Isabelle's `eval` mechanism.

Overview of Tactics. Overall, we provide tactics for goals of the following form:

- The solution of an initial value problem with interval uncertainty.

$$t \in [t^-; t^+] \to x \in [x^-; x^+] \to t \in \mathit{ex\text{-}ivl}(x) \wedge \phi_f(x, t) \in [l; u]$$

- Bound on the solution of an initial value problem together with its variational equation.

$$t \in [t^-; t^+] \to x \in [x^-; x^+] \to d \in [d^-; d^+] \to \\ t \in \textit{ex-ivl}(x) \wedge \phi_f(x,t) \in [l;u] \wedge \mathrm{D}\phi|_x \cdot_{bl} d \in [dl; du]$$

- Bounds on definite integrals by encoding them in the obvious way as an ODE and solving the ODE with *one-step*.

$$\int_a^b f(x)dx \in [l;u]$$

- Bounds on the Poincaré map.

$$\forall x \in [x^-; x^+].\ \textit{returns-to}\ \Sigma\ x \wedge P_\Sigma(x) \subseteq [pl; pu]$$

- Bounds on the derivatiave of the Poincaré map.

$$\forall x \in [x^-; x^+].\ \forall d \in [d^-; d^+].\ \textit{returns-to}\ \Sigma\ x \wedge P_\Sigma(x) \subseteq [pl; pu] \wedge \mathrm{D}P|_x \cdot_{bl} d \in [dl; du]$$

7.6 Plotting

It is instructive to visualize the computations of the rigorous ODE solver. We therefore instrument the ODE solver such that it outputs enclosures while running. For an n-dimensional ODE, the user can provide a function to project n-dimensional zonotopes into the plane. For the resulting two-dimensional zonotope, we use the algorithm *hull-of-zonotope* from section 5.2.3 to compute a list of line segments making up the boundary of the two-dimensional projection. This list of line segments is output in a format that can be read by `gnuplot`. One line in the output format is a list of four numbers in scientific notation, which represent the endpoints of a line segment of the boundary of the zonotope. A file containing such output can then be plotted using gnuplot with the following command:

```
plot "filename" using 1:2:3:4 with vectors nohead
```

7.7 Experimental Evaluation

This section contains a comparison with the performance and precision of other tools for rigorous ODE solving and/or reachability analysis on some selected problems. Please note that these experiments compare tools with very different underlying algorithms and data structures, each of which with their particular strengths and weaknesses. Moreover, performance and precision often depends crucially on a proper choice of parameters, where I cannot claim to be able to identify the best ones. I did, however, try to reach some local optimum in the sense that I started with default parameters, modified trying to reach a given precision and then trying to improve performance while keeping the precision.

The experiments in this section were carried out in a virtual machine operated by a 64-bit Ubuntu 16.04 Linux on a laptop computer with a 2,6 GHz Intel® Core™ i7 CPU and 16 GB RAM. All computations were single-threaded and run with an increased clock frequency of about 3.4 GHz. Isabelle is run with the 64-bit version of PolyML.

7.7.1 Tools

The tools that I used as part of this experimental evaluation are denoted as follows. I will also list the parameters (with default values) for which I tried to find suitable values for good results in this experimental evaluation.

GRK. The Guaranteed Runge-Kutta methods by Bouissou *et al.* [24], which inspired the usage of Runge-Kutta methods and affine arithmetic. It is implemented in OCaml. I will only cite the results reported in their paper.

Flow*. A tool for reachability analysis of continuous and hybrid systems with nonlinear dynamics [30]. Available online[2] in the current version 2.1.2. It is implemented in C++ and is based on Taylor models. Relevant parameters are Taylor model order and an error estimation for the remainder terms.

Flow*-P. This denotes **Flow*** with manually inserted intermediate Poincaré maps, (or pseudo-invariants, as Bak [10] calls them).

CORA. Reachability analysis of continuous and hybrid systems [3, 4], in version CORA 2016, which is available online[3]. It uses zonotopes and linearizes nonlinear dynamics and is implemented in Matlab. Relevant parameters are the (fixed) step size and zonotope order.

CORA-P. This denotes **CORA** with manually inserted intermediate Poincaré maps.

CAPD. CAPD is a library for nonrigorous and validated numerics for dynamical systems[4]. I use version 5.0.6. CAPD is based on high-order Taylor expansions and implemented in C++.

VNODE. This denotes VNODE-LP by Nedialkov [106]. VNODE-LP is implemented using C++ and literate programming, its correctness can therefore be reviewed by a human expert. It is based on high-order Taylor series expansions or Hermite-Obreschkoff. I use version 0.3 which is available online [5].

Coq. This denotes Mahboubi *et al.*'s [94] work on formally verified approximations of definite integrals in the interactive theorem prover Coq [13]. I will only cite the results reported in their paper.

Isabelle. My verified algorithms for zonotope enclosures with Runge-Kutta steps (from section 7.2.5). Relevant parameters are floating point precision, error tolerance for adaptive step size control, and maximum zonotope order.

[2] `http://flowstar.org`
[3] `http://www.i6.in.tum.de/Main/SoftwareCORA`
[4] `http://capd.ii.uj.edu.pl/`
[5] `http://www.cas.mcmaster.ca/~nedialk/vnodelp/`

Isabelle-P. My verified algorithms for reachability analysis with intermediate Poincaré maps (from section 7.3), which are manually chosen a-priori. Other parameters are as for **Isabelle**.

7.7.2 Oil Reservoir

The oil reservoir problem is the running example of Bouissou *et al.*'s paper [24]:

$$\begin{aligned}\dot{y} &= z \\ \dot{z} &= z^2 - \frac{3}{10^{-3} + y^2}\end{aligned}$$

The goal is to solve the following ODE with initial condition $y(0) = 10$ and $z(0) = 0$ on the time interval $[0; 50]$. Figure 7.7 depicts enclosures as computed by the verified algorithm **Isabelle**.

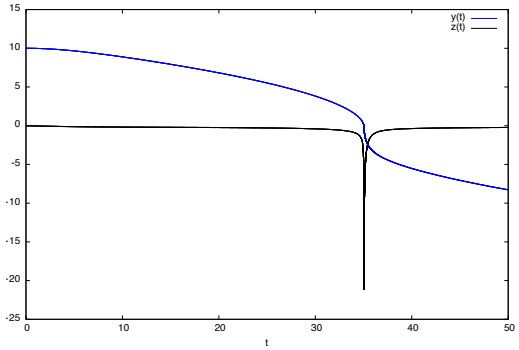


Figure 7.7: Enclosures for the temporal evolution of the oil reservoir problem.

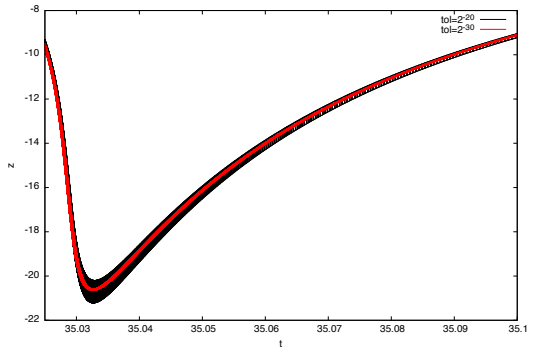


Figure 7.8: Enclosures around $t = 35$ of the oil reservoir problem.

The ODE is stiff around time $t = 35$, i.e., small step sizes are required to maintain precise enclosures. Figure 7.8 zooms in on that detail of the evolution.

Results. Table 7.1 summarizes the time required by the different tools to produce an enclosure with a prescribed width (the *final error*) at time $t = 50$. **CORA** is not listed, because it does not feature an adaptive step size control. When the fixed step size is too large, it fails to maintain precision around time $t = 35$. When the fixed step size is too small, **CORA** requires too many steps (and therefore too much time) to integrate over the whole interval $[0; 50]$. **CAPD** and **VNODE** clearly outperform the other tools, producing much tighter enclosures in fractions of the time. The performance of **Flow*** is comparable to that of **GRK**. On these examples, **Isabelle** is slower than those two by a factor between 7 and 50.

Settings. The error tolerances for the adaptive step-size control are set to $1 \cdot 10^{-6}$ resp. $1 \cdot 10^{-9}$ for **Isabelle**, **Flow***, and **GRK** for final enclosures of width $5 \cdot 10^{-2}$ resp. $5 \cdot 10^{-3}$. I used **VNODE** with the default parameters and **CAPD** with Taylor series of order 20 and error tolerance of $1 \cdot 10^{-10}$.

tool	time [s]	width of final enclosure
CAPD	< 0.01	$5 \cdot 10^{-13}$
VNODE	< 0.01	$5 \cdot 10^{-13}$
Isabelle-P	54.2	$5 \cdot 10^{-2}$
	417	$5 \cdot 10^{-3}$
GRK	4	$5 \cdot 10^{-2}$
	12	$5 \cdot 10^{-3}$
Flow*	7.3	$5 \cdot 10^{-2}$
	8.4	$5 \cdot 10^{-3}$

Table 7.1: Computation time and size of final enclosure (error) for the oil reservoir problem. The entries for GRK are cited from Bouissou *et al.*'s paper [24]

7.7.3 Laub-Loomis

The Laub-Loomis model [88] is a seven-dimensional ODE for studying enzymatic activities, it was a benchmark problem of the ARCH friendly software competition [31]:

$$\begin{cases} \dot{x}_1 &= 1.4x_3 - 0.9x_1 \\ \dot{x}_2 &= 2.5x_5 - 1.5x_2 \\ \dot{x}_3 &= 0.6x_7 - 0.8x_2x_3 \\ \dot{x}_4 &= 2 - 1.3x_3x_4 \\ \dot{x}_5 &= 0.7x_1 - x_4x_5 \\ \dot{x}_6 &= 0.3x_1 - 3.1x_6 \\ \dot{x}_7 &= 1.8x_6 - 1.5x_2x_7 \end{cases}$$

Analysis is to be carried out for time in $[0;20]$. With this experiment, we study the performance on a higher-dimensional, nonlinear ODE. Moreover we investigate how the different tools deal with uncertainties in the initial condition. We therefore study boxes of width $w = 0$ $w = 0.02$ and $w = 0.2$ centered around $x_1(0) = 1.2$, $x_2(0) = 1.05$, $x_3(0) = 1.5$, $x_4(0) = 2.4$, $x_5(0) = 1$, $x_6(0) = 0.1$, $x_7(0) = 0.45$.

Enclosures for $w = 0.01$ as produced by **Isabelle** are plotted in figure 7.9

Results. Table 7.2 summarizes the computation time required depending on the initial uncertainty w. With larger initial uncertainty, many tools fail to produce sensible enclosures for the whole interval $[0;20]$. After some time (roughly about the indicated t_{max}), the tools produce ever-growing enclosures, a typical result of the wrapping effect getting out of control.

As in the previous example (the oil reservoir problem), **CAPD** and **VNODE** are superior in both precision (not reported here) and computation time. They suffer, however, from medium-sized ($w = 0.01$) uncertainty in the initial data. **CAPD** manages to produce sensible enclosures for slightly longer time spans.

Isabelle is more robust in handling larger initial sets, and the performance impact (from $w = 0$ to $w = 0.01$) is negligible. **Isabelle** is slower than **CORA** and **Flow*** by a factor between

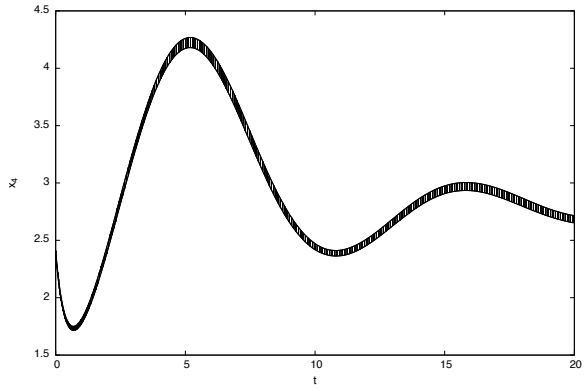


Figure 7.9: Enclosures of the temporal evolution of x_4 of the Laub-Loomis model.

5 and 20. **Isabelle** handles the largest initial set only until $t = 9$, whereas **Flow*** and **CORA** can enclose the solution over the whole time interval.

Settings. For **Isabelle**, the maximum zonotope order is set to 60, error tolerance is set to 2^{-12}. **Flow*** computes with a fixed step size of 0.05, fixed Taylor model order 4, remainder estimation of 0.1, and symbolic remainders for 50 steps. **CORA** uses a fixed step size of 0.05, zonotope order of 50 and for $w = 1$ polynomial zonotopes of order 4.

	time [s]		
tool	$(w = 0)$	$(w = 0.02)$	$(w = 0.2)$
CAPD	< 0.01	$(t_{max} = 11)$	$(t_{max} = 2.5)$
VNODE	< 0.01	$(t_{max} = 8)$	$(t_{max} = 1.7)$
Isabelle	48.2	51.3	$(t_{max} = 9)$
Flow*	6.6	10.0	25.7
CORA	2.5	2.9	23.0

Table 7.2: Computation time for enclosing solutions of the Laub-Loomis model on the interval $t \in [0; 20]$. Entries $(t_{max} = \ldots)$ indicate that enclosures explode around $t = t_{max}$.

7.7.4 Van der Pol.

The Van-der-Pol oscillator (Figure 7.10, plotted from the output of **Isabelle**) is given by the following ODE:

$$\begin{aligned}\dot{x} =& y\\ \dot{y} =& (1 - x^2)y - x\end{aligned}$$

We consider initial value problems $x_0 \in 1.4 + 3w \cdot [-1; 1], y_0 = 2.25$ and vary the size of the initial set with the parameter w. Depending on the capabilities of the tool, we either compute the Poincaré map for returning to $y = 2.25$ from above or until time $t = 7$, which is about the return time. We will see that in this system, intermediate Poincaré sections are very effective. We include one at $x = -1$ for **Flow*-P**, **CORA-P**, and **Isabelle-P**.

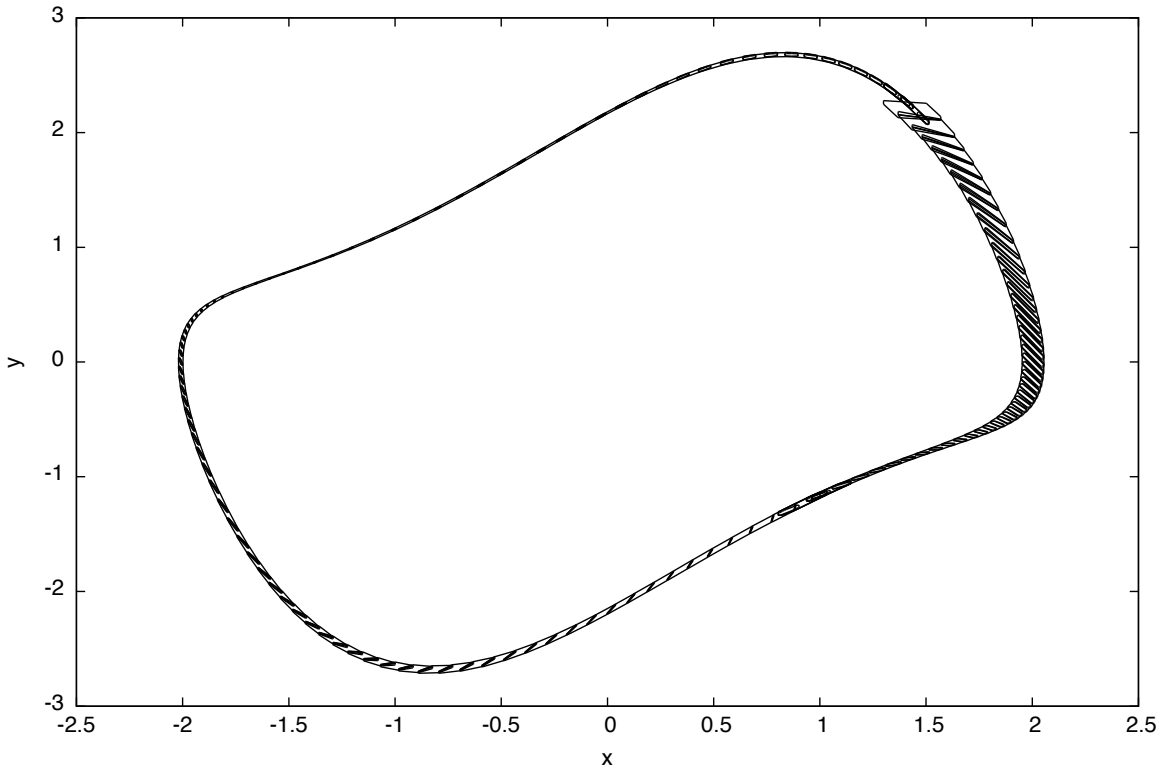


Figure 7.10: Plot of the van der Pol system ($w = 0.2$)

Results. Table 7.3 displays computation time for different sizes of the initial set, with ($t_{max} = \ldots$) indicating failures to produce meaningful enclosures. Intermediate Poincaré maps speed up computations for **Isabelle** and even allow **Isabelle-P** to compute enclosures for the largest initial set. **Flow*** does not profit from the intermediate Poincaré maps in **Flow*-P**. On those examples, **Isabelle-P** can be faster than **CORA**, and up to a factor 5 slower than **Flow***. Here again, **CAPD** and **VNODE** can not handle larger initial sets.

Settings. For **Isabelle**, the maximum zonotope order is set to 20, error tolerance is set to 2^{-12}. **Flow*** computes with fixed Taylor model order 4, remainder estimation of $1 \cdot 10^{-5}$, and adaptive step size control. **CORA** uses a fixed step size of 0.01, zonotope order of 50 and splitting for width ≥ 0.5. **CAPD** and **VNODE** use standard settings.

	time[s]			
tool	$(w = 0.01)$	$(w = 0.1)$	$(w = 0.2)$	$(w = 0.5)$
Isabelle	2.9	3.1	5.0	$(t_{max} = 3)$
Isabelle-P	3.1	2.2	2.2	8.8
Flow*	0.5	0.6	0.8	1.5
Flow*-P	1.6	1.0	1.4	2.5
CORA	4.0	5.8	6.0	$(t_{max} = 3)$
CORA-P	7.8	8.0	7.5	8.2
CAPD	< 0.01	$(t_{max} = 3)$	$(t_{max} = 1.2)$	$(t_{max} = 0.6)$
VNODE	< 0.01	$(t_{max} = 6)$	$(t_{max} = 3)$	$(t_{max} = 0.8)$

Table 7.3: Computation time for the van der Pol system. Entries $(t_{max} = \ldots)$ indicate that enclosures explode around $t = t_{max}$.

7.7.5 Lorenz.

For an experiment with a chaotic system, we consider the classical Lorenz system in Jordan normal form, which is given with approximate numerical constants as follows:

$$\begin{aligned}\dot{x} &= 11.8x - 0.29(x+y)z\\ \dot{y} &= -22.8y + 0.29(x+y)z\\ \dot{z} &= -2.67z + (x+y)(2.2x - 1.3y)\end{aligned}$$

We experiment with initial sets of width 0.01 (in x and y) centered around $(x_0, 2.21, 27)$, where we vary x_0 between 0.84 and 0.94. The dynamics exhibits more chaotic behavior for smaller values of x_0. Figure 7.11 displays enclosures for the evolution from $x_0 = 0.92$ (the innermost), $x_0 = 0.88$, and 0.84 (the outermost). As can be seen in the close-up of figure 7.12, we include an intermediate Poincaré section at $x = 2$. This Poincaré section is also used for **Flow*-P** and **CORA-P**.

Results. Without intermediate Poincaré maps, **Isabelle** and **CORA** fail to produce meaningful enclosures, they are therefore omitted in this analysis. It can be seen in table 7.4 that all tools require more time for more chaotic initial values. **Isabelle-P** produces the tightest enclosures and is slower by a factor between 1.2 (vs. **Flow***) and 5 (vs. **CORA-P**). **CAPD** and **VNODE** fail for the initial sets of width 0.01, we therefore include results for easier initial sets of width 0.01, where **VNODE** produces less accurate enclosures for all initial sets, and **CAPD** is less accurate than **Isabelle-P** for the more chaotic half of initial values (despite the smaller initial set).

Settings. **VNODE** and **CAPD** with standard settings, **Flow*** with remainder estimation 10^{-4}, fixed order 6 and adaptive step size control. **Isabelle-P** with zonotope order 30 and error tolerance 2^{-14}. **CORA-P** with zonotope order 20 and fixed step size 0.005.

		x_0					
tool		0.94	0.92	0.90	0.88	0.86	0.84
Isabelle-P	time[s]	11.5	11.5	11.6	11.6	12.1	13.1
	final width	0.1	0.1	0.1	0.1	0.2	0.3
Flow*	time[s]	6.1	6.3	6.7	7.5	8.3	10.9
	final width	0.15	0.2	0.2	0.4	0.4	0.6
Flow*-P	time[s]	4.7	4.8	4.8	4.9	5.1	5.5
	final width	0.2	0.2	0.3	0.3	0.4	0.5
CORA-P	time[s]	2.6	2.6	2.7	2.7	2.7	4
	final width	0.2	0.25	0.3	0.35	0.4	0.6
CAPD (0.001)	time[s]	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
	final width	0.1	0.1	0.1	0.2	0.4	8
VNODE (0.001)	time[s]	< 0.01	< 0.01	< 0.01			
	final width	1.3	2.4	37	∞	∞	∞

Table 7.4: Computation time (time[s]) and width of final enclosures (final width) for different initial values (x_0) of the Lorenz system.

7.7.6 Integrals

We finally compare our results with the only other formally verified tool, namely the Coq developments by Mahboubi *et al.* [94]. They only work with integrals. For this comparison, we use the setup from section 7.5 to encode the integrals as ODE. We compare **Isabelle** with all the results they report in their paper. They report on six definite integrals, which they approximate to various given values e for the approximation error.

Derivative of arctan. The first example is the integral of the derivative of the arctan:

$$\left| \int_0^1 \frac{1}{1+x^2} \, \mathrm{d}x - \frac{\pi}{4} \right| \leq e$$

For low precision, **Isabelle** takes a similar amount of time, medium precision (10^{-9}) is still possible, but higher precision seems unfeasible with the current approach in **Isabelle**:

e	time (**Coq**) [s]	time (**Isabelle**) [s]
10^{-3}	0.3	0.4
10^{-6}	0.3	1.4
10^{-9}	0.6	52
10^{-12}	1.0	> 60
10^{-15}	1.7	
10^{-18}	2.9	

Ahmed's integral. The second example, Ahmed's integral [2] involves more operators:

$$\left| \int_{-1}^1 \frac{\arctan(\sqrt{x^2+2})}{\sqrt{x^2+2}(x^2+1)} \, \mathrm{d}x - \frac{5\pi^2}{96} \right| \leq e$$

Here, low precision is still feasible, but **Isabelle** cannot compete with medium and high precision results of **Coq**.

e	time (**Coq**) [s]	time (**Isabelle**) [s]
10^{-3}	0.5	1.3
10^{-6}	1.2	15
10^{-9}	2.8	> 60
10^{-12}	5.5	
10^{-15}	11.2	

Trigonometric. The third example involves trigonometric functions, which are harder to approximate with Taylor models (and also zonotopes):

$$\left| \int_0^\pi \frac{x \sin x}{1+(\cos x)^2} \, \mathrm{d}x - \frac{\pi^2}{4} \right| \leq e$$

Here, the Taylor models used in **Coq** are clearly superior: **Isabelle** achieves an accuracy of 10^{-3} with reasonable effort, but higher accuracies become overly expensive.

e	time (**Coq**) [s]	time (**Isabelle**) [s]
10^{-3}	1.1	8.2
10^{-6}	2.3	240
10^{-9}	5.0	>240
10^{-12}	11.5	
10^{-15}	27.2	

Helfgott. The fourth example is from Helfgott, according to Mahboubi *et al.*in the spirit of [62].

$$\left| \int_0^1 \left| (x^4 + 10x^3 + 19x^2 - 6x - 6) \exp x \right| \, \mathrm{d}x - 11.14731055005714 \right| \leq e$$

Unfortunately, it can not be solved with **Isabelle**, because the ODE solver requires a (at least twice) differentiable function, which is not the case here because of the absolute value in the integrand.

Chebyshev. The fifth example is the 12-th coefficient of a Chebychev expansion:

$$\left| \int_{-1}^1 \mathrm{cheb}(x) \, \mathrm{d}x + 3.2555895745 \cdot 10^{-6} \right| \leq e$$

Where

$$\mathrm{cheb}(x) := \Bigg(\left(2048x^{12} - 6144x^{10} + 6912x^8 3584x^6 + 840x^4 - 72x^2 + 1\right) \exp\left(-(x - \frac{3}{4})^2\right) \sqrt{1 - x^2} \Bigg)$$

This integral can not be solved with **Isabelle**: the ODE solver requires the ODE to be defined and differentiable on an open set. But because of the square root, cheb is only defined on the closed interval $[-1; 1]$ and needs to be considered at the boundary values -1 and 1.

Rump. The sixth and last example (suggested by Rump [124]) is a challenge because of its large number of oscillations:

$$\left| \int_0^8 \sin(x + \exp x) \, \mathrm{d}x - 0.3474 \right| \leq e$$

Here, also **Coq** has difficulties and requires much more time than on the previous examples, but still performs better than **Isabelle**, which can only produce 1 correct digit in about 10 minutes.

e	time (**Coq**) [s]	time (**Isabelle**) [s]
10^{-1}	81.0	700
10^{-2}	123.6	> 700
10^{-3}	183.4	
10^{-4}	277.6	

7.7.7 Interpretation of Results

VNODE and CAPD are highly optimized and the methods of choice for small initial sets or point initial conditions. High order Taylor series expansions allow much more precision than the fixed third order scheme implemented in Isabelle. For medium accuracy, Isabelle can also produce meaningful results in reasonable time (compared to the other tools for reachability analysis). Adaptive step size control can be an essential feature, therefore Isabelle can outperform CORA on the oil reservoir problem.

Taylor models as implemented in Flow* or polynomial zonotopes as implemented in CORA are the superior data structure to represent large non-convex sets of higher dimension (as they appear in the reachability analysis of the Laub-Loomis model). VNODE and CAPD are superior for point initial conditions but cannot handle larger uncertainties. For a medium sized initial uncertainty, Isabelle can still compete, but non-convex set representations are essential for high-dimensional non-linear dynamics (splitting the reachable sets is not an option because of the high dimension).

Intermediate Poincaré maps are a useful means to reduce the wrapping of non-convex sets, as could be seen in the van der Pol and Lorenz systems. On these examples, Isabelle-P compares very well with the other (non-verified) tools for reachability analysis.

7.8 Related Work

In this section, I discuss related work on numerics of differential equations in interactive theorem provers and discuss relations to non-verified work, in particular work on reachability analysis of continuous and hybrid systems via flow-pipe constructions (mostly targeted towards engineering applications) and rigorous numerical tools targeted at dynamical systems.

7.8.1 Numerics of Differential Equations in ITPs

Spitters and Makarov [95] implement Picard iteration to calculate solutions of ODEs in the interactive theorem prover Coq. The numerical bounds that they obtain are restricted to relatively short existence intervals. Boldo *et al.* [19] approximate the solution of one particular partial differential equation with a C-program and verify its correctness in Coq.

7.8.2 Other Tools

Apart from the tools presented in section 7.7.1, I would like to mention Bak's [10] approach of automatically deducing intermediate Poincaré maps (he calls them "pseudo-invariants"), implemented e.g. in the tool HYST [11], which can be used as a front-end to other tools. The state-of-the-art tool for analysis of hybrid systems with linear dynamics SpaceEx [39] can handle systems with hundreds of variables, but cannot be used for non-linear dynamics as in the experiments. COSY [14] is based on Taylor models and reportedly precise in the analysis of non-linear dynamics, but unfortunately not freely available.

Rigorous computation of flow-pipes is also part of tools for analyzing non-linear hybrid systems, e.g., HySAT/iSAT [38], Ariadne [12], or Acumen [132].

7.9 Discussion.

We presented a formally verified analyzer for continuous systems given by ODEs. Its performance can be in the range of other, non-verified tools. With the introduction of intermediate Poincaré maps, we can handle larger initial sets or chaotic dynamics better that some of the non-verified tools. Our approach does, however not scale to high precision computations. But it tolerates larger uncertainties than e.g., CAPD, or VNODE.

Comparable tools are a about one or two orders of magnitude faster. But this is still reasonable as e.g., our method does not use native floating point numbers, where we lose a large factor. Setting this in context with other verification work: a factor of 7 to 26 has been reported [36] for a verified implementation of an LTL model checker, where the verified implementation was also tuned for efficiency.

In my earlier paper presented at TACAS [71], the overall reachability analysis algorithm was based on more heuristics, which turned out to be unnecessarily complicated. The manual declaration of intermediate Poincaré maps is perfectly feasible for the presented examples. Moreover, devising intermediate Poincaré maps can be delegated to a tool like HYST [10, 11].

7.9.1 Design Choices

There is no single best approach to reachability analysis of ODEs, this can be seen at the variety of tools and algorithms for rigorous ODE solving. Many of my design decisions were guided by the principle "reasonably efficient/precise" while simple enough for a formalization with the existing libraries.

Affine Arithmetic seemed a straightforward generalization of interval arithmetic, but less intricate to formalize than e.g., Taylor models.

Runge-Kutta methods have already been used with affine arithmetic by Bouissou *et al.* [24]. As a fixed scheme, the verification and implementation is straightforward. Without (at the time) formal results about multivariate Taylor series expansions, the more flexible alternative of Taylor series methods seemed further out of reach.

Splitting as a means to counteract the wrapping effect of non-convex sets is only effective for low-dimensional systems. An alternative could have been the use of more complex data structure like Taylor models that directly represent non-convex sets. It seems, however, that splitting is also necessary for Taylor model based analysis tools, as could be seen in the example on the Lorenz equations in section 7.7.5. Neumaier [107] discusses advantages and limitations of Taylor models.

Intermediate Poincaré maps were inspired by Tucker's algorithm and provided a means to reduce the size of currently reachable sets, thereby avoiding non-convexity or wrapping. An alternative to the geometric algorithm for resolving intersections was proposed by Althoff and Krogh [7].

7.9.2 Summary

In summary, the algorithms this chapter are the result of a pragmatic attempt to hit a sweet spot between ease of implementation and verification on the one hand and performance and precision on the other hand. Many decisions were taken with an analysis of the Lorenz equation as motivation but are general enough to work for other (low dimensional) ODEs as well. The overall design is modular such that it is now possible to include other set representations like e.g., Taylor models (which are now available in Isabelle/HOL [133]) or different approximation schemes like Taylor series methods into the overall framework.

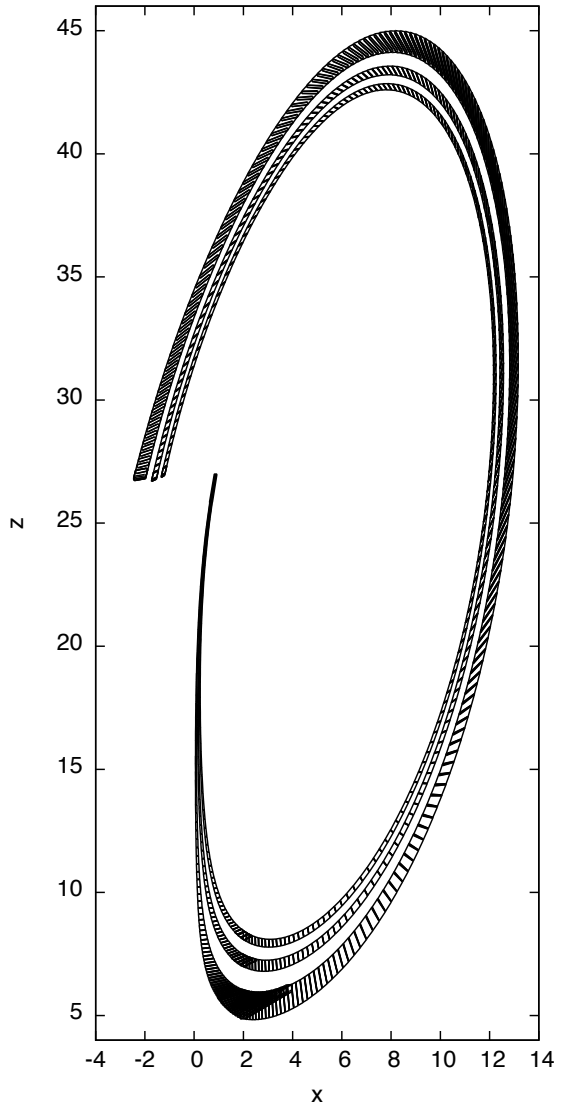


Figure 7.11: Lorenz system, enclosures for initial sets around $x_0 = 0.92$ (innermost), $x_0 = 0.88$, and 0.84 (outermost).

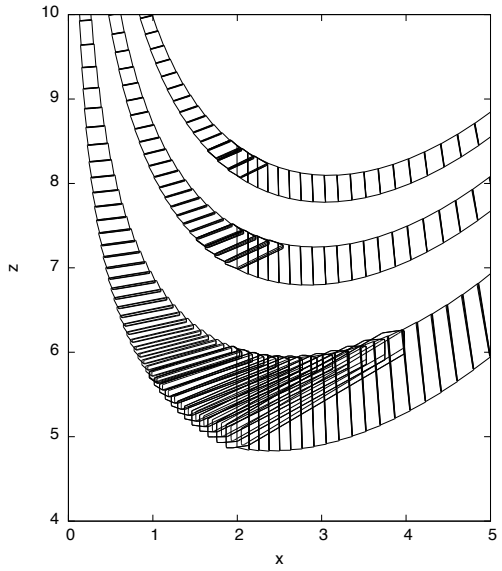


Figure 7.12: Lorenz system, close-up on intermediate Poincaré section at $x = 2$.

8
Smale's 14th Problem

In 1963, meteorologist Edward Lorenz [92] introduced the following system of ODEs as a simplified model for atmospheric dynamics:

$$\begin{aligned}\dot{x} &= -\sigma x + \sigma y \\ \dot{y} &= -xz + \rho x - y \\ \dot{z} &= xy - \beta z\end{aligned}$$

Lorenz observed that even the smallest perturbation in initial values would lead to completely different long-term behavior of the system. Referring to the original motivation, he asked: "Does the Flap of a Butterfly's Wings in Brazil Set Off a Tornado in Texas?" and the term *butterfly effect* entered popular culture. The Lorenz system tends to evolve to a complicated structure, the so-called *Lorenz attractor* (figure 8.1), which became an iconic example of deterministic chaos: According to Sparrow [129] "the number of man, woman, and computer hours spent on [the Lorenz equations ...] must be truly immense".

Despite its popularity and the amount of effort put into its study, nobody managed to prove that the Lorenz attractor is chaotic in a rigorous mathematical sense.

The numerically observed dynamics inspired a geometric model, whose behavior could be described with explicit symbolic formulas and could therefore be analyzed rigorously [144, 50].

Understanding the dynamics of Lorenz' original ODE, however, remained out of reach. In 1998, Fields medalist Stephen Smale put the problem of understanding Lorenz' ODE on his list of eighteen unsolved mathematical problems for the 21st century [127]:

> **Problem 14: Lorenz Attractor**
> Is the dynamics of the ordinary differential equations of Lorenz that of the geometric Lorenz attractor of Williams, Guckenheimer, and Yorke?

8.1 Tucker's Proof

Tucker solved this problem with the help of a computer program. He proved that the Lorenz attractor is singular hyperbolic, it therefore exhibits the same dynamics as the geometric Lorenz attractor of Williams, Guckenheimer, and Yorke.

Tucker does so with the help of a Poincaré section. This is a distinguished set in the phase space, in this case a square on the plane $z = 27$, namely $\Sigma = [-6;6] \times [6;6] \times \{27\}$. On Σ, the Poincaré map P is defined: For a point $x_0 \in \Sigma$, the Poincaré map $P(x_0)$ is the point

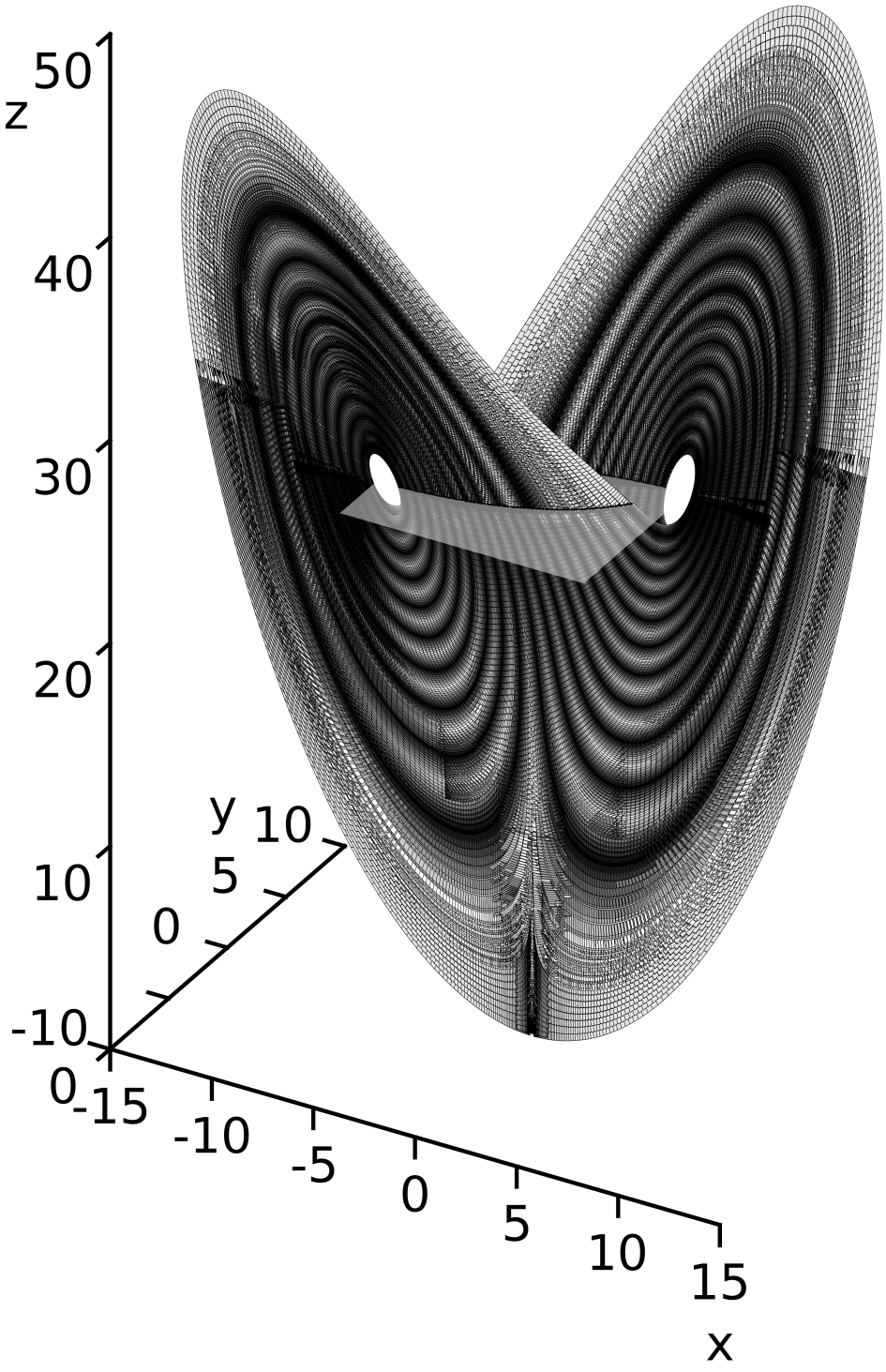


Figure 8.1: An enclosure of the Lorenz attractor. Shading of enclosures varies with initial condition. Plotted from the output of *check-result-c1* (*input-data*).

where the flow first returns to Σ. This reduces the three-dimensional, continuous dynamics ϕ to (discrete) iterations of the two-dimensional map P. Tucker then analyzes the dynamics of P with a rigorous ODE solver. In particular he identifies a trapping region for P on Σ and quantifies sensitive dependence on initial values.

8.1.1 Trapping Region.

Tucker proves that there is a (compact) trapping region $N \subseteq \Sigma$, such that solutions starting in N will remain in N. He does so by subdividing N into a large number of small rectangles. For every small rectangle, Tucker's program computes safe numeric bounds for all solutions evolving from the small rectangle. In a number of time-discretization steps, the evolution is followed until it eventually returns to Σ. Upon return, the program checks that the returned enclosure is contained in N. If this process succeeds for every small rectangle, one can conclude the following theorem.

Theorem 8.1 (Trapping Region)**.** $\forall x \in N - \Gamma.\ P(x) \in N$

Note that there exists a set Γ on which P is not defined: Γ is the set of points, from which solutions tend to the origin in infinite time. Γ is therefore explicitly excluded in the above theorem. We will detail on the role of Γ in section 8.1.3 on normal form theory.

8.1.2 Sensitive Dependence.

Sensitive dependence on initial conditions can be quantified with the help of the derivative: A deviation in the direction of a vector $v \in \mathbb{R}^2$ is propagated (in linear approximation) like the derivative at x, i.e., $P(x+v) \approx x + \mathsf{D}P|_x \cdot v$. Here $\mathsf{D}P|_x \cdot v$ is the matrix of partial derivatives of P (the Jacobian matrix) at the point x, multiplied with the vector v.

A mathematically precise notion of chaos is given by the class of singular hyperbolic systems [103]. A hyperbolic system contracts deviations in *stable* directions and expands deviations in *unstable* directions. Both are relevant for the dynamics of the attractor: Stable directions make solutions tend to the attractor, whereas unstable directions lead to sensitive dependence on initial conditions.

Tucker proves that the Lorenz attractor is (singular) hyperbolic by providing safe overapproximations for the unstable direction: Every $x \in N$ is equipped with a cone $\mathfrak{C}(x)$, which contains the unstable direction. This is also verified by Tucker's computer program: In addition to the Poincaré map, the program keeps bounds on its matrix of partial derivatives. The program tracks how initial deviations (inside the cone associated to an initial rectangle) are propagated by the derivative $\mathsf{D}P$. The cone field needs to be forward invariant (otherwise it would not contain the unstable direction) and the expansion needs to be large enough that the enclosed directions are actually expanding. An example of such a computation is shown in figure 8.2.

Tucker's program establishes factors $\mathcal{E}(x)$ and $\mathcal{E}^{-1}(x)$, which quantify the expansion properties of P:

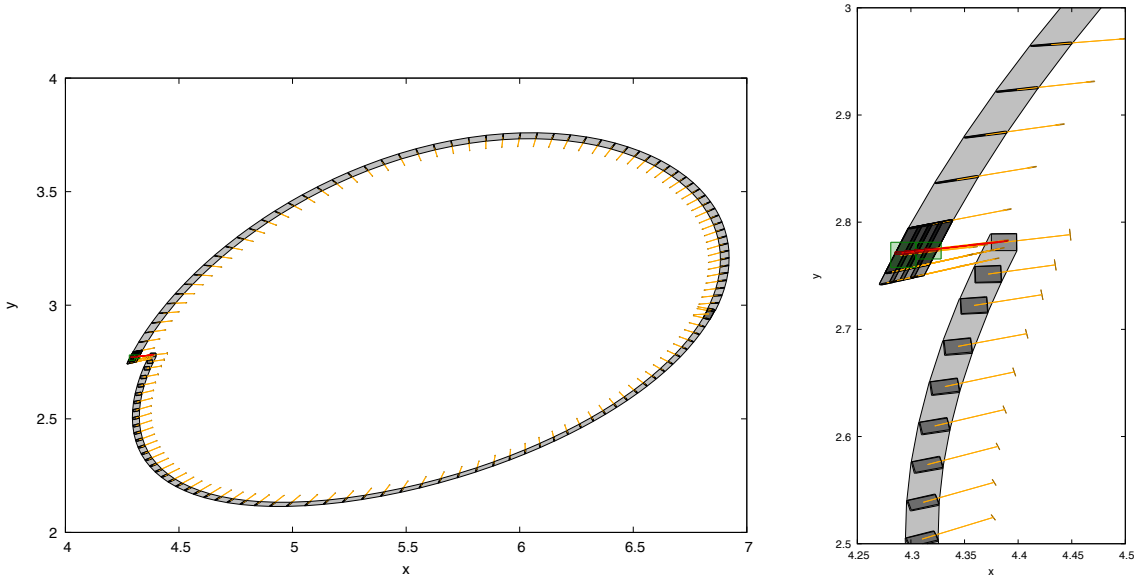


Figure 8.2: Enclosures for the flow and cones evolving from $X_0 = [4.375; 4.4] \times [2.77; 2.79] \times \{27\}$ with a representation of a cone between 1.5° and 11.5° (detail on the right).

Theorem 8.2 (Derivatives, Cones, and Expansion)**.**

1. $\forall x \in N - \Gamma.\ \forall v \in \mathfrak{C}(x).\ \mathrm{D}P|_x \cdot v \in \mathfrak{C}(P(x))$
2. $\forall x \in N - \Gamma.\ \forall v \in \mathfrak{C}(x).\ \|\mathrm{D}P|_x \cdot v\| \geq \mathcal{E}(x)\,\|v\|$
3. $\forall x \in N - \Gamma.\ \forall v \in \mathfrak{C}(x).\ \|\mathrm{D}P|_x \cdot v\| \geq \mathcal{E}^{-1}(P(x))\,\|v\|$

This theorem states that 1., the cone field $\mathfrak{C}$ is forward invariant under the action of the derivative of P: the image of every cone is slimmer than the cones onto which they are mapped. 2., the vectors v satisfy lower bounds on how much they are expanded: the length $\|\mathrm{D}P|_x \cdot v\|$ of the return of the deviation vector v is lower bounded by its length $\|v\|$ times an expansion factor $\mathcal{E}(x)$. They also satisfy a pre-expansion bound $\mathcal{E}^{-1}(x)$ (this does not denote $\frac{1}{\mathcal{E}}$) for the pre-image of x, which is required for technical reasons in Tucker's proof.

8.1.3 Normal Form Theory.

In the previous section on sensitive dependence, I mentioned singular hyperbolic systems. This term classifies hyperbolic sets that contain a hyperbolic fixed point. For the Lorenz equations, the origin $(0, 0, 0)$ is such a hyperbolic fixed point. The origin is a fixed point, because the ODE evaluates to 0. It is hyperbolic, because solutions tend to it in the two stable directions given by the y and z axis and expand in the unstable direction given by the x-axis.

This singular point poses problems for the aforementioned approach of using rigorous numerical methods: there are solutions that tend to the origin as time goes to infinity. In such a situation, a time-discretization algorithm is at a loss, because it would need

infinitely many steps. To remedy this problem, Tucker's program interrupts computations in a small cube $L = [-0.1; 0.1] \times [-0.1; 0.1] \times [-0.1; 0.1]$ around the origin. Inside the cube, where the numerical methods would fail, the evolution of solutions can be described with classical, analytical means: more than half of Tucker's thesis is devoted to accurate analytical expressions (a so-called normal form) for the flow inside the cube L. These expressions can be used to provide explicit bounds on how solutions exit the cube L and continue with numerical computations.

Informal Theorem 8.3. *There is an explicit form that bounds the dynamics inside the cube* $L = [-0.1; 0.1] \times [-0.1; 0.1] \times [-0.1; 0.1]$.

In the rest of this chapter, we present how the verified algorithm *poincare'* of chapter 7 is used to certify Tucker's computations. We show in particular how we formally prove the theorems 8.1 and 8.2. We will also make a precise statement for the informal theorem 8.3.

8.2 The Input Data and its Interpretation

It is not necessary to verify precisely the set N that Tucker used, but coming up with a forward invariant set is slightly more involved than certifying one. We therefore use the output of Tucker's program as a starting point to set up the input for our ODE solver. The output of Tucker's program is available online[1] as a file containing 7258 lines. Since any other forward invariant with suitable cone field and expansion estimates would do just as well, we are free to modify Tucker's data slightly. We preprocessed Tucker's data by merging the information of some of the lines and slightly coarsening some of the numerical bounds.

This results in a list of 400 elements, which we call *input-data* and will be the basis for all further interpretations:

Definition 8.4 (Input Data)**.** *input-data* :: *result list is a list of* 400 *elements of type result*.

$$\begin{aligned}\textbf{datatype}\ \mathit{result} = \mathit{Result}\ &(\mathit{invoke\text{-}nf} : \mathbb{B})\\ &(\mathit{min\text{-}deg} : \mathbb{R})\ (\mathit{max\text{-}deg} : \mathbb{R})\\ &(\mathit{expansion} : \mathbb{R})\ (\mathit{preexpansion} : \mathbb{R})\\ &(\mathit{gridx0} : \mathbb{Z})\ (\mathit{gridx1} : \mathbb{Z})\ (\mathit{gridy0} : \mathbb{Z})\ (\mathit{gridy1} : \mathbb{Z})\\ &(\mathit{inf\text{-}retx} : \mathbb{Z})\ (\mathit{inf\text{-}rety} : \mathbb{Z})\ (\mathit{sup\text{-}retx} : \mathbb{Z})\ (\mathit{sup\text{-}rety} : \mathbb{Z})\end{aligned}$$

Elements *res* of type *result* are interpreted as initial rectangles as follows. The properties *gridx0*, *gridx1*, *gridy0*, and *gridy1* encode a rectangle on Σ, which we denote by $N(res)$. The union of all elements of *input-data* represents the upper branch N^+ of the forward invariant set N. It is plotted in figure 8.3.

[1] `http://www2.math.uu.se/~warwick/main/rodes/ResultFile`

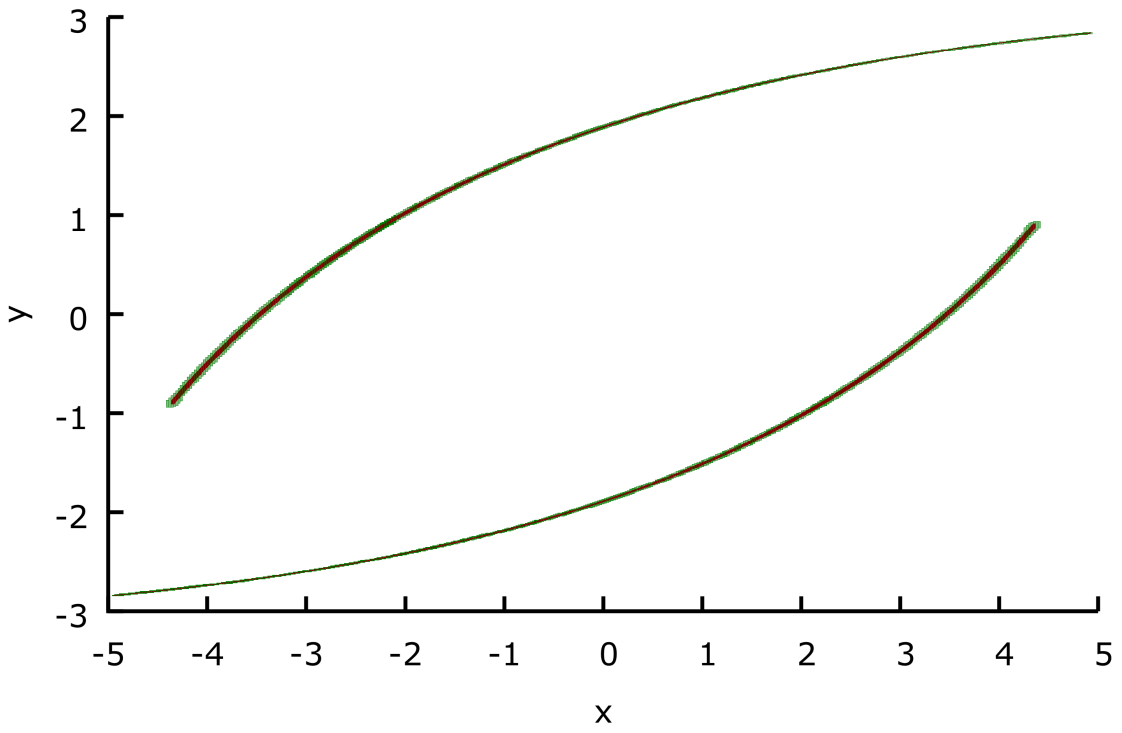


Figure 8.3: N in green (N^+ the upper and $N^- = S(N^+)$ the lower branch) and enclosure of $P(N^+)$ in red.

Definition 8.5.

$$\begin{aligned} N(res) :=& [((\mathit{gridx0}\ res - 1) \cdot 2^{-8}, (\mathit{gridy0}\ res - 1) \cdot 2^{-8}, 27), \\ & ((\mathit{gridx1}\ res + 1) \cdot 2^{-8}, (\mathit{gridy1}\ res + 1) \cdot 2^{-8}, 27)] \\ N^+ :=& \bigcup_{res \in \mathit{input\text{-}data}} N(res) \\ N^- :=& \{(-x, -y, z) \mid (x, y, z) \in (N^+)\} \\ N :=& N^+ \cup N^- \end{aligned}$$

The input data also contains information on the image of an initial rectangle. It is encoded in *inf-retx*, *inf-rety*, *sup-retx*, *sup-rety*: We select the elements within those bounds with *return-of*:

$$\begin{aligned} \mathit{return\text{-}of}\ res := \{&res' \in \mathit{input\text{-}data} \mid \\ & \mathit{gridx0}\ res' \in [\mathit{inf\text{-}retx}\ res; \mathit{sup\text{-}retx}\ res] \wedge \\ & \mathit{gridy0}\ res' \in [\mathit{inf\text{-}rety}\ res; \mathit{sup\text{-}rety}\ res]\} \end{aligned}$$

min-deg and *max-deg* define the cone $\mathfrak{C}$ associated with the rectangle: the conic hull of the line segment between the boundary vectors.

Definition 8.6.

$$\begin{aligned}\mathfrak{C}\ res =& \mathit{cone\ hull}\ (\mathit{segment}\\ &(\cos(\mathit{rad}\ (\mathit{min\text{-}deg}\ res), \sin(\mathit{rad}\ (\mathit{min\text{-}deg}\ res), 0)\\ &(\cos(\mathit{rad}\ (\mathit{max\text{-}deg}\ res)), \sin(\mathit{rad}\ (\mathit{max\text{-}deg}\ res), 0)))))\end{aligned}$$

There $\mathit{rad}\ x = \frac{x \cdot \pi}{180}$ is the radian of the angle given in degrees, $\mathit{segment}\ x\ y$ is the line segment $\{(1-u)\cdot a + u \cdot b \mid u \in [0;1]\}$, and $\mathit{cone\ hull}\ S = \{c \cdot x \mid 0 \le c \wedge x \in S\}$ the conic hull of a set S.

The elements in *input-data* also encode a conefield $\mathfrak{C}$ and expansion estimates as follows. $\mathit{results\text{-}at}\,(x)$ yields the set of result elements that cover a point x (the rectangles overlap at the boundary). We need to respect this to ensure that $\mathfrak{C}$, $\mathcal{E}$, and $\mathcal{E}^{-1}$ are well defined.

Definition 8.7.

$$\begin{aligned}&\mathit{results\text{-}at}\,(x) := \{res \in \mathit{input\text{-}data} \mid x \in N(res)\}\\ &\mathfrak{C}(x) := \bigcup_{res \in \mathit{results\text{-}at}\,(x)} \mathfrak{C}(res)\\ &\mathcal{E}(x) := \min_{res \in \mathit{results\text{-}at}\,(x)} \mathit{expansion}\,(res)\\ &\mathcal{E}^{-1}(x) := \min_{res \in \mathit{results\text{-}at}\,(x)} \mathit{preexpansion}\,(res)\end{aligned}$$

One last property is *invoke-nf*, which encodes if the numerical computations need to be interrupted and the results of the normal form need to be invoked. First, we define abstractly when this is necessary, namely on the *stable manifold* of the origin. That is, the set of all points, which tend to the origin in infinite time. We restrict our attention to the part of the stable manifold whose trajectories do not intersect Σ for positive time.

Definition 8.8.

$$\Gamma := \{x \mid [0;\infty] \subseteq \mathit{ex\text{-}ivl}\ x \wedge (\forall t > 0.\ \phi(x,t) \notin \Sigma) \wedge (\phi(x,t) \longrightarrow_{t\to\infty} (0,0,0))\}$$

When *invoke-nf* is true, the computations will be interrupted once the reachable sets arrive at the small cube $L = [-0.1;0.1] \times [-0.1,0.1] \times [-0.1;0.1]$ inside which the normal form estimates are valid. In our computations, solutions are guaranteed to enter the cube L through a rectangle $\mathit{fst}\,(T)$ and the tangent vectors are in the cone that contains $\mathit{snd}\,(T)$:

$$T := ([-0.1;0.1], [-0.00015;0.00015], 0.1) \times \begin{pmatrix} [0.8,1.7] \\ [0.0005;0.002] \\ 0 \end{pmatrix}$$

That is, sets are very slim in the y-direction, and the expanding direction is closely around the x axis. From Tucker's analysis ([138, Proposition 3.1]), we devised the following bounds for the sets E_1, E_2 through which solutions emanating from T exit the cube L:

$$E_1 := ([-0.12;-0.088], [-0.024;0.024], [-0.012;0.13]) \times \begin{pmatrix} 0 \\ [-0.56;0.56] \\ [-0.6;-0.08] \end{pmatrix}$$

$$E_2 := ([0.088; 0.12], [-0.024; 0.024], [-0.012; 0.13]) \times \begin{pmatrix} 0 \\ [-0.56; 0.56] \\ [0.08; 0.6] \end{pmatrix}$$

When we interrupt computations close to L, we check that the sets entering L do so within T and continue computations from $E_1 \cup E_2$. Since we have not verified Tucker's normal form theory, we need to trust the following assumption, which makes informal theorem 8.3 precise.

Assumption 8.9 (Normal Form Theory Bounds)**.**

$$T \curvearrowright'_L (E_1 \cup E_2)$$

8.3 Checking the Input Data

In the previous section, we only defined what the *input-data* encodes, now we describe how to check that the numerical bounds prescribed by the *input-data* are actually correct. This involves three steps: First, we need to find a suitable setup to be able to use the algorithm *poincare*, which computes derivatives and not cones. Second, we set up the check that a single element of the *input-data* is correct. Third, we check all elements of the *input-data*, from which we conclude the formal counterparts of theorems 8.1 and 8.2.

8.3.1 Representation of Cones

Concerning the checking of cone conditions, first note that $\mathfrak{C}$ *res* is an infinite cone, i.e., an unbounded set of vectors. In contrast to that, all of our numerical algorithms are tailored towards bounded enclosures. We therefore perform the computations with the line segment connecting the two tangent vectors with unit length. *matrix-segment* $x_1\ y_1\ x_2\ y_2\ e$ encodes a line segment (parameterized by e) in a matrix (such that it can be used as matrix initial condition DX of *poincare'*, section 7.4). *mat-seg-of-deg* uses this to define the line segment between the endpoints of unit vectors with given angles u, v to the x axis. A cone can therefore represented with the help of *mat-seg-of-deg*:

Lemma 8.10 (Matrix Representation of Cone)**.**

$$\mathfrak{C}(res) = \textit{cone hull} \left\{ \begin{pmatrix} m_{(1,1)} \\ m_{(2,1)} \\ 0 \end{pmatrix} \middle| m \in \textit{mat-seg-of-deg}\ (\textit{min-deg}\ res)\ (\textit{max-deg}\ res) \right\}$$

with

$$\textit{matrix-segment}\ x_1\ y_1\ x_2\ y_2\ e := \begin{pmatrix} x_1 + e \cdot (x_2 - x_1) & 0 & 0 \\ y_1 + e \cdot (y_2 - y_1) & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

$$\textit{mat-seg-of-deg}\ u\ v :=$$
$$\textit{matrix-segment}\,(\cos\ (\textit{rad}\ u))(\sin\ (\textit{rad}\ u))(\cos\ (\textit{rad}\ v))(\sin\ (\textit{rad}\ v))[0; 1]$$

Algorithm 1 Check Result

1: **function** *check-result-c1* (res)
2: $\quad X_0 \leftarrow N(res)$
3: $\quad DX_0 \leftarrow$ *mat-seg-of-deg* $(\textit{min-deg } res)\ (\textit{max-deg } res)$
4: $\quad RES \leftarrow$ *poincare* $X_0\ DX_0\ \Sigma$
5: $\quad \bigcup_i (P_i \times DP_i) \leftarrow$ *split-along* $N\ RES$
6: $\quad RET \leftarrow$ *get-results* $(\textit{inf-retx } res, \textit{inf-rety } res)\ (\textit{sup-retx } res, \textit{sup-rety } res)$
7: $\quad$ **return** $\forall i.\ \exists ret \in RET.\ \textit{returns-within } res\ P_i\ DP_i\ ret$

8.3.2 Checking a Single Result Element

Algorithm 1 outlines how to check that a single result element $res \in$ *input-data* represents correct numerical bounds. It works as follows: X_0 is the initial rectangle, DX_0 the initial data for the derivatives, which encodes the associated cone with angles *min-deg res* and *max-deg res*. Then the ODE solver returns with a union of return images RES, which are split along the boundaries of the individual rectangles making up N. This splitting ensures that each individual element (P_i, DP_i) resulting from the splitting is contained in exactly one individual element of N. We write singleton parts of the result of this splitting P_i, DP_i. In RET, there are all elements of the *input-data* within which *res* is specified to return. The final check makes sure that every part P_i, DP_i of the splitting returns within one element *ret* of the collection RET. It is defined as follows and precisely formulates that X and DX, which emanate from a result *res* and hit the result *ret*, satisfy the prescribed bounds on cones and expansion.

$$
\begin{aligned}
&\textit{returns-within } res\ X\ DX\ ret := \\
&\quad X \subseteq N(ret) \wedge \\
&\quad \textit{check-cone-bounds } (\textit{min-deg } res)\ (\textit{max-deg } res)\ X\ DX \wedge \\
&\quad \|DX\| \geq \mathcal{E}(res) \wedge \|DX\| \geq \mathcal{E}^{-1}(ret)
\end{aligned}
$$

check-cone-bounds is checked using affine arithmetic: It assumes that u_x and u_y are on the line segment encoding a cone according to *mat-seg-of-deg*, therefore checks that $u_z = 0$ and ignores the other entries of the argument matrix. It further checks that the segment is on the right side ($0 < u_x$) and that the boundary angles L and U (given in degrees) also represent a cone pointing to the right side. The main purpose is in the last line, the check that the angle of the vector (u_x, u_y) with the horizontal axis is between L and U.

$$
\begin{aligned}
&\textit{check-cone-bounds } L\ U \begin{pmatrix} x \\ y \\ z \end{pmatrix} \begin{pmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{pmatrix} := \\
&\quad -90 < L \wedge L \leq U \wedge U < 90 \wedge \\
&\quad 0 < u_x \wedge u_z = 0 \wedge \\
&\quad \tan(\textit{rad } L) \leq \frac{u_y}{u_x} \wedge \frac{u_y}{u_x} \leq \tan(\textit{rad } U)
\end{aligned}
$$

Correctness of *check-result-c1* states that the set $N(res)$ is mapped into the part *return-of res* of the forward invariant set. Vectors in the cone $\mathfrak{C}(res)$ are mapped by the derivative $\mathrm{D}P$

into the cone field with the prescribed expansion estimates. The theorem states that the derivative exists and is defined when approaching x within $\Sigma_{\leq} = \{(x, y, z) \mid z \leq 27\}$.

Theorem 8.11 (Correctness of *check-result-c1*)**.**

$$\begin{aligned}
&\textit{check-result-c1}\,(res) = \textit{return True} \longrightarrow \\
&\quad \forall x \in N(res) - \Gamma.\ \forall dx \in \mathfrak{C}(res).\textit{returns-to}\ \Sigma\ x \wedge P(x) \in N(\textit{return-of}\ res) \wedge \\
&\qquad (\exists DP.\ (P\ \textit{has-derivative}\ DP)\ (\textit{at}\ x\ \textit{within}\ \Sigma_{\leq}) \\
&\qquad\quad (\|DP(dx)\| \geq \mathcal{E}(res) \cdot \|dx\|) \wedge \\
&\qquad\quad (\exists ret \in \textit{return-of}\ res. \\
&\qquad\qquad P(x) \in N(ret) \wedge DP(dx) \in \mathfrak{C}(ret) \wedge \|DP(dx)\| \geq \mathcal{E}^{-1}(ret) \cdot \|dx\|))
\end{aligned}$$

The theorem follows rather directly from the definition of algorithm 1 and the specifications and definitions of the occurring functions.

8.3.3 Checking All Results

We prove that all *input-data* is correct:

Theorem 8.12 (Global Numerical Results)**.**

$$\forall res \in \textit{input-data}\,.\ \textit{check-result-c1}\ res = \textit{return True}$$

Theorem 8.12 is proved by computing *check-result-c1*$\,(res)$ for every $res \in$ *input-data*. The computations are carried out by evaluating the statement

$$\textit{Parallel.forall}\ (\lambda res.\ \textit{check-result-c1}\ res)\ \textit{input-data}$$

with Isabelle/HOL's evaluation engine `eval`.

As a result of those computations, we can state the main theorem of this thesis: a formal verification of Tucker's computations.

Theorem 8.13 (Correctness of Tucker's Computations)**.** *Assumption 8.9 implies theorem 8.1 and theorem 8.2.*

It follows from combining the individual instances of theorem 8.11 for all input data (theorem 8.12) in a suitable way.

Parallel.forall results in parallel processing (using multithreading in PolyML [98]) of the 400 individual elements of *input-data*. Further parallelism is introduced when enclosures are split during reachability analysis. Split sets can be processed in parallel until they reach the next (intermediate) Poincaré section, where they might be (partially merged) upon resolving the intersection (section 7.3.2).

Figure 8.1 shows the plot of an enclosure for the Lorenz attractor resulting from the verified computation. The plot (and its detail in figure 8.5) hints at the intermediate Poincaré sections that were manually set up (for some initial rectangles) at about $z = 27$, $z = 30$, $x = \pm 5$,$x = \pm 1.5$,$x = \pm 1$, $x = \pm 0.75$, $x = \pm 0.1$, and $z = 0.1$. The black part of figure 8.3 is

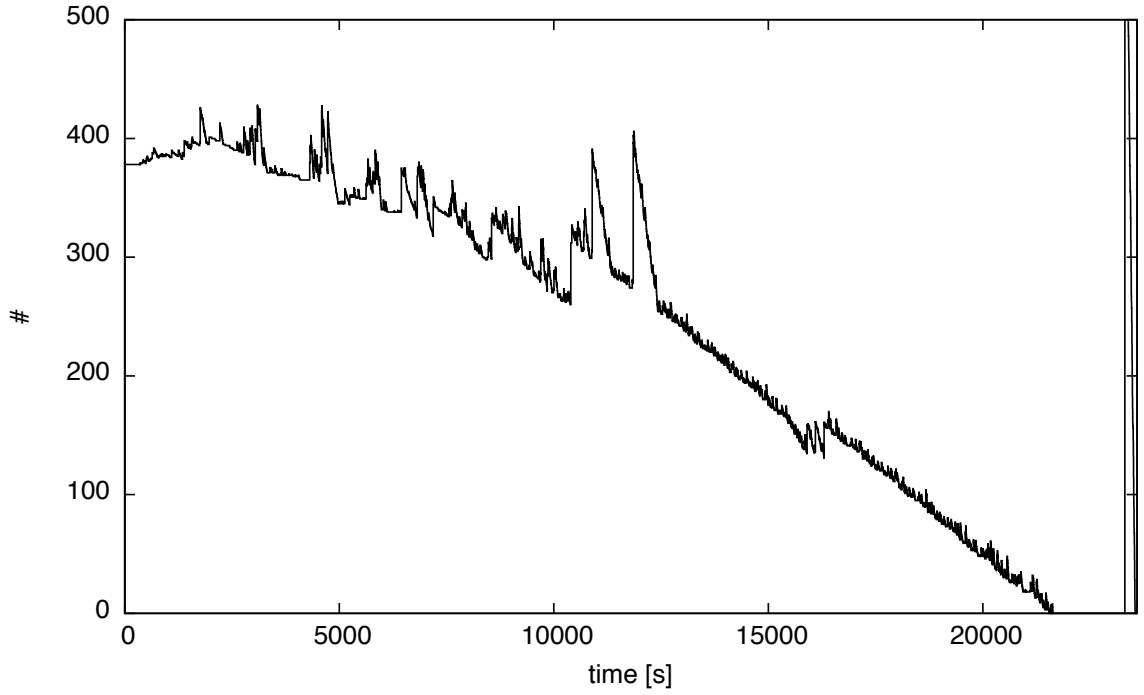


Figure 8.4: ML Statistics: Tasks waiting for Execution

an enclosure for $P(N^+)$ resulting from these computations, and it is as expected and verified contained in N.

The timing results of a computation on a machine with 22 cores[2] are given below:

- Elapsed Time: 6h 33min 9s
- CPU Time: 131h 52min 40 s
- Parallelization Factor: 20.13
- Garbage Collection Time: 42min 36s

A parallelization factor of 20 signifies a good saturation of the available cores, the progress of parallel tasks waiting for execution is shown in figure 8.4. There one can see, that in the first half of the elapsed time, tasks are spawned (mostly because the *input-data* was sorted in such a way that the result elements close to the stable manifold (the ones that perform most splitting) are processed first. In the second half, it is mostly working off tasks in the queue. There appears to be a longer sequential computation in the end, which seems to be responsible for the fact that the parallelization factor is significantly lower than 22.

To compare the general run time with Tucker's C++ program, I compiled Tucker's program in a Virtual Machine running Ubuntu 4.20 and gcc version 3.3.4 on a machine with a 2,6 GHz Intel® Core™ i7 CPU and 16 GB RAM. The program finished after a total computation time of 30h and 24min. A direct comparison is hard, because the algorithms and data

[2] Intel®Xeon®CPU E5-2699 v4 @ 2.20GHz

structures are very different, but a factor of less than five compared to a C++ program signifies reasonable performance of the verified algorithm.

In earlier developments [70], an enclosure for the Lorenz attractor was computed with neither derivative nor cones. This earlier version verified an enclosure for the Lorenz attractor in about 7000 CPU hours. With the present version algorithms, such a computation (without derivatives and cones) can be performed in about 3 CPU hours. The speedup compared to the earlier work is mostly due to less aggressive splitting of reachable sets, and a smaller number of intermediate Poincaré sections: In the earlier work [70], intermediate Poincaré sections were introduced heuristically on-the-fly, and in the present work only where they are really effective. This is beneficial, because resolving the intersection incurs overapproximations.

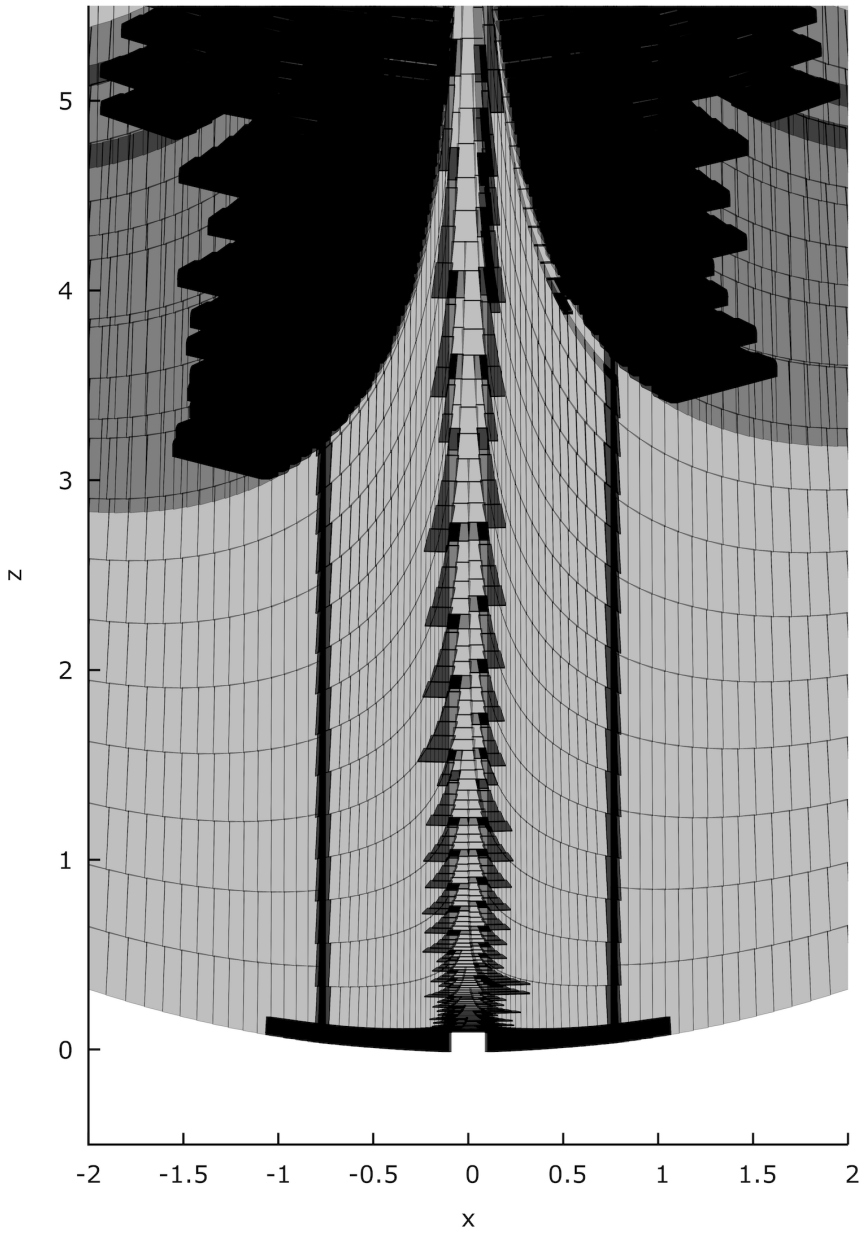


Figure 8.5: Enclosure of $P(N^+)$ (detail around the origin)

9 Conclusion

This chapter concludes my thesis: modern ITP technology is capable for the formal verification of advanced rigorous numerical ODE solvers. It requires sufficient formalized mathematical background (chapters 2 and 3), the right abstractions (enclosures for expressions in chapter 4, algebraic reasoning about geometry in chapter 5), and a suitable setup for a (in this case Lammich's Autoref) framework for specifying and verifying high-level algorithms (chapter 6).

My work brings a new quality of trust to rigorous numerics for dynamical systems and I managed to apply it to a celebrated result.

This result is specialized, but most of the formalized techniques are standard and useful for any ODE. This can be seen in the successful participation in a software competition for non-verified software and by the fact that the theoretical foundations could be used for other formalization projects [16].

Implementing an ODE solver leaves much space for design decisions. In my work, I chose a compromise between ease of formalization and efficiency/precision. The overall structure that emerged, however, should make it possible to re-visit any of those decisions and do modular changes (e.g., replace zonotopes with Taylor models or choose some adaptive high-order enclosure method instead of the present second order Runge-Kutta method).

I would like to prelude the end of this dissertation with an overview of the efforts required for the formalization (estimated by the number of lines of code or proof), which will also underpin the claim that most of the formalization is not specialized for the Lorenz attractor. An outlook on possible future work ends this dissertation.

9.1 Code Size

Table 9.1 shows some statistics on the size in terms of lines of code of several programs related to this verification. RODES is the rigorous ODE solver used by Tucker, it consists of 3800 lines of C++ code and builds on a library for interval arithmetic (Profil/BIAS) of about twice the size. Similar to the sum of those two is the size of the generated SML code. This indicates that the generated code is not overly complicated. The verification required more effort, but the largest part is generic: the part specific to the Lorenz attractor (in particular about cones) makes up only about 3000 lines of code.

	chapter	language	lines of code/proof
RODES		C++	3800
Profil/BIAS		C++	8800
generated ODE solver		SML	12000
Flow, Poincaré map	3	Isabelle/HOL theory	13000
Affine Arithmetic	4		12000
Intersection	5		5000
Refinement/Enclosures	6		6000
Verification of ODE solver	7		12000
Lorenz Attractor	8		3000

Table 9.1: Size of Code and Formalization

9.2 Future Work

There are several possible directions for future work: towards a formal proof of Smale's 14th problem, towards formal analysis of hybrid (discrete-continuous) systems, and general improvements of performance and precision of the ODE solver.

9.2.1 A Formal Proof of Smale's 14th Problem

One possible direction for future work is towards a complete formal proof of Smale's 14th problem. This involves formalization of two orthogonal topics.

First, a formalization of the normal form theory part of Tucker's proof, which involves in particular multivariate formal power series and a number of analytic estimations, proving existence and convergence of specifically constructed formal power series. Tucker generalized this technique to arbitrary ODEs with saddle fixed points [139]. Interestingly, also this part of Tucker's proof involves rigorous numerical computations, but the main difficulty lies really in the analytical estimates. Tucker's programs `smalldiv.cc` and `coeffs.cc` help devising the normal form, but neither their specification nor their implementation is particularly interesting, they essentially only evaluate a large number of fixed arithmetic expressions.

Second, concluding from the numerical results that the Lorenz equations actually support a robust singular hyperbolic attractor with sensitive dependence on initial conditions. This would require a well-developed theory of differentiable manifolds (to my knowledge, the only formalization of manifolds is in Mizar [115, 120]) and advanced results about dynamics on manifolds, in particular invariant foliations along the lines of Hirsch *et al.* [64].

9.2.2 Formally Verified Analysis of Hybrid Systems

The present work could be extended to target applications that are ubiquitous in engineering applications, i.e., hybrid systems. One would need to formalize hybrid systems with a suitable model, e.g., the classic hybrid automata [63], differential dynamic logic (which is already formalized in Isabelle/HOL [16]), more applied formalisms like Acumen [132], or following the work of rigorous simulation of Matlab/Simulink models [23]. Much of the

work on intermediate Poincaré sections could be used for switching hyperplanes (guards) of hybrid automata.

9.2.3 Improving the Verified ODE Solver

The present algorithms for enclosing ODEs are a pragmatic attempt, more efficient or precise techniques could be incorporated to obtain a versatile, formally verified toolbox. Representing enclosures with Taylor models, as formalized by Traut [133] would improve accuracy on nonlinear systems, high-order Taylor series methods would be the method of choice for small initial conditions and high accuracy, and the matrix exponential is a must for high-dimensional linear systems.

Bibliography

[1] Mohammad Abdulaziz and Lawrence C. Paulson. An Isabelle/HOL formalisation of green's theorem. In Jasmin Christian Blanchette and Stephan Merz, editors, *Interactive Theorem Proving: 7th International Conference, ITP 2016, Nancy, France, August 22-25, 2016, Proceedings*, pages 3–19, Cham, 2016. Springer International Publishing. ISBN 978-3-319-43144-4. doi: 10.1007/978-3-319-43144-4_1.

[2] Zafar Ahmed. Ahmed's integral: the maiden solution. *arXiv preprint arXiv:1411.5169*, 2014.

[3] M. Althoff. An introduction to CORA 2015. In *Proc. of the Workshop on Applied Verification for Continuous and Hybrid Systems*, pages 120–151, 2015.

[4] M. Althoff and D. Grebenyuk. Implementation of interval arithmetic in CORA 2016. In *Proc. of the 3rd International Workshop on Applied Verification for Continuous and Hybrid Systems*, pages 91–105, 2016.

[5] M. Althoff, O. Stursberg, and M. Buss. Reachability analysis of nonlinear systems with uncertain parameters using conservative linearization. In *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*, pages 4042–4048, Dec 2008. doi: 10.1109/CDC.2008.4738704.

[6] Matthias Althoff and Bruce H Krogh. Zonotope bundles for the efficient computation of reachable sets. In *Decision and Control and European Control Conference (CDC-ECC), 2011 50th IEEE Conference on*, pages 6814–6821. IEEE, 2011.

[7] Matthias Althoff and Bruce H. Krogh. Avoiding geometric intersection operations in reachability analysis of hybrid systems. In *Proceedings of the 15th ACM International Conference on Hybrid Systems: Computation and Control*, HSCC '12, pages 45–54, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1220-2.

[8] Jeremy Avigad, Johannes Hölzl, and Luke Serafin. A formally verified proof of the central limit theorem. *Journal of Automated Reasoning*, pages 1–35, 2017. ISSN 1573-0670. doi: 10.1007/s10817-017-9404-x. URL `http://dx.doi.org/10.1007/s10817-017-9404-x`.

[9] Ralph-Johan Back and Joakim Wright. *Refinement calculus: a systematic introduction*. Springer Science & Business Media, 2012.

[10] Stanley Bak. Reducing the wrapping effect in flowpipe construction using pseudo-invariants. In *Proceedings of the 4th ACM SIGBED International Workshop on Design, Modeling, and Evaluation of Cyber-Physical Systems*, CyPhy '14, pages 40–43, New York, NY, USA, 2014. ACM.

[11] Stanley Bak, Sergiy Bogomolov, Thomas A. Henzinger, Taylor T. Johnson, and Pradyot Prakash. Scalable static hybridization methods for analysis of nonlinear systems. In *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*,

HSCC '16, pages 155–164, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-3955-1. URL http://doi.acm.org/10.1145/2883817.2883837.

[12] Andrea Balluchi, Alberto Casagrande, Pieter Collins, Alberto Ferrari, Tiziano Villa, and Alberto L. Sangiovanni-Vincentelli. Ariadne: a framework for reachability analysis of hybrid automata. In *Proceedings of the 17th International Symposium on Mathematical Theory of Networks and Systems (MTNS 2006)*, Kyoto, Japan, July 2006.

[13] Yves Bertot and Pierre Castéran. *Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions*. Springer Science & Business Media, 2013.

[14] Martin Berz and Kyoko Makino. Verified integration of ODEs and flows using differential algebraic methods on high-order Taylor models. *Reliable Computing*, 4(4): 361–369, 1998. ISSN 1385-3139. doi: 10.1023/A:1024467732637.

[15] Brandon Bohrer. Differential dynamic logic. *Archive of Formal Proofs*, February 2017. ISSN 2150-914x. http://isa-afp.org/entries/Differential_Dynamic_Logic.shtml, Formal proof development.

[16] Brandon Bohrer, Vincent Rahli, Ivana Vukotic, Marcus Völp, and André Platzer. Formally verified differential dynamic logic. In *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs*, CPP 2017, pages 208–221, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4705-1. URL http://doi.acm.org/10.1145/3018610.3018616.

[17] S. Boldo and G. Melquiond. Flocq: A unified library for proving floating-point algorithms in Coq. In *2011 IEEE 20th Symposium on Computer Arithmetic*, pages 243–252, July 2011. doi: 10.1109/ARITH.2011.40.

[18] Sylvie Boldo, François Clément, Jean-Christophe Filliâtre, Micaela Mayero, Guillaume Melquiond, and Pierre Weis. Formal proof of a wave equation resolution scheme: The method error. In Matt Kaufmann and Lawrence C. Paulson, editors, *ITP 2010*, volume 6172 of *LNCS*, pages 147–162, 2010. ISBN 978-3-642-14051-8. doi: 10.1007/978-3-642-14052-5_12.

[19] Sylvie Boldo, François Clément, Jean-Christophe Filliâtre, Micaela Mayero, Guillaume Melquiond, and Pierre Weis. Wave equation numerical resolution: A comprehensive mechanized proof of a C program. *Journal of Automated Reasoning*, 50(4):423–456, 2013. ISSN 0168-7433. doi: 10.1007/s10817-012-9255-4.

[20] Sylvie Boldo, Catherine Lelay, and Guillaume Melquiond. Coquelicot: A user-friendly library of real analysis for Coq. *Mathematics in Computer Science*, 9(1):41–62, 2015. ISSN 1661-8289. URL http://dx.doi.org/10.1007/s11786-014-0181-1.

[21] Sylvie Boldo, Catherine Lelay, and Guillaume Melquiond. Formalization of real analysis: a survey of proof assistants and libraries. *Mathematical Structures in Computer Science*, 26(7):1196–1233, 2016. doi: 10.1017/S0960129514000437.

[22] Sylvie Boldo, François Clément, Florian Faissole, Vincent Martin, and Micaela Mayero. A Coq formal proof of the lax-milgram theorem. In *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs*, pages 79–89. ACM, 2017.

[23] Olivier Bouissou, Samuel Mimram, and Alexandre Chapoutot. Hyson: Set-based simulation of hybrid systems. In *Rapid System Prototyping (RSP), 2012 23rd IEEE International Symposium on*, pages 79–85. IEEE, 2012.

[24] Olivier Bouissou, Alexandre Chapoutot, and Adel Djoudi. Enclosing temporal evolution of dynamical systems using numerical methods. In Guillaume Brat, Neha Rungta, and Arnaud Venet, editors, *NASA Formal Methods*, volume 7871 of *LNCS*, pages 108–123. Springer, 2013. ISBN 978-3-642-38087-7. doi: 10.1007/978-3-642-38088-4_8.

[25] Nicolas Bourbaki. *General Topology: Chapters 1–4*. Springer Berlin Heidelberg, 1995. ISBN 978-3-540-64241-1. URL `http://doi.org/10.1007/978-3-642-61701-0`.

[26] Nicolas Brisebarre, Mioara Joldeş, Érik Martin-Dorel, Micaela Mayero, Jean-Michel Muller, Ioana Paşca, Laurence Rideau, and Laurent Théry. Rigorous polynomial approximation using Taylor models in Coq. In Alwyn E. Goodloe and Suzette Person, editors, *NASA Formal Methods*, LNCS, pages 85–99. Springer, 2012.

[27] Christophe Brun, Jean-François Dufourd, and Nicolas Magaud. Designing and proving correct a convex hull algorithm with hypermaps in Coq. *Computational Geometry*, 45(8): 436 – 457, 2012. ISSN 0925-7721. doi: http://dx.doi.org/10.1016/j.comgeo.2010.06.006. Geometric Constraints and Reasoning.

[28] Lukas Bulwahn. The new quickcheck for Isabelle. In Chris Hawblitzel and Dale Miller, editors, *Certified Programs and Proofs: Second International Conference, CPP 2012, Kyoto, Japan, December 13-15, 2012. Proceedings*, pages 92–108, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN 978-3-642-35308-6. doi: 10.1007/978-3-642-35308-6_10.

[29] Florian Bünger. Shrink wrapping for taylor models revisited. *Numerical Algorithms*, Sep 2017. ISSN 1572-9265. URL `https://doi.org/10.1007/s11075-017-0410-1`.

[30] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In Natasha Sharygina and Helmut Veith, editors, *CAV*, volume 8044 of *LNCS*, pages 258–263. Springer, 2013. ISBN 978-3-642-39798-1. doi: 10.1007/978-3-642-39799-8_18.

[31] Xin Chen, Matthias Althoff, and Fabian Immler. ARCH-COMP17 category report: Continuous systems with nonlinear dynamics. In Goran Frehse and Matthias Althoff, editors, *ARCH17. 4th International Workshop on Applied Verification of Continuous and Hybrid Systems*, volume 48 of *EPiC Series in Computing*, pages 160–169. EasyChair, 2017.

[32] Alonzo Church. A Formulation of the Simple Theory of Types. *The Journal of Symbolic Logic*, 5(2):56–68, 1940. URL `http://www.jstor.org/stable/2266170`.

[33] Pieter Collins, Milad Niqui, and Nathalie Revol. A validated real function calculus. *Mathematics in Computer Science*, 5(4):437–467, 2011. ISSN 1661-8270. doi: 10.1007/s11786-011-0102-5.

[34] Luiz Henrique de Figueiredo and Jorge Stolfi. Affine arithmetic: Concepts and applications. *Numerical Algorithms*, 37(1-4):147–158, 2004.

[35] Peter Deuflhard and Folkmar Bornemann. *Scientific computing with ordinary differential equations*, volume 42. Springer Science & Business Media, 2012.

[36] Javier Esparza, Peter Lammich, René Neumann, Tobias Nipkow, Alexander Schimpf, and Jan-Georg Smaus. A fully verified executable LTL model checker. In *International Conference on Computer Aided Verification*, pages 463–478. Springer, 2013.

[37] Jacques D. Fleuriot and Lawrence C. Paulson. Mechanizing nonstandard real analysis. *LMS Journal of Computation and Mathematics*, 3:140–190, 2000. ISSN 1461-1570. doi: 10.1112/S1461157000000267.

[38] Martin Fränzle, Christian Herde, Stefan Ratschan, Tobias Schubert, and Tino Teige. Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. *Journal on Satisfiability, Boolean Modeling and Computation*, 1:209–236, 2007.

[39] 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. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *CAV*, volume 6806 of *LNCS*, pages 379–395. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-22109-5.

[40] Nathan Fulton, Stefan Mitsch, Jan-David Quesel, Marcus Völp, and André Platzer. KeYmaera X: An axiomatic tactical theorem prover for hybrid systems. In Amy P. Felty and Aart Middeldorp, editors, *CADE*, volume 9195 of *LNCS*, pages 527–538. Springer, 2015. doi: 10.1007/978-3-319-21401-6_36.

[41] Antoine Girard. Reachability of uncertain linear systems using zonotopes. In Manfred Morari and Lothar Thiele, editors, *Hybrid Systems: Computation and Control*, volume 3414 of *LNCS*, pages 291–305. Springer, 2005. ISBN 978-3-540-25108-8. doi: 10.1007/978-3-540-31954-2_19.

[42] Antoine Girard and Colas Le Guernic. Zonotope/hyperplane intersection for hybrid systems reachability analysis. In Magnus Egerstedt and Bud Mishra, editors, *Hybrid Systems: Computation and Control*, volume 4981 of *LNCS*, pages 215–228. Springer, 2008. ISBN 978-3-540-78928-4. doi: 10.1007/978-3-540-78929-1_16.

[43] Jean-Yves Girard. The system F of variable types, fifteen years later. *Theoretical computer science*, 45:159–192, 1986.

[44] Georges Gonthier. Formal proof–the four-color theorem. *Notices of the AMS*, 55(11):1382–1393, 2008.

[45] Georges Gonthier, Andrea Asperti, Jeremy Avigad, Yves Bertot, Cyril Cohen, François Garillot, Stéphane Le Roux, Assia Mahboubi, Russell O'Connor, Sidi Ould Biha, Ioana Pasca, Laurence Rideau, Alexey Solovyev, Enrico Tassi, and Laurent Théry. A machine-checked proof of the odd order theorem. In Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, editors, *Interactive Theorem Proving: 4th International Conference, ITP 2013, Rennes, France, July 22-26, 2013. Proceedings*, pages 163–179, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-39634-2. doi: 10.1007/978-3-642-39634-2_14.

[46] Michael Gordon, Robin Milner, and Christopher Wadsworth. Edinburgh LCF: a mechanized logic of computation. volume 78 of *Lecture Notes in Computer Science*. Springer-Verlag Heidelberg, 1979. ISBN 978-3-540-09724-2. doi: 10.1007/3-540-09724-4.

[47] Michael J. C. Gordon. Hol: A proof generating system for higher-order logic. In Graham Birtwistle and P. A. Subrahmanyam, editors, *VLSI Specification, Verification and Synthesis*, pages 73–128, Boston, MA, 1988. Springer US. ISBN 978-1-4613-2007-4. doi: 10.1007/978-1-4613-2007-4_3.

[48] Sebastien Gouezel. Ergodic theory. *Archive of Formal Proofs*, December 2015. ISSN 2150-914x. `http://isa-afp.org/entries/Ergodic_Theory.shtml`, Formal proof development.

[49] Sebastien Gouezel. Lp spaces. *Archive of Formal Proofs*, October 2016. ISSN 2150-914x. `http://isa-afp.org/entries/Lp.shtml`, Formal proof development.

[50] John Guckenheimer and Robert F Williams. Structural stability of Lorenz attractors. *Publications Mathématiques de l'Institut des Hautes Études Scientifiques*, 50(1):59–72, 1979.

[51] Florian Haftmann and Tobias Nipkow. Code generation via higher-order rewrite systems. In Matthias Blume, Naoki Kobayashi, and Germán Vidal, editors, *Functional and Logic Programming*, volume 6009 of *Lecture Notes in Computer Science*, pages 103–117. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-12250-7. doi: 10.1007/978-3-642-12251-4_9.

[52] Florian Haftmann and Makarius Wenzel. Constructive type classes in Isabelle. In *International Workshop on Types for Proofs and Programs*, pages 160–174. Springer, 2006.

[53] Florian Haftmann and Makarius Wenzel. Local theory specifications in Isabelle/Isar. In *International Workshop on Types for Proofs and Programs*, pages 153–168. Springer, 2008.

[54] Florian Haftmann, Alexander Krauss, Ondřej Kunčar, and Tobias Nipkow. Data refinement in Isabelle/HOL. In Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, editors, *Interactive Theorem Proving: 4th International Conference, ITP 2013, Rennes, France, July 22-26, 2013. Proceedings*, pages 100–115, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-39634-2.

[55] Thomas Hales, Mark Adams, Gertrud Bauer, Dat Tat Dang, John Harrison, Truong Le Hoang, Cezary Kaliszyk, Victor Magron, Sean McLaughlin, Thang Tat Nguyen, et al. A formal proof of the Kepler conjecture. *arXiv preprint arXiv:1501.02155*, 2015.

[56] John Harrison. *Theorem Proving with the Real Numbers*. PhD thesis, University of Cambridge, 1996.

[57] John Harrison. Floating point verification in HOL light: The exponential function. In Michael Johnson, editor, *Algebraic Methodology and Software Technology: 6th International Conference, AMAST'97 Sydney, Australia, December13–17, 1997 Proceedings*, pages 246–260, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg. ISBN 978-3-540-69661-2. doi: 10.1007/BFb0000475.

[58] John Harrison. A HOL theory of Euclidean space. In Joe Hurd and Tom Melham, editors, *Theorem Proving in Higher Order Logics, 18th International Conference, TPHOLs 2005*, volume 3603 of *Lecture Notes in Computer Science*, pages 114–129, 2005.

[59] John Harrison. Floating-point verification using theorem proving. In Marco Bernardo and Alessandro Cimatti, editors, *Formal Methods for Hardware Verification: 6th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2006, Bertinoro, Italy, May 22-27, 2006, Advanced Lectures*, pages 211–242, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-34305-9. doi: 10.1007/11757283_8.

[60] John Harrison. Formalizing basic complex analysis. *From Insight to Proof: Festschrift in Honour of Andrzej Trybulec*, 10(23):151–165, 2007.

[61] John Harrison. The HOL Light theory of euclidean space. *Journal of Automated Reasoning*, 50(2):173–190, 2013. ISSN 1573-0670. doi: 10.1007/s10817-012-9250-9. URL `http://dx.doi.org/10.1007/s10817-012-9250-9`.

[62] HA Helfgott. Major arcs for goldbach's problem. *arXiv preprint arXiv:1305.2897*, 2013.

[63] Thomas A Henzinger. The theory of hybrid automata. In *Verification of Digital and Hybrid Systems*, pages 265–292. Springer, 2000.

[64] Morris W Hirsch, Charles Chapman Pugh, and Michael Shub. *Invariant manifolds*, volume 583. Springer, 1977.

[65] Morris W Hirsch, Stephen Smale, and Robert L Devaney. *Differential Equations, Dynamical Systems, and an Introduction to Chaos*. Elsevier Academic Print, 2013.

[66] Johannes Hölzl. Proving inequalities over reals with computation in Isabelle/HOL. In Gabriel Dos Reis and Laurent Théry, editors, *Programming Languages for Mechanized Mathematics Systems (ACM SIGSAM PLMMS'09)*, pages 38–45, 2009.

[67] Johannes Hölzl, Fabian Immler, and Brian Huffman. Type classes and filters for mathematical analysis in Isabelle/HOL. In Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, editors, *Interactive Theorem Proving: 4th International Conference,*

ITP 2013, Rennes, France, July 22-26, 2013. Proceedings, pages 279–294, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-39634-2. URL https://doi.org/10.1007/978-3-642-39634-2_21.

[68] Brian Huffman and Ondřej Kunčar. Lifting and transfer: A modular design for quotients in Isabelle/HOL. In *International Conference on Certified Programs and Proofs*, pages 131–146. Springer, 2013.

[69] Fabian Immler. Formally verified computation of enclosures of solutions of ordinary differential equations. In Julia M. Badger and Kristin Yvonne Rozier, editors, *NASA Formal Methods: 6th International Symposium, NFM 2014, Houston, TX, USA, April 29 – May 1, 2014. Proceedings*, pages 113–127, Cham, 2014. Springer International Publishing. ISBN 978-3-319-06200-6. URL https://doi.org/10.1007/978-3-319-06200-6_9.

[70] Fabian Immler. A verified enclosure for the Lorenz attractor (rough diamond). In Christian Urban and Xingyuan Zhang, editors, *Interactive Theorem Proving: 6th International Conference, ITP 2015, Nanjing, China, August 24-27, 2015, Proceedings*, pages 221–226, Cham, 2015. Springer International Publishing. ISBN 978-3-319-22102-1. URL https://doi.org/10.1007/978-3-319-22102-1_14.

[71] Fabian Immler. Verified reachability analysis of continuous systems. In Christel Baier and Cesare Tinelli, editors, *Tools and Algorithms for the Construction and Analysis of Systems: 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015, Proceedings*, pages 37–51, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg. ISBN 978-3-662-46681-0. URL https://doi.org/10.1007/978-3-662-46681-0_3.

[72] Fabian Immler. A verified algorithm for geometric zonotope/hyperplane intersection. In *Proceedings of the 2015 Conference on Certified Programs and Proofs*, CPP '15, pages 129–136, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3296-5. URL http://doi.acm.org/10.1145/2676724.2693164.

[73] Fabian Immler. Affine arithmetic. *Archive of Formal Proofs*, September 2017. ISSN 2150-914x. http://isa-afp.org/entries/Affine_Arithmetic.shtml, Formal proof development.

[74] Fabian Immler. A verified ODE solver and the Lorenz attractor. *Journal of Automated Reasoning*, 61(1):73–111, 2018. ISSN 1573-0670. URL https://doi.org/10.1007/s10817-017-9448-y.

[75] Fabian Immler and Johannes Hölzl. Numerical analysis of ordinary differential equations in Isabelle/HOL. In Lennart Beringer and Amy Felty, editors, *Interactive Theorem Proving: Third International Conference, ITP 2012, Princeton, NJ, USA, August 13-15, 2012. Proceedings*, pages 377–392, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN 978-3-642-32347-8. URL https://doi.org/10.1007/978-3-642-32347-8_26.

[76] Fabian Immler and Johannes Hölzl. Ordinary differential equations. *Archive of Formal Proofs*, September 2017. ISSN 2150-914x. http://isa-afp.org/entries/Ordinary_Differential_Equations.shtml, Formal proof development.

[77] Fabian Immler and Alexander Maletzky. Gröbner bases theory. *Archive of Formal Proofs*, May 2016. ISSN 2150-914x. `http://isa-afp.org/entries/Groebner_Bases.shtml`, Formal proof development.

[78] Fabian Immler and Christoph Traut. The flow of ODEs. In Jasmin Christian Blanchette and Stephan Merz, editors, *Interactive Theorem Proving: 7th International Conference, ITP 2016, Nancy, France, August 22-25, 2016, Proceedings*, pages 184–199, Cham, 2016. Springer International Publishing. ISBN 978-3-319-43144-4. URL `https://doi.org/10.1007/978-3-319-43144-4_12`.

[79] Fabian Immler and Christoph Traut. The flow of ODEs: Formalization of variational equation and Poincaré map. *Journal of Automated Reasoning*, 2018. ISSN 1573-0670. URL `https://doi.org/10.1007/s10817-018-9449-5`.

[80] K. D. Joshi. *Introduction to General Topology*. John Wiley and Sons, 1983.

[81] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, et al. sel4: Formal verification of an os kernel. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 207–220. ACM, 2009.

[82] Donald Knuth. *Axioms and Hulls*. Springer, Berlin New York, 1992. Number 606 in Lecture Notes in Computer Science.

[83] Robbert Krebbers and Bas Spitters. Type classes for efficient exact real arithmetic in Coq. *CoRR*, abs/1106.3448, 2011.

[84] Robbert Krebbers and Bas Spitters. Computer certified efficient exact reals in Coq. In *Conference on Intelligent Computer Mathematics (CICM 2011)*, volume 6824 of *LNAI*, pages 90–103, 2011.

[85] Peter Lammich. Refinement for monadic programs. *Archive of Formal Proofs*, 2012. ISSN 2150-914x. `http://afp.sf.net/entries/Refine_Monadic.shtml`, Formal proof development.

[86] Peter Lammich. Automatic data refinement. In *International Conference on Interactive Theorem Proving*, pages 84–99. Springer, 2013.

[87] Ari Laptev. *European Congress of Mathematics: Stockholm, June 27-July 2, 2004*. European Mathematical Society, 2005. doi: 10.4171/009.

[88] M. T. Laub and W. F. Loomis. A molecular network that produces spontaneous oscillations in excitable cells of dictyostelium. *Molecular Biology of the Cell*, 9:3521–3532, 1998.

[89] Colas Le Guernic and Antoine Girard. Reachability analysis of hybrid systems using support functions. In Ahmed Bouajjani and Oded Maler, editors, *Computer Aided Verification*, volume 5643 of *Lecture Notes in Computer Science*, pages 540–554. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-02657-7. doi: 10.1007/978-3-642-02658-4_40.

[90] Xavier Leroy. Formal verification of a realistic compiler. *Communications of the ACM*, 52(7):107–115, 2009.

[91] David R Lester. Topology in PVS: continuous mathematics with applications. In *Second workshop on Automated formal methods*, AFM '07, pages 11–20, 2007. doi: 10.1145/1345169.1345171.

[92] Edward N. Lorenz. Deterministic nonperiodic flow. *Journal of the Atmospheric Sciences*, 20(2):130–141, 1963. doi: 10.1175/1520-0469(1963)020<0130:DNF>2.0.CO;2.

[93] Marco Maggesi. A formalization of metric spaces in HOL Light. *Journal of Automated Reasoning*, pages 1–18, 2017. ISSN 1573-0670. doi: 10.1007/s10817-017-9412-x. URL `http://dx.doi.org/10.1007/s10817-017-9412-x`.

[94] Assia Mahboubi, Guillaume Melquiond, and Thomas Sibut-Pinote. Formally verified approximations of definite integrals. In Jasmin Christian Blanchette and Stephan Merz, editors, *Interactive Theorem Proving: 7th International Conference, ITP 2016, Nancy, France, August 22-25, 2016, Proceedings*, pages 274–289, Cham, 2016. Springer International Publishing. ISBN 978-3-319-43144-4.

[95] Evgeny Makarov and Bas Spitters. The Picard algorithm for ordinary differential equations in Coq. In Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, editors, *Interactive Theorem Proving*, volume 7998 of *LNCS*, pages 463–468. Springer Berlin Heidelberg, 2013.

[96] Érik Martin-Dorel and Guillaume Melquiond. Proving tight bounds on univariate expressions with elementary functions in Coq. *J. Autom. Reason.*, 57(3):187–217, October 2016. ISSN 0168-7433. doi: 10.1007/s10817-015-9350-4.

[97] Per Martin-Löf and Giovanni Sambin. *Intuitionistic type theory*, volume 9. Bibliopolis Napoli, 1984.

[98] David CJ Matthews and Makarius Wenzel. Efficient parallel programming in Poly/ML and Isabelle/ML. In *Proceedings of the 5th ACM SIGPLAN workshop on Declarative aspects of multicore programming*, pages 53–62. ACM, 2010.

[99] Laura I. Meikle and Jacques D. Fleuriot. Mechanical theorem proving in computational geometry. In Hoon Hong and Dongming Wang, editors, *Automated Deduction in Geometry*, volume 3763 of *Lecture Notes in Computer Science*, pages 1–18. Springer Berlin Heidelberg, 2006. ISBN 978-3-540-31332-8. doi: 10.1007/11615798_1.

[100] Tarek Mhamdi, Osman Hasan, and Sofiène Tahar. On the formalization of the Lebesgue integration theory in HOL. In Matt Kaufmann and Lawrence C. Paulson, editors, *Interactive Theorem Proving: First International Conference, ITP 2010, Edinburgh, UK, July 11-14, 2010. Proceedings*, pages 387–402, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-14052-5. doi: 10.1007/978-3-642-14052-5_27.

[101] J Strother Moore, Tom Lynch, and Matt Kaufmann. A mechanically checked proof of the correctness of the kernel of the AMD5k86 floating point division algorithm.

Unpublished, 1996. URL http://www.cs.utexas.edu/~moore/publications/divide_paper.pdf.

[102] Ramon E Moore, R Baker Kearfott, and Michael J Cloud. *Introduction to interval analysis*. SIAM, 2009.

[103] C Morales, M Pacifico, and Enrique Pujals. Singular hyperbolic systems. *Proceedings of the American Mathematical Society*, 127(11):3393–3401, 1999.

[104] César Muñoz and David Lester. Real number calculations and theorem proving. In J. Hurd and T. Melham, editors, *Theorem Proving in Higher Order Logics (TPHOLs 2005)*, volume 3603 of *LNCS*, pages 195–210, 2005.

[105] Adam Naumowicz and Artur Korniłowicz. A brief overview of Mizar. In Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel, editors, *Theorem Proving in Higher Order Logics: 22nd International Conference, TPHOLs 2009, Munich, Germany, August 17-20, 2009. Proceedings*, pages 67–72, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. ISBN 978-3-642-03359-9. doi: 10.1007/978-3-642-03359-9_5.

[106] Nedialko S. Nedialkov. Implementing a rigorous ODE solver through literate programming. In Andreas Rauh and Ekaterina Auer, editors, *Modeling, Design, and Simulation of Systems with Uncertainties*, pages 3–19, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 978-3-642-15956-5. doi: 10.1007/978-3-642-15956-5_1.

[107] Arnold Neumaier. Taylor forms—use and limits. *Reliable Computing*, 9(1):43–79, Feb 2003. ISSN 1573-1340. doi: 10.1023/A:1023061927787. URL https://doi.org/10.1023/A:1023061927787.

[108] Stefan Berghofer Tobias Nipkow. Random testing in Isabelle/HOL. In J. Cuellar and Z. Liu, editors, *Software Engineering and Formal Methods (SEFM 2004)*, pages 230–239. IEEE Computer Society, 2004.

[109] Tobias Nipkow. Order-sorted polymorphism in Isabelle. *Logical environments*, pages 164–188, 1993.

[110] Tobias Nipkow and Gerwin Klein. *Concrete Semantics: With Isabelle/HOL*. Springer, 2014.

[111] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL: A proof assistant for higher-order logic*. LNCS. Springer, 2002.

[112] Steven Obua. *Flyspeck II: The Basic Linear Programs*. PhD thesis, Technische Universität München, München, 2008.

[113] Russell O'Connor and Bas Spitters. A computer verified, monadic, functional implementation of the integral. *Theoretical Computer Science*, 411(37):3386–3402, 2010. doi: 10.1016/j.tcs.2010.05.031.

[114] Russell O'Connor. Certified exact transcendental real number computation in Coq. In *International Conference on Theorem Proving in Higher Order Logics*, pages 246–261. Springer, 2008. ISBN 978-3-540-71065-3. doi: 10.1007/978-3-540-71067-7_21.

[115] Karol Pąk. Topological manifolds. *Formalized Mathematics*, 22(2):179–186, 2014.

[116] Lawrence C. Paulson. The foundation of a generic theorem prover. *Journal of Automated Reasoning*, 5(3):363–397, 1989. ISSN 1573-0670. doi: 10.1007/BF00248324. URL `http://dx.doi.org/10.1007/BF00248324`.

[117] Lawrence Perko. *Differential Equations and Dynamical Systems*. Springer, 2001. ISBN 978-1-4613-0003-8. doi: 10.1007/978-1-4613-0003-8.

[118] David Pichardie and Yves Bertot. Formalizing convex hull algorithms. In RichardJ. Boulton and PaulB. Jackson, editors, *Theorem Proving in Higher Order Logics*, volume 2152 of *Lecture Notes in Computer Science*, pages 346–361. Springer Berlin Heidelberg, 2001. ISBN 978-3-540-42525-0. doi: 10.1007/3-540-44755-5_24.

[119] Andre Platzer. The complete proof theory of hybrid systems. In *Proceedings of the 2012 27th Annual IEEE/ACM Symposium on Logic in Computer Science*, LICS '12, pages 541–550, Washington, DC, USA, 2012. IEEE Computer Society. ISBN 978-0-7695-4769-5. doi: 10.1109/LICS.2012.64.

[120] Marco Riccardi. The definition of topological manifolds. *Formalized Mathematics*, 19(1): 41–44, 2011.

[121] Albert Rizaldi, Fabian Immler, and Matthias Althoff. A formally verified checker of the safe distance traffic rules for autonomous vehicles. In Sanjai Rayadurgam and Oksana Tkachuk, editors, *NASA Formal Methods: 8th International Symposium, NFM 2016, Minneapolis, MN, USA, June 7-9, 2016, Proceedings*, pages 175–190, Cham, 2016. Springer International Publishing. ISBN 978-3-319-40648-0. URL `https://doi.org/10.1007/978-3-319-40648-0_14`.

[122] Albert Rizaldi, Jonas Keinholz, Monika Huber, Jochen Feldle, Fabian Immler, Matthias Althoff, Eric Hilgendorf, and Tobias Nipkow. Formalising and monitoring traffic rules for autonomous vehicles in Isabelle/HOL. In Nadia Polikarpova and Steve Schneider, editors, *Integrated Formal Methods: 13th International Conference, IFM 2017, Turin, Italy, September 20-22, 2017, Proceedings*, pages 50–66, Cham, 2017. Springer International Publishing. ISBN 978-3-319-66845-1. URL `https://doi.org/10.1007/978-3-319-66845-1_4`.

[123] Clark Robinson. *Dynamical Systems - Stability, Symbolic Dynamics, and Chaos*. CRC Press, 1999. ISBN 0-8493-8493-1. doi: 10.1007/978-1-4613-0003-8.

[124] Siegfried M Rump. Verification methods: Rigorous results using floating-point arithmetic. *Acta Numerica*, 19:287–449, 2010.

[125] Siegfried M. Rump and Masahide Kashiwagi. Implementation and improvements of affine arithmetic. *Nonlinear Theory and Its Applications, IEICE*, 6(3):341–359, 2015. doi: 10.1587/nolta.6.341.

[126] Zhiping Shi, Weiqing Gu, Xiaojuan Li, Yong Guan, Shiwei Ye, Jie Zhang, and Hongxing Wei. The gauge integral theory in hol4. *Journal of Applied Mathematics*, 2013, 2013.

[127] Steve Smale. Mathematical problems for the next century. *The Mathematical Intelligencer*, 20(2):7–15, 1998. ISSN 0343-6993. doi: 10.1007/BF03025291.

[128] Alexey Solovyev and Thomas C Hales. Formal verification of nonlinear inequalities with taylor interval approximations. In *NASA Formal Methods Symposium*, pages 383–397. Springer, 2013.

[129] Colin Sparrow. *The Lorenz Equations: Bifurcations, Chaos, and Strange Attractors*. Number 41 in Applied Mathematical Sciences. Springer, 1982. ISBN 978-0-387-90775-8. doi: 10.1007/978-1-4612-5767-7.

[130] Bas Spitters and Eelis van der Weegen. Type classes for mathematics in type theory. *MSCS, 'Interactive theorem proving and the form. of math.'*, 21:1–31, 2011. doi: 10.1017/S0960129511000119.

[131] Ian Stewart. Mathematics: The Lorenz attractor exists. *Nature*, 406(6799):948–949, 2000.

[132] Walid Taha, Adam Duracz, Yingfu Zeng, Kevin Atkinson, Ferenc A Bartha, Paul Brauner, Jan Duracz, Fei Xu, Robert Cartwright, Michal Konečnỳ, et al. Acumen: An open-source testbed for cyber-physical systems research. In *Internet of Things. IoT Infrastructures: Second International Summit, IoT 360° 2015, Rome, Italy, October 27-29, 2015. Revised Selected Papers, Part I*, pages 118–130. Springer, 2016.

[133] Christoph Traut. Taylor models in Isabelle/HOL. Master's thesis, TU München, Dec 2015.

[134] Dmitriy Traytel, Stefan Berghofer, and Tobias Nipkow. Extending hindley-milner type inference with coercive structural subtyping. In Hongseok Yang, editor, *Programming Languages and Systems: 9th Asian Symposium, APLAS 2011, Kenting, Taiwan, December 5-7, 2011. Proceedings*, pages 89–104, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 978-3-642-25318-8. doi: 10.1007/978-3-642-25318-8_10.

[135] Andrzej Trybulec. Tarski Grothendieck set theory. *Journal of Formalized Mathematics*, 1, 1989.

[136] Warwick Tucker. My thesis: The Lorenz attractor exists. `http://www2.math.uu.se/~warwick/main/pre_thesis.html`, 1998.

[137] Warwick Tucker. The Lorenz attractor exists. *Comptes Rendus de l'Académie des Sciences-Series I-Mathematics*, 328(12):1197–1202, 1999.

[138] Warwick Tucker. A rigorous ODE solver and Smale's 14th problem. *Foundations of Computational Mathematics*, 2(1):53–117, 2002. ISSN 1615-3375.

[139] Warwick Tucker. Robust normal forms for saddles of analytic vector fields. *Nonlinearity*, 17(5):1965, 2004.

[140] Marcelo Viana. What's new on Lorenz strange attractors? *The Mathematical Intelligencer*, 22(3):6–19, 2000.

[141] Wolfgang Walter. *Ordinary Differential Equations*. Springer, 1 edition, 1998.

[142] Markus Wenzel. Type classes and overloading in higher-order logic. In Elsa L. Gunter and Amy Felty, editors, *Theorem Proving in Higher Order Logics: 10th International Conference, TPHOLs '97 Murray Hill, NJ, USA, August 19–22, 1997 Proceedings*, pages 307–322, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg. ISBN 978-3-540-69526-4. URL `http://dx.doi.org/10.1007/BFb0028402`.

[143] Alfred North Whitehead and Bertrand Russell. *Principia mathematica*, volume 2. Cambridge University Press, 1912.

[144] Robert F Williams. The structure of Lorenz attractors. *Publ. Math. IHES*, 50:73–99, 1979.

[145] Lei Yu. A formal model of ieee floating point arithmetic. *Archive of Formal Proofs*, July 2013. ISSN 2150-914x. `http://isa-afp.org/entries/IEEE_Floating_Point.shtml`, Formal proof development.

[146] Roland Zumkeller. Formal global optimisation with Taylor models. In Ulrich Furbach and Natarajan Shankar, editors, *Automated Reasoning*, volume 4130 of *LNCS*, pages 408–422. Springer Berlin Heidelberg, 2006. ISBN 978-3-540-37187-8. doi: 10.1007/11814771_35.