



UNIVERSITÀ
DEGLI STUDI
FIRENZE



UNIVERSITÀ
DEGLI STUDI
DI PERUGIA

[iNSAM]
Istituto Nazionale
di Alta Matematica

Università di Firenze, Università di Perugia, INdAM consorziate nel CIAFM

**DOTTORATO DI RICERCA
IN MATEMATICA, INFORMATICA, STATISTICA
CURRICULUM IN INFORMATICA
CICLO XXXVI**

Sede amministrativa Università degli Studi di Firenze
Coordinatore Prof. Matteo Focardi

**Formal methods for dynamical
systems: invariants, reachability,
inference**

Settore Scientifico Disciplinare INF/01

Dottoranda
Luisa Collodi

Supervisore
Prof. Michele Boreale

Coordinatore
Prof. Matteo Focardi

Abstract

This thesis is centered on Formal Methods for dynamical systems. In particular, we focus on continuous dynamical systems specified by systems of ordinary differential equations (ODEs).

We first consider the task of finding invariants, which are useful in the Safety analysis of systems. We focus on a particular type of invariants, conservation laws. We present a method to compute all the polynomial conservation laws up to a specified order of derivatives and degree for systems of partial differential equations (PDEs). The method is based only on the definition of conservation law and on linear algebraic computations.

We then study conditions and methods to compute reduced linear approximations of nonlinear ordinary differential equations (ODEs) that are accurate also non locally, and present an algorithm that, given an initial set and a finite time horizon, builds a tight overapproximation of the set of its reachable states (reachset) at specified times, relying on Carleman linearization and Krylov projection.

Next, we consider stream differential equations (SDEs), a generalization of differential equations in the framework of *stream calculus*, that is centered on the manipulations of infinite sequences of scalars from a field, or streams. Working within this framework, we provide a method to find all polynomial invariants that fit in a user specified polynomial template, for a given SDE-based initial value problem. We also establish a stream version of Implicit Function Theorem (IFT) for systems of stream polynomial equations, and show the advantages of the stream IFT with respect to the classical IFT from a computational point of view.

We then consider parametric ODEs systems. We introduce an algorithm to compute guaranteed estimates of posterior expectations for the parameters from given observations. We work in a general model, where the relation between observations and parameters is a function, with additive noise; the function can be, in particular, the solution of a ODE. The algorithm relies on a combination of methods based on uncertain probability, Interval Arithmetic and Monte Carlo simulation. Guarantees come in the form of confidence intervals. Finally, we consider a more general and flexible approach to parameter inference based on Probabilistic Programming. In particular, we present an action-based probabilistic programming language equipped with a small-step operational semantics, where discrete and continuous distributions can be freely mixed and unbounded loops are allowed. Our semantics directly leads to an exact sampling algorithm that can be efficiently SIMD-parallelized.

Acknowledgements

I want to express my deepest gratitude to my supervisor Prof. Michele Boreale for his enthusiasm, constant presence and precious guidance. Thank you for introducing me to the fascinating world of research in Theoretical Computer Science, and for the time and effort spent in teaching me since the beginning of my university studies.

Many thanks to Prof. Daniele Gorla for working together and for the many interesting discussions that provided an important contribution to this thesis.

I am particularly grateful to the thesis reviewers, Prof. Ezio Bartocci and Prof. Andrea Vandin, for all the stimulating observations and suggestions that have greatly helped me to improve the thesis.

I want to express my great gratitude to Dr. Cecilia Viscardi and Prof. Fabio Corradi for their support throughout my PhD journey and for showing me the pleasure of sharing knowledge from different backgrounds.

Special thanks also to Prof. Maria Cecilia Verri, Prof. Donatella Merlini and Prof. Andrea Marino for the support and all the teachings they have given me since I was a Bachelor student.

Great thanks to my parents for believing in me unconditionally. Their support in all my choices and their help from every point of view has been essential. A special thank you to my cousin Paolo for his affection and for always standing up for me. A special mention to my grandfathers, who would have been the proudest people of me.

Many thanks also to Prof. Pia Monti, who made me enjoy for the first time the beauty of study and the power of knowledge that has no boundaries.

Last but not least, I want to thank all my friends and colleagues for the time spent together and for sharing doubts and suggestions.

Contents

| | | |
|----------|---|-----------|
| 1 | Introduction | 15 |
| 1.1 | The landscape of formal methods for dynamical systems . . . | 15 |
| 1.1.1 | Dynamical Systems and Ordinary Differential Equations | 15 |
| 1.1.2 | Safety and Reachability | 17 |
| 1.1.3 | Coalgebras and stream differential equations | 17 |
| 1.1.4 | Parameter inference | 18 |
| 1.2 | Outline of the thesis | 18 |
| 1.2.1 | Topics and chapters | 18 |
| 1.2.2 | Papers and publications | 19 |
| 1.3 | Invariants and conservation laws | 21 |
| 1.3.1 | State of the art | 21 |
| 1.3.2 | Our contribution | 22 |
| 1.4 | Linearization and reachability | 24 |
| 1.4.1 | State of the art | 24 |
| 1.4.2 | Our contribution | 25 |
| 1.5 | SDEs, a (co)algebraic approach | 26 |
| 1.5.1 | State of the art | 26 |
| 1.5.2 | Our contribution | 26 |
| 1.6 | Inference: Bayesian parameter estimation | 27 |
| 1.6.1 | State of the art | 27 |
| 1.6.2 | Our contribution | 28 |
| 1.7 | Inference, a more general view: Probabilistic Programming . | 29 |
| 1.7.1 | State of the art | 29 |
| 1.7.2 | Our contribution | 30 |
| 2 | Invariants and conservation laws | 31 |
| 2.1 | Overview | 31 |
| 2.2 | Preliminaries | 33 |
| 2.3 | The method | 34 |
| 2.4 | Experiments | 40 |
| 2.4.1 | Using the same set of indeterminates | 42 |
| 2.4.2 | Using different sets of indeterminates | 43 |
| 2.5 | Analyticity | 44 |
| 2.6 | Conclusion | 45 |

| | | |
|----------|---|------------|
| 3 | Linearization and reachability | 49 |
| 3.1 | Overview | 49 |
| 3.2 | Preliminaries | 53 |
| 3.3 | Carleman linearization | 53 |
| 3.4 | Dimension reduction via Krylov projection | 54 |
| 3.5 | Behaviour of the global error | 58 |
| 3.6 | Relation with the Koopman approach | 59 |
| 3.6.1 | Koopman spectral decomposition | 59 |
| 3.6.2 | Approximate Koopman decomposition | 60 |
| 3.6.3 | Dimension reduction of approximate Koopman decomposition | 61 |
| 3.7 | Application to reachability analysis | 63 |
| 3.8 | Experiments | 66 |
| 3.8.1 | Graphical comparisons | 67 |
| 3.8.2 | Comparison with [72] and [10] | 68 |
| 3.8.3 | Reachsets: comparison with Flow* and CORA | 69 |
| 3.9 | Conclusion | 71 |
| 4 | SDEs, a (co)algebraic approach | 73 |
| 4.1 | Overview | 73 |
| 4.2 | Preliminaries | 76 |
| 4.2.1 | Polynomials and differential equations | 76 |
| 4.2.2 | Streams | 77 |
| 4.2.3 | Coalgebras, SDEs and bisimulation | 78 |
| 4.3 | (Co)algebraic semantics of polynomials and differential equations | 80 |
| 4.4 | Deciding stream equality | 86 |
| 4.4.1 | The algorithm | 86 |
| 4.4.2 | A fixed-point theoretic perspective | 90 |
| 4.5 | Finding polynomial identities | 91 |
| 4.6 | An implicit function theorem for the stream calculus | 97 |
| 4.7 | Relations with the classical IFT | 104 |
| 4.8 | An extended example: three-coloured trees | 106 |
| 4.9 | Classical vs. stream IFT: computational aspects | 108 |
| 4.10 | Conclusion | 110 |
| 5 | Inference: Bayesian parameter estimation | 111 |
| 5.1 | Overview | 111 |
| 5.2 | Preliminaries | 113 |
| 5.2.1 | Framework and problem statement | 113 |
| 5.2.2 | Interval arithmetic, coverings, set inversion | 115 |
| 5.2.3 | Discretized ODEs and neural networks | 117 |
| 5.3 | The core algorithm \mathcal{A} | 119 |
| 5.4 | Confidence intervals for posterior moments | 124 |
| 5.5 | Optimal allocation of computational resources | 125 |
| 5.6 | Experiments | 126 |
| 5.6.1 | Discretized ODEs | 126 |
| 5.6.2 | Feature relevance in neural network classifiers | 129 |

| | | |
|----------|--|------------|
| 5.7 | Conclusion | 130 |
| 6 | Inference, a more general view: Probabilistic Programming | 133 |
| 6.1 | Overview | 133 |
| 6.2 | Preliminaries | 135 |
| 6.3 | Probabilistic programs | 137 |
| 6.4 | Observable semantics | 140 |
| 6.5 | Inference via vectorized Monte Carlo sampling | 143 |
| 6.6 | Implementation and evaluation | 146 |
| 6.6.1 | TSI: a TensorFlow based implementation | 147 |
| 6.6.2 | Evaluation of TSI | 148 |
| 6.7 | Conclusion | 152 |
| 7 | Conclusion | 153 |
| 7.1 | Future work | 154 |
| | Bibliography | 155 |
| A | Proofs of Chapter 2 | 169 |
| A.1 | Proof of Corollary 1 | 169 |
| A.2 | Proof of Proposition 2 | 170 |
| B | Proofs and additional details of Chapter 3 | 173 |
| B.1 | On-the-fly computation of Arnoldi iteration | 173 |
| B.2 | Proofs | 174 |
| B.3 | Further details on experiments | 176 |
| C | Proofs and additional details of Chapter 4 | 179 |
| C.1 | Proof | 179 |
| C.2 | Three-coloured trees example: details | 183 |
| D | Proofs and additional details of Chapter 6 | 185 |
| D.1 | Proofs | 185 |
| D.2 | Importance Sampling | 190 |
| D.3 | Models | 195 |

List of Figures

| | | |
|-----|--|-----|
| 3.1 | Exact and approximate solution for VdP model. | 57 |
| 3.2 | Advection and inflation of a polytope. | 63 |
| 3.3 | Individual trajectories: comparison among different methods. | 67 |
| 3.4 | Exact and approximate solutions for LL over a time horizon of $T = 1$ | 68 |
| 3.5 | Graphical comparison with [72] on the VdP example. | 69 |
| 3.6 | Reachsets computed with CORA, Flow* and ckr. | 71 |
| 4.1 | A weighted automaton. | 97 |
| 4.2 | Execution time as the number of calculated coefficients varies. | 110 |
| 5.1 | Confidence band for the posterior CDF of the simple ball model. | 126 |
| 5.2 | Confidence bands for the marginal CDFs for Fitzhugh- Nagumo model. | 128 |
| 5.3 | Example of relevance map. | 130 |
| 6.1 | Vectorized Monte-Carlo algorithm for expected value estima- tion. | 145 |
| 6.2 | Outline of the translation into Python with TensorFlow. | 150 |

List of Tables

| | | |
|-----|---|-----|
| 2.1 | PDEs considered in the experiments. | 41 |
| 2.2 | Comparison between GEM and POLYCONS with the same in-determinates. | 46 |
| 2.3 | Comparison between GEM and POLYCONS using different in-determinates. | 47 |
| 3.1 | Comparison of Flow*, CORA and CKR. | 70 |
| 5.1 | Comparison of \mathcal{A} , DSA and MCMC on the benchmarks from [52]. | 127 |
| 6.1 | Vectorized Comparison of TSI, webPPL and R2. | 151 |
| 6.2 | Vectorized Confidence intervals and corresponding execution time. | 152 |

Chapter 1

Introduction

1.1 The landscape of formal methods for dynamical systems

1.1.1 Dynamical Systems and Ordinary Differential Equations

A *dynamical system* is a mathematical model, often specified by means of differential equations, that describes how the state of a phenomenon of interest evolves over time, in particular its *trajectory* in a given state space. Differential equations mathematically represent the relation among all the variables involved in the system, including an independent variable t typically representing time, and can be used to analyze and simulate the behaviour of dynamical systems.

In Physics, Newtonian mechanics is most naturally formulated in terms of *ordinary differential equations* (ODE) and dynamical systems. Over time, the range of applications of dynamical systems has extended to Engineering, Chemistry, Biology, Ecology,... For example, systems of differential equations can be used to model biodiesel production [149], to reproduce important characteristics of the human ventricular tissue such as action potential or conduction velocity [37, 88], or to represent connectivity among human brain regions [185]. The analysis of dynamical systems is a challenging task that spans across various disciplines. In Computer Science, over the last three decades, *Formal Methods* for the analysis of dynamical systems have been extensively studied and developed, and represent now a very active area of research. Indeed, in a large number of real-world applications, hardware and software artifacts, while running, must interact with external phenomena whose behaviour can be described as a dynamical system. The analysis of such systems becomes then an integral part of the design and analysis of the software/hardware artifact as a whole. This is particularly evident in such fields as autonomous driving, digital controllers of trains and aircrafts, and more generally embedded and cyber-physical systems [142, 3]. The analysis of dynamical systems conducted with Formal Methods lies at the intersection between Computer Science and Control Theory, and is especially related to establishing *Safety* of a system. Broadly speaking, this means mathematically certifying that the system will never enter a predefined region of the state-space that is considered as dangerous, such as: an aircraft with an excessive angle of attack that might stall [181];

two autonomous vehicles driving too close to each other or too fast; and so on; see [166].

Dynamical systems can be classified as *discrete*, *continuous* or *hybrid*, depending on whether one models time as a discrete or continuous variable, or in a mixed discrete-continuous fashion. The evolution of *discrete systems* can be described by a sequence of state changes taking place at a discrete set of time instants. Any ordinary program running on a computer can be seen as a discrete system: in fact, each program instruction, executed at prescribed time, corresponds to a discrete update of the variables of the program, hence of its state. Mathematically, difference equations and discrete-time Markov chains can be seen as models of specific types of discrete dynamical systems. In *continuous systems* the state evolves continuously over time, and often represents physical quantities, such as the velocity or the acceleration of a car, the temperature of a room, the position of a rocket, and so on. ODEs are the main descriptive formalism of continuous systems. Finally, *hybrid systems* [170, 3] combine discrete and continuous behaviour: the law of continuous change (ODE) can switch among a discrete set of *modes*, with switching happening at discrete times. For instance, the system that describes the motion of a bouncing ball will switch between two modes: one for the ball falling, and one for the ball raising after a bounce. Driver-assistance systems, or the autopilot of an aircraft are examples of more complex systems that can be modeled as hybrid systems. Mathematically, hybrid systems can be modelled using e.g. *hybrid automata* [92], finite state machine with a set of continuous variables whose law of change depends on the discrete *location* (state) of the hybrid automaton.

In this thesis we mainly focus on continuous systems, defined on a real-valued state space. Continuous systems represent the core of the problem, in fact, hybrid systems are built connecting multiple continuous systems; see e.g. the discussion in [46]. Continuous systems are described by differential equations. In detail, we will mainly consider systems that can be defined by finite systems of first order ODEs [104], written as follows:

$$\begin{aligned}\dot{x}_1 &= f_1(t, x_1, \dots, x_n) \\ \dot{x}_2 &= f_2(t, x_1, \dots, x_n) \\ &\vdots \\ \dot{x}_n &= f_n(t, x_1, \dots, x_n).\end{aligned}\tag{1.1}$$

Here, x_i, \dot{x}_i are abbreviations for $x_i(t), \dot{x}_i(t)$, with \dot{x}_i denoting the derivative of variable x_i with respect to the time variable t . A compact vectorial notation will be often used:

$$\dot{x} = f(t, x)\tag{1.2}$$

with $x = [x_1, x_2, \dots, x_n]^T$ and $f = [f_1(t, x), f_2(t, x), \dots, f_n(t, x)]^T$. We refer to (1.2) as the *state equation* of the system, and to x as the vector of the *state variables*. We consider mainly first-order systems, in which only first derivatives of the variables appear. In terms of expressiveness, this is not a limitation, as higher-order systems can be reduced to first-order ones by introducing new variables to represent higher order derivatives.

1.1.2 Safety and Reachability

As many applications are safety-critical, a central point in the analysis of both continuous and hybrid dynamical systems is to verify that “nothing bad will ever happen”, that is proving a Safety property of interest. In other words, it is important to be able to prove that, for a given system, a set of states identified as unsafe will never be reached by system trajectories. For the purpose of proving that a system is unsafe, a scenario with bad behaviour is a sufficient evidence. On the contrary, to prove that a system is safe, stronger evidence than some successful scenarios is necessary. The most direct way to prove Safety of a continuous system consists in identifying *exactly* the set of states reachable by the system, possibly over a finite time horizon of interest, and then checking that it does not contain any unsafe state. However, for reasonably expressive models, in fact even for *linear* systems, this can be a computationally intractable task [46]. Therefore, several techniques to provide tight *overapproximations* of the set of reachable states from a given set of initial states have been developed, collectively described as *reachability analysis* [176, 6, 47]. Proving that an overapproximation of the set of reachable states does not contain unsafe states is sufficient to establish the safety of the system. In Formal Methods, specific approaches rely on dynamic logics [142, 143]; others try to assess safety of a system directly simulating a finite set of its trajectories, and then trying to rigorously extrapolate all the others [65, 79]. Other methods [106, 147, 146] use special functions called *Barrier Certificates* in order to separate the unsafe region from all the possible trajectories starting from a given set of initial conditions. Other methods generalize barrier certificates by means of (*algebraic*) *invariants* [78, 29].

When considering systems that model real-world phenomena, we have to deal with complicated nonlinear ODEs. Moreover the dimension of the space tends to be large, often making the direct application of formal methods analysis practically unfeasible (curse of dimensionality). Thus, methods that reduce the size (number of variables) of the system, while preserving the important aspects of the original dynamics, play a key role in this field. Many methods have been proposed. A few rely on differential elimination and abstract interpretation [143, 59, 142]. In [28, 27], dimensionality reduction is achieved via projection onto a space of lower dimension, and then by working in the embedding coordinate space. Instead, in [41, 42, 39, 177, 40] the ODEs reduction problem is recast into finding an appropriate equivalence relation over ODEs variables. On the same line, in [98], differential bisimulation, an equivalence relation that captures symmetries in the ODE semantics, is shown to lead directly to a partition-refinement algorithm to compute the coarsest ODEs aggregation.

1.1.3 Coalgebras and stream differential equations

A generalization of systems of differential equations is given by systems of *stream* differential equations (SDEs) in the realm of *coalgebra* [154, 155]. In fact, depending on the semantics of the polynomial product one adopts in the definition, different interpretations can be given to differential equations: considering the shuffle product leads us to ordinary differential equations, but if we consider convolution we get stream differential equations. Relations among streams, differential equations, and polynomials are explored in depth in [26] and have several important applications, for

instance to obtain closed forms of algebraic generating functions, or to solve ordinary differential equations [26].

1.1.4 Parameter inference

Often, systems of differential equations involve parameters whose value is not directly measurable, but whose selection in the modeling phase is critical. It is important for applications to assign the correct values to these parameters, or at least to identify a plausible range of values for them. Estimates of parameters values can be obtained starting from real-world observed data and relying on statistical techniques, such as Maximum Likelihood Estimation or Bayesian Inference approaches. We will exclusively follow the last approach, which requires to postulate a prior distribution on the set of possible values. Monte-Carlo methods such as Markov Chain Monte Carlo [125] and Sequential Monte Carlo [63] have been applied to carry out Bayesian Inference in practice. Also hybrid approaches such as [52, 62] have been investigated. Unfortunately, it is in general very hard to get adequate formal, or even statistical, guarantees for the obtained estimates. Recently, combining Bayesian inference and programming languages, a new programming paradigm has emerged: Probabilistic Programming (PP) [85]. Probabilistic programs can be interpreted as models conditioned on observation data, and thus can be used to make inference and predictions. Applications to dynamical systems are described in [85].

1.2 Outline of the thesis

1.2.1 Topics and chapters

A general goal of this thesis is to develop new methods to analyze and predict the behaviour of dynamical systems. We will pursue this goal by adopting a variety of different points of view and combining different approaches, ranging from algebraic geometry, to coalgebra, to Bayesian inference. In particular, we will present methods that can be used for reasoning on safety properties, to approximate reachable sets, and to provide guaranteed estimates in the context of parameter inference and of probabilistic programming. Accordingly, the central part of the thesis will consist of the following five chapters:

- Chapter 2: Invariants and conservation laws.
- Chapter 3: Linearization and reachability.
- Chapter 4: SDEs, a (co)algebraic approach.
- Chapter 5: Inference: Bayesian parameter estimation.
- Chapter 6: Inference, a more general view: Probabilistic Programming.

In the final Chapter 7 we will draw some concluding remarks. A number of proofs and additional technical details have been confined to separate appendices A—D. The following sections of this chapter give a detailed overview of the central chapters 2—6

of the thesis. In each section, we will first describe the State of the Art in the considered subject and then present our contribution.

1.2.2 Papers and publications

This thesis is based on work developed in the papers listed below. We separate those that have already been published or accepted from those that are currently under revision. For each work, we also provide an abstract.

Works published or accepted for publication

- [1] M. Boreale and L. Collodi. A linear algebraic method to compute polynomial PDE conservation laws. *Journal of Symbolic Computation*, 108:55-72, 2022.

Abstract. We present a method to compute polynomial conservation laws for systems of partial differential equations (PDEs). The method only relies on linear algebraic computations and is complete, in the sense it can find a basis for all polynomial fluxes that yield conservation laws, up to a specified order of derivatives and degree. We compare our method to a state-of-the-art algorithm based on the direct approach on a few PDE systems drawn from mathematical physics.

- [2] M. Boreale and L. Collodi. Linearization, model reduction and reachability in nonlinear odes. In *Reachability Problems 2022*, volume 13608 of *Lecture Notes in Computer Science*, pages 49-66. Springer, 2022.

Abstract. In the analysis of nonlinear ordinary differential equations, linear and Taylor approximations are fundamental tools. Such approximations are generally accurate only in a local sense, that is near a given expansion point in space or time. We study conditions and methods to compute linear approximations of nonlinear ODEs that are accurate also non locally. Relying on Carleman linearization and Krylov projection, our method yields a small, hence tractable linear system that is shown to produce accurate approximate solutions, under suitable stability conditions. In the general, possibly non stable case, we provide an algorithm that, given an initial set and a finite time horizon, builds a tight overapproximation of the reachable states at specified times. Experiments conducted with a proof-of-concept implementation have given encouraging results. We also establish a formal relation between our approach and Koopman approximation, a well-known framework for the analysis of nonlinear systems.

- [3] M. Boreale and L. Collodi. Bayesian parameter estimation with guarantees via interval analysis and simulation. In *VMCAI 2023*, volume 13881 of *Lecture Notes in Computer Science*, pages 106-128, Springer, 2023.

Abstract. We give a method to compute guaranteed estimates of Bayesian a posteriori distributions in a model where the relation between the observation y and the parameters θ is a function, possibly involving additive noise parameters ψ , say $y = f(\theta) + h(\psi)$. This model covers the case of (noisy) ODE parameters

estimation and the case when f is computed by a neural network. Applying a combination of methods based on uncertain probability (P-boxes), Interval Arithmetic (IA) and Monte Carlo (MC) simulation, we design an efficient randomized algorithm that returns guaranteed estimates of the posterior CDF of the parameters θ , and moments thereof, given that the observation y lies in a (small) rectangle. Guarantees come in the form of confidence intervals for the CDF values and its moments. Comparison with state-of-the-art approaches on ODES benchmarks shows significant improvement in terms of efficiency and accuracy.

- [4] M. Boreale, L. Collodi, D. Gorla. Products, polynomials and differential equations in the stream calculus. *ACM Transactions on Computational Logic*, volume 25(1), pp 1–26, 2024.

Abstract. We study connections among polynomials, differential equations and streams over a field \mathbb{K} , in terms of algebra and coalgebra. We first introduce the class of (F, G) -products on streams, those where the stream derivative of a product can be expressed as a polynomial function of the streams and their derivatives. Our first result is that, for every (F, G) -product, there is a canonical way to construct a transition function on polynomials such that the resulting unique final coalgebra morphism from polynomials into streams is the (unique) commutative \mathbb{K} -algebra homomorphism – and vice versa. This implies that one can algebraically reason on streams via their polynomial representation. We apply this result to obtain an algebraic-geometric decision algorithm for polynomial stream equivalence, for an underlying generic (F, G) -product. Finally, we extend this algorithm to solve a more general problem: finding all valid polynomial equalities that fit in a user-specified polynomial template.

- [5] M. Boreale and L. Collodi. Guaranteed inference for probabilistic programs: a parallelisable, small-step operational approach. In *VMCAI 2024*, volume 14500 of *Lecture Notes in Computer Science*, pages 141-162, Springer, 2024.

Abstract. We put forward an approach to the semantics of probabilistic programs centered on an action-based language equipped with a small-step operational semantics. This approach provides benefits in terms of both clarity and effective implementation. Discrete and continuous distributions can be freely mixed, unbounded loops are allowed. In measure-theoretic terms, a product of Markov kernels is used to formalize the small-step operational semantics. This approach directly leads to an exact sampling algorithm that can be efficiently SIMD-parallelized. An observational semantics is also introduced based on a probability space of infinite sequences, along with a finite approximation theorem. Preliminary experiments with a proof-of-concept implementation based on TensorFlow show that our approach compares favourably to state-of-the-art tools for probabilistic programming and inference.

- [6] M. Boreale, L. Collodi, D. Gorla. An implicit function theorem for the stream calculus. Accepted for publication in *Logical Methods in Computer Science*, arXiv:2303.11876v3[cs.LO].

Abstract. In the context of the stream calculus, we present an Implicit Function Theorem (IFT) for polynomial systems, and discuss its relations with the classical IFT from calculus. In particular, we demonstrate the advantages of the stream IFT from a computational point of view, and provide a few example applications where its use turns out to be valuable.

Works under revision

- [7] M. Boreale and L. Collodi. Linearization, model reduction and reachability in nonlinear odes. Paper submitted to *Formal Methods in System Design* (revision requested).

Abstract. An expanded version of the conference paper [2].

Other work

The material in the following work has not been included in the thesis.

- [8] MC. Viscardi, S. Monchetti, L. Collodi, G. Bartoli, M. Betti, M. Boreale and F. Corradi. Approximate Bayesian Computation for Probabilistic Damage Identification. In *SIS 2023 International Meeting, SEAS-IN* volume, pages 544-550, PEARSON, 2023.

Abstract. Damage identification analyses are fundamental to guarantee the safety of civil structures. They are often formalised as inverse problems whose solution ignores any source of uncertainty that could be accounted for by using appropriate statistical models. However, these models often exhibit an intractable likelihood function. We propose quantifying uncertainty through a fully Bayesian approach based on Approximate Bayesian Computation (ABC), a class of methods that overcome the evaluation of the likelihood and only require the ability to simulate from the model. Furthermore, we suggest a strategy to reduce ABC computational burden using Neural Networks. Finally, we test the method at work on a damaged beam to discuss its strengths and weaknesses.

1.3 Invariants and conservation laws

1.3.1 State of the art

The verification of Safety properties for dynamical systems is of great importance and has significant practical implications. Unfortunately, it is as crucial as complicated, because it involves dealing with the problem of computing, or at least overapproximating, the set of reachable states, usually called *reachset*, for systems defined by nonlinear differential equations.

In this regard, an indirect but very effective method to prove Safety, based on the concept of *invariant*, can be applied. An invariant is a subset of states of the considered system, with the property that any trajectory of the system that starts from or ‘touches’ the invariant will never leave it. An invariant that includes the initial set

of a given dynamical system can be considered to prove its safety, as it represents an overapproximation of its reachset. Let W an invariant for system (1.2), X_0 and X_U the initial and the unsafe set, respectively; then system (1.2) is safe if W does not contain any unsafe state, more precisely if $X_0 \subseteq W$ and $X_U \cap W = \emptyset$. Specifically, we will mainly consider algebraic invariants. An algebraic invariant is an invariant that can be described as the set of common zeros of a finite set of multivariate polynomials, i.e. as an algebraic variety. Polynomials that vanish on all solutions of the considered system are called *invariant polynomials*.

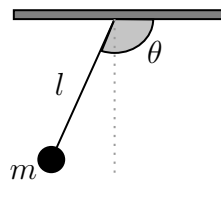
In the case of ODEs of polynomial form, a number of algorithms to compute invariants have been proposed. Some of them generate algebraic invariants by exploiting their polynomial representations. In detail, they rely on the theory of ideals over polynomial rings and on Groebner's bases [57] to reduce the linear and nonlinear invariant generation problem to a constraint solving problem [107, 164]. For instance, in [29, 27] they generate an overapproximation for the set of all the polynomials invariants that are implied by a given initial set and that also fit a given polynomial template, relying on the notion of ideal membership and on the fact that any infinite ascending chain of ideals stabilizes in a finite number of steps. In [163] the notion of invariant ideal is reformulated as the greatest fixed point of a monotone refinement operator over the lattice of ideals in a polynomial ring. They provide an algorithm to compute this monotone operator relying on the Tarski-Knaster fixed point theorem and using basic ideas from commutative algebraic geometry. However, the resulting iteration sequence does not always converge to a fixed point, then they consider a relaxation of the refinement operator using the notion of degree-bounded pseudo-ideals [56].

1.3.2 Our contribution

We focus on a particular type of invariant: *conservation laws*. A conservation law is an equation expressing that certain relations among characteristic quantities of the system are conserved over time, despite the fact that the system evolves. Conservation laws often express physical principles, such as conservation of mass, energy, momentum. For instance, consider a pendulum consisting of a ball of mass $m > 0$ hanging from a rod of length $l > 0$. Its dynamics is modeled by the ODE

$$\frac{d^2}{dt^2}\theta(t) = \frac{g}{l} \cdot \cos(\theta(t))$$

where θ is the angle from the roof to the rod, and $g > 0$ is the (constant) gravity acceleration. Setting $\omega(t) := \dot{\theta}(t)$, $x(t) := \cos(\theta(t))$ and $y(t) := \sin(\theta(t))$, we get the following first order system of ODEs:



$$\begin{cases} \dot{\theta} &= \omega \\ \dot{x} &= -y\omega \\ \dot{y} &= x\omega \\ \dot{\omega} &= \frac{g}{l}x. \end{cases} \quad (1.3)$$

In system (1.3), the mechanical energy, i.e. the sum of kinetic energy and gravitational potential energy, is conserved. This corresponds to the following invariant,

where $l(1 - y)$ is the height of the mass¹

$$mgl(1 - y) = \frac{1}{2}m(l\omega)^2.$$

This equation defines an algebraic variety V in the 4-dimensional space: the set of all (θ, x, y, ω) that make the polynomial $mgl(1 - y) - \frac{1}{2}m(l\omega)^2$ vanish. Every solution of (1.3) must stay in the variety V .

In our treatment of conservation laws, we widen considerably the scope of application of our method by considering *partial* differential equations (PDEs), that is differential equations that may involve multiple independent variables. For example, D'Alembert wave equation in one spatial dimension x can be written, in its simplest form, as the following PDE with two independent variables t, x and one dependent variable u :

$$\frac{\partial u(t, x)}{\partial t^2} = \frac{\partial u(t, x)}{\partial x^2}.$$

Here $u(t, x)$ represents the vertical displacement of a vibrating string at time t and horizontal position x . A number of methods to compute PDE's conservation laws have been proposed. Some of them are linked to the existence of *symmetries*: for example [94] is based on scaling symmetries. Others use a *direct approach* [93, 50] that aims to find multipliers whose linear combinations with the system's equations yield particular type of (divergence) expressions vanishing on its solutions. This methods does not presuppose the existence of symmetries and is more widely applicable.

Our method can be considered as a development of the direct approach, with additional guarantees of completeness. The method itself is quite simple, and ultimately boils down to imposing linear constraints on the coefficients of a generic polynomial template representing a divergence, so as to guarantee that it vanishes on the solutions of the PDE. To be a bit more specific about our contribution, let $\phi = (p_1, p_2, \dots, p_n)$ be a n -tuple of multivariate polynomials. The set of indeterminates consists here of all (dependent and independent) variables of the PDE system at hand. The *divergence* of ϕ is defined as:

$$\nabla \phi := \sum_{j=1}^n D_{x_j} p_j$$

where $D_{x_j} p_j$ is the (total) derivative of p_j with respect to x_j , taken according to the given system of PDEs. We will seek polynomial conservation laws where the divergence is an invariant polynomial, that is vanishes on all solutions of the system [140]. In this case, ϕ is called a *flux* of the system. Our method is relatively complete, in the sense that it can find all the polynomial conservation laws up to a pre-specified degree.

¹In a coordinate system where the origin coincides with the point where the rod is fixed to the roof, and the y axis is positive from the roof downward. We also assume that $\theta(0) = 0$, i.e. that the ball touches the roof initially, which makes the initial mechanical energy equal to mgl .

1.4 Linearization and reachability

1.4.1 State of the art

Many natural and computational phenomena possess an inherently nonlinear nature, which naturally leads to dynamical systems described by nonlinear differential equations. Non-linearity is essential for expressiveness. For example, in neural networks [82], one of the most well-known universal approximators, nonlinearity is crucial, and is introduced by means of activation functions. In more detail, nonlinear ODEs systems [104] are systems of ordinary differential equations in which variables and their derivatives appear in nonlinear combinations. The solution of these systems is very sensitive to initial conditions and represent complicated behaviours. Unfortunately, nonlinear ODEs are difficult to analyze, specifically it is not easy to design control systems or make prediction for them. One way to handle the complexities of nonlinear systems is through *linearization*. That is, it is sometimes possible to approximate a nonlinear ODEs system with a linear one

$$\dot{z} = Az \tag{1.4}$$

with $z \in \mathbb{R}^N$ and $A \in \mathbb{R}^{N \times N}$, and then to apply powerful techniques from linear systems theory to get an approximation of the original solution, taking into account the linearization error. Linearization techniques are relevant to reachability analysis. In fact, reachability analysis for nonlinear systems is much more complicated compared to linear systems, due to the complexity introduced by nonlinear dynamics, in particular many valuable properties, such as the superposition principle, are no longer valid [6]. Moreover, since geometric representations are closed under linear transformations, in the case of linear systems, it is possible to represent the set of initial states and also the reachset, using for instance ellipsoids, zonotopes, or polytopes, because they will be again mapped by the flow of the system to ellipsoids, zonotopes, and polytopes, respectively. The same is not possible for nonlinear systems, because reachsets cannot be computed by a linear map.

There exists a vast literature on the linearization in time or space for nonlinear systems, using Taylor series expansion or the Jacobian matrix; see e.g. [104] and references therein. However, traditional methods are generally accurate only in a local sense, that is near a given expansion point, thus the quality of their approximation decreases moving away from the expansion point. For the same purpose of linearization, also infinite-dimensional approaches can be applied: they require working on infinite dimensional spaces. Examples are Carleman linearization [109] or the Koopman operator approach [124]. In both cases, a linear, but infinite dimensional system is generated in order to approximate the nonlinear one, and then a truncation at a finite cut off is performed to make the computation feasible. In the case of the Koopman operator, the dynamics of the system is lifted from the original state space to a higher dimensional space of smooth functions called *observables*. The dynamics becomes linear in the space of observables, although the dimension of this space is infinite. Instead, Carlemann linearization essentially generates an infinite dimensional linear system, where each dimension corresponds to a monomial. Closely related to Carlemann embedding, the linearization approach presented in [167] generates linear abstractions relying on a change of basis transformations of the original nonlinear

system. It is based on a finite submatrix of the infinite matrix created by the Carleman linearization, such that the rows of this submatrix correspond to monomials whose derivative is a linear combination of monomials that belong the submatrix.

With the aim to approximate globally nonlinear ODEs for reachability analysis, in [103] Jungers and Tabuada propose a linearization technique based on *polyflows*. Polyflows are dynamical systems satisfying a nilpotency property: this basically means that the set of all Lie derivative of the system's dependent variables form a finite dimensional vector space. The method in [103] is based on building polyflows that approximate the original system, using as basis the Lie derivatives of the state variables up to some fixed order. Furthermore, in [103] also asymptotic results have been proposed, but they do not easily yield concrete bounds for the error.

In the field of reachability analysis, Cora and Flow* are two of the most widely used tools. Cora [8] considers vector and matrix set representations and implements operations on these set representations as well as reachability algorithms for various dynamical systems classes. In particular, reachability analysis for nonlinear systems in Cora relies on a mix of techniques, including linearization, in fact it is based on abstraction either to a linear system, or to a polynomial system. Instead, Flow* [47] is based on Taylor models. Taylor models [120, 46] are a combination of bounded degree polynomials obtained applying Picard iteration with a given initial condition, and bloated by an interval. They represent an effective means for computing rigorous bounds on the trajectories also for nonlinear differential equations. In detail, Taylor models are used to represent flowpipes, i.e., a set of states reachable by continuous dynamics from an initial set within a given time interval. Flow* also supports a variety of optimizations including adaptive step sizes, adaptive selection of approximation orders and the heuristic selection of template directions.

1.4.2 Our contribution

In Chapter 3, we explore conditions and methods to devise linear approximations of the solution of nonlinear systems, that can be accurate also non locally. Then we leverage our results in reachability analysis. The main step of our method is the generation of a smaller, computationally tractable, linear system from the original nonlinear system. In detail, we apply Carleman linearization to transform a given polynomial nonlinear system into an infinite linear system, then we blend Carleman linearization with Krylov orthogonal projection techniques in order to achieve dimension reduction, following the approach of [28].

Furthermore, we use the computed reduced linear system to outline a method for reachability analysis inspired by [54]. This method will generate a sequence of overapproximations of the reachsets for the original nonlinear system at specified times, over a finite time horizon, representing reachsets as polytopes. The basic idea of this approach is to perform advection for the vertices of an initial set relying on the reduced linearized system, rather than on the initial nonlinear system. In particular, the solution of the reduced system acts as an approximation of the advection function for the original system, hence we use it to propagate the initial vertices to successive time steps. Finally, the obtained polytope is inflated in order to take into account the approximation and nonlinearity errors, thus obtaining a reachset that is both tight and validated. Experiments conducted with a proof-of-concept implementation of both the

approximation scheme and the reachability algorithm have given encouraging results, especially in terms of accuracy.

1.5 SDEs, a (co)algebraic approach

1.5.1 State of the art

A stream is an infinite sequence of real numbers. The *stream calculus* [154] is based on the presence of a *final coalgebra* structure on the set of streams, arising from an operation of *stream derivative*, which simply consists in removing the first element of a stream. This basically means that streams possess a deterministic automaton structure, that has the property of being *final* in the class of coalgebras: that is, from every coalgebra C in this class, there exists a unique coalgebra morphism (preserving transitions and output values at states, in a precise sense) from C to the coalgebra of streams [155]. On the other hand, starting from a polynomial ODE, or rather an initial value problem (ivp), one can endow the ring of multivariate polynomials with an automaton structure (coalgebra), where the transition function is given by the *Lie derivative* with respect to the given ivp: see [27]. Thus, the set of streams can be seen as a final coalgebra in which to interpret the elements of the coalgebra of polynomials. In practice, polynomials can be seen as a syntax for denoting the behaviours induced by the given initial value problem, whereas the streams represent an abstract (denotational) semantics for them [27]. The obtained semantics is compositional, in particular it transforms the product between polynomials into the *shuffle product* between streams; and ODEs into *Stream Differential Equations* (SDEs), a sort of directly implementable recipes for the stepwise generation of streams [154].

In [26], different types of stream products are considered (convolution, shuffle, Hadamard,...), and connections among polynomials, streams, ODEs and SDEs have been further explored. In detail, in [26] a general class of products on streams, called (F, G) -products, is introduced: they are the class of streams where the stream derivative of a product can be expressed as a polynomial of the streams themselves and their derivatives. (F, G) -products are useful to uniformly accommodate all the different notions of products: this allows one to derive an operational semantics for polynomials appropriate for a generic type of stream product (under certain technical conditions). In fact, [26] establishes that for every (F, G) -product, there is a canonical way to construct a derivative for polynomials such that the induced unique final coalgebra morphism from polynomials into streams is the unique \mathbb{K} -algebra homomorphism, and vice versa. This leads to a full abstraction, and implies that it is possible to algebraically reason on streams, through their polynomial representations, and applying powerful techniques from algebraic geometry [57]. Relying on this result, in [26] an algorithm to decide stream equivalence is proposed, based on an algebraic-geometric procedure.

1.5.2 Our contribution

In Chapter 4 we generalize the algorithm presented in [26] and describe a method to find *all* valid polynomial equations of a given pre-specified form, for a generic (F, G) -

product. To this aim, we use polynomial templates [27], that basically represent a way to compactly specify sets of polynomials. Given a template T and an initial value problem (\mathcal{L}, ρ) , our algorithm returns the intersection between the set of polynomials described by the template and the kernel of the homomorphism μ from the coalgebra of polynomials to that of streams. Essentially, the generated set corresponds to the set of polynomials p such that $p^{(j)}(\rho) = 0$ for each $j \geq 0$, where ρ is the initial condition. Otherwise said, the algorithm returns the set of all the invariant polynomials that are instances of the template T for the given initial value problem. In practice, our algorithm builds two chains of sets. Firstly, a descending chain of vector spaces is generated to represent template parameter valuations such that all the derivatives of p up to order j vanish when computed on the initial conditions ρ . In addition, an ascending chain of ideals is built to detect the stabilization of the previous sequence.

We also present an Implicit Function Theorem (IFT) for systems of polynomial equations. This theorem provides sufficient syntactic conditions under which a system of polynomial equations has a unique stream solution, and gives a method to build it effectively by an SDE of polynomials for the stream solution. The formulation of this theorem is very similar to the classical theorem, but involves stream derivatives and the corresponding version of the Jacobian. We discuss the mathematical relation between the stream IFT and the classical IFT and show that the stream IFT has a neat computational advantage over the classical one.

1.6 Inference: Bayesian parameter estimation

1.6.1 State of the art

Systems of ODEs frequently involve parameters that have a great impact on their behaviour, but that are not directly measurable by experiments, or are such that there is inherent uncertainty on their values. This uncertainty must be rigorously taken into account when the model is used to make predictions, in order to avoid reaching conclusions about system characteristics that are unfounded, or making predictions that are too optimistic, given the model uncertainty. More formally, we will consider parametric ODEs systems of the form

$$\dot{x}(t) = f(x(t), \theta) \tag{1.5}$$

and initial condition $x(0) = x_0$. The problem of providing guaranteed estimates for the values of parameters θ is crucial, especially for safety verification. In general, statistical inference techniques are applied. Statistical inference [44] aims to learn the general characteristics of a population using only a subset of members of that population. Considering ODEs, estimates for parameters are inferred relying on a finite number of noisy observations of the trajectories of a real-world system. In this contest, probability measures play an essential role, since they numerically quantify uncertainty and/or rational beliefs about unknown quantities. In particular, the Bayes' theorem provides a rational method to update a given probability measure as more evidence or data become available. The process of inductive learning via the Bayes' theorem is called Bayesian inference [97].

Several works as [80, 179] follow a Bayesian point of view for ODEs parameter estimation. They assume that a known prior distribution on the unknown parameter values is given and aim to compute their posterior distribution, that essentially describes the uncertainty on parameters conditioned on a collection of observed data. In general, obtaining exact values and formulas for posterior quantities as means and variances is difficult. However, such quantities can be approximated generating random sample values of the parameters from their posterior distributions and applying Monte Carlo methods [97].

Sophisticated sampling schemes for sampling parameters have been developed. For instance, when considering ODEs systems, parametric inference techniques based on Monte Carlo Markov Chain (MCMC)[125] are widely used, they create a Markov chain such that its stationary distribution is the desired posterior distribution. In practice, it is not easy to assess when a Markov chain has reached its stationary regime, that is when to start sampling, then usually an approximation is made. Also particles-based Sequential Monte Carlo (SMC) methods have been applied [63]. Unfortunately, if the simulation is performed with only a finite number of steps or particles which is always the case, formal guarantees of correctness are hard to achieve: this makes safety verification very complicated.

MCMC/SMC techniques are very demanding from a computational point of view and require an explicit expression of the likelihood. Likelihood is a function that essentially quantifies how accurately the parameters explain observations, and it is not always available. This has led to the development of approximate Bayesian computation (ABC) [117], that avoids computation of the likelihood function. However, it shares the same difficulties as MCMC and SMC about formal guarantees and fails to adequately model observations errors when applied to ODEs parameter estimation [5].

With the aim of overcoming these limitations in the context of ODEs parameter estimation, a method to approximate the posterior distribution has been described in [52]. It iteratively partitions the space of parameters into finitely many disjoint cells and computes analytically interval bounds on the posterior likelihood for each cell, then the cells with the highest probability are refined through iterations. Finally, likelihood bounds are normalized to obtain bounds on posterior probabilities of each cell. This computation is achieved by using an inexpensive reachability analysis approach that relies on sensitivity analysis and numerical simulations.

1.6.2 Our contribution

We describe an alternative hybrid approach for ODEs parameter estimation that combines methods based on Monte Carlo simulation, uncertain probability and Interval Arithmetic (IA). It efficiently computes sharp estimates of posterior quantities, such as Cumulative Distribution Functions (CDFs), and their expectations, equipped with formal guarantees of accuracy. In particular, formal guarantees of accuracy are given in the form of confidence intervals for the posterior CDF of the parameters and moments thereof. They are established relying on an exponential tail inequality for the sum of independent random variables: Hoeffding's bound.

Our approach consists of two phases. The first phase is deterministic: after collecting observations, the parameter search space is reduced by computing, via interval analysis [100], a tight approximation of the set of *feasible* parameter values,

that is those parameters values that can actually lead to the set of given observations, for some noise value. The second phase computes confidence intervals for posterior expectations of the parameters given the observations, and is based on a randomized algorithm based on Monte Carlo simulations. Differently from [52], in our case the Monte Carlo phase introduces a level of controlled aleatoric uncertainty: hence the generated bounds are not certain, but come equipped with confidence levels. Accepting a controlled level of uncertainty allows us to achieve greater accuracy and greater efficiency in estimation.

We have put our algorithm at work on a few problems of ODEs parameter estimation from the literature. Our method compares very favourably to state-of-the-art techniques, both in terms of accuracy and execution time. Our approach can be applied not only to the estimation of parameters of ODEs, but to the more general case of a model where the relation between observations and parameters is a function, such as a Neural Network.

1.7 Inference, a more general view: Probabilistic Programming

1.7.1 State of the art

Probabilistic Programming (PP) is a powerful programming paradigm used to formally represent and reason with uncertainty. PP can also be applied to infer statistical conclusions from uncertain data and real-world observations. Basically, a probabilistic program [85, 14] is an ordinary program with the additional possibility to sample from known distributions, and to condition the values of variables based on external observations/data. The last feature is essential, as it enables the integration of information from the external world into a program, and thus to exploit empirical knowledge to update program variables. Although probabilistic programs typically consist of a few lines of code, they are often hard to analyze [85].

A probabilistic program implicitly defines a probability distribution for its output variables, called posterior because it is obtained taking into account the observed data. In principle, PP can be applied to parameter inference in dynamical systems as well. For instance, it is easy to write a probabilistic program that computes stepwise the solution of a discretized parametric ODE, where the values of the parameters are chosen according to predefined distributions. Moreover, noisy versions of the solution might be observed at discrete time instants. Inference on the parameters can then be performed by, roughly speaking, accepting only those computations of the probabilistic program that yield computed solutions compatible with the observations. However, compared to the approach outlined in Section 1.6, inference via PP can be much more general and flexible. For example, observations might follow more complicated patterns, like for example being performed at random; or one might perform inference not only on the parameters, but also on sequences of traversed states that are not directly observable, like in Hidden Markov Models [132].

In recent years, we have seen a great proliferation of PP languages, as well as tools and techniques to perform inference based on them. The semantics of programming languages can be described informally via natural languages, but it is very difficult to

reason about the properties of programs only relying on an informal prose. A formal mathematical description of a programming language is therefore highly desirable. Most work [14, 174, 95, 35, 61, 60] on the semantics of PP follows the denotational approach initiated by Kozen [111]. In [111], programs are interpreted as continuous linear operators (functions) on a Banach space of distributions. A major result is that the behaviour of a probabilistic program is completely determined by its behaviour on inputs whose distribution is a point mass. A few works instead pursue a more operational approach. For instance, in [45] Aditya et al. consider a big-step sampling semantic equivalent to the denotational one, and, based on that, present a Monte Carlo based sampling algorithm for a PP language. An operational approach is also proposed in [99, 87]: in both works, only discrete-space Markov decision processes are considered. Further references to PP can be found in [85].

In general, the use of denotational semantics with the aim of inference has practical limitations, and existent methods for inference in PP are often disconnected from formal semantics. For instance, most practical implementations are based on sampling after a finite number of steps: it is not easy to take correctly into account unbounded or even infinite computations in these types of implementation. Moreover, severe performance issues can arise.

1.7.2 Our contribution

We depart from the denotational tradition of PP and introduce an *action-based* probabilistic programming language, together with a small-step operational semantics. Informally speaking, we describe computations as (infinite) sequences of states, as opposed to the denotational approach, based on functions obtained by sequentially composing the effect of successive program statements.

The small-step semantics is formalized in terms of a Markov kernel and directly leads to an exact sampling algorithm that can be efficiently SIMD-parallelized. It describes how the individual steps of a computation take place: this brings benefits both in terms of clarity of presentation and in terms of effective implementation. On top of the operational semantics, an observational semantics is introduced: it is based on a probability space of infinite sequences of states and is given in terms of expectation of measurable functions, conditioned on non-failure. Along with the exact semantics we prove an approximation theorem: it provides lower and upper bounds of the exact semantics, based on the semantics of the program truncated at a chosen finite execution length.

Preliminary experiments, conducted on a number of nontrivial probabilistic programs drawn from the literature using a TensorFlow-based implementation, show that our approach compares favorably to State-of-the-art tools for probabilistic programming and inference.

Chapter 2

Invariants and conservation laws

2.1 Overview

In this chapter, we focus on the concept of invariant. In particular, we consider a specific type of invariant: conservation laws, and we outline an algorithm to find all the polynomial conservation laws up to a fixed degree for a given system of partial differential equations, only relying on linear algebraic computations.

More formally, let Σ be a system of partial differential equations (PDEs) in n independent variables. A conservation law of Σ is vector of n expressions, called *fluxes*, whose divergence vanishes on the solutions of the system [140, Ch.4]. In the case of ODEs ($n = 1$), this is the same as a first integral of the system. Conservation laws often express physical principles, such as conservation of mass, energy, momentum and so on, and as such are of fundamental importance to gain a qualitative insight into the phenomenon being studied. Moreover, the existence of many conservation laws points to complete integrability of Σ [2]. Conservation laws are also crucial in applications, in particular numerical methods: knowledge of conservation laws of Σ makes it possible to apply effective numerical schemes, including finite volume and finite elements methods; see e.g., [116, Ch.12].

Methods to systematically search conservation laws have traditionally been linked to the existence of symmetries, on account of a celebrated theorem by Emmy Noether [140, Ch.4]. A variety of algorithms based on a *direct approach*, which does not presuppose the existence of symmetries and is more widely applicable, have also been developed: see [140, Ch.4] and e.g. [11, 183, 135, 93, 50] and references therein. In the direct approach, one first finds *multipliers*, whose linear combinations with the system's equations yield divergence expressions vanishing on solutions, then inverts the divergences to obtain the corresponding fluxes.

In this chapter, we put forward a new method to automatically find conservation laws of polynomial form. The method can be applied to any polynomial PDE system, independently of the existence of symmetries, and as such, it might be classified as 'direct' itself. Its distinctive features are:

- (a) its completeness is given in terms of *fluxes*: differently from most direct methods,

the user directly specifies an ansatz (order and degree) of the searched polynomial fluxes, rather than that of multipliers;

- (b) computationally, it only relies on equational rewriting and linear algebraic operations — at least when it is applied to leading linear systems.

An additional benefit of this method is, in our opinion, its conceptual simplicity: it just takes some elementary algebra to give a rather detailed description of its functioning. This we do below.

Let Σ be a polynomial system of PDEs. Under a certain nondegeneracy condition, the set of invariant polynomials of Σ up to a given differential order coincides with the ideal generated by Σ , denoted $\langle \Sigma \rangle$, in a sub-ring of the differential polynomials. Now, a polynomial conservation law is a vector of polynomials, the fluxes, whose divergence is an invariant, that is belongs to $\langle \Sigma \rangle$. Equivalently, upon performing polynomial division of such a divergence by a basis of $\langle \Sigma \rangle$, the obtained remainder is zero. Symbolically, we represent a *set* of polynomials as a formal linear combination of monomials with unknown coefficients, or *template*; and a set of candidate conservation laws as a single vector of user-specified flux templates. We then seek necessary and sufficient conditions on the coefficients for the remainder of the resulting divergence to vanish upon division by $\langle \Sigma \rangle$. This results in a homogeneous linear algebraic system for the unknown coefficients. When the solutions of this system are substituted back into the original flux templates, all polynomial conservation laws fitting the specified templates are obtained. In the case of PDEs whose leading term is linear, a basis of $\langle \Sigma \rangle$ is Σ itself, under an appropriate monomial ordering; and taking the remainder of a polynomial effectively corresponds to rewriting it into a normal form. This justifies our claim that for leading linear systems, computationally the method requires no more than equational rewriting and linear algebra. Additionally, trivial laws can be easily filtered out from the output.

In this context, an important goal will be to identify easy to check syntactic conditions on Σ that guarantee nondegeneracy, which is essential for completeness. We will show that the Riquier's format [150, 153], a generalization of Cauchy-Kovaleskvaya's one [140, Ch.4], does imply the considered notion of nondegeneracy. We will cover the cases of both formal and real analytic solutions of PDEs.

On the negative side, it may be observed that the completeness of the algorithm holds relative to polynomial fluxes only up to a specified degree. We will comment on this limitation when we will compare our algorithm to the direct method.

Structure of the chapter The rest of the chapter is organized as follows. Background on PDEs and their solutions is given in Section 2.2 . The method is then presented in Section 2.3. A few experiments and a comparison with a state-of-the-art algorithm based on the direct approach are discussed in Section 2.4. For most part of the chapter we will be concerned with formal power series solutions of PDEs; the obtained results carry over to real analytic solutions with very minor modifications, which are dealt with in Section 2.5. Concluding remarks and further comparison with related work are in Section 2.6. A few technical proofs have been confined to Appendix A.

2.2 Preliminaries

We introduce some standard terminology and notation on PDEs and their solutions. Let $X = \{x_1, \dots, x_n\}$ and $\mathcal{U} = \{u^1, \dots, u^m\}$ be disjoint, nonempty sets of independent and dependent variables, respectively. We let t, x, y, \dots range over X and u, v, \dots , possibly with superscripts u^i, \dots , range over \mathcal{U} . Let X^\otimes , ranged over by τ, ξ, \dots be the set of monomials that can be formed from variables in X — that is, the free commutative monoid generated by X — with ϵ denoting the identity monomial. For $\tau = x_1^{k_1} \cdots x_n^{k_n}$, we let $|\tau| \triangleq k_1 + \cdots + k_n$. We let $\mathcal{D} \triangleq \{u_\tau : u \in \mathcal{U}, \tau \in X^\otimes\}$ denote the set of *derivatives*; here u_ϵ will be identified with u . We let $\mathbb{R}[X \cup \mathcal{D}]$, ranged over by f, g, \dots, p, q, \dots , be the set of (*differential*) *polynomials* with coefficients in \mathbb{R} and indeterminates in $X \cup \mathcal{D}$. The *order* of a polynomial p is $\max\{|\tau| : u_\tau \text{ occurs in } p\}$. For example, $f = xv_z u_{xy} + v_y^2 + u + 5x$ is a differential polynomial of degree 3 and order 2.

A system of polynomial PDEs is a finite set $\Sigma \subseteq \mathbb{R}[X \cup \mathcal{D}]$. The order of Σ is the maximum order of polynomials in Σ . In what follows, we shall consider an arbitrarily fixed, *finite* $D \subseteq \mathcal{D}$ that is a superset of the derivatives occurring in Σ and let $\mathcal{P} \triangleq \mathbb{R}[X \cup D]$. The set \mathcal{P} will act as our ‘universe’ of differential polynomials, in the sense that we will be interested in finding conservation laws whose divergence (see Section 2.3) lies in \mathcal{P} . For example, in the case of the wave equation, with $X = \{x, y\}$, $\mathcal{U} = \{u\}$ and $\Sigma = \{u_{xx} - u_{yy}\}$, we might fix $D = \{u, u_x, u_y, u_{xy}, u_{xx}, u_{yy}\}$ if we are interested in laws with fluxes built out of $\{u, u_x, u_y\}$. Elements of \mathcal{P} are multivariate polynomials in a finite number of indeterminates in the usual algebraic sense: in particular, they can be evaluated at any point $(x, u_D) \in \mathbb{R}^{X \cup D} \cong \mathbb{R}^k$, with $k \triangleq |X \cup D|$, where we are implicitly fixing an arbitrary total order on $X \cup D$. Therefore, any subset $P \subseteq \mathcal{P}$ induces an *algebraic variety* $\mathcal{V}(P) \subseteq \mathbb{R}^k$, defined as $\mathcal{V}(P) \triangleq \{(x, u_D) \in \mathbb{R}^k : p(x, u_D) = 0 \text{ for each } p \in P\}$. In particular, as $\Sigma \subseteq \mathcal{P}$, we can consider $\mathcal{V}(\Sigma)$, the algebraic variety induced by Σ .

We will be mostly concerned with formal power series solutions of Σ — but see Section 2.5 on analyticity. For a monomial $\tau = x_1^{k_1} \cdots x_n^{k_n}$ and $x^0 = (x_1^0, \dots, x_n^0) \in \mathbb{R}^n$, we let $(x - x^0)^\tau \triangleq (x_1 - x_1^0)^{k_1} \cdots (x_n - x_n^0)^{k_n}$ be a monomial in the terms $x_i - x_i^0$. A formal power series centered at $x^0 \in \mathbb{R}^n$ is a formal sum of monomials¹ $F = \sum_{\tau \in X^\otimes} c_\tau (x - x^0)^\tau$, with $c_\tau \in \mathbb{R}$. The value of F at x^0 , denoted $F(x^0)$, is the constant coefficient c_ϵ . Sum, product and partial derivative $\partial F / \partial x_i$ ($x_i \in X$, extension to monomials denoted by $\partial^{|\tau|} F / \partial \tau$) of formal power series are defined as usual and satisfy the expected properties. Let $U = (U^1, \dots, U^m)$ be a tuple of formal power series centered at x^0 . We let $U_D \triangleq (\frac{\partial^{|\tau|}}{\partial \tau} U^i)_{u_\tau^i \in D}$ denote the tuple of formal power series corresponding to the derivatives in D , and let $U_D(x^0)$ be the tuple $(\frac{\partial^{|\tau|}}{\partial \tau} U^i(x^0))_{u_\tau^i \in D}$. We say U is a formal *solution* of Σ centered at x^0 if for each $p \in \Sigma$ the formal power

¹Rigorously, a formal power series centered at x^0 is a function $F : \{x_1 - x_1^0, \dots, x_n - x_n^0\}^\otimes \rightarrow \mathbb{R}$. Accordingly, when writing polynomial expressions of such series, each $x_i \in X$ is to be interpreted as the formal power series $F = x_i^0 + 1 \cdot (x_i - x_i^0)$, that is: $F(\epsilon) = x_i^0$, $F(x_i - x_i^0) = 1$ and $F((x - x^0)^\tau) = 0$ for any $\tau \neq \epsilon, x_i$.

series $p(x, U_D(x))$ centered at x^0 is zero. The formal D -solution variety of Σ is

$$\mathcal{S}(\Sigma) \triangleq \{(x^0, u_D^0) \in \mathcal{V}(\Sigma) : \text{there is a formal solution } U \text{ of } \Sigma \text{ centered at } x^0 \\ \text{such that } U_D(x^0) = u_D^0\}.$$

We say Σ is formally D -locally solvable if its algebraic and D -solution varieties coincide, $\mathcal{V}(\Sigma) = \mathcal{S}(\Sigma)$. In what follows, we shall omit the qualification “ D -” if the set D is clear from the context.

Remark 1 (on local solvability). Ignoring for the time being the distinction between analytic and formal solutions, we see the above definition of local solvability is more flexible, when compared to the usual one [140, Ch.2, Def.2.70] that requires D to be the set of *all* derivatives up to the order of Σ . For instance, the following system of order 2

$$\Sigma = \{u_x - v, v_{xx} - u_y\}$$

is not locally solvable in the sense of [140, Ch.2, Def.2.70], simply because there are 2nd order differential consequences, like $u_{xy} - v_y$, that are not algebraic consequences. However Σ is D -locally solvable for, say, $D = \{u, v, u_x, u_y, v_{xx}\}$. Generally speaking, our definition of D -local solvability appears to be a sensible extension of the usual one, when considering systems that are not in Cauchy-Kovalevskaya form, but satisfy more general forms, like Riquier’s format [150]. This point will be further discussed in the next section.

2.3 The method

Our main object of interest is a notion invariant, a sort of logical consequence of Σ .

Definition 1 (invariant polynomials). We say $p \in \mathcal{P}$ is an invariant polynomial of Σ if, for each $x^0 \in \mathbb{R}^n$, the systems Σ and $\Sigma \cup \{p\}$ have the same solutions centered at x^0 . We let $\text{Inv}(\Sigma)$ denote the set of polynomial invariants of Σ that are in \mathcal{P} .

We will rely on a simple algebraic-geometric characterization of $\text{Inv}(\Sigma)$. Let us introduce the necessary terminology; see [57, Ch.1-2] for a more comprehensive treatment. For any $W \subseteq \mathbb{R}^k$, we let $\mathcal{I}(W) \subseteq \mathcal{P}$ be the ideal of polynomials that vanish on W , that is $\mathcal{I}(W) \triangleq \{p \in \mathcal{P} : p(w) = 0 \text{ for each } w \in W\}$. Moreover, for any $P \subseteq \mathcal{P}$, we let $\langle P \rangle \subseteq \mathcal{P}$ be the ideal generated by P . The ideal $\mathcal{I}(\mathcal{V}(P))$ is called the *real radical* of P . In the following definition, we consider the real radical of Σ .

Definition 2 (D -nondegeneracy). We say Σ is D -nondegenerate if Σ is D -locally solvable and $\langle \Sigma \rangle = \mathcal{I}(\mathcal{V}(\Sigma))$.

Lemma 1. $\langle \Sigma \rangle \subseteq \text{Inv}(\Sigma)$, with equality if Σ is D -nondegenerate.

PROOF. First suppose $p \in \langle \Sigma \rangle$, that is $p = \sum_j q_j f_j$ for some $q_j \in \mathcal{P}$ and $f_j \in \Sigma$. By definition, any solution of Σ , however centered, makes each f_j , hence p , identically zero.

Suppose now Σ is D -locally solvable and $\langle \Sigma \rangle = \mathcal{I}(\mathcal{V}(\Sigma))$, and consider any $p \in \mathcal{I}nv(\Sigma)$. Consider any $(x^0, u_D^0) \in \mathcal{V}(\Sigma) = \mathcal{S}(\Sigma)$. By definition of local solvability, there is a solution U of Σ centered at x^0 such that $U_D(x^0) = u_D^0$. Then we have: $p(x^0, u_D^0) = p(x^0, U_D(x^0)) = p(x, U_D(x))|_{x=x^0} = 0$, where the last equality stems from p being an invariant. Since $(x^0, u_D^0) \in \mathcal{V}(\Sigma)$ is arbitrary, we have shown that $p \in \mathcal{I}(\mathcal{V}(\Sigma)) = \langle \Sigma \rangle$. \square

Consider now any Groebner basis Δ of $\langle \Sigma \rangle$. For each $p \in \mathcal{P}$, we can then consider the unique remainder of the polynomial division of p by Δ , that is

$$S_\Delta p \stackrel{\Delta}{=} p \bmod \Delta.$$

In what follows, we shall write $S_\Delta p$, leaving Δ implicit. By Lemma 1, $S_\Delta p = 0$ ensures that p is an invariant. The converse as well can be stated if Σ is a nondegenerate. We introduce below an important class of nondegenerate systems.

We recall that a *ranking* [153] of the derivatives is a total ordering on \mathcal{D} such that, for all $u, v \in U$, $\tau, \xi \in X^\otimes$ and $x_i \in X$:

1. $u_\tau < u_{\tau x_i}$;
2. $u_\tau < v_\xi$ implies $u_{\tau x_i} < v_{\xi x_i}$.

Let us say Σ is *leading linear* if its elements are of the form $u_\tau + f$, where $u_\tau > v_\xi$ for each v_ξ occurring in f ; in this case, we let $\text{dom}(\Sigma)$ be the set of such leading derivatives u_τ . In what follows, $D_{x_i}p$ denotes the formal total derivative of p along $x_i \in X$: this is computed like the usual derivative of p along x_i , just taking into account that $D_{x_i}u_\tau = u_{\tau x_i}$. As $D_{x_i}D_{x_j}p = D_{x_j}D_{x_i}p$, the notation $D_\tau p$ for $\tau \in X^\otimes$ is well defined. For $k \geq 0$, let us denote the k -th prolongation of Σ and D as $\Sigma^{(k)} \stackrel{\Delta}{=} \{D_\tau p : p \in \Sigma, |\tau| \leq k\}$ and $D^{(k)} \stackrel{\Delta}{=} \{u_{\xi\tau} : u_\xi \in D \text{ and } |\tau| \leq k\}$, respectively. A leading linear Σ is *passive* if it implies all its integrability conditions, that is: whenever $u_\tau + f \in \Sigma$ and $u_\xi + g \in \Sigma$ and $\tau\xi' = \xi\tau'$ then, for some k , $(D_{\xi'}f - D_{\tau'}g) \in \langle \Sigma^{(k)} \rangle$. In fact, for passivity it is sufficient to check the finitely many integrability conditions with $\xi' = \sigma/\tau$ and $\tau' = \sigma/\xi$, where σ is the least common multiple of τ and ξ ; see [153, Cor.1]. A leading linear, passive system is also called a *Riquier basis* in [153]: from now on we shall adopt this terminology², with the further specification that any two distinct elements in a Riquier basis must have distinct leading derivatives. This implies no loss of expressiveness. For a leading linear system Σ , we can define the sets of *principal derivatives* $\mathcal{P}_r(\Sigma) \stackrel{\Delta}{=} \{u_{\tau\xi} : u_\tau \in \text{dom}(\Sigma) \text{ and } \xi \in X^\otimes\}$, and *parametric derivatives* $\mathcal{P}_a(\Sigma) \stackrel{\Delta}{=} \mathcal{D} \setminus \mathcal{P}_r(\Sigma)$. The following proposition ensures D -nondegeneracy for Riquier bases, whenever the principal derivatives occurring in Σ coincide with its leading derivatives, as well as with the principal derivatives in D .

Proposition 1. *Let Σ be a Riquier basis and assume $D \cap \mathcal{P}_r(\Sigma) = \text{dom}(\Sigma)$. Then Σ is D -nondegenerate. Moreover, Σ is a Groebner basis of $\mathcal{I}(\mathcal{V}(\Sigma))$ w.r.t. a suitable monomial order.*

²The *coherent* systems in [31] are an equivalent notion.

PROOF. We first show that Σ is D -locally solvable. An initial data specification for Σ is a function $\rho : \mathcal{Pa}(\Sigma) \rightarrow \mathbb{R}$. A solution of the initial value problem (Σ, ρ) at $x^0 \in \mathbb{R}^n$ is a solution U of Σ centered at x^0 such that, for each $u_\tau^i \in \mathcal{Pa}(\Sigma)$, $\frac{\partial^{|\tau|}}{\partial \tau} U_\tau^i(x^0) = \rho(u_\tau^i)$. The formal Riquier existence theorem [153, Th.2] ensures that for each x^0 and ρ there is a unique solution U of the problem (Σ, ρ) .

Now, let $(x^0, u_D^0) \in \mathcal{V}(\Sigma)$. Below, we will denote the component of u_D^0 corresponding to $u_\tau^i \in D$ by $u_\tau^{0,i}$. Let us define ρ as follows: $\rho(u_\tau^i) = u_\tau^{0,i}$ for each $u_\tau^i \in D \cap \mathcal{Pa}(\Sigma)$, and arbitrarily for any other element of $\mathcal{Pa}(\Sigma)$. According to the formal existence theorem, there is a solution U of (Σ, ρ) . In particular, we have that for each $u_\tau^i \in D \cap \mathcal{Pa}(\Sigma)$, $\frac{\partial^{|\tau|}}{\partial \tau} U_\tau^i(x^0) = \rho(u_\tau^i) = u_\tau^{0,i}$. We now proceed to show the same for each $u_\tau^i \in \text{dom}(\Sigma)$. We proceed by induction on the ranking of u_τ^i . Let $u_\tau^i + f \in \Sigma$, where $f = f(x, u_{D'})$, for some $D' \subseteq D$ that only contains elements smaller than u_τ^i in the ranking. In the base case, D' only contains parametric derivatives. In any case, we have

$$\begin{aligned} \frac{\partial^{|\tau|}}{\partial \tau} U_\tau^i(x^0) &= -f(x^0, U_{D'}(x^0)) \\ &= -f(x^0, u_{D'}^0) \\ &= u_\tau^{0,i} \end{aligned}$$

where the first equality follows from U being a solution of Σ , the second one from either the result on parametric derivatives above (base case) or the induction hypothesis (inductive step), and the third one from $(x^0, u_D^0) \in \mathcal{V}(\Sigma)$ and $u_\tau^i + f \in \Sigma$. Overall, we have shown that $u_D^0 = U_D(x^0)$, hence $(x^0, u_D^0) \in \mathcal{S}(\Sigma)$.

We now show that $\langle \Sigma \rangle = \mathcal{I}(\mathcal{V}(\Sigma))$. First, we show that Σ is a Groebner basis for $\langle \Sigma \rangle$, once we consider the lexicographic monomial order on $(X \cup D)^\otimes$ induced by the ranking $<$ on \mathcal{D} , augmented by the rules $x_i < x_j < u_\tau$ for every $i < j$ and $u_\tau \in \mathcal{D}$. To see this, take any $0 \neq p \in \langle \Sigma \rangle$, and assume by contradiction that the leading monomial of p is not divisible by the leading derivative of any element in Σ . By the chosen monomial order and the condition $D \cap \mathcal{Pr}(\Sigma) = \text{dom}(\Sigma)$, p has no occurrences of principal derivatives. Now partition D as $D = D_0 \cup D_1$, where D_0 are parametric derivatives and $D_1 = \text{dom}(\Sigma)$ are principal. Consider any tuple $(x^0, u_{D_0}^0) \in \mathbb{R}^{X \cup D_0}$. For each $u_\tau \in \text{dom}(\Sigma)$, we define $u_\tau^0 \in \mathbb{R}$, by induction on the ranking and using the same notation as above, as follows: $u_\tau^0 \triangleq -f(x^0, u_{D'}^0)$, where $u_\tau + f \in \Sigma$. By construction, the tuple $(x^0, u_D^0) = (x^0, u_{D_0}^0, u_{D_1}^0)$ so defined is in $\mathcal{V}(\Sigma)$. Hence, recalling that $p \in \langle \Sigma \rangle \subseteq \mathcal{I}(\mathcal{V}(\Sigma))$, we have: $0 = p(x^0, u_D^0) = p(x^0, u_{D_0}^0)$. Since this holds for arbitrary $(x^0, u_{D_0}^0)$ and $p \in \mathbb{R}[X \cup D_0]$, we have shown that $p = 0$, contradicting the initial assumption.

We now show that $\langle \Sigma \rangle = \mathcal{I}(\mathcal{V}(\Sigma))$. The inclusion $\langle \Sigma \rangle \subseteq \mathcal{I}(\mathcal{V}(\Sigma))$ is obvious, so consider any $f \in \mathcal{I}(\mathcal{V}(\Sigma))$, and write $f = f_0 + p$, where $f_0 \in \langle \Sigma \rangle$ and $p = f \bmod \Sigma$, hence $p \in \mathbb{R}[X \cup D_0]$. For arbitrary $(x^0, u_{D_0}^0)$, let us build $(x^0, u_D^0) = (x^0, u_{D_0}^0, u_{D_1}^0)$ exactly as above. As $(x^0, u_D^0) \in \mathcal{V}(\Sigma)$, we have $0 = f(x^0, u_D^0) = f_0(x^0, u_D^0) + p(x^0, u_D^0) = p(x^0, u_{D_0}^0)$. Since this holds for arbitrary $(x^0, u_{D_0}^0)$, we deduce $p = 0$, hence $f \in \langle \Sigma \rangle$. \square

The above result implies that, for Riquier bases, provided the set of leading deriva-

tives coincides with the set of principal derivatives occurring in Σ , D can be taken to consist of all the derivatives occurring in Σ , plus an arbitrary finite set of parametric derivatives.

Example 1. The system Σ in Remark 1 is a Riquier basis w.r.t., for instance, a graded lexicographic ranking with $u < v$ and $x < y$. In particular, no integrability conditions arise. Take D as specified in the remark. As $D \cap \mathcal{P}_r(\Sigma) = \{u_x, v_{xx}\} = \text{dom}(\Sigma)$, Proposition 1 ensures that Σ is D -locally solvable.

It may also be the case that the condition $D \cap \mathcal{P}_r(\Sigma) = \text{dom}(\Sigma)$ is not satisfied, because there are principal derivatives in D that are not in $\text{dom}(\Sigma)$, so one cannot apply Proposition 1 directly to Σ . In this case, it is enough to expand Σ by adding the corresponding equations. Formally, one applies the following lemma, possibly several times, until a Σ is obtained that satisfies $D \cap \mathcal{P}_r(\Sigma) = \text{dom}(\Sigma)$. The proof of the lemma is an immediate consequence of the definition of Riquier basis and of invariant polynomial.

Lemma 2. *Let Σ be a Riquier basis, $u_\tau \in \mathcal{P}_r(\Sigma) \setminus \text{dom}(\Sigma)$ and $u_\xi + f \in \Sigma$ with $\tau = \xi\zeta$. Then $\Sigma' \triangleq \Sigma \cup \{D_\zeta(u_\xi + f)\}$ is still a Riquier basis. If $D_\zeta(u_\xi + f) \in \mathcal{P}$, $\text{Inv}(\Sigma) = \text{Inv}(\Sigma')$.*

Let us now introduce conservation laws.

Definition 3 (conservation laws). *A n -tuple of polynomials $\Phi = (p_1, \dots, p_n)$ is a (polynomial) conservation law for Σ if its divergence $\text{div} \Phi \triangleq \sum_{j=1}^n D_{x_j} p_j$ is an invariant polynomial for Σ . The components p_i in Φ are called fluxes.*

For any finite $Z \subseteq (X \cup D)^\otimes$, let \mathcal{P}_Z be the set of polynomials that can be formed from monomials in Z . We let $\text{CL}(\Sigma, Z) \subseteq \mathcal{P}_Z^n$ denote the set of conservation laws with fluxes in \mathcal{P}_Z . A typical choice will be, $Z = Y^{\leq d}$, for some $Y \subseteq X \cup D$ and $d \geq 1$, that is the monomials in Y^\otimes of degree $\leq d$: this will give all the conservation laws with fluxes of degree $\leq d$ that can be built out Y . In what follows, we will refer to Z as to the chosen ansatz. We shall assume that D is chosen large enough³ to ensure that $\text{div} \Phi \in \mathcal{P}$ whenever $\Phi \in \mathcal{P}_Z^n$. $\text{CL}(\Sigma, Z)$ is clearly a finite-dimensional vector space: our goal is to give a method to compute a basis for this space.

With this goal in mind, we introduce a way of representing succinctly sets of polynomials by means of *templates*. Fix a set of $s \geq 1$ distinct symbols, the *parameters* $a = \{a_1, \dots, a_s\}$. A linear expression $\ell = \sum_j \lambda_j a_j$ is linear combination those s parameters with real coefficients $\lambda_j \in \mathbb{R}$. A template is a polynomial expression with linear expressions as coefficients, $\pi = \sum_{j=1}^h \ell_j \cdot \alpha_j$, for monomials $\alpha_1, \dots, \alpha_h \in (X \cup D)^\otimes$. We say π is *complete* for the ansatz Z if $\ell_j = a_{i_j}$ are pairwise distinct parameters, and $Z = \{\alpha_1, \dots, \alpha_h\}$. For $v = (v_1, \dots, v_s) \in \mathbb{R}^s$, we let $\ell[v] \in \mathbb{R}$ and $\pi[v] \in \mathcal{P}$ be the real value and the polynomial, respectively, obtained by replacing each a_i with v_i , for $i = 1, \dots, s$. For any $A \subseteq \mathbb{R}^s$, we let $\pi[A] = \{\pi[v] : v \in A\} \subseteq \mathcal{P}$. For a set of linear expressions $L = \{\ell_1, \dots, \ell_t\}$, we let $\text{span}(L) \triangleq \{v \in \mathbb{R}^s : \ell[v] = 0 \text{ for each } \ell \in L\} \subseteq \mathbb{R}^s$ denote the vector space of parameter evaluations that annihilate all expressions in L .

³This may imply expanding Σ via Lemma 2 in order to apply Proposition 1 and ensure nondegeneracy.

It is now convenient to regard both polynomials and templates as elements of a larger polynomial ring, $\mathbb{R}[a \cup X \cup D]$. Let Δ be a Groebner basis of $\langle \Sigma \rangle$ in \mathcal{P} w.r.t. some monomial order. Δ is still a Groebner basis in the larger polynomial ring, with respect to any monomial on $(a \cup X \cup D)^\otimes$ that conservatively extends the original one on $(X \cup D)^\otimes$ (see Appendix A.1 for additional details about this technical point). Then we can conservatively extend the function S to templates

$$S\pi \stackrel{\Delta}{=} \pi \bmod \Delta.$$

In the special case of $\Sigma = \Delta$ a Riquier basis, this means to rewrite π applying repeatedly the equations in Σ from left to right, treating the parameters a_i s as arbitrary constants, until no leading derivative remains. The following lemma gives a simple substitution property for templates; for a proof, see [29, Lemma 3].

Lemma 3. *Let π be a template. Then $S\pi$ is still a template. Moreover, for each $v \in \mathbb{R}^s$, $S(\pi[v]) = (S\pi)[v]$.*

We extend the total derivative operator D_{x_i} to templates as expected, by setting $D_{x_i}(\sum_j \ell_j \alpha_j) \stackrel{\Delta}{=} \sum_j \ell_j D_{x_i} \alpha_j$. Then, for a tuple of templates $\Pi = (\pi_1, \dots, \pi_n)$, we let $\mathbf{div} \Pi \stackrel{\Delta}{=} \sum_{i=1}^n D_{x_i} \pi_i$ denote its divergence. A *C.L. template for Z* is a tuple $\Pi = (\pi_1, \dots, \pi_n)$ of complete templates for Z , such that the components π_i s are formed from pairwise disjoint subsets of parameters in a .

Given a C.L. template Π , we are interested in those $v \in \mathbb{R}^s$ such that $\Pi[v] \stackrel{\Delta}{=} (\pi_1[v], \dots, \pi_n[v])$ is a conservation law. That is, those v 's such that $\mathbf{div}(\Pi[v])$ is an invariant for Σ . That is, according to Lemma 1 and Lemma 3, those v 's such that $S(\mathbf{div} \Pi[v]) = (S(\mathbf{div} \Pi))[v] = 0$. In other words, we are interested in the v 's that annihilate all the linear expressions (coefficients) of $S(\mathbf{div} \Pi)$. This reasoning leads to the following result. A detailed proof is reported in Appendix A.1.

Corollary 1 (completeness for CL). *Let Π be a C.L. template for the ansatz Z . Let L be the set of linear expressions (coefficients) of $S(\mathbf{div} \Pi)$ and $V = \text{span}(L)$. Then $\Pi[V] \stackrel{\Delta}{=} \{\Pi[v] : v \in V\} \subseteq \text{CL}(\Sigma, Z)$, with equality if Σ is D -nondegenerate.*

Note, from the above result, that once we have a basis B of V , then $\Pi[B]$ is a basis for $\Pi[V]$. To sum up, our method conceptually consists of the following steps, assuming a Grobner basis Δ of $\langle \Sigma \rangle$ has been pre-computed. Given an ansatz Z :

1. using n disjoint sets of parameters, build a C.L. template for Z , $\Pi = (\pi_1, \dots, \pi_n)$;
2. compute the divergence template $\mathbf{div} \Pi$ and its normal form $r = S(\mathbf{div} \Pi)$;
3. extract from r its coefficients (linear expressions), $L = \{\ell_1, \dots, \ell_h\}$;
4. compute a basis B for $V = \text{span}(L)$;
5. return $\Pi[B]$, a basis of $\Pi[V]$.

Remark 2 (trivial, equivalent and independent laws). As an optional final step in the above method, one might want to remove trivial conservation laws from $\Pi[B]$. Recall that a conservation law $\Phi = (p_1, \dots, p_n)$ is *trivial* if it is a linear combination of

trivial laws of the first kind (each flux p_i in Φ is an invariant for Σ) and of the second kind ($\mathbf{div} \Phi$ is zero as a polynomial⁴). Two laws Φ_1 and Φ_2 are *equivalent* if their difference $\Phi_1 - \Phi_2$ is a trivial law.

Note any law $\Phi = (p_1, \dots, p_n)$ is equivalent to $\tilde{\Phi} = (Sp_1, \dots, Sp_n)$, as $\Phi - \tilde{\Phi} \in \langle \Sigma \rangle^n$ is a trivial law of the first kind by Lemma 1. Therefore, without loss of generality, that is up to equivalence, it is always possible to choose an ansatz Z such that $Sp = p$ for each $p \in \mathcal{P}_Z$. Syntactically, this means that no monomial in Z is divisible by the leading monomial of any element in Δ . For leading linear systems, this amounts to making sure that *no principal derivative occurs in Z* . Now further assume that Σ is D -nondegenerate. Then, by virtue of Lemma 1, the only trivial law of the first kind in $\text{CL}(\Sigma, Z)$ is $(0, \dots, 0)$ (n times). In this case, it is therefore sufficient to search and remove from $\Pi[B]$ trivial laws of the second kind, which is computationally easy.

An even less redundant representation of the space $\Pi[V]$ can be obtained by requiring that the set of returned laws, say $C \subseteq \Pi[V]$, satisfies the following property of *independence* up to triviality: if a linear combination of the laws in C , $\Psi = \sum_{\lambda_\Phi \in C} \lambda_\Phi \cdot \Phi$ ($\lambda_\Phi \in \mathbb{R}$), is trivial, then for each $\Phi \in C$, $\lambda_\Phi = 0$. If Σ is nondegenerate and the ansatz Z is chosen as specified above, again triviality of Ψ is equivalent to $\mathbf{div} \Psi = 0$ as a polynomial. Hence independence up to triviality of C is equivalent to the usual linear independence of the set of divergences $\{\mathbf{div} \Phi : \Phi \in C\}$ in the vector space of polynomials \mathcal{P} . Therefore, given $\Pi[B]$, computing a set C with the desired independence property is a matter of applying familiar linear algebraic techniques.

We illustrate the above method with a simple example.

Example 2 (Wave equation). Let Σ consists of the wave equation $u_{xx} - u_{yy}$. We can fix $D = \{u, u_x, u_y, u_{xy}, u_{xx}, u_{yy}\}$. We consider the ansatz $Z = Y^{\leq 2}$ where $Y = \{u_x, u_y\}$, and compute a basis for $\text{CL}(\Sigma, Z)$. Σ is a Riquier basis w.r.t. any ranking where $u_{xx} > u_{yy}$, and $D \cap \text{Pr}(\Sigma) = \{u_{xx}\} = \text{dom}(\Sigma)$: hence Σ is D -nondegenerate by Proposition 1. Now we have the following steps.

1. CL template:

$$\begin{aligned} \Pi &= (\pi_1, \pi_2) \quad \text{where} \quad \pi_1 = a_1 + a_2 u_x + a_3 u_x u_y + a_4 u_y^2 + a_5 u_x^2 + a_6 u_y \\ &\quad \pi_2 = a_7 + a_8 u_x + a_9 u_x u_y + a_{10} u_y^2 + a_{11} u_x^2 + a_{12} u_y. \end{aligned}$$

2. Divergence template and its normal form:

$$\begin{aligned} \mathbf{div} \Pi &= 2a_{10} u_y u_{yy} + a_{12} u_{yy} + a_2 u_{xx} + a_3 u_y u_{xx} + 2a_5 u_x u_{xx} + a_9 u_{yy} u_x + \\ &\quad u_y u_{xy} (2a_4 + a_9) + u_x u_{xy} (2a_{11} + a_3) + u_{xy} (a_6 + a_8) \\ S(\mathbf{div} \Pi) &= u_y u_{yy} (2a_{10} + a_3) + u_y u_{xy} (2a_4 + a_9) + u_{yy} u_x (2a_5 + a_9) + \\ &\quad + u_{yy} (a_{12} + a_2) + u_x u_{xy} (2a_{11} + a_3) + u_{xy} (a_6 + a_8). \end{aligned}$$

3. Linear expressions in $S(\mathbf{div} \Pi)$:

$$L = \{2a_4 + a_9, 2a_{10} + a_3, 2a_{11} + a_3, 2a_5 + a_9, a_6 + a_8, a_{12} + a_2\}.$$

⁴The general definition requires $(\mathbf{div} \Phi)(x, U_D(x))$ to be identically 0 for any smooth function U . In the polynomial case, this is equivalent to $\mathbf{div} \Phi = 0$.

4. Basis in \mathbb{R}^{12} for $V = \text{span}(L)$:

$$B = \{e_1, e_2 - e_{12}, e_3 - \frac{1}{2}(e_{10} + e_{11}), e_7, -e_6 + e_8, -\frac{1}{2}(e_4 + e_5) + e_9\}.$$

5. Basis of $\Pi[V]$, with trivial laws TL removed:

$$\Pi[B] \setminus TL = \{(-2u_y u_x, u_y^2 + u_x^2), (-u_y^2/2 - u_x^2/2, u_y u_x), (u_x, -u_y)\}.$$

For instance, the second law above is usually interpreted as conservation of mechanical (potential+kinetic) energy; see [140, Ch4]. As Σ is nondegenerate, $\Pi[V] = \text{CL}(\Sigma, Z)$.

In principle, the method can be applied ‘as is’ also to non leading-linear systems. However, since Proposition 1 cannot be invoked, D -nondegeneracy hence completeness (equality in Corollary 1) may not be guaranteed. Also, identification of trivial laws may be somewhat more laborious.

Example 3. Consider the single PDE system $\Sigma = \{u_y^2 + u_x^2 - 1\}$, a special case of the Eikonal equation. We can fix $D = \{u, u_x, u_y, u_{xy}, u_{xx}, u_{yy}\}$ and consider the ansatz $Z = Y^{\leq 3}$, where $Y = \{u, u_x, u_y\}$. We fix a complete flux template $\Pi = (\pi_1, \pi_2)$ and apply the algorithm with Σ as a Groebner basis; to this purpose, we fix the lexicographic monomial order induced by $y > x$, so that u_y^2 is the leading term of $u_y^2 + u_x^2 - 1$. Proceeding like in the previous example, we obtain as a result a vector space $\Pi[V] \subseteq \text{CL}(\Sigma, Z)$ (Corollary 1); or better, a concrete basis $\Pi[B]$ of it. In order to identify trivial laws more easily, we normalize the fluxes in $\Pi[B]$, that is, we consider $S\Pi[B] \triangleq \{(Sp, Sq) : (p, q) \in \Pi[B]\}$. Clearly, $\Pi[B]$ and $S\Pi[B]$ are equivalent up to triviality, in particular $S\Pi[B]$ is still a basis of $\Pi[V]$. Removing zeros and trivial laws of the second kind from $S\Pi[B]$, we are left with two laws

$$\{(-u_y u_x^2 - 2u_y, u_x^3), (-u_x^3, -u_y u_x^2 + u_y)\}.$$

We remark that Σ is not D -locally solvable.

2.4 Experiments

We present experimental results obtained from a proof-of-concept Python implementation⁵ of the algorithm described in Section 2.3. We shall refer to this implementation of the algorithm as POLYCONS.

We apply POLYCONS to some classic PDEs drawn from mathematical physics and presented, in equational form, in Table 2.1. Some of the original equations have been transformed into lower order equivalent systems: this is beneficial for efficiency, but not strictly necessary for our method to work. An exception is the sine-Gordon (s-G) equation, whose original form is: $u_{tt} - u_{xx} + \sin u = 0$. In this case, the transformation is necessary to remove the transcendent nonlinearity⁶ $\sin u$. For a suitable choice of

⁵Code and examples available at <https://github.com/luisacollodi-stud/conservationLaws.git>. Experiments run on a 2.5 GHz Intel Core i5 machine under Windows.

⁶Strictly speaking, the resulting system is not equivalent to the original sine-Gordon equation, as we have replaced $\sin u$ with a generic sinusoid $v = A \sin u + B \cos u$, for A, B arbitrary constants.

| | |
|------------------------------------|---|
| Korteweg-de Vries (KdV) | $u_t = -uu_x - u_{xxx}$ |
| Drinfel'd-Sokolov-Wilson (DSW) | $u_t = -3vv_x$ $v_t = -2uv_x - u_xv - 2v_{xxx}$ |
| Boussinesq (Bou) | $u_t = -v_x$ $v_t = -u_x + 3uu_x + u_{xxx}$ |
| sine-Gordon (s-G) | $u_t = -v$ $v_t = -u_{xx} - s$ $s_x = cu_x$ $c_x = -su_x$ $s_t = cu_t$ $c_t = -su_t$ |
| Euler's incompressible fluid (Eul) | $u_t = -(uu_x + vv_y + ww_z + p_x)$ $v_t = -(uv_x + vv_y + ww_z + p_y)$ $w_t = -(uw_x + vw_y + ww_z + p_z)$ $u_x = -(v_y + w_z)$ |

Table 2.1: PDEs considered in the experiments.

a ranking on \mathcal{D} , each of the considered systems is a Riquier basis where all occurring principal derivatives are also leading. Hence Proposition 1, possibly after expansion of Σ via Lemma 2, implies D -nondegeneracy, for appropriate choices of D . For each system, the leading derivatives are those on the left-hand side of the equalities. In particular, for the sine-Gordon equation we consider the graded lexicographic order on \mathcal{D} induced by $x < t$ and $u < c < s$, in which derivatives are first graded by total degree and then lexicographically:

$$u < c < s < u_x < u_t < c_x < c_t < s_x < s_t < \dots$$

For Euler equations, the (plain or graded) lexicographic order induced by $u > v > w > p$ and $t > x > y > z$ suffices. For the remaining three systems, we consider the following total order on \mathcal{D} :

$$\begin{aligned} & v < v_x < v_{xx} < \dots < u < u_x < u_{xx} < \dots \\ < v_t < v_{tx} < v_{ttx} < \dots < u_t < u_{tx} < u_{ttx} < \dots \\ < v_{tt} < v_{ttx} < v_{tttx} < \dots < u_{tt} < u_{ttx} < u_{tttx} < \dots \end{aligned}$$

This is equivalent to view each derivative ω_τ , with $\omega \in \{u, v\}$ and $\tau \in \{t, x\}^\otimes$, as a monomial ω_τ and then to consider the lexicographic order induced by $t > u > v > x$ on $\{u, v, t, x\}^\otimes$. This is easily checked to be a ranking.

To frame our experiments in the general context of conservation laws methods, we also provide a comparison with the results obtained on the same examples by applying the direct approach [140, Ch.4]. Generally speaking, methods that follow this approach comprise the following two steps: (1) once a set of indeterminates — independent and differential variables — has been fixed, find all multipliers, that is functions depending on those indeterminates, which, when linearly combined with the equations of the system, yield divergence expressions vanishing on its solutions; (2) invert those

divergence expressions to find the corresponding flows. Step (1) is carried out by first applying the variational derivative (a.k.a Euler) operator, which gives conditions on the searched multipliers in terms of linear PDEs, and then solving the resulting PDEs. Step (2) typically relies on homotopy operators, which reduce the inversion problem to computing some 1-dimensional integral. See e.g. [140, 11, 93, 50] for details. Here, we have chosen GEM [50], a state-of-art algorithm also implemented in a Maple package, as a representative of methods based on the direct approach.

Both for GEM and POLYCONS the returned laws depend, of course, on the initially chosen set of indeterminates, say Y . This set is used in very different ways by the two algorithms, though: it serves to build multipliers in GEM, and fluxes in POLYCONS. Moreover, in the case of POLYCONS, one has also to specify a maximum degree of monomials. These differences complicate a direct comparison between the two methods. We prefer to divide the experiments, and the ensuing comparisons, into two parts, depending on whether the same or different sets of indeterminates are used with the two considered algorithms:

1. always use the same set of indeterminates Y with GEM and POLYCONS;
2. when Y is used with POLYCONS, use $Y \cup Y'$ with GEM, where Y' is obtained from the first derivatives of the indeterminates in Y .

The rationale behind the second form of comparison is the following: if a differential indeterminate occurs in the fluxes, it is likely that some of its first derivatives will occur in the resulting divergence expression, hence in its multipliers. This way, the two algorithms will hopefully be compared on a more equal footing.

2.4.1 Using the same set of indeterminates

For each experiment, we fix a finite set of indeterminates $Y \subseteq X \cup \mathcal{P}_a(\Sigma)$ and use Y as an ansatz for GEM, and each of $Z = Y^{\leq d}$, for $d = 2, 3, 4$, as possible ansätze for POLYCONS. The results of these experiments are reported in Table 2.2. In the case of POLYCONS, the returned sets of laws have been checked to be independent up to triviality, in the sense of Remark 2. In addition to the execution time \mathbf{t} and to the overall number \mathbf{n} of independent laws found, the table also displays the number \mathbf{n}^* of extra laws Φ returned by the considered algorithm but not by the other. Here, Φ must be a *genuinely* new law, that is we require that: (a) Φ is not equivalent, up to triviality, to a linear combination of the laws returned by the other algorithm; and (b) that this holds even after S -normalization of the fluxes, that is rewriting Φ to eliminate the principal derivatives.

We see that \mathbf{n}^* for POLYCONS is generally quite large, with a maximum value of 101. On the contrary, \mathbf{n}^* for GEM never exceeds 1. The only reason for a polynomial law not to be found by POLYCONS is that one of its fluxes either has a degree higher than d or contains an indeterminate not in Y . For example, in the case of the Boussinesq equation with $Y = \{u, u_x, u_{xx}, v, v_x\}$ and $d = 3$, the only law returned by GEM and not by POLYCONS is:

$$\left(-\frac{u^3}{2} + \frac{u^2}{2} - \frac{1}{2}uu_{xx} + \frac{v^2}{2}, -\frac{3}{2}u^2v + \frac{u(2v + u_{tx})}{2} - vu_{xx} - \frac{1}{2}u_xu_t \right)$$

which contains two *principal* derivatives not in the chosen Y , that is u_t and u_{tx} . One can eliminate principal derivatives by normalization, that is by applying the function $S(\cdot)$ to both fluxes, thus getting the equivalent law:

$$\left(-\frac{u^3}{2} + \frac{u^2}{2} - \frac{1}{2}uu_{xx} + \frac{v^2}{2}, -\frac{3}{2}u^2v + \frac{1}{2}u(2v - v_{xx}) + \frac{u_xv_x}{2} - u_{xx}v \right).$$

But this law is not among those returned by POLYCONS either, nor is a linear combination of them. In fact, it contains v_{xx} , a parametric derivative which is not in the fixed set Y . A similar case occurs for the sine-Gordon equation. In the case of the Korteweg-de Vries (KdV) equation, the only law returned by GEM and not by POLYCONS has degree 5, higher than the maximum $d = 4$ considered for POLYCONS.

As an example of law found by POLYCONS and not by GEM, consider the case of the KdV equation and $Y = \{t, x, u, u_x, u_{xx}, u_{xxx}, u_{xxxx}\}$. With $d = 4$, exactly 23 extra laws are found, including for example

$$\left(u^3 - 3uu_{xx} - 6u_x^2, \frac{3}{4}u^4 - 12uu_x^2 - 3uu_{xxx} - 9u_xu_{xxx} + 3u_{xx}^2 \right).$$

For another example, consider the Boussinesq equation with $Y = \{t, x, u, u_x, u_{xx}, u_{xxx}, u_{xxxx}, v, v_x, v_{xx}\}$ and $d = 4$. One of the 101 laws found by POLYCONS and not by GEM is

$$\left(u_x^4 + 3uu_x^2u_{xx}, 3uu_xv_{xx} + u_x^3v_x \right).$$

Euler's equations (Eul) govern the flow of an inviscid, incompressible fluid in three spatial dimensions x, y, z and time t : here, $\mathbf{u} = (u, v, w)$ is the fluid's velocity vector and p is its pressure. With $Y = \{u, v, w, p\}$, among the laws found by both GEM and POLYCONS — in the latter case already with $d = 3$ — we have

$$\left(u^2 + v^2 + w^2, 2pu + u^3 + uv^2 + uw^2, 2pv + u^2v + v^3 + vw^2, 2pw + u^2w + v^2w + w^3 \right).$$

This is the conservation of kinetic energy, more commonly expressed in divergence form, vectorially:

$$\frac{\partial}{\partial t} \left(\frac{1}{2}|\mathbf{u}|^2 \right) + \nabla \cdot \left(\frac{1}{2}|\mathbf{u}|^2\mathbf{u} + p\mathbf{u} \right) = 0$$

where $\nabla \cdot$ and $|\cdot|$ are the gradient and 2-norm of a vector, respectively.

Overall, execution times grow as expected with the cardinality of Y and, in the case of POLYCONS, also with the template degree d . Execution times of POLYCONS are generally higher than GEM's. This is compensated by the large number of extra laws n^* found by POLYCONS. The large values of n^* also suggest that a more fair comparison between GEM and POLYCONS should take into account the different roles played by the indeterminates set Y in the two approaches. This is explored in the next subsection.

2.4.2 Using different sets of indeterminates

Given a set of indeterminates Y for POLYCONS, we will let the ansatz Y_{GEM} for GEM consist of Y plus all the first derivatives of the indeterminates in Y , normalized. Explicitly

$$Y_{\text{GEM}} \triangleq Y \cup \{u_\tau : u_\tau \text{ occurs in } S(v_{\xi x_i}), \text{ for some } v_\xi \in Y, x_i \in X\}.$$

The results of the new experiments are reported in Table 2.3. The comparison gives now more equilibrate results in terms of execution times, which are comparable. Yet, POLYCONS still consistently finds more conservation laws than GEM, although n^* never exceeds 11 now. In each case, there is precisely one law returned by GEM and not by POLYCONS: for KdV this is a degree 5 law, in the other four cases the fluxes found by GEM, even after normalization, contain some derivative not belonging to Y .

Overall, the above comparison indicates that, when confining to polynomial laws, GEM and POLYCONS somehow complement with each other. In principle, GEM finds all conservation laws with multipliers built out of a given ansatz of indeterminates. POLYCONS finds all conservation laws with fluxes built out of a given ansatz of indeterminates, up to a given degree. The completeness result of POLYCONS is seemingly less strong, because of the upper bound on the degree, but more direct, because expressed in terms of fluxes. More generally, it is unclear how the completeness of the direct approach in terms of multipliers translates into completeness in terms of fluxes. In other words, it is unclear what is a finite ansatz for multipliers, yielding all laws with polynomial fluxes built out of a given Y . Our attempt with Y_{GEM} falls short of achieving this goal, as seen from the nonzero values of n^* for POLYCONS. Note that, at least at an initial stage of exploration of a PDE system, one might want to reason more naturally in terms of fluxes, rather than in terms of multipliers. In this respect, POLYCONS's higher value of n^* points at least to a practical advantage over the multiplier-based direct approach.

A profiling of the code of POLYCONS shows that the most time-consuming phase of its execution is by far the application of the substitution that solves the linear system to the original flux templates Π (step 4). This suggests that there is much room for improvement, by devising data structures for polynomials that support efficient linear substitution operations.

2.5 Analyticity

The definitions, results and algorithm of sections 2.2 and 2.3 carry over to real analytic solutions of PDEs: essentially, all we have to do is to replace the word “formal” with the word “analytic” throughout the sections. The only difference is in Proposition 1, where we need to strengthen the conditions on Σ to ensure existence of analytic solutions of the involved initial value problems. To this purpose, we shall rely on Riquier's analyticity theorem. We introduce the necessary definitions and the analytic counterpart of Proposition 1 below.

A ranking $>$ over \mathcal{D} is *weakly orderly* if whenever $|\tau| > |\xi|$ then $u_\tau > u_\xi$ for each u ; it is a *Riquier ranking* if whenever $u_\tau > u_\xi$ for some u then $v_\tau > v_\xi$ for all v . A *passive orthonomic* system is a Riquier basis with the following two additional properties: (a) for each element $u_\tau + f \in \Sigma$ (u_τ leading), f does not contain principal derivatives; (b) whenever u_τ and u_ξ are distinct leading derivatives, neither $\tau \leq \xi$ nor $\xi \leq \tau$ hold true. The following is the analytic counterpart of Proposition 1; the straightforward proof is reported in Appendix A.2.

Proposition 2 (analytic D -nondegeneracy). *Let Σ be passive orthonomic w.r.t. a ranking which is Riquier and weakly orderly. Assume $D \cap \text{Pr}(\Sigma) = \text{dom}(\Sigma)$. Then Σ*

is analytically D -nondegenerate. Moreover, Σ is a Groebner basis of $\mathcal{I}(\mathcal{V}(\Sigma))$ w.r.t. a suitable monomial order.

All the examples considered in the previous section are seen to be passive ortho-
nomic for some suitable ranking which is Riquier and weakly orderly. More precisely,
for the sine-Gordon and Euler systems, one considers the same graded lexicographic
rankings introduced in the previous section. On the contrary, the already introduced
ranking for the KdV, DSW and Boussinesq fails to be weakly orderly: one can instead
consider the graded lexicographic ranking induced by $u > v$ and $t > x$. Obviously, a
change in the ranking also leads to a change in the leading derivatives of these systems.

Note that, for a system that is both formally and analytically D -nondegenerate,
the sets of formal and analytic invariant polynomials in the ring \mathcal{P} coincide with
 $\langle \Sigma \rangle$ (Lemma 1). Hence the set $\Pi[V] = CL(\Sigma, Z)$ (Corollary 1), considered in a
formal or analytic sense, is just the same. This gives one some freedom in the choice
of the ranking when it comes to actually computing $\Pi[V]$. Of course, the concrete
representation of $\Pi[V]$, that is the the basis $\Pi[B]$ concretely returned by the algorithm,
does depend on the chosen ranking. Moreover, identifying and filtering equivalent and
trivial laws out from $\Pi[B]$ may be non obvious, in case principal derivatives w.r.t. the
chosen ranking occur in Z (see Remark 2).

2.6 Conclusion

We have put forward a method to compute PDE polynomial conservation laws. Under
a certain nondegeneracy condition, the method is complete, relatively to a user
specified polynomial template for fluxes. Computationally, the proposed method is
based entirely on equational rewriting and linear algebraic operations. This should
be contrasted with the direct approach, that heavily relies on variational tools (Euler
operator), coupled with linear PDE solving and symbolic integration (homotopy).
The simplicity of our method's underlying principles is, we believe, an additional
benefit in terms of the audience that can be reached.

In Section 2.4, we have discussed at length the differences between our work and the
direct approach. Somewhat halfway between our method and the direct approach, one
might place the work of W. Hereman and collaborators based on scaling symmetries
[94, 144]. Like in our case, their starting point is a polynomial template. In their
case, though, the template represents a candidate density, that is a flux corresponding
to time. Moreover, only monomials invariant under the same scaling symmetry of the
PDE are involved in the template. Taking the time total derivative of the candidate
density, they seek conditions on the unknown coefficients for the resulting expression
to be a (spatial) divergence: this is done by equating its variational derivative to zero
and forming a linear system for the unknown coefficients. The solution of the system
is then substituted into the spatial divergence expression, and homotopy operators are
used to recover the spatial fluxes. The methods has limitations, in that only applies to
evolution equations and requires the existence of scaling symmetries.

| Σ | Indeterminates Y | GEM | | | POLYCONS | | | | | | | | |
|----------|---|-----------------|--------------|----------------|-----------------|--------------|----------------|-----------------|--------------|----------------|-----------------|--------------|----------------|
| | | $\mathbf{t(s)}$ | \mathbf{n} | $\mathbf{n^*}$ | $d = 2$ | | | $d = 3$ | | | $d = 4$ | | |
| | | | | | $\mathbf{t(s)}$ | \mathbf{n} | $\mathbf{n^*}$ | $\mathbf{t(s)}$ | \mathbf{n} | $\mathbf{n^*}$ | $\mathbf{t(s)}$ | \mathbf{n} | $\mathbf{n^*}$ |
| KdV | $\{u, u_x, u_{xx}, u_{xxx}\}$ | 1.8 | 3 | 1 | 0.4 | 2 | 1 | 1.5 | 4 | 2 | 3.9 | 6 | 4 |
| | $\{t, x, u, u_x, u_{xx}, u_{xxx}\}$ | 3.7 | 5 | 1 | 0.7 | 2 | 1 | 3.4 | 6 | 4 | 17.6 | 14 | 10 |
| | $\{t, x, u, u_x, u_{xx}, u_{xxx}, u_{xxxx}\}$ | 4.1 | 7 | 1 | 1.1 | 3 | 2 | 6.9 | 11 | 9 | 40.1 | 29 | 23 |
| DSW | $\{u, u_x, v, v_x, v_{xx}\}$ | 2.7 | 2 | 0 | 0.7 | 2 | 1 | 2.7 | 4 | 2 | 8.4 | 5 | 3 |
| | $\{t, x, u, u_x, v, v_x, v_{xx}, v_{xxx}\}$ | 3.3 | 3 | 0 | 1.5 | 3 | 2 | 12.3 | 11 | 9 | 95.0 | 29 | 26 |
| | $\{t, x, u, u_x, u_{xx}, v, v_x, v_{xx}, v_{xxx}, v_{xxxx}\}$ | 18.1 | 4 | 0 | 2.9 | 5 | 4 | 33.1 | 24 | 21 | 458.2 | 78 | 74 |
| Bou | $\{u, u_x, u_{xx}, v, v_x\}$ | 3.3 | 4 | 1 | 0.6 | 4 | 2 | 2.6 | 7 | 3 | 7.8 | 8 | 5 |
| | $\{t, x, u, u_x, u_{xx}, u_{xxx}, v, v_x\}$ | 3.7 | 7 | 1 | 1.4 | 7 | 5 | 10.6 | 20 | 16 | 81.6 | 40 | 34 |
| | $\{t, x, u, u_x, u_{xx}, u_{xxx}, u_{xxxx}, v, v_x, v_{xx}\}$ | 20.7 | 10 | 1 | 2.8 | 13 | 11 | 32.4 | 43 | 36 | 441.7 | 110 | 101 |
| s-G | $\{u, u_x, v, c\}$ | 3.9 | 2 | 0 | 0.4 | 5 | 5 | 1.3 | 8 | 8 | 3.4 | 12 | 10 |
| | $\{u, u_x, u_{xx}, v, v_x, c\}$ | 4.7 | 4 | 1 | 1.1 | 8 | 8 | 6.4 | 14 | 14 | 20.7 | 24 | 21 |
| | $\{t, x, u, u_x, u_{xx}, v, v_x, c\}$ | 9.0 | 4 | 1 | 1.9 | 12 | 12 | 17.4 | 32 | 32 | 89.2 | 69 | 66 |
| Eul | $\{u, v, w, p\}$ | 2.1 | 5 | 0 | 0.04 | 4 | 0 | 2.8 | 5 | 0 | 8.7 | 5 | 0 |

Table 2.2: Comparison between GEM and POLYCONS using the same set of indeterminates Y . Legenda: \mathbf{t} = execution time in seconds, \mathbf{n} = number of independent laws found, $\mathbf{n^*}$ = number of independent laws found by one algorithm but not by the other (see text), d = degree of the complete polynomial template for POLYCONS.

| Σ | GEM | | POLYCONS | | | | | |
|----------|--|------------------|---------------------------------------|------------------|------------------|------------------|--|--|
| | Indet. Y_{GEM} | t(s) n n* | Indet. Y | $d = 2$ | $d = 3$ | $d = 4$ | | |
| | | | | t(s) n n* | t(s) n n* | t(s) n n* | | |
| KdV | $\{u, u_x, u_{xx}, u_{xxx}, u_{xxxx}\}$ | 2.6 3 1 | $\{u, u_x, u_{xx}, u_{xxx}\}$ | 0.5 2 1 | 1.3 4 2 | 3.1 6 4 | | |
| DSW | $\{u, u_x, u_{xx}, v, v_x, v_{xx}, v_{xxx}, v_{xxxx}\}$ | 12.3 3 1 | $\{u, u_x, v, v_x, v_{xx}, v_{xxx}\}$ | 1.0 3 2 | 5.3 7 5 | 24.0 12 10 | | |
| Bous | $\{u, u_x, u_{xx}, u_{xxx}, u_{xxxx}, v, v_x, v_{xx}\}$ | 13.0 4 1 | $\{u, u_x, u_{xx}, u_{xxx}, v, v_x\}$ | 0.8 5 3 | 4.3 10 7 | 19.5 14 11 | | |
| s-G | $\{u, u_x, u_{xx}, v, v_x, c\}$ | 4.0 4 1 | $\{u, u_x, v, c\}$ | 0.2 5 5 | 1.1 8 8 | 3.2 12 11 | | |
| Eul | $\{u, v, w, p, u_y, u_z, v_x, v_y, v_z, w_x, w_y, w_z, p_t, p_x, p_y, p_z\}$ | 3.5 6 1 | $\{u, v, w, p\}$ | 0.04 4 0 | 2.8 5 0 | 8.7 5 0 | | |

Table 2.3: Comparison between GEM and POLYCONS with different indeterminates, Y and Y_{GEM} . Legenda: see Table 2.2.

Chapter 3

Linearization and reachability

3.1 Overview

The analysis of systems of nonlinear ordinary differential equations poses formidable challenges to theoreticians and practitioners. Among the great variety of existing techniques, many concentrate on specific properties, for instance stability (e.g. via Lyapunov functions [104, Ch.4]) and safety (e.g. via barrier certificates [146, 106, 86]). Other techniques focus on computing detailed, effective descriptions of the set of reachable states over a given time horizon, possibly taking into account uncertainties on the initial states, see e.g. [176, 54, 46, 47, 8, 9] and references therein. These descriptions, variously called reachsets, flowpipes etc., are typically obtained in a piecewise fashion; that is, by sewing together local approximations over different regions of the state space and/or time. Here, we are interested in: (a) conditions and methods by which a single linear approximation of a system can be computed that can be accurate also non locally; (b) understanding if such approximations can be leveraged in reachability analysis.

Given a nonlinear system of ODEs in the state variables $x = (x_1, \dots, x_n)^T$

$$\dot{x} = f(x_1, \dots, x_n) \tag{3.1}$$

(see Section 3.2 for precise definitions), approximation can take place either in space, like when linearizing the system's equations around a point $x = x_0$; or in time, like when Taylor expanding the ODE's solution around a time $t = t_0$. With traditional methods, the resulting description will typically exhibit only a limited, *local* accuracy: the quality of the approximation will tend to get very bad as one gets away from $x = x_0$ and/or $t = t_0$.

Our goal is to devise approximations of nonlinear systems that can be accurate also non-locally. In particular, under suitable assumptions, accuracy should remain good over a long, possibly infinite time horizon. In our method, a crucial step in achieving this goal is the computation of a 'small', hence computationally tractable, linear ODE system that approximates (3.1). In perspective, this linear system might replace the original system not only for the purpose of global reachability analysis, but also for tasks such as runtime verification [129, 161].

The proposed technique is related to *Carleman embedding* [18, 110], which is used to transform a given nonlinear system like (3.1) into an *infinite* linear system (Section

3.3). For the purpose of effective computation, this infinite system is truncated at a finite cut off, obtaining a linear system

$$\dot{z} = Az \tag{3.2}$$

of dimension M , typically with $M \gg n$. The z variables represent the (approximate) evolution of certain functions of the original state variables x , say $z = \alpha(x) = (\alpha_1(x), \dots, \alpha_M(x))^T$. The elements of α are chosen in such a way that an *observable* of interest of the state x , say $g(x)$, can be expressed as a linear combination of them. In typical cases, the α_j 's will be monomial functions and g a linear combination of them. One is interested in the observable dynamics $g(t) := g(x(t))$.

In order to achieve dimension reduction, we blend Carleman embedding with Krylov orthogonal projection techniques [160] (Section 3.4). Basically, working in the z -coordinate space, we reduce the system (3.2) via projection of the matrix A onto an appropriate subspace of dimension $m \ll M$, thus obtaining a reduced linear ODE system of dimension m . From the reduced system, approximations $\hat{x}(t)$ of the exact solutions of the original system (3.1) can be readily obtained. Composing the original observable g and the approximate solution $\hat{x}(t)$ yields $\hat{g}(t)$, which we call *reduced observable dynamics*. Distinctive features of the proposed approximation scheme are the following.

- (a) The equations of the reduced linear ODE system, while depending on the given observable function g , are independent of the initial state. In fact, the error between the exact and reduced observable dynamics is guaranteed to be $O(t^m)$ near $t = 0$ for *any* given initial condition. Moreover, the reduced linear system can be computed without having to store the whole matrix A , which can be quite large.
- (b) Under suitable stability conditions, concrete and sharp error bounds can be given over finite time horizons (Section 3.5). In a special case, even an infinite time horizon can be considered.

Our work is also related to the Koopman approach [124], where the system's dynamics is lifted from the state space X to a higher dimensional space of observables, smooth functions $X \rightarrow \mathbb{R}$. The advantage of doing so is that the dynamics becomes linear in the space of observables, although the dimension of this space is infinite. Under suitable conditions, one can decompose the description of the system's dynamics in the observable space as a (in general, infinite) linear combination of eigenfunctions (Koopman spectral decomposition), thus obtaining important qualitative and quantitative insight into the system. We study the formal relationship between our approach and Koopman's (Section 3.6), and prove that, under a rather natural assumption, our reduced observable dynamics $\hat{g}(t)$ coincides with a reduced Koopman spectral decomposition, $\hat{g}_K(t)$.

Next, we leverage the above discussed features (a) and (b) of our approach in an algorithm to compute overapproximations of reachsets of the original nonlinear system (3.1), at specified times over a finite time horizon (Section 3.7). The resulting algorithm CKR (for Carleman-Krylov Reachability) works in the general, not necessarily stable case. The basic idea is to perform advection of the vertices of an initial convex set (polytope), relying on the reduced, linearized system rather than on (3.1).

Similarly to other proposals [54, 128, 184], compensation of the errors resulting from nonlinearities is reduced to an optimization problem. Experiments¹ conducted with a proof-of-concept implementation of both the approximation scheme and the reachability algorithm have given encouraging results, also in comparison to recently proposed approaches based on Carleman linearization (Section 3.8). Concerning reachsets, we offer a comparison with state-of-the-art reachability tools, on a few examples drawn from the literature. A few concluding remarks are reported in the final section (Section 3.9).

To sum up, our main contributions are the following:

1. Integration of Carleman linearization and Krylov dimension reduction into a technique to compute a reduced observable dynamics.
2. For the reduced observable dynamics, error bounds that are practically less conservative than those obtained in other approaches based on Carleman linearization.
3. Application of the reduced observable dynamics to the design of CKR, a general purpose reachability algorithm with good accuracy.
4. A proof that the reduced observable dynamics coincides with a reduced Koopman spectral decomposition.

Related work There exists a vast literature on the linearization of nonlinear systems. In particular, techniques based on Carleman embedding [18, 110] and the Koopman approach [124] have recently received a renewed attention. Most related to our work and motivations, Jungers and Tabuada [103] have recently proposed a technique for global approximation of nonlinear ODES by linear ODES, based on *polyflows*. These are systems that are exactly linearizable via a change of variables. The only systems admitting exact polyflow solutions are those where the set of all Lie derivatives of the state variables w.r.t. the vector field f form a finite-dimensional vector space: hence they can fundamentally be regarded as linear systems in a higher dimensional space. The technique in [103] is based on building polyflows that approximate the original system, using as a basis the Lie derivatives up to some order N ; the resulting system plays a role somewhat similar to the truncated Carleman embedding (3.2). As $N \rightarrow +\infty$, the approximation of [103] becomes exact in the interval of convergence of the Taylor expansion of the solution for any given x_0 . Note that this is an asymptotic result that does not easily yield concrete bounds for a fixed N . On the contrary, our results provide concrete error bounds for any fixed m and finite time horizon — and also for an *infinite* time horizon under suitable stability assumptions. Systems that are exactly linearizable via polynomial changes of variables are the subject of [163, 167, 28]. In [28] we have considered Carleman embedding and Krylov-based approximations, essentially from a local point of view. Here, we provide novel analyses of both local and global errors, and leverage them in CKR, a new reachability algorithm.

General error bounds for the truncated Carleman linearization have been recently considered in [10, 72]. The time interval of validity of these bounds is quite small,

¹The datasets analysed during the current study are available in the Github repository <https://github.com/Luisa-unifi/CKR>.

contrary to ours; moreover, in practice they appear to be significantly more conservative than ours. This is further discussed in Section 3.8, also based on experimental evidence. In [73], an efficient reachability analysis algorithm relying on Carleman linearization is presented, limited to the class of weakly nonlinear, dissipative systems. We do not have such restrictions, but it is well possible that the approach in [73] is more effective than ours limited to the mentioned class of systems. Dimension reduction is not considered in any of [72, 10, 73].

As mentioned earlier, our work is also related to the Koopman approach [124]. In this framework, global analysis techniques have recently emerged, see in particular [123]. Our method too is centered on a basis of observable functions, the aforementioned α , but our goal in the present work is quite specific, with an emphasis on finite dimensionality, error bounds that are valid over a prescribed time horizon and reachability. The formal relationship between our approach and Koopman's is discussed in detail in Section 3.6.

There is also a vast literature on model reduction techniques. A reduction technique particularly close to ours is the one described in [38]. It is based on approximate differential equivalence and has been successfully compared with CORA. Essentially, a partition of the variables set of the original model is generated and each variable of the reduced model is associated with an ODE giving the dynamics of the sum of the variables in a partition block. A threshold $\epsilon > 0$ is also considered in the partition process in order to capture parameter uncertainty and error tolerances, this leads to approximation error. Similarly to our method, the approach of [38] applies to non linear ODEs and the error is bounded formally in terms of over-approximations of the reachable set. However, the reduced model they get is non linear and relies on a different theoretical basis. In addition, differently from our case, they also admit sources of uncertainty about the parameters or related to finite-precision measurements, and thus, the corresponding error is caused by the model uncertainty and not specifically by the reduction process.

In the field of tools for continuous and hybrid systems, like Flow* [47] and CORA [8], a mix of approximations techniques are employed [120, 46, 47, 8, 7, 9]. In particular, Flow* [46, 47] is based on Taylor models: Picard iteration is used to obtain a polynomial of given degree that approximates the exact solution over time with respect to uncertain initial states. CORA relies on a mix of techniques, including linearization of the nonlinear ODE equations at various points in the state space [7, 9]. Here we are primarily interested in approximations that are accurate for as long as possible, and in connecting them to reachability analysis. As argued empirically in Section 3.8, our approach brings significant benefits in terms of accuracy.

Structure of the chapter The rest of the chapter is organized as follows. Section 3.2 introduces some basic definitions, such as nonlinear system and Lie derivative. Section 3.3 presents Carleman linearization and its truncation. In Section 3.4 a dimension reduction method based on Krylov spaces is put forward, and results to bound the resulting local error are given. Section 3.5 studies the global error qualitatively, both in the presence and in the absence of stability. Section 3.6 establishes a formal relation of our approach with the Koopman approximation framework. Section 3.7 presents CKR, a reachability algorithm based on the results in sections 3.3 and 3.4. Section 3.8 is entirely devoted to illustrate some experimental results. A few concluding remarks

are drawn in Section 3.9. For the sake of readability, computational details and proofs have been confined to a separate appendix (Appendix B).

3.2 Preliminaries

For $x = (x_1, \dots, x_n)^T$ a vector of state variables, we consider a system of ODES

$$\dot{x} = f(x) \quad (3.3)$$

where $f = (f_1, \dots, f_n)^T$ is a vector field of locally Lipschitz analytic functions defined on some open subset $\Omega \subseteq \mathbb{R}^n$. For $x_0 \in \Omega$, we let $x(t; x_0)$ be the unique solution of the ODE system with the initial condition $x(0) = x_0$: the unique solution exists and is real analytic in an open interval containing the origin $t = 0$ (Picard-Lindelöf theorem).

For a real analytic function g defined on some open subset of \mathbb{R}^n that includes the trajectories $x(t; x_0)$ for $x_0 \in \Omega$, we will be interested in studying the *observable* of the system (3.3) via g , that is the function $g \circ x(t; x_0)$.

Recall that $\mathcal{L}_f(g) := \langle \nabla g, f \rangle = \sum_{j=1}^n \frac{\partial g}{\partial x_j} \cdot f_j$ is the Lie derivative of g (w.r.t. f), and $\mathcal{L}_f^{(k)}(g)$ is the k -th Lie derivative, defined inductively by $\mathcal{L}_f^{(k+1)}(g) := \mathcal{L}_f(\mathcal{L}_f^{(k)}(g))$. We shall omit the subscript f whenever it is understood from the context.

3.3 Carleman linearization

In this section, we introduce a method of linearization of the system (3.3) which is strongly related to Carleman embedding [110]. Generally speaking, one can apply the following method to $g(x(t; x_0))$, for any suitable observable function g , so we will describe the method in terms of such a generic g . As we will see, for the purposes of building approximate solutions, it will be sufficient to apply the results to each of the n identity functions $g = x_i$, for $i = 1, \dots, n$ in turn.

Let us fix a set $\mathcal{A} = \{\alpha_1, \alpha_2, \dots\}$ of functions $\alpha_i : \mathbb{R}^n \rightarrow \mathbb{R}$. For instance \mathcal{A} might be all monomial functions. We assume that, over its domain of definition, the observable function g can be represented in a *unique* way as a linear combination of functions from \mathcal{A} up to a cutoff $M > 0$. In other words, we assume there are unique $v = (\lambda_1, \dots, \lambda_M)^T \in \mathbb{R}^M$ and basis vector $\alpha := (\alpha_1, \dots, \alpha_M)^T$ such that

$$g = \sum_{i=1}^M \lambda_i \alpha_i = v^T \alpha. \quad (3.4)$$

Otherwise, all we require from the functions in \mathcal{A} is that they are analytic², and that the Lie derivative of each α_i can in turn be expressed as a unique linear combination of elements from \mathcal{A} . That is, for each $i \geq 1$, there is a unique sequence of real coefficients a_{ij} ($j \geq 1$) such that

$$\mathcal{L}(\alpha_i) = \sum_{j \geq 1} a_{ij} \alpha_j. \quad (3.5)$$

²This can be weakened to analyticity in some open set containing all the trajectories $x(t; x_0)$ for $x_0 \in \Omega$.

For the sake of simplicity, we shall assume that, for each i , only finitely many coefficients a_{ij} here are nonzero; this assumption is true e.g. for g a polynomial and \mathcal{A} equal to the set of all monomials. We let A denote the $M \times M$ matrix of the coefficients a_{ij} for $1 \leq i, j \leq M$, and B be the $M \times k$ matrix of possibly nonzero elements $b_{i,j} = a_{i,M+j}$; that is, k is chosen large enough to ensure that, for $1 \leq i \leq M$, we have $a_{ij} = 0$ for each $j > M + k$. We let $\psi \triangleq (\alpha_{M+1}, \dots, \alpha_{M+k})^T$.

For any fixed initial condition $x_0 \in \Omega$ of the original system (3.3), we can form the linear system of ODES and the initial condition described below. Note that for each fixed $x_0 \in \Omega$, $\psi(x(t; x_0)) : I \rightarrow \mathbb{R}^k$ is a real analytic function of t defined in an interval I containing the origin. This function will in general not be explicitly available, as it depends on the solution $x(t; x_0)$. The *Carleman linearization* (or embedding) of (3.3) is given by the following linear, non-autonomous system in the variables $z = (z_1, \dots, z_M)^T$ and initial condition

$$\dot{z} = Az + B\psi(x(t; x_0)) \quad (3.6)$$

$$z(0) = \alpha(x_0) =: z_0. \quad (3.7)$$

The following result is an almost immediate consequence of the existence and uniqueness of the solution of ODES (Picard-Lindelöf). For a detailed proof, see [28, Th.3].

Theorem 1 (Carleman linearization). *Let $x_0 \in \Omega$. Then $\alpha(x(t; x_0))$ is the unique solution of the system (3.6) with $z(0)$ as in (3.7).*

Note that we cannot *explicitly* build the system (3.6), as the function $\psi(x(t; x_0))$ is in general not available – even when ψ and B are available. In the next section, this will lead us to consider an approximation where we neglect the “remainder” $\psi(x(t; x_0))$, the *truncated* Carleman linearization of dimension M

$$\dot{z} = Az \quad z(0) = z_0 (= \alpha(x_0)). \quad (3.8)$$

We illustrate the truncated Carleman linearization with the following example, an instance of the Van der Pol oscillator (VdP, see [178]). This system is used as a benchmark in a number of papers on reachability for nonlinear ODES.

Example 4. Consider the system $\dot{x} = f$ where $f := (x_2, -x_1 + x_2(1 - x_1^2))^T$. We fix as \mathcal{A} the set of all monomial functions, and $\alpha = (x_1, x_2, x_1^2 x_2)^T$, hence $M = 3$ is the dimension. It is immediate to check that $A = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix}$. Therefore, letting $z = (z_1, z_2, z_3)^T$, for each $x_0 \in \mathbb{R}^2$, the truncated Carleman linearization of dimension 3 is: $\dot{z} = Az$ with $z(0) = \alpha(x_0)$.

In cases arising in applications, the matrix A can in practice be too large to be explicitly generated. Thus (3.8) will be the starting point to build a further approximation and reduction, as detailed in the next section.

3.4 Dimension reduction via Krylov projection

We discuss to reduce the dimension of (3.8) while keeping certain, still local, accuracy guarantees. The discussion in this section expands that in [28, Sect.4]. The basic

idea is to reduce the dimension of the problem by projecting A onto an appropriate subspace of \mathbb{R}^M of dimension $m \ll M$. The differential equations of the reduced linear system will depend on g , but not on the initial state x_0 . The method is amenable to an “on the fly” implementation, in the following sense: it only requires building the Lie derivatives of g until a prescribed order m . In detail, the order m coincides with the dimension of the obtained linear system of ODES and give rise to approximate solutions of (3.3) that are locally accurate. The behaviour of the global error will be discussed in Section 3.5.

Recall that our goal is to approximate a target function $g(x(t; x_0))$. For the sake of notation, we will adopt the following abbreviation for this function:

$$g(t; x_0) := g(x(t; x_0)). \quad (3.9)$$

Fix $m \geq 1$ and order the elements in \mathcal{A} in such a way the the first M functions, $\alpha = (\alpha_1, \dots, \alpha_M)^T$ are those appearing in the (unique) decompositions of the Lie derivatives of g from 0 through $m - 1$: that is, for each $j = 0, \dots, m - 1$ there is a (unique) vector $u_j \in \mathbb{R}^M$ such that $\mathcal{L}^{(j)}(g) = u_j^T \alpha$; here $u_0 = v$. We assume without loss of generality that $m \leq M$ (typically, $m \ll M$). From (3.5) and from the definitions of the matrices A , B and of the functions α and ψ , it follows that, componentwise

$$\mathcal{L}(\alpha) = A\alpha + B\psi. \quad (3.10)$$

From the definition (3.4) of g and the linearity of $\mathcal{L}(\cdot)$, it follows that $\mathcal{L}(g) = v^T A\alpha + v^T B\psi$. From the assumed uniqueness of the decomposition of Lie derivatives in \mathcal{A} , we have that $v^T A = u_1^T$ and $v^T B = 0$ hence

$$\mathcal{L}(g) = v^T A\alpha.$$

Taking the Lie derivative of the above equation, we have $\mathcal{L}^{(2)}(g) = v^T A(A\alpha + B\psi) = v^T A^2\alpha + v^T AB\psi$, where $v^T AB = 0$ again as a consequence of the uniqueness of the decomposition of $\mathcal{L}^{(2)}(g)$ in \mathcal{A} . Proceeding similarly for the subsequent derivatives, that is iterating (3.10) and exploiting the linearity of $\mathcal{L}(\cdot)$ and the uniqueness of the decomposition of $\mathcal{L}^{(j)}(g)$ in \mathcal{A} , for $0 \leq j \leq m - 1$, we arrive at the following conclusions.

$$v^T A^j \alpha = \mathcal{L}^{(j)}(g) \quad (0 \leq j \leq m - 1) \quad (3.11)$$

$$v^T A^{j-1} B = 0 \quad (1 \leq j \leq m - 1). \quad (3.12)$$

Now, we consider the m -dimensional Krylov space³ generated by v and A^T , that is the subspace of \mathbb{R}^M

$$\mathcal{K}_m := \text{span}\{v, A^T v, (A^T)^2 v, \dots, (A^T)^{m-1} v\}. \quad (3.13)$$

Comparing (3.13) and (3.11), we see that \mathcal{K}_m is the subspace of \mathbb{R}^M spanned by the (column) vectors of the coefficients of the Lie derivatives of g from 0 through $m - 1$. Here we assume without loss of generality that $v \neq 0$ and that \mathcal{K}_m has dimension m — that is, m is small enough that the m vectors listed on the right-hand side of (3.13) are linearly independent. Let $V = [v_1 | \dots | v_m]$ be an orthonormal basis of

³For an introduction to Krylov spaces, see e.g. [160].

\mathcal{K}_m , represented as a $M \times m$ matrix (see at the end of the section for computational considerations). Now consider the projection of A^T onto \mathcal{K}_m and represent it w.r.t. the basis V , in other words we consider the $m \times m$ matrix

$$H_m := V^T A^T V. \quad (3.14)$$

Given a vector of m distinct state variables $y = (y_1, \dots, y_m)^T$, we let the *reduced* linear system derived from (3.6) and the corresponding initial condition, derived from (3.7), be defined as:

$$\begin{aligned} \dot{y} &= H_m^T y \\ y(0) &= V^T z_0 =: y_0. \end{aligned} \quad (3.15)$$

Note that the reduced equations (3.15) do not depend on $x_0 \in \Omega$. Informally speaking, the solution $y(t; y_0)$ of the reduced system describes the evolution of the vector $\alpha(x(t; x_0))$, projected onto the subspace \mathcal{K}_m , in the coordinates of the basis V . Recalling that $v \in \mathbb{R}^M$ is the vector of the coefficients of g with respect to α , as in (3.4), it is then natural to consider the following approximation of $g(t; x_0)$.

Definition 4 (reduced observable dynamics). *For each $x_0 \in \Omega$ and $y_0 = V^T x_0$, we define the function:*

$$\hat{g}(t; x_0) := v^T V y(t; y_0). \quad (3.16)$$

In fact, we will see that $v_1 = v/\|v\|_2$, while v is orthogonal to v_j for $j > 1$. Hence (3.16) can be simplified to

$$\hat{g}(t; x_0) = \|v\|_2 y_1(t; y_0). \quad (3.17)$$

In order to study the quality of this approximation, we introduce the error function relative to g

$$\epsilon_g(t; x_0) := g(t; x_0) - \hat{g}(t; x_0). \quad (3.18)$$

The following result confirms that this error is small near $t = 0$. Indeed, the Taylor expansions of $\hat{g}(t; x_0)$ and $g(t; x_0)$ up to order $m - 1$ coincide: this is a consequence of the fact that the coordinates (in α) of the Lie derivatives of g from 0 to $m - 1$ span \mathcal{K}_m .

Theorem 2. *For each $x_0 \in \Omega$, the function $\epsilon_g(t; x_0)$ is $O(t^m)$ around $t = 0$.*

PROOF. Fix any $0 \leq j \leq m - 1$. Note that $\frac{d^j}{dt^j} g(t; x_0)|_{t=0} = \mathcal{L}^{(j)}(g)|_{x=x_0} = \alpha^T(x_0) A^{Tj} v$, where in the last step we have applied (3.11). On the other hand, from the definition of $\hat{g}(t; x_0) = v^T V y(t; y_0)$ and (3.15), we have: $\frac{d^j}{dt^j} \hat{g}(t; x_0)|_{t=0} = v^T V H_m^{Tj} V^T \alpha(x_0) = \alpha^T(x_0) V H_m^j V^T v = \alpha(x_0)^T V V^T A^{Tj} V V^T v$, where in the last step we have applied the definition (3.14) of H_m , and the fact that $V^T V = I$. Now, $V V^T v = v$, because $v \in \mathcal{K}_m$, and $V V^T A^{Tj} v = A^{Tj} v$, again because $A^{Tj} v \in \mathcal{K}_m$. Therefore $\frac{d^j}{dt^j} \hat{g}(t; x_0)|_{t=0} = \alpha(x_0)^T V V^T A^{Tj} V V^T v = \alpha^T(x_0) A^{Tj} v = \frac{d^j}{dt^j} g(t; x_0)|_{t=0}$. This proves that the Taylor coefficients of $\epsilon_g(t; x_0)$ from 0 through $m - 1$ are zero. \square

Explicit local bounds of the error function can be obtained from the Taylor theorem with remainder in Lagrange form, assuming we can construct validated enclosures S and E of $x(t; x_0)$ and $y(t; y_0)$, respectively — which for

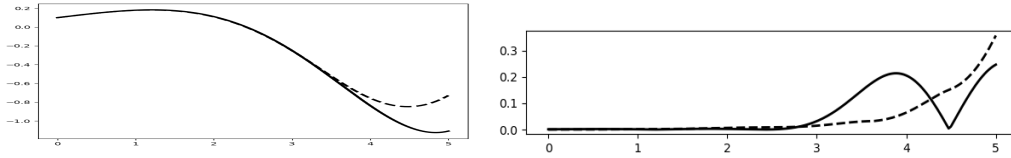


Figure 3.1: For Example 5, left: $g(t; x_0)$ and $\widehat{g}(t; x_0)$ 5; right: $|h(\tau; x_0)|$ and bound (3.23).

small t is possible by standard techniques, see e.g. [136] and references therein. We state the result in a form suitable for application to reachability analysis, where an initial set X_0 is explicitly considered. Below, we let $\rho_{X_0} := \inf_{x_0 \in X_0} \{\rho : \rho \text{ is the radius of convergence of the Taylor series of } \epsilon_g(t; x_0) \text{ from } t = 0\}$.

Corollary 2. *Consider a set $X_0 \subseteq \Omega$. Fix t s.t. $\rho_{X_0} > t > 0$ and compact sets $S \subseteq \mathbb{R}^n$ and $E \subseteq \mathbb{R}^m$ such that $X_0 \subseteq S$, $V^T X_0 \subseteq E$ and for each $(\tau, x_0) \in [0, t] \times X_0$ we have $x(\tau; x_0) \in S$ and $y(\tau; y_0) \in E$, where $y_0 = V^T \alpha(x_0)$. Define*

$$\gamma^-(t; S, E) := \frac{t^m}{m!} \left(\min_{x \in S} \mathcal{L}^{(m)}(g)(x) - \max_{y \in E} v^T V (H_m^T)^m y \right) \quad (3.19)$$

$$\gamma^+(t; S, E) := \frac{t^m}{m!} \left(\max_{x \in S} \mathcal{L}^{(m)}(g)(x) - \min_{y \in E} v^T V (H_m^T)^m y \right). \quad (3.20)$$

Then for each $x_0 \in X_0$ and $\tau \in [0, t]$, $\gamma^-(t; S, E) \leq \epsilon_g(\tau; x_0) \leq \gamma^+(t; S, E)$.

PROOF. Corollary 2 For each $x_0 \in X$, we have $\epsilon_g(t; x_0) = g(t; x_0) - \widehat{g}(t; x_0)$ and the Taylor series of $g(t; x_0)$ and $\widehat{g}(t; x_0)$ coincide up to and including order $m - 1$ (Theorem 2). Hence, for any t in the interval of convergence of this series, $\epsilon_g(t; x_0)$ equals the difference of the Lagrange remainders of $g(t; x_0)$ and of $\widehat{g}(t; x_0)$, say L_m and \widehat{L}_m . In other words, for each $\rho_{X_0} > t > 0$, there are $t_1^*, t_2^* \in (0, t)$ s.t.

$$\begin{aligned} \epsilon_g(t; x_0) &= L_m(t; t_1^*) - \widehat{L}_m(t; t_2^*) \\ &= \frac{t^m}{m!} g^{(m)}(t_1^*; x_0) - \frac{t^m}{m!} \widehat{g}^{(m)}(t_2^*; y_0) \\ &= \frac{t^m}{m!} \mathcal{L}^{(m)}(g)(x(t_1^*; x_0)) - \frac{t^m}{m!} v^T V (H_m^T)^m y(t_2^*; y_0) \end{aligned} \quad (3.21)$$

where the last equality follows from: (a) $\frac{d^m}{dt^m} g(x(t; x_0))|_{t=t_1^*} = \mathcal{L}^{(m)}(g)(x(t_1^*; x_0))$; and (b) differentiating m times the function $\widehat{g}(t; x_0) = v^T V y(t; y_0)$ by applying (3.15), that is $\dot{y} = H_m^T y$. By assumption, $x(t_1^*; x_0) \in S$ and $y(t_2^*; y_0) \in E$. Hence, by replacing in (3.21) these two expressions with generic $x \in S$ and $y \in E$, and then taking the $\max_{x \in S}$ and the $\min_{y \in E}$, we obtain the inequality $\epsilon_g(t; x_0) \leq \gamma^+(\dots)$ in the statement. The other inequality is obtained similarly. \square

There exists a well-known algorithm for the efficient, “on the fly” construction of the matrices V, B, H_m , the Arnoldi iteration [160] (see Appendix B).

Example 5. Let us consider again the VdP system in Example 4 and let us build the reduced linear system (3.15) for $m = 2$ and $g = x_1$. The choice of $\alpha = (x_1, x_2, x_1^2 x_2)^T$ guarantees that g and $\mathcal{L}(g) = \mathcal{L}^{(m-1)}(g)$ are both representable w.r.t. α , as required: in particular, $g = \alpha^T v_1$ and $\mathcal{L}(g) = \alpha^T v_2$ with $v = v_1 := (1, 0, 0)^T$ and $v_2 := (0, 1, 0)^T$. Thus $\mathcal{K}_m = \text{span}\{v_1, v_2\}$ has $V := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}^T$ has a basis, and this is

already orthonormal. We define $H = V^T A^T V = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$, where A is the same as in Example 4. Writing $x_0 = x = (x_1, x_2)^T$ for a generic initial state, we have $\hat{g}(t; x) = v^T V y(t; y_0) = y_1(t; y_0)$, where $y(t; y_0)$ is the solution⁴ of (3.15) with initial condition $y_0 := V^T \alpha(x) = (x_1, x_2)^T$. For $x_0 = (0.1, 0.1)^T$, we plot the exact $x_1(t; x_0)$ (dashed) and approximate $\hat{g}(t; x_0)$ (solid) solutions for $t \in [0, 5]$ in Fig. 3.1, left.

3.5 Behaviour of the global error

We study, mostly from a qualitative point of view, the behaviour of the error function ϵ_g . In what follows, we will assume the orthonormal basis $V = [v_1 | \dots | v_m]$ of the Krylov space \mathcal{K}_m is generated via the Arnoldi Algorithm. This means that the vectors v_j are an orthonormalized version of the vectors $(A^T)^j v$ in (3.13), inductively built as follows: $v_1 := v / \|v\|_2$ and $v_j := w_j / \|w_j\|_2$ for $j = 2, \dots, m$, where $w_j := A^T v_{j-1} - \sum_{k=1}^{j-1} \mu_k v_k$ with $\mu_k := \langle A^T v_{j-1}, v_k \rangle$. We will let r_m denote the projection of $A^T v_m$ onto \mathcal{K}_m^\perp , the orthogonal complement of \mathcal{K}_m . Explicitly: $r_m := A^T v_m - V V^T A^T v_m$. We define the *remainder function* $h : \mathbb{R}^n \rightarrow \mathbb{R}$ as follows for $x \in \mathbb{R}^n$:

$$h(x) := v_m^T B \psi(x) + r_m^T \alpha(x). \quad (3.22)$$

We have seen that $\hat{g}(t; x)$ represents faithfully $g(t; x)$ up to order $m - 1$ (Theorem 2). Informally speaking, the remainder function h has two error terms, corresponding to whatever of the m -th derivative of $g(t; x)$ cannot be represented: either because it involves elements ψ of \mathcal{A} outside α (term $v_m^T B \psi(x)$), or because it falls outside \mathcal{K}_m (term $r_m^T \alpha(x)$). One's hope here is that $|h(x)|$ is small when computed along the trajectories of $x(t; x_0)$, for x_0 in the initial set. The following theorem provides a general error bound in terms of $h(x)$. We say that a real square matrix is *stable* if all its eigenvalues have a nonnegative real part, and every purely imaginary eigenvalue, if any, has geometric multiplicity equal to the algebraic one (or, equivalently, in the Jordan decomposition of the matrix, the block corresponding to any such eigenvalue has order 1; cf. e.g. the proof of [104, Th.4.5]).

Theorem 3 (global error bound). *For any $t > 0$ such that $x(\tau; x_0)$ is defined for $\tau \in [0, t]$:*

$$|\epsilon_g(t; x_0)| \leq \|v\|_2 \int_0^t |h(x(\tau; x_0))| \cdot |(e^{(t-\tau)H_m^T})_{1,m}| d\tau. \quad (3.23)$$

If additionally H_m is stable then there is a constant $D > 0$ independent of t such that

$$|\epsilon_g(t; x_0)| \leq \|v\|_2 D \int_0^t |h(x(\tau; x_0))| d\tau. \quad (3.24)$$

Qualitatively speaking, (3.24) says that, for a stable H_m , the behaviour of the global error is determined by $|h(x(\tau; x_0))|$: if this function decays fast enough to be integrable over $[0, +\infty)$, then $\epsilon_g(t; x_0)$ will be globally bounded. In Appendix B, we also report a special case of this situation that can be characterized analytically (Corollary B.2.1).

If H_m is not stable, (3.23) still applies. In this case, the norm of the matrix exponential $e^{(t-\tau)H_m^T}$ will eventually dominate, making the bound useless for large t .

⁴An explicit expression for y_1 is: $y_1(t; y_0) = -1/3 e^{t/2} (\sqrt{3}(x_1 - 2x_2) \sin(\frac{1}{2} t \sqrt{3}) - 3x_1 \cos(\frac{1}{2} t \sqrt{3}))$.

Yet, there may be a time horizon within which $|h(x(\tau; x_0))|$ and/or the exponential are small enough to make the bound (3.23) useful. This will be typically the case if $x(t; x_0)$ hence $|h(x(\tau; x_0))|$ are bounded, for instance in systems that exhibit a limit-cycle behaviour, like VdP. For Example 5, we plot $|h(x(t; x_0))|$ (solid) and the right-hand side of (3.23) (dashed) for $t \in [0, 5]$ in Fig. 3.1, right.

These considerations prompt for use of the approximation $\hat{g}(t; x_0)$ inside a scheme for reachability analysis. As for error control, the evaluation of the upper bounds (3.23) and (3.24) requires knowledge of the solution $x(t; x_0)$, or at least of a bound on its norm on an interval of interest, which are in general not available. However, useful bounds can be obtained from enclosures of the solution taken at successive, small time intervals. This will be developed in Section 3.7. As an interlude, in the next section we clarify the relation of the present approach with Koopman's.

3.6 Relation with the Koopman approach

There is a rich and deep theory on the Koopman approach, a recent survey of which can be found in the book [124]. Our aim here is merely to establish a precise formal relationship between our approach and Koopman's. The bottom line is that, under certain conditions, our reduced observable dynamics (Definition 4) coincides with a reduced form of Koopman spectral decomposition (cf. Proposition 4). In the rest of the section, by $x(t; x_0)$ we denote the unique solution of the ODE system (3.3) with initial condition $x(0) = x_0 \in \Omega$. For simplicity, we further assume that Ω is invariant for the flow of the ODE, that is, for each $x_0 \in \Omega$ we have $x(t; x_0) \in \Omega$ for each t in the interval of definition of the solution.

3.6.1 Koopman spectral decomposition

In this subsection, we review the Koopman spectral decomposition. We mostly follow the treatment in [124, Ch.1]. A *Koopman eigenfunction* of the eigenvalue $\lambda \in \mathbb{C}$ of (3.3) is an analytic function $\phi : \Omega \rightarrow \mathbb{C}$ such that $\phi(x(t; x_0)) = e^{\lambda t} \phi(x_0)$ for each $x_0 \in \Omega$ and each t in the interval of definition of $x(t; x_0)$. In terms of Lie derivatives, ϕ satisfies

$$\mathcal{L}(\phi) = \sum_{j=1}^n \frac{\partial \phi}{\partial x_j} \cdot f_j = \lambda \phi. \quad (3.25)$$

Solving (3.25) for λ and ϕ one can find pairs of (eigenvalue, eigenfunction). Note that for $n > 1$ (3.25) is a partial differential equation in ϕ .

Example 6. Consider the one variable ($n = 1$) ODE $\dot{x}_1 = -x_1 - x_1^3$. Equation (3.25) is a ODE in ϕ with a parameter λ , that is: $\left(\frac{d}{dx_1} \phi(x_1)\right) (-x_1^3 - x_1) = \lambda \phi(x)$. For any $\lambda \neq 0$, this has the solution (up to a constant factor): $\phi_\lambda(x_1) = (x_1^2 + 1)^{\lambda/2} x_1^{-\lambda}$.

In general, Koopman eigenvalues and eigenfunctions are not easy to find, and there may be infinitely many of them — in fact, Koopman eigenvalues can even form dense sets. At any rate, let $g : \Omega \rightarrow \mathbb{C}$ be an observable of interest; this is the same as the function g considered in Section 3.3, but here we do allow g to take on complex values.

Ideally, one is interested in finding a *Koopman mode expansion* of g , if it exists, in the following sense: for a countable set of eigenfunctions $\{\phi_j\}_{j=0}^{\infty}$ and corresponding eigenvalues $\{\lambda_j\}_{j=0}^{\infty}$, we have on Ω

$$g = \sum_{i \geq 0} \nu_i \phi_i. \quad (3.26)$$

The coefficients $\nu_j \in \mathbb{C}$ are called *Koopman modes* of g . On the right-hand side of (3.26) we take the convergence of the series to be uniform on Ω . Note that, once a Koopman mode expansion of g is known, one obtains a complete characterization of the dynamics of the observable g . Indeed, from (3.26), for every $x_0 \in \Omega$ and t in the interval of definition of $x(t; x_0)$

$$g(x(t; x_0)) = \sum_{i \geq 0} \nu_i e^{\lambda_i t} \phi_i(x_0). \quad (3.27)$$

We refer to this as *Koopman spectral decomposition* of g (w.r.t. f).

3.6.2 Approximate Koopman decomposition

In practice, the Koopman spectral decomposition, even when it exists, is hard to compute due to its infinite dimensional nature. It is therefore natural to resort to finite-dimensional approximations. We start from the truncated Carleman linearization (3.8) introduced in Section 3.3. Taking the Lie derivative of α componentwise, we have seen in (3.10) that $\mathcal{L}(\alpha) = A\alpha + B\beta$; or, taking the transpose, $\mathcal{L}(\alpha^T) = \alpha^T A^T + \beta^T B^T$. Abbreviating $L := A^T$, for each function $h = u^T \alpha = \alpha^T u$ ($u \in \mathbb{C}^M$), we have the following expression for the Lie derivative of h

$$\mathcal{L}(h) = \alpha^T L u + \beta^T B^T u. \quad (3.28)$$

In particular, for any eigenvalue λ of the matrix L (hence of A) and corresponding *right* eigenvector u , letting $\phi := \alpha^T u$ we have: $\mathcal{L}(\phi) = \alpha^T L u + \beta^T B^T u = \lambda \alpha^T u + \beta^T B^T u = \lambda \phi + \beta^T B^T u$. One's hope here is that $\|\beta^T B^T u\| \approx 0$, so that one could write

$$\mathcal{L}(\phi) \approx \lambda \phi$$

and have (3.25) *approximately* satisfied. With this motivation, we proceed as follows. Let us say that an eigenvalue is *non-defective* if its geometric and algebraic multiplicity coincide. Let $\lambda_1, \dots, \lambda_m \in \mathbb{C}$ be the list of nondefective eigenvalues of L (hence of A), where each eigenvalue is repeated a number of times equal to its multiplicity. We assume $m \geq 1$. Also, let u_1, \dots, u_m and w_1^T, \dots, w_m^T be corresponding right and left eigenvectors, respectively: thanks to non-defectiveness, without loss of generality we can assume that $w_i^T u_j = \delta_{ij}$ (Kronecker's delta). Note that these eigenvectors have, in general, complex components. Let U , resp. W , be the $M \times m$ matrix that has the u_i 's, resp. w_i 's, as columns: then $W^T U = I_m$, the $m \times m$ identity matrix. Let $\tilde{\phi}_i := \alpha^T u_i$ ($1 \leq i \leq m$) denote the approximate Koopman eigenfunctions corresponding to the m right eigenvectors. Now, instead of (3.26), we equate g to an approximate Koopman mode expansion

$$g = \sum_{i=1}^m \tilde{\nu}_i \tilde{\phi}_i \quad (3.29)$$

where the approximate Koopman modes $\tilde{v}_i \in \mathbb{C}$ can be determined as follows. From (3.29), $\alpha^T v = g = \sum_{i=1}^m \tilde{v}_i \alpha^T u_i$. Therefore letting $\tilde{v} := (\tilde{v}_1, \dots, \tilde{v}_m)^T$, we have that (3.29) is equivalent to $\alpha^T U \tilde{v} = \alpha^T v$. A sufficient condition on \tilde{v} for this to hold is that $U \tilde{v} = v$ (this is also necessary if every function can be expressed in at most one way w.r.t. the sequence \mathcal{A}). Left-multiplying this equality by W^T , we have $W^T U \tilde{v} = \tilde{v} = W^T v$. Hence, we can define the vector of approximate Koopman modes as

$$\tilde{v} := W^T v. \quad (3.30)$$

Finally, in analogy with (3.27), we can *define* the approximate spectral decomposition of g as follows.

Definition 5 (approximate Koopman spectral decomposition). *With the above notation, we let the approximate Koopman decomposition w.r.t. A be:*

$$g_K(t; x_0) := \sum_{i=1}^m \tilde{v}_i e^{\lambda_i t} \tilde{\phi}_i(x_0). \quad (3.31)$$

The following proposition relates the approximate Koopman spectral decomposition to the truncated Carleman linearization (3.8) introduced in Section 3.3. Its proof is easy and mainly consists of matrix manipulations. In the statement of the proposition, we also consider complex solutions $z(t) = a(t) + ib(t)$ of the linear ODE systems $\dot{z} = Az$, with A a real matrix: this must be interpreted in the sense that both $a(t)$ and $b(t)$ are real solutions of the given ODE system. A complex solution $z(t)$ is real — that is, its imaginary $b(t)$ part is identically zero — if and only if the initial condition $z(0)$ is real; see [51, Ch.2, Sect.1.4].

Proposition 3 (Koopman vs Carleman). *Let $z(t; z_0)$ be the unique solution of the linear ODE system $\dot{z} = Az$ (truncated Carleman linearization) with initial condition $z_0 = WU^T \alpha(x_0) \in \mathbb{C}^M$. Then*

$$g_K(t; x_0) = v^T z(t; z_0). \quad (3.32)$$

In particular, if A is diagonalisable then $z_0 = \alpha(x_0)$.

When $z_0 = \alpha(x_0)$, we have the approximation $\alpha(x(t; x_0)) \approx z(t; z_0)$ (truncated Carleman linearization): then, in the diagonalisable case, the previous proposition yields the approximation⁵ $g_K(t; x_0) \approx g(x(t; x_0))$. This suggests that one could take $g_K(t; x_0) := v^T z(t; z_0)$ as a more direct and general definition of approximate Koopman decomposition.

3.6.3 Dimension reduction of approximate Koopman decomposition

More often than not, the size of A will be so large to make the effective computation of its eigenvalues and eigenvectors unfeasible. So whether we consider (3.31) or

⁵Under suitable conditions, concrete error bounds for this approximation can be given in a style similar to [124, Ch.2.4.2].

(3.32) — as seen, the two are equivalent at least in the diagonalisable case — the approximate Koopman spectral expansion of g can be difficult to compute as outlined in the preceding section.

Just as in done in Section 3.4, one way to try to circumvent this difficulty is to project $L(= A^T)$ onto an appropriate subspace K of \mathbb{R}^M of dimension $m \ll M$, and work within the coordinate space of K . It is well known that, depending on the choice of K , the eigen-values and vectors of the projected matrix will often be excellent approximations of (a subset of) the original eigen-values and vectors. This is the case, for example, when $K = \mathcal{K}_m$, the Krylov subspace generated by L and v ; but we need not assume this in the description given below.

From Section 3.4, recall that we let $V = [v_1 | \cdots | v_m] \in \mathbb{R}^{M \times m}$ be a matrix whose columns form an orthonormal basis of K in \mathbb{R}^M , and let $H := H_m = V^T L V$ be the projection of L onto K ; note that H is a $m \times m$ real matrix. Here we assume for simplicity that H is diagonalisable: this is often the case in practice, even if A is not. We let $\hat{\lambda}_1, \dots, \hat{\lambda}_m$ be a list of the eigenvalues of H , each repeated according to its multiplicity. Let $\hat{U} = [\hat{u}_1 | \cdots | \hat{u}_m] \in \mathbb{C}^{m \times m}$ (resp. $\hat{W}^T = [\hat{w}_1 | \cdots | \hat{w}_m]^T \in \mathbb{C}^{m \times m}$) be a matrix whose columns (resp. rows) form a basis of right (resp. left) eigenvectors of H . We can assume without loss of generality that $\hat{W}^T \hat{U} = I_m$. Now we will use $V \hat{U}$ and $(V \hat{W})^T = \hat{W}^T V^T$ as approximations of the right- and left-eigenvectors of L , respectively. With this in mind, we can define the *reduced* versions of:

- (a) the approximate Koopman eigenfunctions as $\hat{\phi}_i := \alpha^T V \hat{u}_i$, hence $\hat{\phi} := \hat{U}^T V^T \alpha$;
- (b) the approximate Koopman modes, as $\hat{v} := (V \hat{W})^T v$.

Note that $\hat{w}_i^T \hat{u}_j = (V \hat{u}_i)^T (V \hat{u}_j)$, as the columns of V form an orthonormal basis. So, in analogy with (3.27) and (3.31), the reduced, approximate Koopman spectral decompositions is defined as:

$$\hat{g}_K(t; x_0) := \sum_{i=1}^m \hat{v}_i e^{\hat{\lambda}_i t} \hat{\phi}_i(x_0). \quad (3.33)$$

On the other hand, recall from Definition 4 that the reduced observable dynamics of g is

$$\hat{g}(t; x_0) := v^T V y(t; y_0) \quad (3.34)$$

where $y(t; y_0)$ is the solution of the $m \times m$ linear homogeneous system $\dot{y} = H^T y$ with initial condition $y(0) = y_0 := V^T \alpha(x_0)$. The next proposition says that the reduced Koopman spectral decomposition coincides with our reduced observable dynamics, in case H is diagonalisable.

Proposition 4. *Suppose H is diagonalisable. Then $\hat{g}_K(t; x_0) = \hat{g}(t; x_0)$.*

We conclude with some remarks. The above result has been proven under the assumption that H is diagonalisable. On the other hand, the definition of reduced observable dynamics $\hat{g}(t; x_0)$ in (3.34) makes sense and is natural in the general case. Informally speaking, the system $\dot{y} = H^T y$ is just the orthogonal projection of the system $\dot{z} = L^T z = Az$ onto the subspace K . Since $z(t; z_0) \approx \alpha(x(t; x_0))$, and $z \approx Vy$, we have $v^T V y(t; y_0) \approx v^T \alpha(x(t; x_0)) = g(x(t; x_0))$, which is the exact dynamics. Moreover, when $K = \mathcal{K}_m$ (Krylov space) as discussed in Section 3.4 there is a way to compute $\hat{g}(t; x_0)$ on the fly via the Arnoldi algorithm, see also Appendix B.1. Additionally, $\hat{g}(t; x_0)$ is also locally accurate in the sense of Theorem 3.

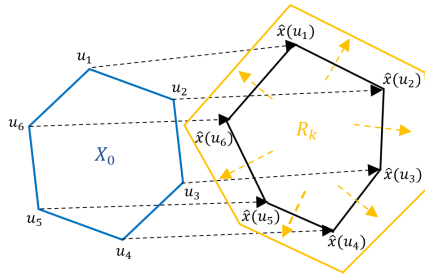


Figure 3.2: Advection and inflation of a polytope.

3.7 Application to reachability analysis

We will apply the linearization scheme outlined in sections 3.3 and 3.4 to compute an approximation $\hat{x}(t; x_0)$ of the flow $x(t; x_0)$, and then use it to compute an overapproximation of the reachable set of the nonlinear system (3.1) at fixed times: t_1, t_2, \dots . This goal will be achieved by applying the scheme of Section 3.3 to each of the observable functions $g = x_i$, for $i = 1, \dots, n$ in turn. Using the notation in that section, for each $i = 1, \dots, n$, let $v^{(i)}$ the coefficient vector of x_i in the chosen basis α , that is $x_i = v^{(i)T} \alpha$, and $V^{(i)}, H_m^{(i)}$ the corresponding basis and reduced matrix. We define the approximate flow by $\hat{x}(t; x_0) := (\hat{x}_1(t; x_0), \dots, \hat{x}_n(t; x_0))^T$, where, as an instance of (3.17), we have

$$\hat{x}_i(t; x_0) := \|v^{(i)}\|_2 y_1^{(i)}(t; y_0) \quad (i = 1, \dots, n) \quad (3.35)$$

with $y^{(i)}(t; y_0)$ the solution of the linear initial value problem (3.15) for $g = x_i$. Moreover, we will also consider the general case where we are given an initial set X_0 rather than an individual initial state x_0 .

The proposed reachability method is inspired by the CHECKMATE algorithm in [54]. For the sake of simplicity, we will represent the initial set X_0 as well as the successive reachsets R_1, R_2, \dots as convex polytopes⁶ (see below). Let $0 = t_0, t_1, \dots, t_N = t$ be time points, with $\Delta_k := t_k - t_{k-1} > 0$ for $k \geq 1$. The basic idea is to use (approximations of) the *advection maps*

$$x_0 \mapsto x(t_k; x_0) \quad (k = 1, 2, \dots)$$

to propagate the initial polytope's vertices $\{u_1, \dots, u_p\}$ to successive time points t_k . At the k -th stage, the polytope resulting from the advected vertices is suitably inflated to compensate for nonlinearities and approximation errors, thus obtaining the actual polytope R_k that over-approximates the reachable set at time t_k ; cf. Fig. 3.2. The main difference between [54] and us is that, while they approximate the advection maps via numerical integration of the original system (3.3), we adopt the maps $x_0 \mapsto \hat{x}(t_k; x_0)$. Theorem 3 suggests that $\hat{x}(t_k; x_0)$ is a good approximation of $x(t_k; x_0)$, but does not provide a direct way of bounding the resulting error. Instead, we will keep track of the approximation error via Corollary 2.

Concerning the computation of the approximate advection maps, we recall that, as a solution of the linear system (3.15), each component in (3.35) can be written

⁶The method can be extended without much difficulty to more sophisticated and scalable types of sets, like zonotopes.

explicitly as:

$$\widehat{x}_i(t; x_0) = \|v^{(i)}\|_2 \left(e_1^{tH_m^{(i)T}} \right) V^{(i)T} \alpha(x_0) \quad (i = 1, \dots, n) \quad (3.36)$$

where $e_1^{(\dots)}$ denotes the first row of the exponential matrix. As a function of x_0 , each $\widehat{x}_i(t; x_0)$ is a linear combination of the components of the basis $\alpha(x_0)$. For instance, it is a polynomial in x_0 if the elements of the basis are monomials.

In more detail, given p vectors (vertices) $u_1, \dots, u_p \in \mathbb{R}^n$, we assume that X_0 is the convex hull generated by those vertices, $X_0 = \text{ch}(u_1, \dots, u_p) := \{\sum_{i=1}^p \lambda_i u_i : \lambda_i \geq 0 \text{ and } \sum_{i=1}^p \lambda_i = 1\}$. It is easy to compute the polytope generated by the advected vertices at time t_k , given by $P_k := \text{ch}(\widehat{x}(t_k; u_1), \dots, \widehat{x}(t_k; u_p))$. We let the matrix-vector pair (C_k, b_k) denote a halfspace representation of P_k , that is $P_k = \{x \in \mathbb{R}^n : C_k x \leq b_k\}$. In fact, below we will only make use of the matrix C_k ; we assume without loss of generality the rows of this matrix, say $c_1^T, \dots, c_{\ell_k}^T$, are unitary, $\|c_j\|_2 = 1$. With the notation used in Corollary 2, we let $\gamma^{-(i)}, \gamma^{+(i)}$ denote the bounds in (3.19), (3.20) applied to $g = x_i$, for $i = 1, \dots, n$. In Definition 6 below, for $k \geq 1$ we shall adopt the abbreviations

$$\gamma_k^{(i)} := \max\{|\gamma^{-(i)}(\Delta_k; S_k, E_k^{(i)})|, |\gamma^{+(i)}(\Delta_k; S_k, E_k^{(i)})|\} \quad (3.37)$$

for given compact sets $S_k \supseteq \{x(\tau; \xi) : (\tau, \xi) \in [0, \Delta_k] \times R_{k-1}\}$ and $E_k^{(i)} \supseteq \{y(\tau; \zeta) : (\tau, \zeta) \in [0, \Delta_k] \times V^{(i)T} R_{k-1}\}$. We let $\gamma_k := (\gamma_k^{(1)}, \dots, \gamma_k^{(n)})^T$, with $\gamma_0 := (0, \dots, 0)^T$. For any nonnegative vector $\zeta \in \mathbb{R}^n$, we will let $[-\zeta, \zeta]$ denote the hyper-rectangle $[-\zeta_1, \zeta_1] \times \dots \times [-\zeta_n, \zeta_n] \subseteq \mathbb{R}^n$. Below, we assume $\Delta_k < \rho_{R_{k-1}}$ for each $k \geq 1$.

Definition 6 (reachsets R_k). *With the notation introduced above, for $k = 0, 1, 2, \dots$ we define the sequence of vectors $\eta_k = (\eta_k^{(1)}, \dots, \eta_k^{(\ell_k)})^T \in \mathbb{R}^{\ell_k}$ and of polytopes $R_k \subseteq \mathbb{R}^n$, as follows. $\eta_0 := 0$, $R_0 := X_0$ and, for $k \geq 1$, inductively:*

$$\eta_k^{(j)} := \max_{\substack{\xi \in R_{k-1} \\ \delta \in [-\gamma_k, \gamma_k]}} c_j^T (\widehat{x}(\Delta_k; \xi) + \delta) \quad (j = 1, \dots, \ell_k) \quad (3.38)$$

$$R_k := \{x \in \mathbb{R}^n : C_k x \leq \eta_k\}. \quad (3.39)$$

We note the following important facts about the above definition. (1) Computing η_k requires γ_k , whose computation in turn only requires enclosures S_k and E_k for ‘small’ flows $x(\tau; \xi)$ and $y(\tau; \zeta)$, for $\tau \in [0, \Delta_k]$ (cf. Corollary 2). (2) In the definition of R_k , one actually modifies a polytope P_k obtained by directly advecting the *initial* X_0 (sort of ‘long’ advection), not the preceding set R_{k-1} .

The correctness of the method is expressed by the following lemma, which also gives additional guarantees about the enclosure sets⁷ S_k .

Lemma 4 (correctness of R_k). *For each $k = 0, 1, \dots, N$ and $x_0 \in X_0$, we have $x(t_k; x_0) \in R_k$. Consequently, $S_k \supseteq \{x(\tau; x_0) : (\tau, x_0) \in [t_{k-1}, t_k] \times X_0\}$ for each $k \geq 1$.*

⁷These are useful in case one wants a flowpipe encapsulating the flow $x(t; x_0)$ for all t 's in a given interval, not only at specified time points t_k 's.

PROOF. Let $x_0 \in X_0$. Note that the second part of the statement follows from the first part ($x(t_k; x_0) \in R_k$ for $k \geq 0$), the additivity of the flow of ODES and the definition of S_k . Indeed, given $k \geq 1$ and $\tau \in [t_{k-1}, t_k]$, say $t_{k-1} + \tau'$ with $0 \leq \tau' \leq \Delta_k$, we have: $x(\tau; x_0) = x(\tau'; \xi)$ where $\xi = x(t_{k-1}; x_0) \in R_{k-1}$, but $x(\tau'; \xi) \in S_k$ by definition of S_k .

We proceed by induction on k to prove that $x(t_k; x_0) \in R_k$ for each $k \geq 0$. The base case $k = 0$ is trivial, so assume $k \geq 1$. By the additivity property of the flow of ODES, we have that $x(t_k; x_0) = x(\Delta_k; \xi)$ where $\xi = x(t_{k-1}; x_0) \in R_{k-1}$, by induction hypothesis. By Corollary 2 and the definition of γ_k , it follows that $|x_i(t_k; x_0) - \hat{x}_i(\Delta_k; \xi)| = |x_i(\Delta_k; \xi) - \hat{x}_i(\Delta_k; \xi)| \leq \gamma_k^{(i)}$ for $i = 1, \dots, n$. Hence $x(t_k; x_0) = x(\Delta_k; \xi) = \hat{x}(\Delta_k; \xi) + \delta$ for some vector $\delta \in [-\gamma_k, \gamma_k]$. Then for each row c_j^T of the matrix A_k , we have

$$c_j^T x(t_k; x_0) = c_j^T (\hat{x}(\Delta_k; \xi) + \delta) \leq \eta_k^{(j)} \quad (3.40)$$

by definition of $\eta_k^{(j)}$. Since this holds for each $j = 1, \dots, \ell_k$, we have the wanted $x(t_k; x_0) \in R_k$. \square

The overall workflow of the method is summarized in Algorithm 1, which we christen `ckr`, for *Carleman-Krylov Reachability*. The timesteps Δ_k , for $k = 1, 2, \dots, N$, are such that $\Delta_k = t_k - t_{k-1}$ and $t_N = T$, the time horizon, which is assumed to be in the interval of definition of $x(t; x_0)$ for each $x_0 \in X_0$. In an actual implementation, the timesteps might be chosen adaptively. In the pseudo-code, we use the abbreviation $\text{hs}(P)$ to denote a halfspace representation (C, b) of a polytope P .

Example 7. Let us reconsider the VdP system of Example 5. We fix $m = 2$, $X_0 = [0.1, 0.2] \times [0.1, 0.2]$, $T = \Delta = 0.1$ and a basis α of monomials. The approximate advection functions \hat{x}_1 and \hat{x}_2 are computed in the cycle 2–6 of Algorithm 1: we have already detailed the computation of \hat{x}_1 in Example 5; one proceeds similarly for \hat{x}_2 . Overall, writing $x_0 = x = (x_1, x_2)^T$, one obtains $\hat{x}(\Delta; x) := (\hat{x}_1(\Delta; x), \hat{x}_2(\Delta; x))^T$ where for $\Delta = 0.1$:

$$\hat{x}_1(\Delta; x) = 0.99 \cdot x_1 + 0.10 \cdot x_2 \quad \hat{x}_2(\Delta; x) = -0.10 \cdot x_1^2 \cdot x_2 - 0.10 \cdot x_1 + 1.09 \cdot x_2.$$

Let us see how the first (and only, for this example) reachset R_1 is computed. The four vertices $U = \{u_1, u_2, u_3, u_4\}$ of X_0 are advected at time T obtaining $U_1 = \{u'_1, u'_2, u'_3, u'_4\} := \{\hat{x}(\Delta; u_1), \hat{x}(\Delta; u_2), \hat{x}(\Delta; u_3), \hat{x}(\Delta; u_4)\}$. For instance, for $u_1 = (0.10, 0.10)^T$, one has $u'_1 = \hat{x}(\Delta; u_1) = (0.20, 0.08)^T$. For the convex hull of U_1 , a halfspace representation (C_1, b_1) is computed (step 10); to compensate for errors, vector b_1 is replaced by a slightly larger (componentwise) η_1 , computed via optimization (step 14), giving rise to the representation $R_1 = (C_1, \eta_1)$ returned as output. More precisely: $C_1 = \begin{bmatrix} -0.11074 & -0.99546 & 0.99544 & 0.11392 \\ -0.99385 & 0.09513 & -0.09541 & 0.99349 \end{bmatrix}^T$, $b_1 = (-0.11067, -0.10006, 0.20011, 0.22134)^T$ and $\eta_1 = (-0.11065, -0.10002, 0.20015, 0.22142)^T$.

Remark 3 (computational considerations). Concerning Algorithm 1, a few considerations are in order.

1. The computation of the enclosures $S_k, E_k^{(i)}$ in steps 11 and 12 can be achieved using any library available to this purpose. We rely on `CORA` [8] in our implementation.

Algorithm 1 CKR

Input: $f = (f_1, \dots, f_n)$, vector field in the variables $x = (x_1, \dots, x_n)$; $U \subseteq_{\text{fin}} \mathbb{R}^n$ s.t. $X_0 = \text{ch}(U)$; $m \geq 1$, order of approximation; $T \geq 0$, time horizon; $(\Delta_k)_{k=1}^N$, timesteps.

Output: RL , a list of reachsets.

- 1: $\alpha :=$ vector of elements of \mathcal{A} to repr. each of $\mathcal{L}_f^{(j)}(x_i)$ ($0 \leq j \leq m-1$, $1 \leq i \leq n$)
- 2: **for** $i = 1, \dots, n$ **do**
- 3: $v^{(i)} :=$ vector of coefficients of x_i w.r.t. α
- 4: $V^{(i)}, H_m^{(i)} := \text{Arnoldi}(f, v^{(i)}, \alpha, m)$ ▷ relies on $u \mapsto A^T u$
- 5: $\hat{x}_i := (t, x) \mapsto \|v^{(i)}\|_2 (e_1^{tH_m^{(i)T}})^T V^{(i)T} \alpha(x)$ ▷ Cf. (3.36). For $t = t_k$ is an adv. map
- 6: **end for**
- 7: $R_0 := \text{hs}(\text{ch}(U))$
- 8: $RL := [R_0]$
- 9: **for** $k = 1, 2, \dots, N$ **do**
- 10: $(C_k, b_k) := \text{hs}(\text{ch}(\hat{x}(t_k, U)))$ ▷ k -th advected polytope
- 11: $S_k := \text{enclosure}(f, \Delta_k, R_{k-1})$
- 12: $E_k^{(i)} := \text{enclosure}((H_m^{(i)})^T, \Delta_k, (V^{(i)})^T R_{k-1})$ ($1 \leq i \leq n$)
- 13: $\gamma_k := \text{apply (3.37) to } \Delta_k, S_k, E_k^{(i)}$ ($1 \leq i \leq n$)
- 14: $\eta_k := \text{apply (3.38) to } R_{k-1}, \gamma_k, A_k$
- 15: $R_k := (C_k, \eta_k)$ ▷ k -th reachset
- 16: append(RL, R_k)
- 17: **end for**
- 18: **return** RL

2. Solving the non-convex optimization problem (3.38) at step 14 is arguably the most demanding aspect of the algorithm, especially if one is interested in building a certified implementation. In the case of a polynomial vector field and basis, certified upper bounds can be obtained via Sum-Of-Squares (SOS) programming [148], which preserves correctness. In our current implementation, we rely on a general purpose global (non-convex) optimization procedure. We leave for future work the exploration of SOS techniques.
3. On the other hand, the optimization problems in steps 21 (C_k) and 23 (ρ_k) are unproblematic: for a polynomial basis, an upper bound on C_k can be effectively computed via interval arithmetic, while ρ_k can be obtained by a few projections on the c_j 's.
4. Numerical computation of the exponential matrix $e^{(t_k - \tau)H_m^T}$ in step 5 is not problematic, given that m is typically quite small. In a certified implementation, one might compute the exponential via interval arithmetic [81].

3.8 Experiments

In this section, we present some experimental results obtained by applying the approximation scheme and method in the preceding sections⁸. The section is divided into

⁸Code and examples available at <https://github.com/Luisa-unifi/CKR>.

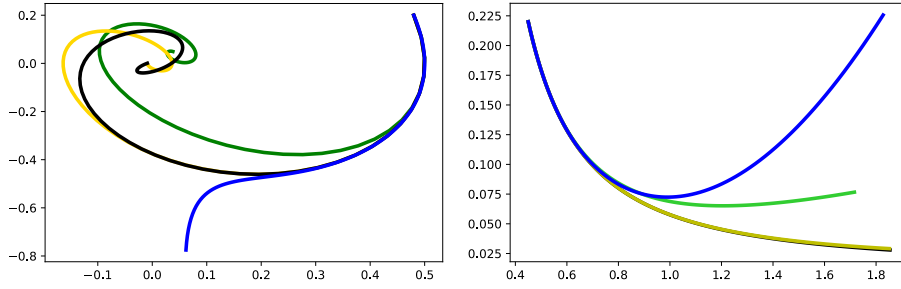


Figure 3.3: Individual trajectories starting from $x_0 = (0.485, 0.2)^T$ in the time interval $[0, 1]$ for systems (3.41)(a) (left) and (3.41)(b) (right), computed as follows: (i) $x(t; x_0)$, the exact solution computed numerically (yellow); (ii) $\hat{x}(t; x_0)$, our approximate solution (3.36) (black); (iii) $x_T(t; x_0)$ with $x_{T,i}(t; x_0) := \sum_{j=0}^{m-1} \mathcal{L}^{(j)}(x_i)|_{x=x_0} \frac{t^j}{j!}$, the Taylor expansion of order $m-1$ of the solution from $t=0$ (blue), limited to $t=0.2$ for system (3.41)(a); (iv) $x_L(t; x_0)$, the solution of the linearized system $\dot{x} = f(x_0) + J|_{x=x_0} \cdot (x - x_0)$, where J is the Jacobian of $f(x)$ in (3.3) (green).

three parts. First, we compare graphically different approximation methods, including ours, on two examples drawn from the literature, and on a more substantial example drawn from System Biology. Next, we compare our approximation method and our error bound with those proposed in [72, 10]. Finally, we illustrate the result of applying a proof-of-concept Python implementation of Algorithm 1, CKR. In particular, we compare CKR with two state-of-the-art tools for reachability analysis, CORA [8] and Flow* [47] on some examples drawn from the literature. In all the examples, for our method we consider a basis of monomial functions.

3.8.1 Graphical comparisons

We analyze the following two nonlinear systems.

$$(a) \begin{cases} \dot{x}_1 &= 4x_2(x_1 + \sqrt{3}) \\ \dot{x}_2 &= -4(x_1 + \sqrt{3})^2 - 4(x_2 + 1)^2 + 16 \end{cases} \quad (b) \begin{cases} \dot{x}_1 &= x_1(1.5 - x_2) \\ \dot{x}_2 &= -x_2(3 - x_1) \end{cases} \quad (3.41)$$

System (3.41)(a) is taken from [28], while system (3.41)(b) is an instance of Lotka-Volterra in 2D. System (3.41)(a) has the origin as a stable equilibrium point, while (3.41)(b) is not stable at the origin. For a time horizon of $T=1$, we show in Fig. 3.3 trajectories of exact and approximate solutions, computed with various methods, including ours, as explained in the caption. Our approximation (black curve) is very close to the exact solution (yellow curve).

We consider also a more substantial example drawn from System Biology, the Laub-Loomis model, whose description we report from [48].

$$(LL) \begin{cases} \dot{x}_1 &= 1.4x_3 - 0.9x_1 & \dot{x}_5 &= 0.7x_1 - x_4x_5 \\ \dot{x}_2 &= 2.5x_5 - 1.5x_2 & \dot{x}_6 &= 0.3x_1 - 3.1x_6 \\ \dot{x}_3 &= 0.6x_7 - 0.8x_3x_2 & \dot{x}_7 &= 1.8x_6 - 1.5x_7x_2 \\ \dot{x}_4 &= 2 - 1.3x_4x_3 \end{cases} \quad (3.42)$$

Like in the previous example, we make a graphical comparison of the exact solution, computed numerically, with approximate solutions obtained with various methods,

including ours. We fix the initial condition to $x_0 = (1.2, 1.05, 1.5, 2.4, 1.0, 0.1, 0.45)^T$ and the time horizon to $T = 1$. The plots in Fig. 3.4 show that $\hat{x}(t; x_0)$ is quite accurate w.r.t. to the exact solution.

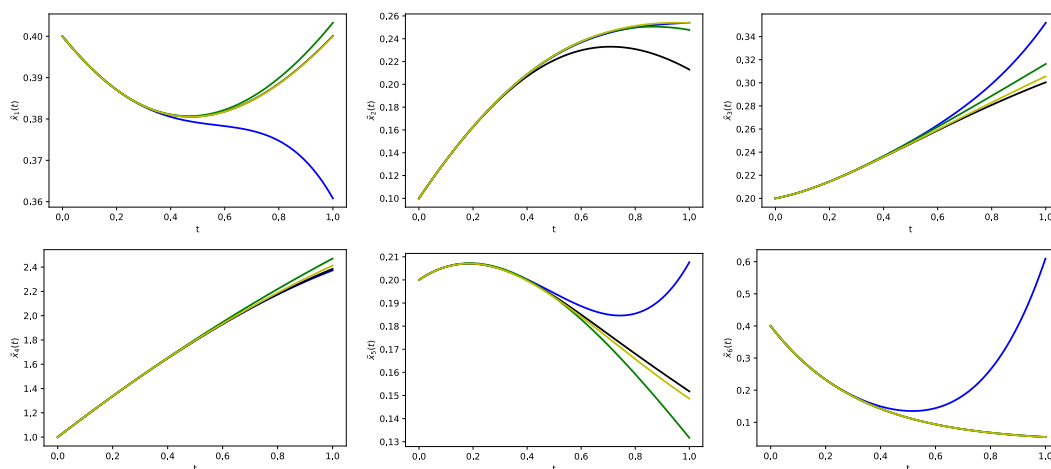


Figure 3.4: Exact and approximate solutions for LL over a time horizon of $T = 1$. The color code is the same as in Fig. 3.3: yellow/exact, black/our approximation, blue/Taylor, green/linearization with jacobian. Different values of m in $\{4, 5, 6\}$ are considered, depending on x_i . For layout reasons, the plot of x_7 is reported in a separate figure in Appendix B.3.

3.8.2 Comparison with [72] and [10]

The works [72] and [10] consider error bounds for the truncated Carleman approximation error, and show convergence as the truncation order N goes to infinity. We do not consider asymptotic convergence in N , but rather concrete error bounds on a time interval of interest, for any *fixed* approximation order m (a different parameter). Confining the comparison to error bounds on a time interval, we note the following.

- a) the main bound in [72, Th.4.3] is valid up to a time T^* that depends on the initial condition;
- b) the bounds in [10, Th.6.1] (cf. e.g. their eq. (6.28)) are valid up to a time τ^* that depends on the truncation order N .

These results can be compared with our Theorem 3, in particular (3.23), which applies to the system obtained by the combined Carleman+Krylov reduction. We do not have restrictions on the time interval width, although we do need an a priori bound on the norm of the solution on the time interval of interest (in order to bound the polynomial $h(x)$). As detailed below, we do not regard the latter as a practically significant limitation, at least in the comparison with [72, 10]. Moreover, our bounds appear to be in practice significantly less conservative than those in [72, 10], at least on the challenging VdP example.

The time widths in [72, 10], resp. T^* and τ^* , will typically be rather small. As an example, for the VdP system, [72, Sect.4.4] reports $T^* = 0.58$, while [10, Sect.8.1] reports $\tau^* = 0.2$. For such short time intervals, *rough but validated* compact

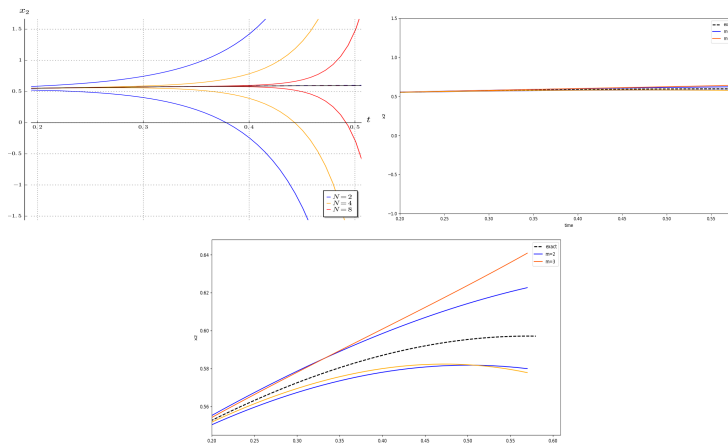


Figure 3.5: Graphical comparison with [72] on the VdP example. **Left:** approximated solution and error envelopes for $x_2(t; x_0)$ from [72]. **Center:** on the same scale, our approximation and error bounds. **Right:** close up of our plot. See the main text for details.

enclosures S of the exact solution $x(t; x_0)$ are easy to compute relying on classical techniques such as those reported in [136]. As an example, for the VdP system and initial condition $x_0 = (0, 0.5)$ considered in [72, Sect.4.4], using the CORA library one quickly finds an enclosing rectangle $S = [-0.028, 0.344] \times [0.458, 0.647]$. Once such an enclosure of $x(t; x_0)$ for the target time interval is available, it is possible to bound the factor $|h(x(\tau))|$ in our eq. (3.23), and compare the resulting bound to that of [72]. We report in Fig. 3.5 a graphical comparison for the VdP instance of [72, Sect.4.4]. On the left, we report from [72, Fig.1] a plot of the ‘envelopes’ of the exact solution (x_2 component) obtained using the truncated Carleman approximation and their error bounds; the exact solution is the dashed black line in the middle. On the center of the figure, we report on the same scale a plot of the envelopes obtained using our reduced order approximation $\hat{g}(t; x_0)$ ($g = x_2$) and our error bound (3.23), considering two different values of our parameter m ; we see that our envelopes are extremely tight around the exact solution (black, dashed). A close up of this plot is reported on the right of the figure. A similar comparison can be carried out with the example in [10, Sect.8.1], in particular with their error bounds reported in [10, Fig.2]. For instance, considering the initial condition $x_0 = (\sqrt{2}, \sqrt{2})$, which lies in the rectangle they consider, already for $m = 4$ our bound (3.23) yields a maximum error of 0.00585 at $t = \tau^*$, which appears to be well below the minimum error yielded by their bound, over all the considered values of N ([10, Fig.2], solid red line). This case can be easily extended to a more systematic comparison.

3.8.3 Reachsets: comparison with Flow* and CORA

Flow* [47] and CORA [8] are state-of-the-art tools for reachability analysis; they are quite effective at building (over-approximations of) reachsets. The purpose of the following comparison is showing that building reachsets around our approximate solutions $\hat{x}(t; x_0)$, as we do in ckr, can be beneficial for accuracy. We compare the reachsets R_k produced by ckr with those produced by Flow* and CORA on three examples, one stable and two unstable. Specifically, we consider: (1) the system in

| Sys | TH | Termination | | | | | Accuracy (average area) | | | | | Execution time | | | | |
|---------|----|-------------|----------|----------|----------|----------|-------------------------|-------|-------|-------------|-------------|----------------|-------------|--------|-------|--------------|
| | | Flow* | | | CORA | CKR | Flow* | | | CORA | CKR | Flow* | | | CORA | CKR |
| | | m=4 | m=8 | m=10 | all m | all m | m=4 | m=8 | m=10 | all m | m=4/5 | m=4 | m=8 | m=10 | m=4 | m=4/5 |
| (3.41)b | 1 | 1 | 1 | 1 | 1 | 1 | 0.02 | 0.02 | 0.02 | 0.01 | 0.01 | 0.12 | 1.45 | 3.90 | 0.17 | 13.31 |
| | 3 | 3 | 3 | 3 | 2.2* | 3 | 22.75 | 6.20 | 6.18 | 1.27* | 2.82 | 0.98 | 4.67 | 14.94 | 0.47* | 50.31 |
| | 5 | 2.7* | 5 | 5 | 2.2* | 5 | 99.67* | 4.37 | 4.35 | 1.27* | 1.57 | 2.74* | 8.55 | 25.24 | 0.49* | 94.79 |
| (3.43) | 1 | 1 | 1 | 1 | 0.6* | 1 | 5.16 | 3.34 | 3.33 | 5.34* | 0.95 | 0.38 | 6.92 | 23.06 | 4.28* | 14.27 |
| | 3 | 1.3* | 1.5* | 1.5* | 0.6* | 3 | 8.37* | 6.81* | 6.10* | 5.34* | 0.72 | 5.04* | 21.90* | 67.76* | 4.18* | 37.96 |
| | 5 | 1.3* | 1.5* | 1.5* | 0.6* | 5 | 8.37* | 6.81* | 6.10* | 5.34* | 0.62 | 4.94* | 19.84* | 76.48* | 5.08* | 64.42 |
| VdP | 1 | 1 | 1 | 1 | 1 | 1 | 0.37 | 0.37 | 0.37 | 0.15 | 0.12 | 0.13 | 1.71 | 5.03 | 2.02 | 13.72 |
| | 3 | 3 | 3 | 3 | 3 | 3 | 0.16 | 0.15 | 0.15 | 0.09 | 0.05 | 0.42 | 5.05 | 15.42 | 5.13 | 37.05 |
| | 5 | 5 | 5 | 5 | 5 | 5 | 0.15 | 0.13 | 0.13 | 0.18 | 0.07 | 0.77 | 8.54 | 24.93 | 10.36 | 65.66 |

Table 3.1: Comparison of Flow*, CORA and CKR on: system (3.41)(b) with $X_0 = [0.40, 0.52] \times [0.18, 0.27]$; system (3.43) with $X_0 = [-0.5, 0.3] \times [-0.7, 0.8]$; the VdP system of Example 5 with $X_0 = [1.00, 1.50] \times [2.00, 2.45]$. Legend: **Sys** = system's equation reference, **TH** = time horizon, **Termination** = time at which the algorithm stops, either by natural termination or by breakdown (marked with *), **Accuracy** = average area of reachsets, **m** = approximation order (see Appendix B.3 for further details). In each row, the best achieved results are marked in **boldface**.

(3.41)(b) (unstable); (2) a new system (stable) defined by:

$$\dot{x}_1 = -x_1^3 + x_2 \quad \dot{x}_2 = -x_1^3 - x_2^3. \quad (3.43)$$

and finally, (3) the VdP system introduced in Example 5 (unstable). VdP also exhibits a limit cycle behaviour. We also stress-test the capabilities of the algorithms in terms of initial sets by considering relatively large X_0 's.

We measure the quality of the results as the average area of the reachsets in correspondence of the timesteps returned by each algorithm, until natural or premature termination: $\frac{1}{N} \sum_{k=1}^N a(P_k)$, where $a(P_k)$ denotes the area of the polygon P_k corresponding to the reachset at time t_k ($P_k = R_k$ for CKR). We report the obtained results in Table 3.1, together with the time at which the different algorithms stop, possibly due to an explosion of the overapproximation (breakdown time). For the sake of completeness, we also report a column with execution times⁹.

As far as accuracy is concerned, in all cases the sets produced by CKR are tighter than those produced by the other two, often significantly so. Concerning termination, CKR is the only algorithm to complete its execution over the whole time horizon in all the

⁹It should be noted, though, that it makes little sense to compare a proof-of-concept implementation with highly optimized tools in this respect. At any rate, all execution times are below 100 seconds.

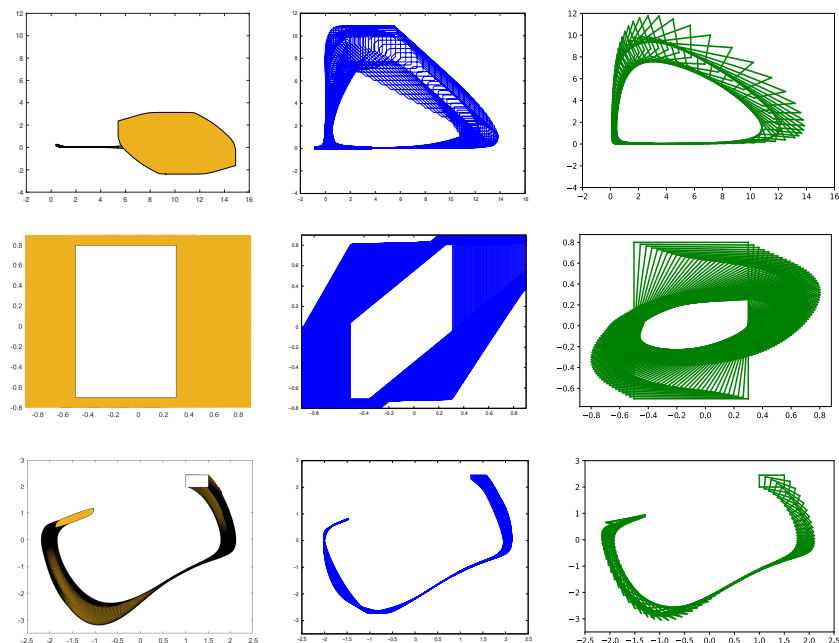


Figure 3.6: Reachsets computed with CORA (**left**), Flow* (**center**) and CKR (**right**). **Top row**: system (3.41)(b), **Central row**: system (3.43), **Bottom row**: VdP. In all cases, $T = 5$. See text for details on m, X_0 .

considered cases. The difference among the results produced by the three algorithms can be appreciated also graphically from a few plots of the computed reachsets, see Fig. 3.6. Additional details on these experiments are reported in Appendix B.3.

3.9 Conclusion

We have presented an approach to effectively compute, given a nonlinear ODE system, a linear system which is at the same time small and useful to produce globally accurate approximate solutions, under suitable conditions. We have argued that the method can also bring some benefit to classical reachability analysis in terms of accuracy.

As for future work, it would be interesting to further explore the scalability of the CKR algorithm. Application to Runtime Verification and Model Predictive Control also deserves further attention.

Chapter 4

SDEs, a (co)algebraic approach

4.1 Overview

In this chapter we investigate a connection among polynomials, differential equations and *streams*, i.e., infinite sequences of elements from a set [155]. At a very informal level, this connection can be expressed by the following correspondences: polynomials = syntax; differential equations = operational semantics; streams = abstract (denotational) semantics. There are two important motivations behind this standpoint. (1) Diverse notions of *product* (convolution, shuffle,...) arise in streams, in relation to different models – discrete computations, combinatorial sequences, analytic functions, and more [17, 155]. There is also a close analogy between several forms of products and forms of parallelism arising in concurrency. Our aim is to uniformly accommodate such diverse notions, by automatically deriving an operational semantics for polynomials that is adequate for a given *generic* stream product. (2) Once adequate polynomial syntax and operational semantics have been obtained, one can apply powerful techniques both from algebraic geometry (Groebner bases [57]) and from coalgebra (coinduction [155]) for reasoning on streams. This includes devising algorithms for deciding stream equivalence. Again, one would like to do so in a uniform fashion w.r.t. an underlying notion of stream product.

Technically, achieving these goals amounts to defining a fully abstract semantics from polynomials to streams, which is essential for algebraic-geometric reasoning on streams. Moreover, one wants the resulting construction to be as much as parametric as possible with respect to the underlying notion of stream product.

As hinted above, we will pursue these goals by relying on tools from algebra and coalgebra. Indeed, it is well-known that, when polynomial coefficients and stream elements are drawn from a field \mathbb{K} , both polynomials and streams form commutative \mathbb{K} -algebras, i.e., rings with an additional vector space structure over \mathbb{K} . Note that, while this algebra structure is fixed for polynomials, it varies with the underlying product for streams. On the other hand, streams also possess a *coalgebraic* structure, arising from the operation of stream derivative. On the side of polynomials, it is also natural to interpret a differential equation $\dot{x}_i = p_i$ as a transition $x_i \rightarrow p_i$: thus one expects a transition structure, hence a coalgebra, over polynomials as well. How to appropriately extend transitions from individual variables x_i to monomials and polynomials, though, nontrivially depends on the notion of stream product one wants to model.

Our first result is that the above outlined goals can be achieved for the class of (F, G) -products on streams, where, basically, the derivative of a product of two streams can be expressed as a polynomial of the streams themselves and their derivatives. As an example, convolution, shuffle, Hadamard and infiltration products (see e.g. [17]) all fall in this class. One can then define a coalgebra structure on polynomials, depending on the given (F, G) -product and differential equations, such that the unique morphism from this coalgebra to the coalgebra of streams is also a commutative \mathbb{K} -algebra *homomorphism*. And vice versa: every homomorphism that satisfies the given differential equations is the unique morphism. Thus, full abstraction is achieved.

A major application of this result is an algorithm based on an algebraic-geometric procedure for deciding stream equivalence, i.e. if two polynomials denote the same stream. This procedure is then smoothly extended to an algorithm to find, for instance, *all* valid polynomial identities up to a given degree. These algorithms are illustrated on specific (F, G) -products (convolution, shuffle), by automatically finding nontrivial valid polynomial equations, for a few examples of SDEs. Note that algebraically solving such equations in the ring of streams [155, 157] leads in turn to closed forms for generating functions of sequences [71, 173].

Moreover, we add another tool to the stream calculus: an Implicit Function Theorem (IFT) for systems of stream polynomial equations. Indeed, while SDEs represent a powerful computational device, depending on the problem at hand streams may be more naturally expressed in an algebraic fashion, that is as the (unique) solution of systems of polynomial equations. In analogy with the classical IFT from calculus [152, 112, 168], our main result provides sufficient syntactic conditions under which a system of polynomial equations has a unique stream solution. The theorem also provides an equivalent system of SDEs, that is useful to actually compute the stream solution. A crucial step toward proving the result is devising a stream version of the chain rule from calculus, whereby one can express the derivative of a function $\mathbf{F}(x, y_1(x), \dots, y_n(x))$ w.r.t. x in terms of the partial derivatives of \mathbf{F} w.r.t. y_i and the ordinary derivative of the y_i w.r.t. x .

To sum up, we make the following two main contributions. (1) A unifying treatment of stream products, implying that, under reasonable assumptions, coalgebra morphisms from polynomials to streams are also commutative \mathbb{K} -algebra homomorphisms (full abstraction) – and vice versa. (2) Algorithms for deciding polynomial stream equivalence and finding valid stream polynomial identities, that rely on the full abstraction result. (3) An Implicit Function Theorem (IFT) for systems of stream polynomial equations. The rest of the chapter is organized as follows. In Section 4.2 we introduce the necessary background on polynomials, differential equations, streams and coalgebras. Section 4.3 contains our main result, the coincidence of coalgebra morphisms and algebra homomorphisms, under certain conditions on the underlying stream product. As a major application of this result, we present in Section 4.4 an algorithm for deciding stream equality. This is expanded in Section 4.5, where we present a method to find all valid polynomial identities that fit a given template. This result is related to existing algorithms for linear weighted automata/expressions. In Section 4.6, in the setting of the stream calculus and of polynomial equations, we obtain a version of the IFT whose form resembles closely the classical one (Theorem 8). In Section 4.7, beyond the formal similarity, we discuss the precise mathematical relation of the stream IFT with the classical IFT (Theorem 9). As an extended example

of application of the stream IFT, in Section 4.8 we apply the result to the problem of enumerating *three-colored trees* [71, Sect.4, Example 14], a typical class of combinatorial objects that are most naturally described by algebraic equations. In Section 4.9 we discuss the computational aspects of the stream IFT. Section 4.10 briefly draws some concluding remarks and discusses potential directions for future work. For ease of reading, the proofs of some intermediate technical results have been confined to a separate appendix, Appendix C.

Related work Rutten’s *stream calculus* [155, 157], a coinductive approach to the analysis of infinite sequences (streams), is a major source of inspiration for our work. [155] studies streams, automata, languages and formal power series in terms of coalgebra morphisms and bisimulation. In close analogy with classical analysis, [157] presents coinductive definitions and proofs for a calculus of *behavioural differential equations*, also called *stream differential equations* (SDEs) in later works. A number of applications to difference equations, analytical differential equations, continued fractions and problems from combinatorics, are presented. Convolution and shuffle products play a central role in the stream calculus; a duality between them, mediated by a variation of Laplace transform, exists [156].

A coinductive treatment of analytic functions and Laplace transform is also presented by Escardo and Pavlovic [141]. Basold et al. [17] enrich the stream calculus with two types of products, Hadamard and infiltration, and exhibit a duality between the two, mediated by a so-called *Newton transform*. Although these works form a conceptual prerequisite of our study, they do not offer a unifying treatment of the existing disparate notions of stream product, nor any algorithmic treatment of the induced stream equivalences.

Boreale [30] and Bonchi et al. [21] consider an operational approach to streams and convolution product based on weighted automata, which basically correspond to linear SDEs and expressions. They offer an equivalence checking algorithm for such automata and the recognized streams, based on a linear-algebraic construction; however, the polynomial case is not addressed. Related to this is the work of Bonchi et al. [16], where algorithms for equality of streams specified by linear SDEs, are presented. Our results here generalize these algorithms, as we can also work with polynomial SDEs. This will be made precise and discussed in the final part of Section 4.5.

We also mention [27, 29], that adopt a coinductive approach to reason on polynomial ODEs. The ring of multivariate polynomials is employed as a syntax, with *Lie derivatives* inducing a transition structure. An algebraic-geometric algorithm to decide polynomial equivalence is presented. This algorithm as well has inspired our decision method: in particular, as Lie derivatives are precisely the transition structure induced in our framework by the shuffle product, the decision algorithms of [27, 29] are in essence special cases of our algorithms in Section 4.4 and Section 4.5. Furthermore, [31, 32] extend the framework of [27, 29] to polynomial partial differential equations, which pose significant additional challenges.

Somewhat related to ours is the work of Winter on coalgebra and polynomial systems: see e.g. [182, Ch.3]. Importantly, Winter considers polynomials in *non-commuting* variables: under suitable assumptions, this makes his systems of equations isomorphic to certain context-free grammars; see also [113]. The use of noncom-

muting variables sets Winter's treatment in mathematical realm that is totally different from the one considered here. In particular, the algebraic geometric concepts we rely on here, like ideals and Groebner bases, are not applicable in Winter's framework.

In enumerative combinatorics [71, 173], formal power series defined via polynomial equations are named *algebraic series*. [71, Sect.4] discusses several aspects of algebraic series, including several methods of reduction, involving the theory of resultants and Groebner bases. We compare our approach to algebraic series in Section 4.6, Remark 11.

More closely related to ours is the work of Hansen, Kupke and Rutten [89]. There the authors prove that, when the SDEs defining given operations on streams obey a GSOS syntactic format, the final coalgebra morphism is also a homomorphism from the free term algebra to the algebra (w.r.t. the given operations) of streams [89, Sect.8]. It is interesting to note that our notion of (F, G) -product falls in the abstract GSOS format. However, we work with the algebra of polynomials, which besides being a commutative ring and vector space over \mathbb{K} , possesses additional structure arising from monomials. All this structure is essential for algebraic-geometric reasoning, and sets our approach apart from those based on term algebras: for one thing, in term algebras there is no obvious analog of Hilbert's basis theorem, a result deeply related to the well-ordering of monomials (cf. Dickson's lemma, [57, Ch.2]), and a crucial ingredient in our decision algorithm. One might consider more complicated GSOS frameworks enriched with equational theories: indeed, an abstract version of the GSOS format has also been discussed in the framework of *bialgebras* [89, Sect.9] and *distributive laws* [25]. Bialgebras require a substantial background in category theory, which we have preferred to avoid here so as to keep our approach as elementary and accessible as possible. A more technical discussion on this point is deferred to Section 4.3, Remark 6.

4.2 Preliminaries

4.2.1 Polynomials and differential equations

Let us fix a finite, non empty set of symbols or *variables* $X = \{x_1, \dots, x_n\}$ and a distinct variable $x \notin X$. Informally, x will act as the independent variable, while x_1, \dots, x_n will act as dependent variables, or functions, defined by differential equations (see below). Notationally, we write \mathcal{X} to denote $\{x\} \cup X$. We fix a generic field \mathbb{K} of characteristic 0; $\mathbb{K} = \mathbb{R}$ and $\mathbb{K} = \mathbb{C}$ are typical choices. We let $\mathcal{P} \triangleq \mathbb{K}[\mathcal{X}]$, ranged over by p, q, \dots , be the set of polynomials with coefficients in \mathbb{K} and indeterminates in \mathcal{X} . We let \mathcal{M} , ranged over by m, m', \dots , be the set of *monomials*, that is the free commutative monoid generated by \mathcal{X} . As usual, we shall denote polynomials as formal finite sums of distinct monomials with nonzero coefficients in \mathbb{K} : $p = \sum_{i \in I} r_i m_i$, for $r_i \in \mathbb{K}$ and $m_i \in \mathcal{M}$. By slight abuse of notation, we shall write the zero polynomial and the empty monomial as 0 and 1, respectively. Over \mathcal{P} , one can define the usual operations of sum $p + q$ and product $p \cdot q$, with 0 and 1 as identities, and enjoying commutativity, associativity and distributivity, which make \mathcal{P} a ring; multiplication of $p \in \mathcal{P}$ by a scalar $r \in \mathbb{K}$, denoted rp , is also defined and makes $(\mathcal{P}, +, 0)$ a vector space over \mathbb{K} . Therefore, $(\mathcal{P}, +, \cdot, 0, 1_\pi)$ forms a commutative \mathbb{K} -algebra.

We shall also fix a set $\mathcal{D} = \{\dot{x}_1 = p_1, \dots, \dot{x}_n = p_n\}$ of *differential equations*, one for each $x_i \in X$, where the p_i 's belong to \mathcal{P} and are called *drifts*. An *initial condition* for \mathcal{D} is a vector $\rho = (r_1, \dots, r_n) \in \mathbb{K}^n$. The pair (\mathcal{D}, ρ) forms an *initial value problem*. Informally, each $x_i \in X$ represents a placeholder for a function whose derivative is given by p_i , and whose value at the origin is $x_i(0) = r_i$. This terminology is borrowed from the theory of differential equations. However, note that, depending on the semantics of polynomial product one adopts (see next section), \mathcal{D} can be given diverse interpretations, including *stream differential equations* (SDE, for convolution, see next subsection) in the sense of Rutten [155], and of course `ODES` (for shuffle). In the literature on bialgebras and distributive laws, initial value problems are sometimes referred to as *coequations* [58].

Notationally, it will be sometimes convenient to regard \mathcal{D} and ρ as functions $\mathcal{D} : X \rightarrow \mathcal{P}$ and $\rho : X \rightarrow \mathbb{K}$, respectively, such that $\mathcal{D}(x_i) = p_i$ and $\rho(x_i) = r_i$. It is also convenient to extend \mathcal{D} and ρ to x by letting $\mathcal{D}(x) = 1$ and $\rho(x) = 0$; note that, seen as an initial value problem, the last two equations define the identity function. Finally, we let x_0 denote x and, when using \mathcal{D} and ρ as functions, use x_i as a metavariable on \mathcal{X} : this makes $\mathcal{D}(x_i)$ and $\rho(x_i)$ well defined for $0 \leq i \leq n$.

4.2.2 Streams

We quickly review some basic notions taken from [155]. We let $\Sigma\langle\mathbb{K}\rangle \triangleq \mathbb{K}^\omega$, ranged over by σ, τ, \dots , denote the set of *streams*, that is infinite sequences of elements from \mathbb{K} : $\sigma = (r_0, r_1, r_2, \dots)$ with $r_i \in \mathbb{K}$. Often \mathbb{K} is understood from the context and we shall simply write Σ rather than $\Sigma\langle\mathbb{K}\rangle$. When convenient, we shall explicitly consider a stream σ as a function from \mathbb{N} to \mathbb{K} and, e.g., write $\sigma(i)$ to denote the i -th element of σ . By slightly overloading the notation, and when the context is sufficient to disambiguate, the stream $(r, 0, 0, \dots)$ ($r \in \mathbb{K}$) will be simply denoted by r , while the stream $(0, 1, 0, 0, \dots)$ will be denoted by X ; see [155] for motivations behind these notations. Furthermore, a stream made up of all the same element $r \in \mathbb{K}$ will be denoted as $\underline{r} = (r, r, \dots)$. One defines the *sum* of two streams σ and τ as the stream $\sigma + \tau$ defined by: $(\sigma + \tau)(i) \triangleq \sigma(i) + \tau(i)$ for each $i \geq 0$, where the $+$ on the right-hand side denotes the sum in \mathbb{K} . Sum enjoys the usual commutativity and associativity properties, and has the stream $0 = (0, 0, \dots)$ as an identity.

Various forms of stream products, generically denoted by π and with identity 1_π , can also be considered – this is indeed a central theme of our work. In particular, the *convolution product* (written \times) and the *shuffle product* (written \otimes) are defined as follows, for any $i \geq 0$:

$$(\sigma \times \tau)(i) \triangleq \sum_{0 \leq j \leq i} \sigma(j) \cdot \tau(i-j) \quad (\sigma \otimes \tau)(i) \triangleq \sum_{0 \leq j \leq i} \binom{i}{j} \sigma(j) \cdot \tau(i-j), \quad (4.1)$$

where operations on the right-hand side are carried out in \mathbb{K} .¹ Both products are commutative, associative, have $1 = (1, 0, 0, \dots)$ as an identity, and distribute over $+$; multiplication of $\sigma = (r_0, r_1, \dots)$ by a scalar $r \in \mathbb{K}$, denoted $r\sigma = (r r_0, r r_1, \dots)$, is

¹The above operations enjoy alternative, easier to handle formulations based on stream differential equations – see next subsection; there, a crucial notion will be the *derivative* of a stream σ , that is the stream σ' obtained from σ by removing its first element.

also defined and makes $(\Sigma, +, 0)$ a vector space over \mathbb{K} . Therefore, $(\Sigma, +, \pi, 0, 1)$ forms a commutative \mathbb{K} -algebra for both products. Let us record the following useful properties for future use: $X \times \sigma = (0, r_0, r_1, \dots)$ and $r \pi \sigma = (r r_0, r r_1, \dots)$, where $r \in \mathbb{K}$ and $\pi \in \{\times, \otimes\}$. In view of the second equation above, $r \pi \sigma$ coincides with $r\sigma$. The first equation above leads to the so called fundamental theorem of the stream calculus, whereby for each $\sigma \in \Sigma$

$$\sigma = \sigma(0) + X \times \sigma'. \quad (4.2)$$

Less commonly found forms of products, like Hadamard and Infiltration products, will be introduced in the next subsection; equations similar to (4.2) exist also for such products [17, 89].

4.2.3 Coalgebras, SDEs and bisimulation

We quickly review some basic definitions and results about coalgebras and bisimulation; see e.g. [155] for a comprehensive treatment. A *(stream) coalgebra with outputs in \mathbb{K}* is a Moore automaton $C = (S, \delta, o)$, where S is a nonempty set of *states*, $\delta : S \rightarrow S$ is the *transition* function, and $o : S \rightarrow \mathbb{K}$ is the *output* function. A *bisimulation* on C is a binary relation $R \subseteq S \times S$ such that, whenever $(s, t) \in R$, then $o(s) = o(t)$ and $(\delta(s), \delta(t)) \in R$. As usual, there always exists a largest bisimulation on C , denoted \sim : it is the union of all bisimulations and it is an equivalence relation on S . Given two coalgebras C_1 and C_2 , a *coalgebra morphism* between them is a function $\mu : S_1 \rightarrow S_2$ from the states of C_1 to the states of C_2 that preserves transitions and outputs, that is (with obvious notation): $\mu(\delta_1(s)) = \delta_2(\mu(s))$ and $o_1(s) = o_2(\mu(s))$, for each $s \in S_1$. Coalgebra morphisms preserve bisimilarity, in the sense that $s \sim t$ in C_1 if and only if $\mu(s) \sim \mu(t)$ in C_2 . A coalgebra C_0 is *final* in the class of coalgebras with outputs in \mathbb{K} if, from every coalgebra C in this class, there exists a unique morphism μ from C to C_0 . In this case, \sim in C_0 coincides with equality, and the following *coinduction principle* holds: for every C and $s \sim t$ in C , it holds that $\mu(s) = \mu(t)$ in C_0 .

The set of streams Σ can be naturally given a stream coalgebra structure $(\Sigma, (\cdot)', o(\cdot))$, as follows. The *output* of a stream $\sigma = (r_0, r_1, \dots)$ is $o(\sigma) \triangleq r_0$ and its *derivative* is $\sigma' \triangleq (r_1, r_2, \dots)$, that is σ' is obtained from σ by removing its first element, that constitutes the output of σ . In fact, this makes Σ final in the class of all coalgebras with outputs in \mathbb{K} [155]. This also implies that one can prove equality of two streams by exhibiting an appropriate bisimulation relation relating them (coinduction).

It is sometimes convenient to consider an enhanced form of bisimulation on Σ that relies on the notion of *linear closure*.² Given a relation $R \subseteq \Sigma \times \Sigma$, its linear closure \hat{R} is the set of pairs of the form $(\sum_{i=1}^n r_i \sigma_i, \sum_{i=1}^n r_i \tau_i)$, where $n \in \mathbb{N}$, $(\sigma_i, \tau_i) \in R$ and $r_i \in \mathbb{K}$, for every $i \in \{1, \dots, n\}$. We say that R is a *bisimulation up to linearity* if, for every $(\sigma, \tau) \in R$, it holds that $o(\sigma) = o(\tau)$ and $(\sigma', \tau') \in \hat{R}$. If R is a bisimulation up to linearity, then \hat{R} is a bisimulation [155]; since by definition $R \subseteq \hat{R}$, this implies that $R \subseteq \sim$, the bisimilarity on streams, which coincides with equality.

²More general notions that we could have used here are *contextual closure* (see [17, Thm. 2.4]) and works on distributive laws for bialgebras [22]. However, the simpler notion of linear closure suffices for our purposes here.

A *stream differential equation* (SDE) in the unknown σ is a pair of equations of the form $\sigma(0) = r$ and $\sigma' = \phi$, for $r \in \mathbb{K}$ and a stream expression ϕ (that can depend on σ or its components, or even on σ' itself). Under certain conditions on ϕ [89, 155], it can be proven that there is a unique stream σ satisfying the above SDE. In this thesis, we shall focus on the case where ϕ is represented by a polynomial expression – this will be formalized in the next section. More formally, let $\boldsymbol{\sigma} = (\sigma_1, \dots, \sigma_n)$ a n -tuple of streams and $p(X, \boldsymbol{\sigma})$ the result of substituting the variables x and \mathbf{y} in p with the streams X and $\boldsymbol{\sigma}$, respectively.

Definition 7 (SDE [155]). *Given a tuple of polynomials $(p_1, \dots, p_n) \in \mathcal{P}^n$ and $\mathbf{r}_0 = (r_1, \dots, r_n) \in \mathbb{K}^n$, the corresponding system of (polynomial) stream differential equations (SDEs) \mathcal{D} and initial conditions are written as follows*

$$\mathcal{D} = \{y'_1 = p_1, \dots, y'_n = p_n\} \quad \rho = \{y_1(0) = r_1, \dots, y_n(0) = r_n\}. \quad (4.3)$$

The pair (\mathcal{D}, ρ) is also said to form a (polynomial) SDE initial value problem for the variables \mathbf{y} . A solution of (4.3) is a tuple of streams $\boldsymbol{\sigma} = (\sigma_1, \dots, \sigma_n) \in \Sigma^n$ such that $\sigma'_i = p_i(X, \boldsymbol{\sigma})$ and $\sigma_i(0) = r_i$ for $i = 1, \dots, n$.

A natural generalization of the above definition are systems of *rational* SDEs, where the right-hand side of each equation is a fraction of polynomials. Systems of rational SDEs have indeed the same expressive power as polynomial ones: a version of this (well-known) result will be explicitly formulated in Section 4.6 (see Lemma 8).

For a proof of the following theorem (in a more general context), see e.g. [89, 26].

Remark 4 (stream coefficients computation). We record for future use that a SDE initial value problem (\mathcal{D}, ρ) like (4.3) implies a recurrence relation, hence an algorithm, to compute the coefficients of the solution streams σ_i . Indeed, denote by $\sigma_{:k}$ the stream that coincides with σ when restricted to $\{0, \dots, k\}$ and is 0 elsewhere. This notation is extended to a tuple $\boldsymbol{\sigma}$ componentwise. Then we have, for each $i = 1, \dots, n$ and $k \geq 0$:

$$\sigma_i(0) = y_i(0) \quad (4.4)$$

$$\sigma_i(k+1) = \sigma'_i(k) = p_i(X, \boldsymbol{\sigma})(k) = p_i(X, \boldsymbol{\sigma}_{:k})(k) \quad (4.5)$$

where the last step follows from the fact that the k -th coefficient of $p_i(X, \boldsymbol{\sigma})$ only depends on the first k coefficients of $\boldsymbol{\sigma}$. In the literature, this is referred to as *causality* (see [89, 105, 145], just to cite a few).

As an example, consider

$$y' = y^2 \quad y(0) = 1$$

for which we get the recurrence: $\sigma(0) = 1$ and $\sigma(k+1) = \sigma^2(k) = \sum_{j=0}^k \sigma(j) \cdot \sigma(k-j)$. From the computational point of view this is far from optimal. Indeed, in the case of a single polynomial equation ($n = 1$) like this one, a linear (in y) recurrence relation for generating the Taylor coefficients of the solution can always be efficiently built; see [71, 173]. In the case of $n > 1$ equations, the situation is more complicated. We defer to Section 4.9 further considerations on the computation of stream coefficients, including details on an effective implementation of (4.5).

For the time being, we observe that the product operations defined in the preceding subsection enjoy a formulation in terms of SDEs. In particular (see [17, 89, 155]), for given σ and τ , their convolution and shuffle products are the unique streams satisfying the following SDEs (recall that, as a stream, x denotes $(0, 1, 0, 0, \dots)$):

$$(\sigma \times \tau)(0) = \sigma(0) \cdot \tau(0) \quad (\sigma \times \tau)' = \sigma' \times \tau + \sigma \times \tau' - x \times \sigma' \times \tau' \quad (4.6)$$

$$(\sigma \otimes \tau)(0) = \sigma(0) \cdot \tau(0) \quad (\sigma \otimes \tau)' = \sigma' \otimes \tau + \sigma \otimes \tau'. \quad (4.7)$$

From the last equation, note the analogy between shuffle and interleaving of languages. Moreover, the derivative of convolution product is usually defined as

$$(\sigma \times \tau)' = \sigma' \times \tau + \sigma(0) \times \tau'.$$

We shall generally prefer formula (4.6) because it is symmetric. The multiplicative inverse of a stream σ w.r.t. \times and \otimes exists under the condition that $\sigma(0) \neq 0$. In the case of convolution product, the inverse is denoted by σ^{-1} and satisfies the following SDE and initial condition:

$$(\sigma^{-1})' = -\sigma(0)^{-1} \sigma' \times \sigma^{-1} \quad (\sigma^{-1})(0) = \sigma(0)^{-1}. \quad (4.8)$$

Two additional examples of stream products are introduced below; see [17] for the underlying motivations. The *Hadamard product* \odot and the *infiltration product* \uparrow can be defined by the following two SDEs.

$$(\sigma \odot \tau)(0) = \sigma(0)\tau(0) \quad (\sigma \odot \tau)' = \sigma' \odot \tau' \quad (4.9)$$

$$(\sigma \uparrow \tau)(0) = \sigma(0)\tau(0) \quad (\sigma \uparrow \tau)' = (\sigma' \uparrow \tau) + (\sigma \uparrow \tau') + (\sigma' \uparrow \tau'). \quad (4.10)$$

Hadamard product \odot is reminiscent of synchronization in concurrency theory and has $\underline{1} \triangleq (1, 1, 1, \dots)$ as an identity; it is just the componentwise product of two streams, i.e. $(\sigma \odot \tau)(i) = \sigma(i)\tau(i)$, for every $i \geq 0$. Infiltration product \uparrow is again reminiscent of a notion in concurrency theory, namely the fully synchronized interleaving; it has $1 = (1, 0, 0, \dots)$ as an identity.

4.3 (Co)algebraic semantics of polynomials and differential equations

The main result of this section is that, once fixed an initial value problem (\mathcal{D}, ρ) , for every product π (with identity 1_π) defined on streams and satisfying certain syntactic conditions, one can build a coalgebra over polynomials such that the corresponding final *morphism* into Σ is also a commutative \mathbb{K} -algebra *homomorphism* from $(\mathcal{P}, +, \cdot, 0, 1)$ to $(\Sigma, +, \pi, 0, 1_\pi)$. In essence, the polynomial syntax and operational semantics reflects exactly the algebraic and coalgebraic properties of the considered π on streams.

To make polynomials a coalgebra, we need to define the output $o : \mathcal{P} \rightarrow \mathbb{K}$ and transition $\delta : \mathcal{P} \rightarrow \mathcal{P}$ functions. The definition of $o(\cdot)$ is straightforward and only depends on the given initial conditions ρ : we let $o \triangleq o_\rho$ be the homomorphic extension of ρ , seen as a function defined over \mathcal{X} , to \mathcal{P} . Equivalently, seeing ρ as a point in

\mathbb{K}^{n+1} , we let $o_\rho(p) \triangleq p(\rho)$, that is the polynomial p evaluated at the point ρ . It can be easily checked that $o_\rho(1) = 1$.

The definition of δ , on the other hand, depends on π and is not straightforward. We will confine ourselves to products π satisfying SDEs of the form: $(\sigma \pi \tau)' = F(\sigma, \tau, \dots)$, for a given *polynomial* function F . Then, we will require that δ on polynomials mimics this equation. For instance, in the case of shuffle product, we expect that $\delta(pq) = p\delta(q) + q\delta(p)$. Therefore, our first step is to precisely define the class of products on streams that satisfy a polynomial SDE. To this purpose, in what follows we shall consider polynomials $G(y_1) \in \mathbb{K}[y_1]$ and $F(x, y_1, \dots, y_4) \in \mathbb{K}[x, y_1, y_2, y_3, y_4]$. These can be identified with polynomial functions on streams: we shall write $G(\sigma_1)$, $F(x, \sigma_1, \dots, \sigma_4)$ for the evaluation of G, F in $(\Sigma, +, \pi, 0, 1_\pi)$ with specific streams $x = (0, 1, 0, \dots)$ and $\sigma_1, \dots, \sigma_4$.

Definition 8 ((F, G)-product on streams). *Let $(\Sigma, +, \pi, 0, 1_\pi)$ be a commutative \mathbb{K} -algebra, $F \in \mathbb{K}[x, y_1, y_2, y_3, y_4]$ and $G \in \mathbb{K}[y_1]$. We say that π is a (F, G) -product if, for each $\sigma, \tau \in \Sigma$, the following equations are satisfied:*

1. $(\sigma \pi \tau)(0) = \sigma(0)\tau(0)$ and $(\sigma \pi \tau)' = F(x, \sigma, \sigma', \tau, \tau')$;
2. $1_\pi(0) = 1$ and $1'_\pi = G(1_\pi)$.

Remark 5. Notice that $1_\pi(0) = 1$ in Definition 8(2) is a necessary condition, that follows from Definition 8(1). Indeed, let $1_\pi(0) = r \in \mathbb{K}$. Since 1_π is the identity of π , for every σ we must have $\sigma \pi 1_\pi = \sigma$, hence $(\sigma \pi 1_\pi)(0) = \sigma(0)$. On the other hand, by Definition 8(1), $(\sigma \pi 1_\pi)(0) = \sigma(0) 1_\pi(0) = \sigma(0) r$. As σ is arbitrary, we can take $\sigma(0) \neq 0$ and multiply $\sigma(0) r = \sigma(0)$ by $\sigma(0)^{-1}$; this gives $r = 1$. However, we prefer to keep $1_\pi(0) = 1$ explicit in the definition, for the sake of clarity. As an aside, we note that (F, G) -products fall in the GSOS format of [89]: this ensures that, for any given (F, G) , conditions (1) and (2) in Definition 8 univocally define a binary operation π on streams. However, this does not immediately entail that π enjoys the commutative ring axioms for product, a fact that we assume from the outset so as to ensure that Σ forms a commutative \mathbb{K} -algebra. We will return to this point in Remark 6 in Section 4.3.

Example 8. For the products introduced in Section 4.2, the pairs of polynomials (F, G) are defined as follows:

- $F_\times = y_2y_3 + y_1y_4 - xy_2y_4$. Note that $F_\times = y_2y_3 + (y_1 - xy_2)y_4$, where $y_1 - xy_2$ corresponds to $\sigma - x \times \sigma' = \sigma(0)$; this gives the asymmetric definition of convolution.
- $F_\otimes = y_2y_3 + y_1y_4$.
- $F_\odot = y_2y_4$.
- $F_\uparrow = y_2y_3 + y_1y_4 + y_2y_4$.

The identity stream for convolution, shuffle and infiltration is defined by $1_\pi(0) = 1$ and $1'_\pi = 0$, i.e., in these cases the polynomial G is 0. For the Hadamard product, the identity is given by $1_\pi(0) = 1$ and $1'_\pi = 1_\pi$, i.e., the polynomial G in this case is y_1 .

Given a (F, G) -product π on streams, δ is defined in a straightforward manner on monomials, then extended to polynomials by linearity. Below, we assume a total order on variables $x_0 < x_1 < \dots < x_n$ and, for any monomial $m \neq 1$, let $\min(m)$ denote the smallest variable occurring in m w.r.t. such a total order³.

NOTATION. In what follows, the definition of derivative δ (Definition 9) and of coalgebra morphism μ (Theorem 4) depend not only on π , but also on a given initial value problem (\mathcal{D}, ρ) . To avoid excessive notational burden, we will assume that (\mathcal{D}, ρ) is fixed once and for all, and omit the dependence on it from the notation, writing e.g. δ_π in place of $\delta_{\pi, (\mathcal{D}, \rho)}$, and so on.

Definition 9 (transition function δ_π). *Let π be a (F, G) -product on streams. Given an initial value problem (\mathcal{D}, ρ) , we define $\delta_\pi : \mathcal{P} \rightarrow \mathcal{P}$ by induction on the size of $p \in \mathcal{P}$ as follows:*

$$\delta_\pi(1) = G(1) \quad (4.11)$$

$$\delta_\pi(x_i) = \mathcal{D}(x_i) \quad (4.12)$$

$$\delta_\pi(x_i m) = F(x, x_i, \delta_\pi(x_i), m, \delta_\pi(m)) \quad \text{for } m \neq 1 \text{ and } x_i = \min(x_i m) \quad (4.13)$$

$$\delta_\pi \left(\sum_{i \in I} r_i m_i \right) = \sum_{i \in I} r_i \delta_\pi(m_i). \quad (4.14)$$

The formal similarity between the definition given above and the construction in [89, Prop.5.4] is noteworthy. Expressed in the language of monads, an important difference is that the monad \mathcal{V} of vector spaces in [89] should be replaced here with the monad \mathcal{P} of commutative multivariate polynomials; we will return to this point in Remark 6.

Example 9. Returning to the products defined in Section 4.2, we have:

$$\delta_\pi(1) = \begin{cases} 0 & \text{for } \pi \in \{\times, \otimes, \uparrow\} \text{ (convolution, shuffle, infiltration)} \\ 1 & \text{for } \pi = \odot \text{ (Hadamard)} \end{cases}$$

$$\delta_\pi(x_i m) = \begin{cases} \mathcal{D}(x_i) \cdot m + x_i \cdot \delta_\pi(m) - x \cdot \mathcal{D}(x_i) \cdot \delta_\pi(m) & \text{for } \pi = \times \text{ (convolution)} \\ \mathcal{D}(x_i) \cdot m + x_i \cdot \delta_\pi(m) & \text{for } \pi = \otimes \text{ (shuffle)} \\ \mathcal{D}(x_i) \cdot \delta_\pi(m) & \text{for } \pi = \odot \text{ (Hadamard)} \\ \mathcal{D}(x_i) \cdot m + x_i \cdot \delta_\pi(m) + \mathcal{D}(x_i) \cdot \delta_\pi(m) & \text{for } \pi = \uparrow \text{ (infiltration)}. \end{cases}$$

We must now impose certain additional sanity conditions on F to ensure that the final coalgebra morphism induced by δ_π , as just defined, is also an algebra homomorphism. In the rest of this chapter, we will make use of the following abbreviation

$$F_\pi[p; q] \triangleq F(x, p, \delta_\pi(p), q, \delta_\pi(q)).$$

The necessity of the following conditions is self-evident, if one thinks of $F_\pi[p; q]$ as $\delta_\pi(p \cdot q)$ (see Lemma 5 below).

³In Definition 9, we are in effect totally ordering monomials by graded lexicographic order (grlex, see [57, Ch.1]), and then proceeding by induction on this order.

Definition 10 (well-behavedness). Let π be a (F, G) -product on streams. We say that π is well-behaved if, for each initial value problem (\mathcal{D}, ρ) , the following equalities hold, for every $p, q \in \mathcal{P}$, $m_1, m_2, m_i \in \mathcal{M}$, $x_i \in \mathcal{X}$ and $r_i \in \mathbb{K}$:

$$F_\pi[1; q] = \delta_\pi(q) \quad (4.15)$$

$$F_\pi[x_i m_1; m_2] = F_\pi[m_1; x_i m_2] \quad (4.16)$$

$$F_\pi \left[\sum_{i \in I} r_i m_i ; q \right] = \sum_{i \in I} r_i F_\pi[m_i; q] \quad (4.17)$$

$$F_\pi[p; q] = F_\pi[q; p]. \quad (4.18)$$

All products defined in Section 4.2 are well-behaved: the proof of this fact, which is not entirely trivial, is reported in Appendix C (see Proposition C.1.1).

We are now ready to prove the main result of this section, i.e. that μ_π is a commutative \mathbb{K} -algebra homomorphism. Intuitively, its proof consists in showing that μ_π preserves all the operations in \mathcal{P} , by exhibiting in each case an appropriate bisimulation relation in $\Sigma \times \Sigma$ and then applying coinduction. As expected, the most crucial case is product, where one shows that the relation consisting of all pairs $(\mu_\pi(p_1 \cdot \dots \cdot p_k), \mu_\pi(p_1) \pi \dots \pi \mu_\pi(p_k))$ (for $k > 0$) is a bisimulation up to linearity. In this case, Lemma 5 below is used to prove that μ_π preserves transitions: indeed, it connects morphism to homomorphism properties induced by π . The proof is in Appendix C.

Lemma 5. Let π be a well-behaved (F, G) -product. Then, for every $p, q \in \mathcal{P}$, it holds that $\delta_\pi(p \cdot q) = F_\pi[p; q]$.

In the next proof and in the rest of this section, we will use the following notation. Given a polynomial substitution (i.e., a map from variables to polynomials) ζ , and a monomial $m = x_{i_1} \dots x_{i_k}$, we let $m\zeta$ denote the polynomial $\zeta(x_{i_1}) \cdot \dots \cdot \zeta(x_{i_k})$. Similarly, given a stream substitution (i.e., a map from variables to streams) ξ , we let $m\xi$ denote the stream $\xi(x_{i_1}) \pi \dots \pi \xi(x_{i_k})$.

Theorem 4. Let π be a well-behaved (F, G) -product. Then the (unique) coalgebra morphism μ_π from $(\mathcal{P}, \delta_\pi, o_\rho)$ to $(\Sigma, (\cdot)^\prime, o)$ is a commutative \mathbb{K} -algebra homomorphism from $(\mathcal{P}, +, \cdot, 0, 1)$ to $(\Sigma, +, \pi, 0, 1_\pi)$.

PROOF. We prove that $\mu = \mu_\pi$ preserves the ring operations and their identities, as well as multiplication by a scalar.

1. $\mu(r_1 p + r_2 q) = r_1 \mu(p) + r_2 \mu(q)$. We prove that

$$R = \{(\mu(r_1 p + r_2 q), r_1 \mu(p) + r_2 \mu(q)) : p, q \in \mathcal{P}, r_1, r_2 \in \mathbb{K}\}$$

is a bisimulation. Pick up any $(\mu(r_1 p + r_2 q), r_1 \mu(p) + r_2 \mu(q)) \in R$, we need to prove two conditions.

- (a) $\mu(r_1 p + r_2 q)(0) = (r_1 \mu(p) + r_2 \mu(q))(0)$: since μ is a coalgebra morphism and by definitions, $\mu(r_1 p + r_2 q)(0) = o_\rho(r_1 p + r_2 q) = r_1 o_\rho(p) + r_2 o_\rho(q) = r_1 \mu(p)(0) + r_2 \mu(q)(0) = (r_1 \mu(p) + r_2 \mu(q))(0)$.

(b) $\mu(r_1p + r_2q)' = (r_1\mu(p) + r_2\mu(q))'$: since μ is a coalgebra morphism, by (4.14) and by definitions, $\mu(r_1p + r_2q)' = \mu(\delta_\pi(r_1p + r_2q)) = \mu(r_1\delta_\pi(p) + r_2\delta_\pi(q))$, whereas by definition and by the fact that μ is a coalgebra morphism, $(r_1\mu(p) + r_2\mu(q))' = r_1\mu(p)' + r_2\mu(q)' = r_1\mu(\delta_\pi(p)) + r_2\mu(\delta_\pi(q))$. This suffices to conclude, since $(\mu(r_1\delta_\pi(p) + r_2\delta_\pi(q)), r_1\mu(\delta_\pi(p)) + r_2\mu(\delta_\pi(q))) \in R$.

2. Concerning the identity of sum, by part 1 we have that $\mu(0) = \mu(p - p) = \mu(p) - \mu(p) = 0$.
3. $\mu(1) = 1_\pi$. It suffices to prove that

$$R = \{ (\mu(1), 1_\pi) \}$$

is a bisimulation up to linearity. To this aim, we need to check two conditions.

- (a) $\mu(1)(0) = 1_\pi(0)$. Since μ is a coalgebra morphism, by definition of 1, and by Def. 8(2), we have that $\mu(1)(0) = o_\rho(1) = 1 = 1_\pi(0)$.
- (b) $\mu(1)' = 1'_\pi$. Since G is a polynomial in the variable y_1 (i.e. $G = \sum_{i \in I} r_i m_i$, where the m_i 's are monomials in y_1), we have that

$$\begin{aligned} \mu(1)' &= \mu(\delta_\pi(1)) && \mu \text{ is a coalgebra morphism} \\ &= \mu(G(1)) && \text{by (4.14) and (4.12)} \\ &= \sum_{i \in I} r_i \mu(m_i \zeta) && \text{point 1 of this proof} \\ &= \sum_{i \in I} r_i \mu(1) \end{aligned}$$

where ζ is the substitution that maps y_1 to 1, hence all monomials m_i evaluated under ζ yield 1, which justifies the last step above. By Definition 8(2) and definition of G , we have that $1'_\pi = G(1_\pi) = \sum_{i \in I} r_i (m_i \xi) = \sum_{i \in I} r_i 1_\pi$ where ξ is the substitution that maps y_1 to 1_π , hence all monomials m_i evaluated under ξ yield 1_π . This suffices to conclude up to linearity.

4. $\mu(p \cdot q) = \mu(p) \pi \mu(q)$. To prove this fact, let us consider the relation

$$R = \{ (\mu(p_1 \cdot \dots \cdot p_k), \mu(p_1) \pi \dots \pi \mu(p_k)) : p_i \in \mathcal{P}, k > 0 \}$$

and prove that it is a bisimulation up to linearity. Let us consider any $(\sigma, \tau) = (\mu(p_1 \cdot \dots \cdot p_k), \mu(p_1) \pi \dots \pi \mu(p_k)) \in R$. We will prove that (a) $o(\sigma) = \sigma(0) = \tau(0) = o(\tau)$ and (b) $(\sigma', \tau') \in \widehat{R}$ (the linear closure of R). The case $k = 1$ is trivial, so assume $k > 1$. Let $q = p_2 \cdot \dots \cdot p_k$. We check conditions (a) and (b) defined above.

- (a) $\mu(p_1 \cdot q)(0) = (\mu(p_1) \pi \mu(q))(0)$:

$$\begin{aligned} \mu(p_1 \cdot q)(0) &= o_\rho(p_1 \cdot q) && \text{since } \mu \text{ is a coalgebra morphism} \\ &= o_\rho(p_1) o_\rho(q) && \text{by def. of } o_\rho(\cdot) \\ &= \mu(p_1)(0) \mu(q)(0) && \text{since } \mu \text{ is a coalgebra morphism} \\ &= (\mu(p_1) \pi \mu(q))(0) && \text{by Definition 8(1)}. \end{aligned}$$

- (b) $\mu(p_1 \cdot q)' \widehat{R} (\mu(p_1) \pi \mu(q))'$: By assumption F is a polynomial in the variables x, y_1, \dots, y_4 , i.e. $F = \sum_{i \in I} r_i m_i$, where $r_i \in \mathbb{K}$ and the m_i 's are monomials in these variables. Let $\zeta : \{x, y_1, \dots, y_4\} \rightarrow \mathcal{P}$ and $\xi : \{x, y_1, \dots, y_4\} \rightarrow \Sigma$ be the substitutions defined as follows.

$$\begin{array}{ll} \zeta : x \mapsto x & \xi : x \mapsto \mu(x) \\ y_1 \mapsto p_1 & y_1 \mapsto \mu(p_1) \\ y_2 \mapsto \delta_\pi(p_1) & y_2 \mapsto \mu(\delta_\pi(p_1)) \\ y_3 \mapsto q & y_3 \mapsto \mu(q) \\ y_4 \mapsto \delta_\pi(q) & y_4 \mapsto \mu(\delta_\pi(q)). \end{array}$$

Then we have

$$\begin{aligned} \mu(p_1 \cdot q)' &= \mu(\delta_\pi(p_1 \cdot q)) && \mu \text{ is a coalgebra morphism} \\ &= \mu(F_\pi[p_1; q]) && \text{Lemma 5} \\ &= \mu(\sum_{i \in I} r_i (m_i \zeta)) && \text{def. of } F[\cdot; \cdot] \\ &= \sum_{i \in I} r_i \mu(m_i \zeta) && \text{part 1 of this proof} \end{aligned}$$

and

$$\begin{aligned} (\mu(p_1) \pi \mu(q))' &= F(\mu(p_1), \mu(p_1)', \mu(q), \mu(q)') && \text{Def. 8(1)} \\ &= F(\mu(p_1), \mu(\delta_\pi(p_1)), \mu(q), \mu(\delta_\pi(q))) && \mu \text{ coalgebra morphism} \\ &= \sum_{i \in I} r_i (m_i \xi) && \text{def. of } F. \end{aligned}$$

Now, by definition, $(\mu(m_i \zeta), m_i \xi) \in R$ for every $i \in I$. Thus, we conclude up to linearity. \square

To conclude the section, we also present a sort of converse of the previous theorem. That is, μ_π is the only commutative \mathbb{K} -algebra homomorphism that respects the initial value problem, i.e. that satisfies $\mu_\pi(x_i)' = \mu_\pi(\mathcal{D}(x_i))$ and $\mu_\pi(x_i)(0) = \rho(x_i)$. This is an immediate corollary of the following result (proof in Appendix C) and of the uniqueness of the final coalgebra morphism.

Proposition 5. *Let π be a well-behaved (F, G) -product and ν be a commutative \mathbb{K} -algebra homomorphism from $(\mathcal{P}, +, \cdot, 0, 1)$ to $(\Sigma, +, \pi, 0, 1_\pi)$ that respects (\mathcal{D}, ρ) . Then, ν is a coalgebra morphism from $(\mathcal{P}, \delta_\pi, o_\rho)$ to $(\Sigma, (\cdot)', o)$.*

Remark 6 (relations with abstract GSOS and bialgebras). Polynomial syntax and operational semantics might also be described in terms of distributive laws and bialgebras, thus allowing one to leverage known results in this field [25, 89]. Below, we outline this possibility; in doing so, we shall assume a basic knowledge of the language of category theory.

With bialgebras, the algebraic and coalgebraic structures are combined together, and their interaction is modeled via a distributive law. Specifically, a monad is used to describe the (possibly non free) syntax. A distributive law is a natural transformation that is compatible with the monadic and coalgebraic structure. In the case that interests

us, the set of commutative, multivariate polynomials can be modelled as a monad on the category Set (of sets and functions), defined by letting \mathcal{P} be the set of polynomial terms over \mathcal{X} quotiented by the congruence generated by the axioms of commutative \mathbb{K} -algebras. The Eilenberg-Moore algebras for \mathcal{P} are then the commutative \mathbb{K} -algebras. As already noted, the conditions in the definition of (F, G) -product ensure that the product operation and its identity are defined and fall in the abstract GSOS format. The results in [25] imply then the existence of a distributive law λ of the polynomial terms monad over the (copointed) stream functor defining the coalgebraic structure.

We conjecture⁴ that, in this framework, our Theorem 4 — that the coalgebra structure on \mathcal{P} is such that the final coalgebra morphism is a \mathbb{K} -algebra homomorphism for every well behaved (F, G) -product — should follow from Proposition 3 and Theorem 1 in [25], provided one can show that, for a *generic* (F, G) -product, the corresponding distributive law λ preserves the \mathbb{K} -algebra axioms. In fact, we have checked some of the \mathbb{K} -algebra axioms for the distributive laws induced by the SDEs of specific products, viz. convolution and shuffle. Extending this to the general case of a (F, G) -product would presumably involve proving that well-behavedness (our Definition 10) implies preservation of the \mathbb{K} -algebra axioms. This extension appears nontrivial; we leave a thorough exploration of this connection for future work.

4.4 Deciding stream equality

One benefit of a polynomial syntax is the possibility of applying techniques from algebraic geometry to reason about stream equality. We will devise an algorithm for checking whether two given polynomials are semantically equivalent, that is, are mapped to the same stream under μ_π .

4.4.1 The algorithm

First of all, by linearity of $\mu_\pi(\cdot)$, we have that $\mu_\pi(p) = \mu_\pi(q)$ if and only if $\mu_\pi(p) - \mu_\pi(q) = \mu_\pi(p - q) = 0$. Therefore, checking semantic equivalence of two polynomials reduces to the problem of checking if a polynomial is equivalent (bisimilar) to 0. Before introducing the actual algorithm for checking this, we quickly recall a few notions from algebraic geometry; see [57, Ch.1–4] for a comprehensive treatment.

Definition 11 (Ideal). *A set of polynomials $I \subseteq \mathcal{P}$ is an ideal if $0 \in I$ and, for all $p_1, p_2 \in I$ and $q \in \mathcal{P}$, it holds that $p_1 + p_2 \in I$ and $q \cdot p_1 \in I$. Given a set of polynomials S , the ideal generated by S is*

$$\langle S \rangle \triangleq \left\{ \sum_{j=1}^k q_j \cdot p_j : k \geq 0 \wedge \forall j \leq k. (q_j \in \mathcal{P} \wedge p_j \in S) \right\}.$$

By the previous definition, we have that $\langle \emptyset \rangle \triangleq \{0\}$. Trivially, $I = \langle S \rangle$ is the smallest ideal containing S , and S is called a set of *generators* for I . It is well-known that every ideal I admits a finite set S of generators (Hilbert’s basis theorem). By virtue of this result, any infinite ascending chain of ideals, $I_0 \subseteq I_1 \subseteq I_2 \subseteq \dots \subseteq \mathcal{P}$,

⁴We thank one of the reviewers for pointing out this connection.

Algorithm 2 Checking equivalence to zero

Input: $p \in \mathcal{P}$, an initial value problem (\mathcal{D}, ρ) , a well-behaved (F, G) -product π

Output: YES ($\mu_\pi(p) = 0$) or NO ($\mu_\pi(p) \neq 0$)

- 1: **for all** $k \geq 0$ **do**
 - 2: **if** $o_\rho(p^{(k)}) \neq 0$ **then return** NO
 - 3: **if** $p^{(k)} \in \langle \{p^{(0)}, \dots, p^{(k-1)}\} \rangle$ **then return** YES
 - 4: **end for**
-

stabilizes in a finite number of steps: that is, there is $k \geq 0$ s.t. $I_{k+j} = I_k$ for each $j \geq 0$ (Ascending Chain Condition, ACC). A key result due to Buchberger [57] is that, given a finite $S \subseteq \mathcal{P}$, it is possible to decide whether $p \in I = \langle S \rangle$, for any polynomial p . As a consequence, also ideal inclusion $I_1 \subseteq I_2$ is decidable, given finite sets of generators for I_1, I_2 .

Remark 7. These facts are consequences of the existence of a set of generators B for I , called *Groebner basis*, with a special property (see [57, Chpt.2-§6-Cor.2]): $p \in I$ if and only if $p \bmod B = 0$, where ‘ $p \bmod B$ ’ denotes the remainder of the multivariate polynomial division of p by B . Indeed, by [57, Chpt.2-§3-Thm.3], we can define the notion of multivariate polynomial division by a set of polynomials and, when such a set is a Groebner basis [57, Chpt.2-§5-Def.5], we know by [57, Chpt.2-§6-Prop.1] that the remainder of the division, denoted by $p \bmod B$, is unique (though the quotient is not). There exist algorithms to build Groebner bases which, despite their exponential worst-case complexity, turn out to be effective in many practical cases [57, Ch.4].

In what follows, we fix a well-behaved (F, G) -product π , and let δ_π and μ_π denote the associated transition function and coalgebra morphism. Moreover, we denote by $p^{(j)}$ the j -th derivative of p , i.e. $p^{(0)} \triangleq p$ and $p^{(j+1)} \triangleq \delta_\pi(p^{(j)})$. The actual decision procedure is presented as Algorithm 2. This algorithm characterizes the *kernel* of the morphism μ_π , that is $\ker(\mu_\pi) := \{p \in \mathcal{P} : \mu_\pi(p) = 0\}$. Informally speaking, to prove that $\mu_\pi(p) = 0$, one might check if $o_\rho(p^{(j)}) = 0$ for every j , which is of course non effective. But due to ACC, at some point $p^{(j)} \in \langle \{p^{(0)}, \dots, p^{(j-1)}\} \rangle$, which implies that the condition $o_\rho(p^{(j)}) = 0$ holds for *all* j 's.

Formally, the correctness of this algorithm can be proven under an additional mild condition on F : we require that $F \in \langle \{y_3, y_4\} \rangle$ seen as an ideal in $\mathbb{K}[x, y_1, \dots, y_4]$. Explicitly, $F = h_1 y_3 + h_2 y_4$ for some $h_1, h_2 \in \mathbb{K}[x, y_1, \dots, y_4]$. The polynomials F for the products in Section 4.2 all satisfy this condition: for example, $F_\times = y_2 y_3 + (y_1 - x y_2) y_4$.

Theorem 5. *Let π be a well-behaved (F, G) -product, with $F \in \langle \{y_3, y_4\} \rangle$. Then, Algorithm 2 with input p , (\mathcal{D}, ρ) and π terminates and it returns YES if and only if $\mu_\pi(p) = 0$.*

PROOF. Non termination for some input polynomial p would imply that, for all $k \geq 0$, $p^{(k+1)} \notin I_k \triangleq \langle \{p^{(0)}, \dots, p^{(k)}\} \rangle$. This in turn would imply an ever ascending chain of ideals $I_0 \subsetneq I_1 \subsetneq \dots$, contradicting ACC.

If the algorithm returns NO, then for some k we must have (recall that $\sigma^{(k)}$ stands for the k -th stream derivative of σ): $o_\rho(p^{(k)}) = o(\mu_\pi(p)^{(k)}) = (\mu_\pi(p)^{(k)})(0) \neq 0$, thus $\mu_\pi(p) \neq 0$.

Assume now that the algorithm returns YES. Then there exists $k \geq 0$ such that $o_\rho(p^{(j)}) = 0$, for every $0 \leq j \leq k$, and $p^{(k)} \in \langle \{p^{(0)}, \dots, p^{(k-1)}\} \rangle$. Excluding the trivial case $p = 0$, we can assume $k \geq 1$. If we prove that $p^{(k+j)} \in \langle \{p^{(0)}, \dots, p^{(k-1)}\} \rangle$ for every $j \geq 0$, the claim follows: indeed, by $p^{(k+j)} = \sum_{i=0}^{k-1} q_i \cdot p^{(i)}$, for some $q_i \in \mathcal{P}$, and by $o_\rho(p^{(i)}) = 0$ for every $0 \leq i \leq k-1$, it also follows $(\mu_\pi(p))^{(j)} = (\mu_\pi(p))^{(j)}(0) = o_\rho(p^{(k+j)}) = 0$. Now the proof that $p^{(k+j)} \in \langle \{p^{(0)}, \dots, p^{(k-1)}\} \rangle$ is by induction on j . The base case ($j = 0$) holds by assumption. For the induction step, let us consider $p^{(k+j+1)}$. By definition, $p^{(k+j+1)} = \delta_\pi(p^{(k+j)})$; by induction $p^{(k+j)} = \sum_{i=0}^{k-1} q_i \cdot p^{(i)}$, for some $q_i \in \mathcal{P}$. By (4.14) and Lemma 5, $p^{(k+j+1)} = \sum_{i=0}^{k-1} \delta_\pi(q_i \cdot p^{(i)}) = \sum_{i=0}^{k-1} F_\pi[q_i; p^{(i)}]$. By hypothesis $F \in \langle \{y_3, y_4\} \rangle$, hence $F_\pi[q_i; p^{(i)}] \in \langle \{p^{(i)}, p^{(i+1)}\} \rangle$, for every i , therefore $F_\pi[q_i; p^{(i)}] \in \langle \{p^{(0)}, \dots, p^{(k-1)}\} \rangle$, as by hypothesis $p^{(k)} \in \langle \{p^{(0)}, \dots, p^{(k-1)}\} \rangle$. This suffices to conclude. \square

We first illustrate the algorithm with a simple, linear example.

Example 10 (Fibonacci numbers). Consider the initial value problem (\mathcal{D}, ρ) given by the following equations.

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = x_1 + x_2 \end{cases} \quad \begin{cases} \rho(x_1) = 0 \\ \rho(x_2) = 1. \end{cases} \quad (4.19)$$

Let us consider here the convolution product \times . It is easily checked that x_1 defines the Fibonacci numbers: $\mu_\times(x_1) = (0, 1, 1, 2, 3, 5, 8, 13, \dots)$. We want to prove the following equation:

$$\mu_\times(x_1 \cdot (1 - x - x^2)) = \mu_\times(x). \quad (4.20)$$

Equivalently, using Algorithm 2, we check that $\mu_\times(x_1 \cdot (1 - x - x^2) - x) = 0$. Let $p(x, x_1) \triangleq x_1 \cdot (1 - x - x^2) - x$. Then, an execution of Algorithm 2 consists of the following steps.

- **($k = 0$):** $\rho(p) = p(0, 1) = 0$ and $p^{(0)} = p(x, x_1) \notin \langle \emptyset \rangle = \{0\}$.
- **($k = 1$):** $p^{(1)} = x_2 \cdot (1 - x - x^2) - x_1 \cdot (1 + x) + x \cdot x_2 \cdot (1 + x) - 1 = x_2 - x_1 - x_1 x - 1$. Hence, $\rho(p^{(1)}) = 1 - 1 = 0$ and $p^{(1)} \notin \langle p \rangle$.
- **($k = 2$):** $p^{(2)} = x_1 + x_2 - x_2 - (x_2 x + x_1 - x x_2) = 0$. Hence, $\rho(p^{(2)}) = 0$ and trivially $p^{(2)} \in \langle p, p^{(1)} \rangle$.

We conclude that $\mu_\times(p) = 0$.

The algorithm in [27] for polynomial ODEs can be seen as a special case of the algorithm presented here, obtained by letting $\pi = \otimes$, the shuffle product. Indeed, it is not difficult to see that δ_\otimes coincides with the *Lie derivative*, on which the algorithm in [27] is based. Technically, the key step to obtain the present generalization is enucleating a sufficient condition under which, as soon as $I_i = I_{i+1}$, the ideals chain gets stable: this is the syntactic requirement that the derivative of the product is in the ideal generated by the arguments, $F \in \langle \{y_3, y_4\} \rangle$. Let us now discuss a nonlinear example based on shuffle product.

Example 11 (double factorial of odd numbers). Consider the initial value problem (\mathcal{D}, ρ) given by the following equation:

$$\dot{x}_1 = x_1^3 \quad \rho(x_1) = 1. \quad (4.21)$$

Let us consider here the shuffle product \otimes . It is easily checked that $\mu_{\otimes}(x_1) = (1, 1, 3, 15, 105, 945, 10395, 135135, \dots)$, the sequence of double factorials of odd numbers (sequence A001147 in [139]). We want to check the following equation

$$\mu_{\otimes} \left(x_1^2 \left(x - \frac{1}{2} \right) + \frac{1}{2} \right) = 0 \quad (4.22)$$

using Algorithm 2. Let $q(x, x_1) \triangleq x_1^2(x - \frac{1}{2}) + \frac{1}{2}$. An execution of Algorithm 2 consists of the following steps.

- $(k = \mathbf{0})$: $\rho(q) = q(0, 1) = 0$ and $q^{(0)} = q(x, x_1) \notin \langle \emptyset \rangle = \{0\}$.
- $(k = \mathbf{1})$: $q^{(1)} = 2x_1^4x - x_1^4 + x_1^2 = 2x_1^2q$, hence $q^{(1)} \in \langle q \rangle$.

We conclude that $\mu_{\otimes}(q) = 0$. As δ_{\otimes} represents the Lie derivative of polynomials, the equality $\mu_{\otimes}(q) = 0$ can also be interpreted in terms of ODES: the unique solution $x_1(x)$ of the initial value problem (4.22) satisfies the invariant $q(x, x_1(x)) = 0$ for all x 's in the interval of definition of the solution. In turn, this allows to compute algebraically the solution $x_1(x) = \sqrt{\frac{1}{1-2x}}$. See [27, 29] for additional details in the case of ODES.

Remark 8. We can define the generating function associated to Fibonacci numbers, that is the function $g(z)$ whose Taylor series expansion from $z = 0$ is $\sum_{j \geq 0} f_j z^j$, where f_j are the Fibonacci numbers. For z in the radius of convergence of this series, we have

$$g(z) = \frac{z}{1 - z - z^2}. \quad (4.23)$$

From [17] it is known that the convolution product inverse of a given stream σ exists whenever $\sigma(0) \neq 0$. From (4.20) we obtain

$$\mu_{\times}(x_1) = \mu_{\times}(x) \times (\mu_{\times}(1 - x - x^2))^{-1} = \frac{\mu_{\times}(x)}{1 - \mu_{\times}(x) - \mu_{\times}(x)^2}$$

where we use the usual notation $\frac{\sigma}{\tau}$ to denote $\sigma \times \tau^{-1}$. This equation for $\mu_{\times}(x_1)$ is structurally identical to (4.23): this is of course no coincidence, as algebraic identities on streams correspond exactly to algebraic identities on generating functions. Similarly, the equivalence $\mu_{\otimes}(p) = 0$ obtained for the double factorial equations yields the exponential generating function for A001147 when solved algebraically for x_1 , that, as seen in Example 11 above, is $g(z) = \sqrt{\frac{1}{1-2z}}$. For additional details on generating functions and algebraic series, see [26].

Remark 9 (complexity). The exact theoretical complexity of Algorithm 2 is difficult to characterize, as it also depends on the specific (F, G) -product π that is considered. One can try at least to work out some very conservative bounds. Let us denote by d the sum of the degree of the input polynomial p and of the maximal degree of

polynomials in \mathcal{D} , and by n the number of variables. Assume that the algorithm stops at iteration $i + 1$, where i is the least integer such that $p^{(i+1)} \in \langle \{p^{(0)}, \dots, p^{(i)}\} \rangle$. We note that: (a) each iteration of the main loop involves the computation of a Groebner basis, for which known algorithms have an exponential worst case time complexity upper bounded approximately by $O(D^{2^n})$, where D is the maximum degree in the input polynomial set (see [57]); (b) the maximum degree D of the derivatives $p^{(k)}$'s, for $0 \leq k \leq i$, depends on the actual (F, G) -product that is considered. For instance, in the case $\pi = \otimes$, it is not difficult to see that $D \leq i \cdot d$, which gives a worst case time complexity of approximately $O(i^{2^N+1} d^{2^N})$. The number of steps i before stabilization also depends on the actual (F, G) -product. As an example, in the case of shuffle product (Lie derivative), according to a result in [138], the number of steps i before stabilization of an ascending chain of ideals generated by successive Lie derivatives is upper bounded by $d^{N^{O(N^2)}}$. One should stress that these are *very conservative* bounds, and that the algorithms works reasonably well in many practical cases.

4.4.2 A fixed-point theoretic perspective

To set our algorithm in a more general coalgebraic perspective, it is useful to relate it to a characterization of the kernel's morphism in terms of fixed points. Given any coalgebra with outputs in \mathbb{K} , say $C = (S, \delta, o)$, consider the function $\Phi : 2^S \rightarrow 2^S$ defined below, for any $I \subseteq S$:

$$\Phi(I) := \{s \in S : o(s) = 0 \wedge \delta(s) \in I\}. \quad (4.24)$$

We say that I is a *post-fixed point* of Φ if $I \subseteq \Phi(I)$ and a *fixed point* if this inclusion holds with equality. Let us denote by $\text{GFP}(\Phi)$ the greatest fixed point of Φ . An easy application of the Kanster-Tarski fixed point theorem and of the monotonicity of Φ shows that the unique coalgebra morphism μ from C to the final coalgebra of streams can be characterized as $\ker(\mu) = \text{GFP}(\Phi) = \bigcup \{I : I \text{ is a post-fixed point of } \Phi\}$. In terms of the function Φ induced by the polynomials coalgebra $C = (\mathcal{P}, \delta_\pi, o_\rho)$, Algorithm 2, in case of a YES answer, builds precisely a post-fixed point $I = \langle \{p^{(0)}, \dots, p^{(k-1)}\} \rangle$: the minimal post-fixed point that is an ideal and contains the input polynomial p . In general, however, $I \neq \text{GFP}(\Phi)$. This leaves open the problem of actually computing $\text{GFP}(\Phi)$, that is $\ker(\mu_\pi)$, which can be regarded as the main object of interest here. The theory of fixed points ensures that $\text{GFP}(\Phi)$ can be iteratively obtained as $\text{GFP}(\Phi) = \bigcap_{j \geq 0} \Phi^{(j)}(\mathcal{P})$ where, inductively, $\Phi^{(0)}(\mathcal{P}) := S$ and $\Phi^{(j+1)}(\mathcal{P}) := \Phi(\Phi^{(j)}(\mathcal{P}))$. While the resulting procedure is formally correct, it is far from clear how to make it effective.

In the next section, we will introduce a generalization of Algorithm 2 that actually allows one to find all polynomials in $\ker(\mu_\pi)$ *up to a prescribed degree*. We shall adopt a hybrid approach: we will start from a set of polynomials of bounded degree, that can be effectively described via a template; then *refine* this set, similarly to the construction of $\text{GFP}(\Phi)$ described above. However, since applying δ_π can actually *increase* the degree of a polynomial, hence leading outside the initial set, at each iteration we shall also need to add polynomials to the current set. Establishing termination of the resulting procedure, as we shall see, is nontrivial.

4.5 Finding polynomial identities

In the previous section we have described an algorithm to check whether two given polynomials have the same denotation — hence, are semantically equivalent. Now we generalize this algorithm and give a method to find *all* valid polynomial equations of a given form. This can be used, for instance, to find all polynomial equalities up to a given degree. To this aim, we use *polynomial templates* [164], a way to compactly specify sets of polynomials. We note that the template-based algorithm presented in [27] is a special instance of the present one, that can be obtained by letting $\pi = \otimes$.

Fix a tuple of $h \geq 1$ distinct *parameters*, say $\mathbf{a} = (a_1, \dots, a_h)$, disjoint from \mathcal{X} . Let $Lin(\mathbf{a})$, ranged over by ℓ , be the set of *linear expressions* with coefficients in \mathbb{K} and variables in \mathbf{a} ; e.g., by taking $\mathbb{K} = \mathbb{R}$, we have that $\ell = 5a_1 + \frac{1}{2}a_2 - 3a_3$ is one such expression⁵. A *template* is an element of the set $Lin(\mathbf{a})[\mathcal{X}]$, that is, a polynomial with linear expressions as coefficients; we let Θ range over templates. For example, the following is a template:

$$\Theta = (5a_1 + 3a_3)xx_1^2 + (7a_1 - \frac{1}{2}a_2 + 2a_3)xx_2 + (a_2). \quad (4.25)$$

Given $\mathbf{r} = (r_1, \dots, r_h) \in \mathbb{K}^h$, we will let $\ell[\mathbf{r}]$ denote the element of \mathbb{K} resulting from the evaluation of the expression obtained by replacing each a_i with r_i in ℓ ; we let $\Theta[\mathbf{r}] \in \mathbb{K}[\mathcal{X}]$ denote the polynomial obtained by replacing each ℓ with $\ell[\mathbf{r}]$ in Θ . For example, by taking Θ as in (4.25), we have that $\Theta[(1, 2, 1)] = 8xx_1^2 + 8xx_2 + 2$. Given a set $R \subseteq \mathbb{K}^h$, we let $\Theta[R]$ denote the set $\{\Theta[\mathbf{r}] : \mathbf{r} \in R\} \subseteq \mathbb{K}[\mathcal{X}]$.

The (formal) derivative of a template is defined as expected, once linear expressions are treated as constants; note that $\delta_\pi(\Theta)$ is still a template. Because of (4.14), for each Θ and \mathbf{r} , one has $\delta_\pi(\Theta[\mathbf{r}]) = \delta_\pi(\Theta)[\mathbf{r}]$; this holds in general for the j -th derivative (for every $j \geq 0$):

$$\delta_\pi^{(j)}(\Theta[\mathbf{r}]) = \delta_\pi^{(j)}(\Theta)[\mathbf{r}]. \quad (4.26)$$

To make notation lighter, when π is clear from the context, we shall write $\delta_\pi^{(j)}(\Theta)$ as $\Theta^{(j)}$.

We now present an algorithm that, given a template Θ with h parameters, finds all instances p of Θ such that $\mu_\pi(p) = 0$. More precisely, given a template Θ , the algorithm computes the intersection of $\Theta[\mathbb{K}^h]$ with the kernel of μ_π

$$\mathcal{Z}\Theta \triangleq \Theta[\mathbb{K}^h] \cap \ker(\mu_\pi). \quad (4.27)$$

Equivalently, $\mathcal{Z} = \{p \in \Theta[\mathbb{K}^h] : \forall j \geq 0, o_\rho(p^{(j)}) = 0\}$, i.e. \mathcal{Z} is the subset of the instances of Θ whose derivatives of any order vanish, when evaluated at the initial conditions ρ . In order to compute $\mathcal{Z}\Theta$, we shall rely on a special kind of ideals, namely *invariants*, that are defined by relying on the following notation: given $P \subseteq \mathcal{P}$, we denote with $\delta_\pi(P)$ the set $\{\delta_\pi(p) : p \in P\}$.

⁵Differently from [164], we do not allow linear expressions with a constant term, such as $2 + 5a_1 + \frac{1}{2}a_2 - 3a_3$. This minor syntactic restriction does not practically affect the expressiveness of the resulting polynomial templates and simplifies the subsequent treatment.

Algorithm 3 Finding all valid equalities of a given form

Input: $\Theta \in \text{Lin}(\mathbf{a})[\mathcal{X}]$, an initial value problem (\mathcal{D}, ρ) , a well-behaved (F, G) -product

π

Output: (R, I) s.t. $\Theta[R] = \mathcal{Z}\Theta$ and I is the smallest invariant that includes $\Theta[R]$

- 1: $R_0 \triangleq \{\mathbf{r} \in \mathbb{K}^h : o_\rho(\Theta[\mathbf{r}]) = 0\}$
 - 2: $I_0 \triangleq \langle \Theta[R_0] \rangle$
 - 3: **for all** $i \geq 1$ **do**
 - 4: $R_i \triangleq \{\mathbf{r} \in \mathbb{K}^h : \forall j \leq i. o_\rho(\Theta^{(j)}[\mathbf{r}]) = 0\}$
 - 5: $I_i \triangleq \langle \bigcup_{j \leq i} \Theta^{(j)}[R_i] \rangle$
 - 6: **if** $R_i = R_{i-1} \wedge I_i = I_{i-1}$ **then return** (R_{i-1}, I_{i-1})
 - 7: **end if**
 - 8: **end for**
-

Definition 12 (Invariant). *Given an initial value problem (\mathcal{D}, ρ) , an ideal I is a (\mathcal{D}, ρ) -invariant (or simply invariant, if the initial value is understood) if $o_\rho(p) = 0$, for each $p \in I$, and $\delta_\pi(I) \subseteq I$.*

The algorithm we are going to present returns a pair (R, I) , where $R \subseteq \mathbb{K}^h$ is such that $\Theta[R] = \mathcal{Z}\Theta$ and I is the smallest invariant that includes $\Theta[R]$. We calculate these two sets by building two chains: a descending chain of vector spaces and an (eventually) ascending chain of ideals. A pseudo-code description⁶ of this procedure is presented as Algorithm 3. Note that each set R_i is actually a vector space in \mathbb{K}^h : this is a consequence of the fact that, for each linear expression ℓ , the set $V_\ell := \{\mathbf{r} \in \mathbb{K}^h : \ell[\mathbf{r}] = 0\}$ is in turn a vector space, and that R_i can equivalently be described as follows, where $\Theta(\rho)$ denotes the polynomial obtained by replacing each $x_i \in \mathcal{X}$ with $\rho(x_i)$:

$$R_i = \bigcap \{V_\ell : \ell \text{ occurs as a coefficient in one of } \Theta(\rho), \dots, \Theta^{(i)}(\rho)\}.$$

The ideal chain is used to detect the stabilization of the sequence. In fact, in the sequence of vector spaces, $R_{i+1} = R_i$ does not imply that $R_{i+k} = R_i$ for each $k \geq 1$; see Example 12 for an illustration of this phenomenon. For this reason, the algorithm returns the pair (R_m, I_m) , where m is the least integer such that $R_{m+1} = R_m$ and $I_{m+1} = I_m$.

The correctness of Algorithm 3 is proven under the same mild condition on F as we assumed for proving correctness of Algorithm 2. We give with a preliminary lemma stating that the algorithm terminates, and that this happens exactly when the two chains stabilize.

Lemma 6. *Consider a template Θ and a well-behaved (F, G) -product π such that $F \in \langle \{y_3, y_4\} \rangle$. Then Algorithm 3 terminates. Furthermore, let (R_i, I_i) be the pair of sets returned by the algorithm; then, $R_i = R_{i+k}$ and $I_i = I_{i+k}$, for every $k \geq 1$.*

PROOF. We first prove that there exists an i such that $R_{i+1} = R_i$ and $I_{i+1} = I_i$. Indeed, $R_0 \supseteq R_1 \supseteq \dots$ forms an infinite descending chain of finite-dimensional vector spaces,

⁶A proof-of-concept implementation of Algorithm 3 is available. Python code, instructions and examples available from <https://github.com/Luisa-unifi/streams>.

which must stabilize in finitely many steps; hence, we can consider the least i' such that $R_{i'} = R_{i'+k}$ for each $k \geq 1$. Then, $I_{i'} \subseteq I_{i'+1} \subseteq \dots$ forms an infinite ascending chain of ideals, which must stabilize at some $i \geq i'$.

Then, let (R_i, I_i) be the sets returned by the algorithm; the proof is by induction on k . The base case holds by line 6 of the algorithm. For the inductive case, we assume that $R_i = \dots = R_{i+k}$ and $I_i = \dots = I_{i+k}$, and we prove that $R_i = R_{i+k+1}$ and $I_i = I_{i+k+1}$. To this aim, we first show that

$$\Theta^{(i+k+1)}[\mathbf{r}] \in I_i \quad \text{for each } \mathbf{r} \in R_i. \quad (4.28)$$

Indeed, by induction hypothesis and definition of I_{i+k} , we have that $\Theta^{(i+k)}[R_i] = \Theta^{(i+k)}[R_{i+k}] \subseteq I_{i+k} = I_i$; hence, for any $\mathbf{r} \in R_i$, we have that $\Theta^{(i+k)}[\mathbf{r}] = \sum_t q_t \cdot \Theta^{(j_t)}[\mathbf{r}_t]$, with $0 \leq j_t \leq i$ and $\mathbf{r}_t \in R_i$. By (4.26), (4.14) and Lemma 5, we have that

$$\Theta^{(i+k+1)}[\mathbf{r}] = \delta_\pi(\Theta^{(i+k)}[\mathbf{r}]) = \sum_t F_\pi[q_t; \Theta^{(j_t)}[\mathbf{r}_t]].$$

Since $F \in \langle \{y_3, y_4\} \rangle$, we have that $F_\pi[q_t; \Theta^{(j_t)}[\mathbf{r}_t]] \in \langle \{\Theta^{(j_t)}[\mathbf{r}_t], \Theta^{(j_t+1)}[\mathbf{r}_t]\} \rangle$, for each t ; since ideals are closed under sum, $\Theta^{(i+k+1)}[\mathbf{r}] \in \langle \bigcup_t \{\Theta^{(j_t)}[\mathbf{r}_t], \Theta^{(j_t+1)}[\mathbf{r}_t]\} \rangle$. This proves (4.28) since, for each t , we have that $\Theta^{(j_t)}[\mathbf{r}_t], \Theta^{(j_t+1)}[\mathbf{r}_t] \in I_{i+1} = I_i$ (by definition of I_{i+1} and induction).

Let us now come to the proof of the inductive step:

$R_i = R_{i+k+1}$: To see this, observe that, for each $\mathbf{r} \in R_{i+k}(= R_i)$, it follows from (4.28) that $\Theta^{(i+k+1)}[\mathbf{r}] = \sum_t q_t \cdot \Theta^{(j_t)}[\mathbf{r}_t]$, with $0 \leq j_t \leq i$ and $\mathbf{r}_t \in R_i$. By construction of R_i , we have that $o_\rho(\Theta^{(i+k+1)}[\mathbf{r}]) = 0$, which shows that $\mathbf{r} \in R_{i+k+1}$. This proves that $R_{i+k} \subseteq R_{i+k+1}$; the reverse inclusion is by construction and, together with the inductive hypothesis, allows us to conclude.

$I_i = I_{i+k+1}$: By the previous point, definition of I_{i+k} , inductive hypothesis and (4.28), we have that

$$\begin{aligned} I_{i+k+1} &= \langle \bigcup_{j=0}^{i+k} \Theta^{(j)}[R_{i+k}] \cup \Theta^{(i+k+1)}[R_{i+k}] \rangle \\ &= \langle I_{i+k} \cup \Theta^{(i+k+1)}[R_{i+k}] \rangle \\ &= \langle I_i \cup \Theta^{(i+k+1)}[R_i] \rangle \\ &= \langle I_i \rangle \\ &= I_i. \end{aligned}$$

□

Theorem 6 (correctness and relative completeness). *Consider a template Θ and a well-behaved (F, G) -product π such that $F \in \langle \{y_3, y_4\} \rangle$. Let (R_i, I_i) be the pair of sets returned by Algorithm 3. Then:*

- (a) $\Theta[R_i] = \mathcal{Z}\Theta$;
- (b) I_i is the smallest invariant containing $\Theta[R_i]$.

PROOF. Concerning part (a), we first note that each $\Theta[\mathbf{r}] \in \mathcal{Z}\Theta$ is such that $o_\rho(\Theta[\mathbf{r}]^{(j)}) = o_\rho(\Theta^{(j)}[\mathbf{r}]) = 0$, for each $j \geq 0$; this, by definition, implies $\mathbf{r} \in R_j$, for each $j \geq 0$, and so $\mathbf{r} \in R_i$. Conversely, if $\mathbf{r} \in R_i = R_{i+1} = R_{i+2} = \dots$ (here we are using Lemma 6), then, by definition, $o_\rho(\Theta[\mathbf{r}]^{(j)}) = o_\rho(\Theta^{(j)}[\mathbf{r}]) = 0$, for each $j \geq 0$, which implies that $\Theta[\mathbf{r}] \in \mathcal{Z}\Theta$. Note that, in proving both inclusions, we used (4.26).

Concerning part (b), we prove that: (1) I_i is an invariant, (2) $I_i \supseteq \mathcal{Z}\Theta$, and (3) every invariant I that contains $\mathcal{Z}\Theta$ also contains I_i .

1. For each $\mathbf{r} \in R_i$ and $j \in \{0, \dots, i-1\}$, we have that $\delta_\pi(\Theta^{(j)}[\mathbf{r}]) = \Theta^{(j+1)}[\mathbf{r}] \in I_i$, while for $j = i$, we have that $\delta_\pi(\Theta^{(i)}[\mathbf{r}]) = \Theta^{(i+1)}[\mathbf{r}] \in I_{i+1} = I_i$, since $\mathbf{r} \in R_i = R_{i+1}$ (in both cases we used (4.26)).
2. Because of part (a), we have that $I_i \supseteq \Theta[R_i] = \mathcal{Z}\Theta$.
3. Let $I \supseteq \mathcal{Z}\Theta$ be an invariant. We show by induction on j that $\Theta^{(j)}[\mathbf{r}] \in I$, for each $\mathbf{r} \in R_i$; this implies the claim. By part (a), $\Theta^{(0)}[\mathbf{r}] = \Theta[\mathbf{r}] \in \mathcal{Z}\Theta$, as $\Theta[R_i] \subseteq \mathcal{Z}\Theta$. Assuming now that $\Theta^{(j)}[\mathbf{r}] \in I$, by invariance of I , we have that $\Theta^{(j+1)}[\mathbf{r}] = \delta_\pi(\Theta^{(j)}[\mathbf{r}]) \in I$ (again, here we used (4.26)).

□

Concerning complexity, similar considerations to those for the base algorithm apply, see Remark 9. In particular, step 4 of Algorithm 3 can be carried out via simple linear algebraic manipulations, leaving the cost of the Groebner basis computation at step 5 as the asymptotically dominant one. A slight optimization is to perform step 5 only when the condition $R_{i+1} = R_i$ is true.

Example 12. Let us consider the initial value problem given by:

$$\dot{x}_1 = x_1^2 \qquad \rho(x_1) = 1. \qquad (4.29)$$

Let $\pi = \times$ and Θ be the complete template of degree 3 involving x and x_1 :

$$\Theta = a_0 \cdot x^3 + a_1 \cdot x^2 \cdot x_1 + a_2 \cdot x^2 + a_3 \cdot x \cdot x_1^2 + a_4 \cdot x \cdot x_1 + a_5 \cdot x + a_6 \cdot x_1^3 + a_7 \cdot x_1^2 + a_8 \cdot x_1 + a_9.$$

We run Algorithm 3 with Θ .

- At iteration $i = 0$, $\Theta^{(0)} = \Theta$; so, for every $\mathbf{r} \in \mathbb{R}^9$, we have that $o_\rho(\Theta^{(0)}[\mathbf{r}]) = 0$ if and only if $\mathbf{r} \in R_0 := \{\mathbf{r} \in \mathbb{R}^9 : r_9 = -r_7 - r_8 - r_6\}$; I_0 is then built as the ideal generated by all the instances of the polynomial $\Theta^{(0)}[\mathbf{r}]$ for $\mathbf{r} \in R_0$, that is $I_0 = \langle \Theta^{(0)}[R_0] \rangle$. In practice, instead of considering the entire R_0 , it is sufficient to consider a basis of it.
- For $i = 1$, we have that $o_\rho(\Theta^{(1)}[\mathbf{r}]) = 0$ for all $\mathbf{r} \in R_0$; thus, $R_1 := R_0$. So, the vector space equality has been detected, but ideal equality does not hold: in fact, $\Theta^{(1)}[R_1] \not\subseteq I_0$ and, hence, $I_1 = \langle \Theta^{(0)}[R_1] \cup \Theta^{(1)}[R_1] \rangle \neq I_0$. The ideal chain is not yet stabilized and the algorithm goes on with a new iteration of the for loop.
- For $i = 2$, we obtain $R_2 := \{\mathbf{r} \in \mathbb{R}^9 : r_1 = r_7/3 - 2r_5/3 - r_3/3 - r_4/3 + r_6/3, r_9 = -r_7 - r_8 - r_6\}$, which does not coincide with R_1 . This shows that $R_1 = R_0$ does not imply that $R_1 = R_k$ for each $k > 1$: for this reason we use the chain of ideals to detect convergence of the sequence of vector spaces.

- The for loop goes on until the tenth iteration ($i = 10$), where $R_{10} = R_9 := \{\mathbf{r} \in \mathbb{R}^9 : r_0 = 0, r_1 = 0, r_2 = 0, r_4 = 0, r_5 = 0, r_6 = 0, r_7 = 0, r_8 = -r_3, r_9 = -r_8\}$. Hence, also in this case the vector space equality has been detected, but now $\Theta^{(10)}[R_{10}] \subseteq I_9$. Consequently, $I_{10} = \langle \bigcup_{j \leq 10} \Theta^{(j)}[R_i] \rangle = I_9$; thus, the algorithm terminates returning (R_{10}, I_{10}) .

We conclude that all valid polynomial identities that fit Θ are of the form $a_3 \cdot x_1^2 \cdot x - a_3 \cdot x_1 + a_3 = 0$, for any choice of the parameter $a_3 \in \mathbb{R}$. In particular, letting $a_3 = 1$, we deduce that any stream solution σ of the initial value problem (4.29) satisfies $\sigma^2 \cdot x - \sigma + 1 = 0$. This equation can be solved algebraically for σ in the ring of streams with the convolution product [155, 157], yielding:

$$\sigma = \frac{1 - \sqrt{1 - 4x}}{2x}.$$

This is the generating function of the Catalan numbers, the sequence of A000108 in [139]:

$$\sigma = (1, 1, 2, 5, 14, 42, 132, \dots).$$

Again, we refer the reader to [26] for a detailed discussion on the relations with generating functions.

In the final part of this section, we discuss the important special case of linear SDEs. Let $\mathcal{A} \subseteq \mathcal{P}$ be the subset of polynomials of degree ≤ 1 , in other words, affine expressions of the form $p = \sum_{i=0}^n r_i x_i + r_{n+1}$, with $r_i \in \mathbb{K}$. Assume in the given initial value problem (\mathcal{D}, ρ) all the expressions in \mathcal{D} are *linear*⁷, that is of the form $\sum_{i=0}^n r_i x_i$ with $r_i \in \mathbb{K}$: we call this a *linear initial value problem*. The restriction of the function δ_π to \mathcal{A} only depends on the polynomial G , not on F — cf. equalities (4.11)–(4.14). Moreover, since $G(1) \in \mathbb{K}$ and $\mathcal{D}(x_i) \in \mathcal{A}$ for each variable x_i , clearly for each $p \in \mathcal{A}$ we have $\delta_\pi(p) \in \mathcal{A}$. In other words, denoting by $|_{\mathcal{A}}$ function restriction to \mathcal{A} , we see that $C_{\pi, \mathcal{A}} := (\mathcal{A}, \delta_\pi|_{\mathcal{A}}, \rho|_{\mathcal{A}})$ forms a sub-coalgebra (Moore sub-automaton) of $(\mathcal{P}, \delta_\pi, \rho)$. Let $\mu_\pi|_{\mathcal{A}}$ be the final coalgebra morphism from $C_{\pi, \mathcal{A}}$ to Σ . The kernel $\ker(\mu_\pi|_{\mathcal{A}}) = \ker(\mu_\pi) \cap \mathcal{A}$ yields, informally speaking, all valid identifications among the variables x_i and their affine combinations, under the given linear initial value problem. It is easy to compute $\ker(\mu_\pi|_{\mathcal{A}})$ by resorting to Algorithm 3. In fact, we can specialize the algorithm so as to avoid the (costly) computation of the ideals I_i , as described in the next result. In what follows, we let

$$\Theta = \sum_{k=0}^n a_k x_k + a_{n+1} \tag{4.30}$$

with a_0, \dots, a_{n+1} distinct parameters, be the complete template of degree 1.

Theorem 7 (linear version of Algorithm 3). *Let (\mathcal{D}, ρ) be a linear initial value problem and π be a well-behaved product. Let Θ be the template defined in (4.30), and R_0, R_1, \dots be the sequence of vector spaces defined in steps 1 and 4 of Algorithm 3. If $i \geq 0$ is the first index such that $R_i = R_{i+1}$, then $\Theta[R_i] = \ker(\mu_\pi|_{\mathcal{A}})$.*

⁷Affine SDEs with constant terms in the drift expressions can be easily coded up as linear systems, by introducing extra variables and initial conditions.

PROOF. Let $L \in \mathbb{K}^{(n+1) \times (n+1)}$ be the matrix defining the given linear SDEs: for $i = 0, \dots, n$, the row i of L is (l_{i0}, \dots, l_{in}) , where $\dot{x}_i = l_{i0}x_0 + \dots + l_{in}x_n$ is the SDE for x_i (recall that $x_0 = x$). Now let $\mathbf{v}_0 := (\rho(\mathcal{X}), 1)^T \in \mathbb{K}^{n+2}$, seen as a column vector, and let $L' \in \mathbb{K}^{(n+2) \times (n+2)}$ be the matrix obtained from L by first adding an extra zero row, then an extra zero column, then setting $L'[n+1, n+1] := G(1)$ (here indices run from 0 to $n+1$). Also consider the vectors: $(\mathcal{X}, 1)$, obtained from the concatenation of \mathcal{X} and 1, and $\delta_\pi(\mathcal{X}) := (\delta_\pi(x_0), \dots, \delta_\pi(x_n))$. With this notation in place, for every $\mathbf{r} \in \mathbb{K}^{n+2}$ a column vector of parameters, we can write: $\Theta[\mathbf{r}] = \mathbf{r}^T \cdot (\mathcal{X}, 1)^T$ and $\delta_\pi(\Theta[\mathbf{r}]) = \mathbf{r}^T \cdot (\delta_\pi(\mathcal{X}), \delta_\pi(1))^T = \mathbf{r}^T \cdot L' \cdot (\mathcal{X}, 1)^T$; more generally, one can easily check that $\delta_\pi^{(j)}(\Theta[\mathbf{r}]) = L'^j \cdot (\mathcal{X}, 1)^T$; here we also exploit the fact that $\delta_\pi^{(j)}(1) = G(1)^j$ for all $j \geq 1$, a consequence of (4.15)–(4.17). Therefore $o_\rho(\Theta[\mathbf{r}]) = \mathbf{r}^T \cdot \mathbf{v}_0$ and more generally, $o_\rho(\delta_\pi^{(j)}(\Theta[\mathbf{r}])) = \mathbf{r}^T \cdot L'^j \cdot \mathbf{v}_0$ for each $j \geq 0$.

Now define the vectors $\mathbf{v}_j := L'^j \mathbf{v}_0$, for each $j \geq 1$. Then, as $o_\rho(\delta_\pi^{(j)}(\Theta[\mathbf{r}])) = \mathbf{r}^T \cdot \mathbf{v}_j$, by definition of R_j one has $R_j = \{\mathbf{v}_0, \dots, \mathbf{v}_j\}^\perp$, the orthogonal complement of the subspace spanned by $\{\mathbf{v}_0, \dots, \mathbf{v}_j\}$ in \mathbb{K}^{n+2} . Consider the least i s.t. $R_{i+1} = R_i$, which must exist as the R_i 's form a descending chain of finite dimensional vector spaces. Then $\mathbf{v}_{i+1} \in \text{span}\{\mathbf{v}_0, \dots, \mathbf{v}_i\}$, that is $\mathbf{v}_{i+1} = \sum_{j=0}^i \lambda_j \mathbf{v}_j$, for some $\lambda_j \in \mathbb{K}$. We now show that, for each $k \geq 1$, $\mathbf{v}_{i+k} \in \text{span}\{\mathbf{v}_0, \dots, \mathbf{v}_i\}$: this will imply that $R_i = R_{i+1} = R_{i+2} = \dots$, that is the chain of vector spaces has stabilized. For $k = 1$, this holds by definition of the index i . Assume $k > 1$. Then, for some $\lambda'_j \in \mathbb{K}$, we have: $\mathbf{v}_{i+k} = L' \mathbf{v}_{i+k-1} = L' \sum_{j=0}^i \lambda'_j \mathbf{v}_j = \sum_{j=0}^i \lambda'_j L' \mathbf{v}_j = \sum_{j=0}^i \lambda'_j \mathbf{v}_{j+1}$, where in the third equality we have exploited the induction hypothesis. Now the last expression is a linear combination of elements in $\text{span}\{\mathbf{v}_0, \dots, \mathbf{v}_i\}$; in particular, \mathbf{v}_{i+1} is in the span by assumption. This shows that $\mathbf{v}_{i+k} \in \text{span}\{\mathbf{v}_0, \dots, \mathbf{v}_i\}$.

Now, for each $\mathbf{r} \in R_i$, by definition of $R_i = R_{i+1} = \dots$, we have $o_\rho(\delta_\pi^{(j)}(\Theta[\mathbf{r}])) = 0$ for each $j \geq 0$, which implies $\mu_\pi|_{\mathcal{A}}(\Theta[\mathbf{r}]) = 0$. Conversely, if $\mu_\pi|_{\mathcal{A}}(\Theta[\mathbf{r}]) = 0$, then for each $j \geq 0$ we have: $o_\rho(\Theta^{(j)}[\mathbf{r}]) = (\mu_\pi|_{\mathcal{A}}(\Theta[\mathbf{r}]^{(j)}))(0) = \mu_\pi|_{\mathcal{A}}(\Theta[\mathbf{r}])(j) = 0$; then, by definition, $\mathbf{r} \in R_i$. This concludes the proof. \square

It is worthwhile to note that the transition functions on \mathcal{A} for $\pi \in \{\times, \otimes, \uparrow\}$ coincide: for these products, $C_{\mathcal{A}} := C_{\pi, \mathcal{A}}$ can be identified with the coalgebra of (expressions for) *stream weighted automata* [155, 157]. In particular, the final morphism $\mu_{\mathcal{A}} := \mu_\pi|_{\mathcal{A}}$ represents the standard semantics of weighted automata in terms of streams of [155, 157]; see also Example 13 below. On the other hand, for these products the algorithm outlined in Theorem 7 basically corresponds to the partition refinement algorithm for linear weighted automata described in [30, 21]. This algorithm has been subsequently generalized to the case where weights are drawn from a (semi)ring, under certain conditions: see, e.g., [19, 119] and the references therein.

We conclude the section with an example.

Example 13. For our purposes, a finite-state weighted automaton is a finite-state automaton where both states and transitions are labelled with *weights* drawn from a field \mathbb{K} . Weights on states are also called *output weights*. Figure 4.1 displays a weighted automaton with ten states $\{x_1, \dots, x_{10}\}$ with outputs in $\mathbb{K} = \mathbb{R}$; outputs are assumed to be 1 for x_{10} and 0 for any other state. The semantics of weighted automata can be given in terms of streams as described in e.g. [155]. Equivalently, one can associate to each state x_i a linear SDE and an initial condition, as dictated by, respectively, its outgoing transitions and its output weight. For our example,

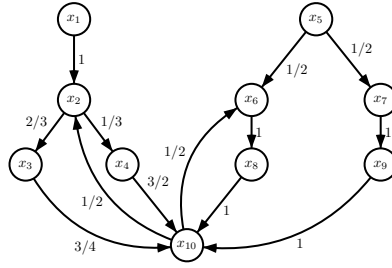


Figure 4.1: A weighted automaton. Output weights (not displayed) are assumed to be 1 for state x_{10} and 0 for any other state.

this leads to: $\mathcal{D} = \{\dot{x}_1 = x_2, \dot{x}_2 = \frac{2}{3}x_3 + \frac{1}{3}x_4, \dots, \dot{x}_{10} = \frac{1}{2}x_2 + \frac{1}{2}x_6\}$, and $\rho(x_1) = 0, \rho(x_2) = 0, \dots, \rho(x_{10}) = 1$. The semantics is then given by the final morphism $\mu_{\mathcal{A}}$.

We run the linear version of Algorithm 3 described in Theorem 7 on this example with $\pi \in \{\times, \otimes, \uparrow\}$; as remarked above, these products coincide on \mathcal{A} . The algorithm stops at iteration $i = 5$, returning a space R_5 such that $\Theta[R_5] = \ker(\mu_{\mathcal{A}})$. Concretely, $\Theta[R_5]$ can be described as the set of all linear expressions that are instances of the template Θ' below, for any choice of the involved free parameters a_1, \dots, a_7 in \mathbb{R} :

$$\Theta' = a_1(x_2 - x_6) + a_2(-4x_3/3 + x_8) + a_3(-4x_3/3 + x_9) + a_4(-2x_3 + x_4) + a_5(-x_1 + x_{10} - 1) + a_6(-x_1 + x_5) + a_7(-x_6 + x_7).$$

This implies for instance that: $\mu_{\mathcal{A}}(x_2) = \mu_{\mathcal{A}}(x_6)$, $(4/3)\mu_{\mathcal{A}}(x_3) = \mu_{\mathcal{A}}(x_8)$, and so on. For $\pi = \odot$, the algorithm stops at iteration $i = 4$ with a different result, expressed by the following template:

$$\Theta'' = a_1(x_2 - x_6) + a_2(-4x_3/3 + x_8) + a_3(x - x_1 - 4x_3/3 - x_6) + a_4(-4x_3/3 + x_9) + a_5(-2x_3 + x_4) + a_6(x_{10} + 4x_3/3 + x_6 - 1) + a_7(-x_1 + x_5) + a_8(-x_6 + x_7).$$

4.6 An implicit function theorem for the stream calculus

Let $\mathcal{E} \subseteq \mathcal{P}$ be a finite, nonempty set of polynomials that we call *polynomial system*. A *stream solution* of \mathcal{E} is a tuple of streams $\sigma = (\sigma_1, \dots, \sigma_n)$ such that $p(x, \sigma) = 0$ for each $p \in \mathcal{E}$. We want to show that, under certain syntactic conditions, any stream solution of \mathcal{E} can be uniquely defined via a polynomial SDE initial value problem (\mathcal{D}, ρ) . Instrumental to establish this result is a close stream analog of the well known Implicit Function Theorem (IFT) from calculus.

Let us introduce some extra notation on polynomials and streams. Beside the variables x , we shall consider $\mathbf{y} = (y_1, \dots, y_n)$, a set of new, distinct *initial value indeterminates* $\mathbf{y}_0 = (y_{01}, \dots, y_{0n})$ and *primed indeterminates* $\mathbf{y}' = (y'_1, \dots, y'_n)$. As usual, we let $y_0 = x$; moreover, by slightly abusing notation, we will let y_{00} denote 0 (the scalar zero). We will assume a fixed total order on all variables ($x = y_0 < y_1 < \dots < y_n$) and, for any monomial $m \neq 1$, on the variables in \mathbf{y} define $\min(m) := \min\{y : y \text{ occurs in } m\}$, where the min is taken according to the fixed total order on variables. In the definition below, we order the individual variables in a

monomial according to $<$ before proceeding to differentiation. It will turn out that the chosen total order is semantically immaterial, see Remark 10 further below. The *total degree* of a monomial m is just its size, that is the number of occurrences of variables in m . Recall that $\mathbb{K}[x, \mathbf{y}_0, \mathbf{y}, \mathbf{y}']$ denotes the set of polynomials having with coefficients in \mathbb{K} and indeterminates in $x, \mathbf{y}_0, \mathbf{y}, \mathbf{y}'$.

Definition 13 (syntactic stream derivative). *The syntactic stream derivative operator $(\cdot)'$: $\mathcal{P} \rightarrow \mathbb{K}[x, \mathbf{y}_0, \mathbf{y}, \mathbf{y}']$ is defined as follows. First, we define $(\cdot)'$ on monomials by induction on the total degree as follows:*

$$\begin{aligned} (1)' &:= 0 & (x)' &:= 1 & (y_i)' &:= y'_i & (1 \leq i \leq n) \\ (y_i \cdot m)' &:= y'_i \cdot m + y_{0i} \cdot (m)' & (0 \leq i \leq n, y_i = \min(y_i \cdot m) \text{ and } m \neq 1). \end{aligned}$$

The operator $(\cdot)'$ is then extended to polynomials in \mathcal{P} by linearity.

As an example, $(xy_1^2 + y_1y_2)' = y_1^2 + y'_1y_2 + y_{01}y'_2$. Note that p' lives in a polynomial ring $\mathbb{K}[x, \mathbf{y}_0, \mathbf{y}, \mathbf{y}']$ that includes \mathcal{P} . We shall write p' as $p'(x, \mathbf{y}_0, \mathbf{y}, \mathbf{y}')$ when wanting to make the indeterminates that may occur in p' explicit. With this notation, it is easy to check that $(\cdot)'$ commutes with substitution, as stated in the following lemma.

Lemma 7. *For every $p(x, \mathbf{y})$ and σ , we have that $(p(X, \sigma))' = p'(X, \sigma(0), \sigma, \sigma')$.*

PROOF. For p a monomial, the proof is by induction on its total degree, and straightforwardly follows from Definition 13. The general case when p is a linear combination of monomials follows then by linearity of the definition of $(\cdot)'$. \square

Remark 10. While the definition of syntactic stream derivative *does* depend on the chosen total order of indeterminates (y_0, \dots, y_n) , Lemma 7 confirms that this order becomes immaterial when the indeterminates are substituted with streams. In particular, if $(\cdot)'$ and $(\cdot)^\dagger$ are two syntactic stream derivative operators, corresponding to two different total orders, Lemma 7 implies that $p'(X, \sigma(0), \sigma, \sigma') = p^\dagger(X, \sigma(0), \sigma, \sigma') = (p(X, \sigma))'$, where the last occurrence of $(\cdot)'$ denotes stream derivative.

Ultimately, this coincidence stems from the fact that the asymmetry in the definition of stream derivative of the convolution product, $(\sigma \times \tau)' = \sigma' \times \tau + \sigma(0) \cdot \tau'$, is only apparent. Indeed, taking into account the equality $\sigma(0) = \sigma - X \times \sigma'$, one can obtain the symmetric rule $(\sigma \times \tau)' = \sigma' \times \tau + \sigma \times \tau' - X \times \sigma' \times \tau'$. Note, however, that the equality $\sigma(0) = \sigma - X \times \sigma'$ cannot be expressed at the syntactic (polynomial) level.

The next lemma is about rational SDEs, and how to convert them into polynomial SDEs. This result has already appeared in the literature in various forms, see e.g. [16, 126]. Here we keep its formulation as elementary as possible, and tailor it to our purposes. Its proof is a routine application of equation (4.8).

Lemma 8 (from rational to polynomial SDEs). *Let $f_i(x, \mathbf{y}_0, \mathbf{y})$ for $i = 1, \dots, n$ and $g(x, \mathbf{y}_0, \mathbf{y})$ be polynomials, and $\mathbf{r}_0 \in \mathbb{K}^n$ be such that $g(0, \mathbf{r}_0, \mathbf{r}_0) \neq 0$. Let $\sigma = (\sigma_1, \dots, \sigma_n)$ be any tuple of streams satisfying the following system of (rational) SDEs and initial conditions:*

$$\sigma'_i = f_i(X, \mathbf{r}_0, \sigma) \cdot g(X, \mathbf{r}_0, \sigma)^{-1} \quad \sigma_i(0) = r_{0i} \quad (i = 1, \dots, n). \quad (4.31)$$

Then, for a new variable w , there is a polynomial $h(x, \mathbf{y}_0, \mathbf{y}, w)$, not depending on σ , such that (σ, τ) , with $\tau := g(x, \mathbf{r}_0, \sigma)^{-1}$, is the unique solution of the following initial value problem of $n + 1$ polynomial SDEs and initial conditions:

$$\sigma'_i = f_i(X, \mathbf{r}_0, \sigma) \cdot \tau \quad \sigma_i(0) = r_{0i} \quad (i = 1, \dots, n) \quad (4.32)$$

$$\tau' = -g(0, \mathbf{r}_0, \mathbf{r}_0)^{-1} \cdot h(X, \mathbf{r}_0, \sigma, \tau) \cdot \tau \quad \tau(0) = g(0, \mathbf{r}_0, \mathbf{r}_0)^{-1}. \quad (4.33)$$

In particular, the polynomial $h(x, \mathbf{y}_0, \mathbf{y}, w)$ is obtained from $g' = g'(x, \mathbf{y}_0, \mathbf{y}, \mathbf{y}')$ by replacing each variable y'_i with the polynomial $f_i(x, \mathbf{y}_0, \mathbf{y}) \cdot w$, for $i = 1, \dots, n$. Conversely, for any (σ, τ) satisfying (4.32) and (4.33), we have that σ also satisfies (4.31).

An important technical ingredient in the proof of the IFT for streams is an operator of *stream partial derivative* $\frac{\partial}{\partial y_i}$ on polynomials: this will allow us to formulate a stream analog of the chain rule from calculus⁸. For our purposes, a chain rule for polynomials suffices; for a more general scenario, see [157, Eq.25], where composition of streams is introduced (and can be used for covering the case of arbitrary functions). The following result is instrumental to formally introduce stream partial derivatives and the chain rule for streams.

Lemma 9. *For every $p \in \mathcal{P}$, any $y'_i \in \mathbf{y}'$ can only occur linearly in p' , i.e., there is a unique $(n + 1)$ -tuple (q_0, q_1, \dots, q_n) of polynomials in $\mathbb{K}[x, \mathbf{y}_0, \mathbf{y}]$ such that $p' = q_0 + \sum_{i=1}^n q_i \cdot y'_i$.*

PROOF. Let us first consider existence. We first consider the case in which p is a monomial and we proceed by induction on its total degree. The base cases follows by the first three cases of Def. 13: for $p = 1$, set all q_i 's to 0; for $p = x$, set $q_0 = 1$ and all other q_i 's to 0; for $p = y_i$, set $q_i = 1$ and all other q_j 's to 0. For the inductive case, consider $p = y_i \cdot m$ with $m \neq 1$ and $y_i = \min(y_i \cdot m)$. By induction hypothesis, there is a unique $(n + 1)$ -tuple $(\hat{q}_0, \hat{q}_1, \dots, \hat{q}_n)$ of polynomials such that $m' = \hat{q}_0 + \sum_{j=1}^n \hat{q}_j \cdot y'_j$. By the fourth case of Def. 13, $p' := y'_i \cdot m + y_{0i} \cdot m'$; then, it suffices to set $q_0 = y_{0i} \cdot \hat{q}_0$, $q_i = y_{0i} \cdot \hat{q}_i + m$, and all other q_j 's to $y_{0i} \cdot \hat{q}_j$. The case when p is a linear combination of monomials follows by linearity.

As to uniqueness, suppose there are two tuples (q_0, q_1, \dots, q_n) and $(\tilde{q}_0, \tilde{q}_1, \dots, \tilde{q}_n)$ of polynomials in $\mathbb{K}[x, \mathbf{y}_0, \mathbf{y}]$ such that $p' = q_0 + \sum_{i=1}^n q_i \cdot y'_i = \tilde{q}_0 + \sum_{i=1}^n \tilde{q}_i \cdot y'_i$. This implies $(q_0 - \tilde{q}_0) + \sum_{i=1}^n (q_i - \tilde{q}_i) \cdot y'_i = 0$. For each $j = 1, \dots, n$, the indeterminate y'_j in the last sum does not occur in any of the terms $(q_i - \tilde{q}_i)$ ($0 \leq i \leq n$), which implies that $(q_j - \tilde{q}_j) = 0$, hence $q_j = \tilde{q}_j$. This in turn implies $q_0 - \tilde{q}_0 = 0$, hence $q_0 = \tilde{q}_0$ as well. \square

For reasons that will be apparent in a while, we introduce the following suggestive notation for the polynomials q_i uniquely determined by p according to Lemma 9:

$$\frac{\partial p}{\partial x} := q_0 \quad \frac{\partial p}{\partial y_i} := q_i \quad (i = 1, \dots, n) \quad \nabla_{\mathbf{y}} p := \left(\frac{\partial p}{\partial y_1}, \dots, \frac{\partial p}{\partial y_n} \right).$$

⁸The chain rule from calculus is: $\frac{d}{dx} f(y_1(x), \dots, y_n(x)) = \sum_{i=1}^n \frac{\partial}{\partial y_i} f(y_1(x), \dots, y_n(x)) \cdot \frac{d}{dx} y_i(x) = \nabla_{\mathbf{y}} f(y_1(x), \dots, y_n(x)) \cdot \left(\frac{d}{dx} y_1(x), \dots, \frac{d}{dx} y_n(x) \right)^T$.

With this notation, the equality for p' in the lemma can be written in the form of a chain rule:

$$p' := \frac{\partial p}{\partial x} + (\nabla_{\mathbf{y}} p) \cdot \mathbf{y}'^T. \quad (4.34)$$

Also, it is easy to check that $\frac{\partial p}{\partial x} \in \mathcal{P}$, so that one may write $\frac{\partial p}{\partial x}(x, \mathbf{y})$ if wanting to emphasize the dependence on indeterminates. Practically, $\frac{\partial p}{\partial y_i}$ ($1 \leq i \leq n$) can be easily computed from p' by taking its quotient with respect to y'_i : this means expressing the syntactic stream derivative as $p' = (\dots) + q \cdot y'_i$, with y'_i not occurring in (\dots) , then letting $\frac{\partial p}{\partial y_i} = q$. Likewise, $\frac{\partial p}{\partial x}$ can be computed by removing from p' all terms divisible by some y'_i . A few examples are discussed below.

Example 14 (partial stream derivatives). Let α be a monomial. For x not occurring in α , $j \geq 1$ and $y_i \neq x$, we have:

- $\frac{\partial}{\partial x} x^j \alpha = x^{j-1} \alpha$, $\frac{\partial}{\partial x} \alpha = 0$ and $\frac{\partial}{\partial y_i} x^j \alpha = 0$;
- for y_i not occurring in α , $\frac{\partial}{\partial y_i} \alpha = 0$.

When y_i occurs in α , the position of y_i in the total order of variables plays a role in the result (only at a syntactic level, cf. Remark 10). As an example, $\frac{\partial}{\partial y_2} 2y_1^2 y_2^2 y_3 = 2y_1^2 y_3 (y_2 + y_{02})$. The partial stream derivative operator is linear. As an example, for $p = x^2 y_1 y_2^3 + 2y_1 y_2^2 + 2x + 1$, we have: $\frac{\partial p}{\partial x} = x y_1 y_2^3 + 2$, $\frac{\partial p}{\partial y_1} = 2y_2^2$ and $\frac{\partial p}{\partial y_2} = 2y_1 (y_2 + y_{02})$. These are all instances of a general rule expressed by equation (4.42), which is established in the proof of Lemma 12.

The following lemma translates the syntactic formula (4.34) in terms of streams. Its proof is an immediate consequence of (4.34) and of Lemma 7.

Lemma 10 (chain rule for stream derivative). For any σ and $\mathbf{r}_0 = \sigma(0)$, we have:

$$(p(X, \sigma))' = \frac{\partial p}{\partial x}(X, \mathbf{r}_0, \sigma) + (\nabla_{\mathbf{y}} p)(X, \mathbf{r}_0, \sigma) \cdot \sigma'^T.$$

Now we assume $|\mathcal{E}| = n$, say $\mathcal{E} = \{p_1, \dots, p_n\}$. Fixing some order on its elements, we will sometimes regard \mathcal{E} as a *vector* of polynomials, and use the notation $\mathcal{E}(x, \mathbf{y})$ accordingly. In particular, we let $\nabla_{\mathbf{y}} \mathcal{E}$ denote the $n \times n$ matrix of polynomials whose rows are $\nabla_{\mathbf{y}} p_i$, for $i = 1, \dots, n$. Evidently, this is the stream analog of the *Jacobian* of \mathcal{E} . Moreover, we let $\frac{\partial \mathcal{E}}{\partial x} := \left(\frac{\partial p_1}{\partial x}, \dots, \frac{\partial p_n}{\partial x} \right)$. The following lemma is an immediate consequence of the fact that $\mathcal{E}(X, \sigma)' = 0$ and of previous lemma, considering \mathcal{E} componentwise.

Lemma 11. Let $\sigma = (\sigma_1, \dots, \sigma_n)$ be a solution of \mathcal{E} and $\mathbf{r}_0 = \sigma(0)$. Then

$$(\nabla_{\mathbf{y}} \mathcal{E})(X, \mathbf{r}_0, \sigma) \cdot \sigma'^T + \left(\frac{\partial \mathcal{E}}{\partial x}(X, \sigma) \right)^T = 0. \quad (4.35)$$

Example 15. Consider the polynomial system made up of a single equation, $\mathcal{E} = \{p\}$, where $p(x, \mathbf{y}) := y - (1 + xy^2)$ with $\mathbf{y} = y = y_1$ (see also [157, pg. 117]). We compute

$$\begin{aligned} (\nabla_{\mathbf{y}} \mathcal{E})(x, \mathbf{r}_0, \mathbf{y}) &= \nabla_y p(x, r_0, y) = \frac{\partial p}{\partial y}(x, r_0, y) = 1 \\ \frac{\partial \mathcal{E}}{\partial x}(x, \mathbf{y}) &= \frac{\partial p}{\partial x}(x, y) = -y^2. \end{aligned}$$

Hence, for any stream solution σ of \mathcal{E} , applying Lemma 11 we get:

$$\sigma' - \sigma^2 = 0.$$

Consider now $\mathcal{E} = \{q\}$ with $q(x, y) := x^2 + y^2 - 1$, where $y = y_1$ and $y_0 = y_{01}$. We compute

$$\begin{aligned} (\nabla_y \mathcal{E})(x, r_0, y) &= y + y_0 \\ \frac{\partial q}{\partial x}(x, y) &= x. \end{aligned}$$

Hence, for any stream solution σ of \mathcal{E} , applying Lemma 11 we get:

$$(\sigma + \sigma(0)) \cdot \sigma' + X = 0.$$

Let us recall a few facts from the theory of matrices and determinants in a commutative ring, applied to the ring Σ . By definition, a matrix of streams $A \in \Sigma^{n \times n}$ is invertible iff there exists a matrix of streams $B \in \Sigma^{n \times n}$ such that $A \times B = B \times A = I$ (the identity matrix of streams); this B , if it exists, is unique and denoted by A^{-1} . It is easy to show that $A \in \Sigma^{n \times n}$ is invertible if and only if $A(0) \in \mathbb{K}^{n \times n}$ is invertible⁹. By general results on determinants, $\det(A \times B) = \det(A) \cdot \det(B)$ (Binet's theorem). For streams, this implies that, if A is invertible, then $\det(A)$ as a stream is invertible, that is $\det(A)(0) \neq 0$. Moreover, again by virtue of these general results, the formula for the element of row i and column j of A^{-1} is given by:

$$A^{-1}(i, j) = (-1)^{i+j} \det(A)^{-1} \cdot \det(A_{ji}) \quad (4.36)$$

where A_{ji} denotes the $(n-1) \times (n-1)$ *adjunct* matrix obtained from A by deleting its j -th row and i -th column. Also note that, for a $n \times n$ matrix of polynomials, say $P = P(x, \mathbf{y}_0, \mathbf{y})$, $\det(P)$ is a polynomial in $x, \mathbf{y}_0, \mathbf{y}$, and $\det(P(X, \mathbf{r}_0, \boldsymbol{\sigma})) = (\det(P))(X, \mathbf{r}_0, \boldsymbol{\sigma})$.

Theorem 8 (IFT for streams). *Let $\mathbf{r}_0 \in \mathbb{K}^n$ be such that $\mathcal{E}(0, \mathbf{r}_0) = 0$ and $(\nabla_{\mathbf{y}} \mathcal{E})(0, \mathbf{r}_0, \mathbf{r}_0)$ is invertible as a matrix in $\mathbb{K}^{n \times n}$. Then there is a unique stream solution $\boldsymbol{\sigma}$ of \mathcal{E} such that $\boldsymbol{\sigma}(0) = \mathbf{r}_0$. Moreover, $(\nabla_{\mathbf{y}} \mathcal{E})(X, \mathbf{r}_0, \boldsymbol{\sigma})$ is invertible as a matrix in $\Sigma^{n \times n}$ and $\boldsymbol{\sigma}$ satisfies the following system of n rational SDEs and initial conditions:*

$$\boldsymbol{\sigma}'^T = -(\nabla_{\mathbf{y}} \mathcal{E})(X, \mathbf{r}_0, \boldsymbol{\sigma})^{-1} \cdot \left(\frac{\partial \mathcal{E}}{\partial x}(X, \boldsymbol{\sigma}) \right)^T \quad \boldsymbol{\sigma}(0) = \mathbf{r}_0. \quad (4.37)$$

⁹Note this is true only because we insist that the inverse matrix must also lie in $\Sigma^{n \times n}$. Working in the field of formal *Laurent* series, which strictly includes Σ , this would be false: e.g. $X(0) = 0$, but X has X^{-1} as an inverse.

Moreover, from (4.37) it is possible to build a system of $n + 1$ polynomial SDEs in $n + 1$ variables and corresponding initial conditions, whose unique solution is (σ, τ) , for a suitable τ .

PROOF. We will first show that the initial value problem given in (4.37) is satisfied by every, if any, stream solution σ of \mathcal{E} such that $\sigma(0) = \mathbf{r}_0$. Indeed, consider any such σ . As $(\nabla_{\mathbf{y}} \mathcal{E})(X, \mathbf{r}_0, \sigma)(0) = (\nabla_{\mathbf{y}} \mathcal{E})(0, \mathbf{r}_0, \mathbf{r}_0)$, which is invertible by hypothesis, the above considerations on matrix invertibility imply that there exists $(\nabla_{\mathbf{y}} \mathcal{E})(X, \mathbf{r}_0, \sigma)^{-1}$ in $\Sigma^{n \times n}$. Multiplying equality (4.35) from Lemma 11 to the left by $(\nabla_{\mathbf{y}} \mathcal{E})(X, \mathbf{r}_0, \sigma)^{-1}$, we obtain that σ satisfies (4.37). Now define the following (matrix of) polynomials:

- $g(x, \mathbf{y}_0, \mathbf{y}) := \det(\nabla_{\mathbf{y}} \mathcal{E})$
- $\tilde{A} := [\tilde{a}_{ij}]$ with $\tilde{a}_{ij} := (-1)^{i+j} \det((\nabla_{\mathbf{y}} \mathcal{E})_{ji})$
- $f_i(x, \mathbf{y}_0, \mathbf{y}) := -\tilde{A}_i \cdot \left(\frac{\partial \mathcal{E}}{\partial x}\right)^T$, where \tilde{A}_i denotes the i -th row of \tilde{A} .

Applying our previous observation on the determinant of a matrix of polynomials, we have that $\det((\nabla_{\mathbf{y}} \mathcal{E})(X, \mathbf{r}_0, \sigma)) = g(X, \mathbf{r}_0, \sigma)$, and similarly $(-1)^{i+j} \det((\nabla_{\mathbf{y}} \mathcal{E})(X, \mathbf{r}_0, \sigma)_{ji}) = \tilde{a}_{ij}(X, \mathbf{r}_0, \sigma)$. Therefore, by the formula for the inverse matrix (4.36), equation (4.37) can be written componentwise as follows

$$\sigma'_i = f_i(X, \mathbf{r}_0, \sigma) \cdot g(X, \mathbf{r}_0, \sigma)^{-1} \quad \sigma_i(0) = r_{i0} \quad (i = 1, \dots, n). \quad (4.38)$$

This is precisely the rational form in (4.31). Then Lemma 8 implies that there is a set \mathcal{D} of $n + 1$ polynomial SDEs in the indeterminates \mathbf{y}, w , and corresponding initial conditions $\rho := (\mathbf{r}_0, g(0, \mathbf{r}_0, \mathbf{r}_0)^{-1})$, satisfied when letting $\mathbf{y}, w = \sigma, \tau$, where $\tau = g(X, \mathbf{r}_0, \sigma)^{-1}$:

$$y'_i = f_i(x, \mathbf{r}_0, \mathbf{y}) \cdot w \quad y_i(0) = r_{i0} \quad (i = 1, \dots, n) \quad (4.39)$$

$$w' = -g(0, \mathbf{r}_0, \mathbf{r}_0)^{-1} \cdot h(x, \mathbf{r}_0, \mathbf{y}, w) \cdot w \quad w(0) = g(0, \mathbf{r}_0, \mathbf{r}_0)^{-1} \quad (4.40)$$

with h obtained from g as described in Lemma 8. Note the SDEs \mathcal{D} we have arrived at are purely syntactic and do not depend on the existence of any specific σ . Now, by Theorem 4 there is a (unique) solution, say (σ, τ) , of the polynomial SDE initial value problem (\mathcal{D}, ρ) defined by (4.39)-(4.40).

We now show that σ is a stream solution of \mathcal{E} . By the last part of Lemma 8, σ satisfies (4.38), which, as discussed above, is just another way of writing (4.37). Now we have

$$\mathcal{E}(0, \sigma)(0) = \mathcal{E}(0, \mathbf{r}_0) = 0$$

$$\begin{aligned} \mathcal{E}(0, \sigma)' &= (\nabla_{\mathbf{y}} \mathcal{E})(X, \mathbf{r}_0, \sigma) \cdot \sigma'^T + \left(\frac{\partial \mathcal{E}}{\partial x}(X, \sigma)\right)^T \\ &= -(\nabla_{\mathbf{y}} \mathcal{E})(X, \mathbf{r}_0, \sigma) \cdot (\nabla_{\mathbf{y}} \mathcal{E})(X, \mathbf{r}_0, \sigma)^{-1} \cdot \left(\frac{\partial \mathcal{E}}{\partial x}(X, \sigma)\right)^T + \left(\frac{\partial \mathcal{E}}{\partial x}(X, \sigma)\right)^T \\ &= -\left(\frac{\partial \mathcal{E}}{\partial x}(X, \sigma)\right)^T + \left(\frac{\partial \mathcal{E}}{\partial x}(X, \sigma)\right)^T \\ &= 0 \end{aligned}$$

where the second equality is just the chain rule on streams (Lemma 10), and the third equality follows from (4.37). As $\mathcal{E}(0, \sigma)(0) = 0$ and $\mathcal{E}(0, \sigma)' = 0$, by e.g. the fundamental theorem of the stream calculus (4.2) it follows that $\mathcal{E}(0, \sigma) = 0$. This completes the *existence* part of the statement.

As to *uniqueness*, consider any tuple of streams $\zeta \in \Sigma^n$ that is a stream solution of \mathcal{E} and such that $\zeta(0) = \mathbf{r}_0$. As shown above, (ζ, ξ) , with $\xi = g(X, \mathbf{r}_0, \zeta)^{-1}$, satisfies the polynomial SDE initial value problem (\mathcal{D}, ρ) defined by (4.39)-(4.40). By uniqueness of the solution (Theorem 4), $(\zeta, \xi) = (\sigma, \tau)$. \square

The above theorem guarantees existence and uniqueness of a solution of \mathcal{E} , provided that there exists a unique tuple of “initial conditions” $\mathbf{r}_0 \in \mathbb{K}^n$ for which \mathcal{E} satisfies the hypotheses of Theorem 8. The existence and uniqueness of such a \mathbf{r}_0 must be ascertained by other means. In particular, it is possible that the algebraic conditions $\mathcal{E}(0, \mathbf{r}_0) = 0$ and $\det((\nabla_{\mathbf{y}} \mathcal{E})(x, \mathbf{r}_0, \mathbf{r}_0)) \neq 0$ are already sufficient to uniquely determine \mathbf{r}_0 . There are powerful tools from algebraic geometry that can be applied to this purpose, such as elimination theory: we refer the interested reader to [57] for an introduction. For now we shall content ourselves with a couple of elementary examples. An extended example is presented in Section 4.8.

Example 16. Let us consider again $\mathcal{E} = \{p\}$ with $p(x, y) := y - (1 + xy^2)$, described in Example 15. Note that $p(0, r_0) = 0$ uniquely identifies the initial condition $r_0 = \sigma(0) = 1$. Also note that $\nabla_y p = 1$ is invertible at $y = r_0 = 1$: hence Theorem 8 applies. The system (4.37) followed by the transformation of Lemma 8 becomes the following polynomial system of SDEs and initial conditions:

$$\begin{aligned} y' &= y^2 w & y(0) &= 1 \\ w' &= 0 & w(0) &= 1. \end{aligned}$$

Note that the SDEs and initial condition for w define the constant stream $1 = (1, 0, 0, \dots)$, hence the above system can be simplified to the single SDE and initial condition: $y' = y^2$ and $y(0) = 1$. The unique stream solution of this initial value problem is $\sigma = (1, 1, 2, 5, 14, 42, \dots)$, the stream of Catalan numbers. Hence σ is the only stream solution of \mathcal{E} .

More generally, any set of *guarded* polynomial equations [16] of the form $\mathcal{E} = \{y_i - (c_i + xp_i) : i = 1, \dots, n\}$ satisfies the hypotheses of Theorem 8 precisely when $\mathbf{r}_0 = (c_1, \dots, c_n)$. Indeed, $\mathcal{E}(0, \mathbf{r}_0) = 0$, while $\nabla_{\mathbf{y}} \mathcal{E} = I$, the $n \times n$ identity matrix, which is clearly invertible. The SDEs and initial conditions (\mathcal{D}, ρ) determined by the theorem are given by $y_i' = p_i$ and $y_i(0) = c_i$ for $i = 1, \dots, n$, plus the trivial $w' = 0$ and $w(0) = 1$, that can be omitted.

For a non guarded example, consider $\mathcal{E} = \{q\}$ where $q := x^2 + y^2 - 1$, again from Example 15. $q(0, r_0) = 0$ gives two possible values, $r_0 = \pm 1$. Let us fix $r_0 = 1$. We have $\nabla_y p = y + y_0$, which is $\neq 0$ when evaluated at $y = y_0 = r_0$. Applying Theorem 8 and Lemma 8 yields the following SDEs and initial conditions:

$$\begin{aligned} y' &= -xw & y(0) &= 1 \\ w' &= \frac{xw^2}{2} & w(0) &= \frac{1}{2}. \end{aligned}$$

The SDE for w arises considering equation (4.8) for the multiplicative inverse of a stream, in detail, letting $w = \frac{1}{y+1}$, we get: $w' = -w(0) \cdot (y+1)' \cdot w = -\frac{1}{2} \cdot (-xw) \cdot w = \frac{xw^2}{2}$.

The unique solution of the derived initial value problem is the stream $\sigma = (1, 0, -1/2, 0, -1/8, 0, -1/16, \dots)$; these are the Taylor coefficients of the function $\sqrt{1-x^2}$ around $x = 0$. This stream is therefore the unique solution of \mathcal{E} with $r_0 = 1$. If we fix $r_0 = -1$, we obtain $-\sigma$ as the unique solution, as expected.

Remark 11 (relation with algebraic series). Recall from [71, 173] that a stream σ is *algebraic* if there exists a nonzero polynomial $p(x, y)$ in the variables x, y such that $p(X, \sigma) = 0$. For $|\mathcal{E}| > 1$, algebraicity of the solution is not in general guaranteed. [71, Th.8.7] shows that a sufficient condition for algebraicity in this case is that \mathcal{E} be *zero-dimensional*, i.e. that \mathcal{E} has finitely many solutions when considered as a set of polynomials with coefficients in $\mathcal{C}(x)$, the fraction field of univariate polynomials in x with coefficients in \mathcal{C} . In this case, in fact, for each variable y_i one can apply results from elimination theory to get a single nonzero polynomial $p(x, y_i)$ satisfied by σ_i . See also the discussion in Section 4.8.

On the other hand, we do not require zero-dimensionality of \mathcal{E} in Theorem 8. Moreover, for the case of polynomials with rational coefficients, [26, Cor.5.3] observes that the unique solution of a polynomial SDE initial value problem like (4.3) is a tuple of algebraic streams. Then, an immediate corollary of Theorem 8 is that, under the conditions stated for \mathcal{E} and \mathbf{r}_0 , the unique stream solution of \mathcal{E} is algebraic, even for positive-dimensional systems — at least in the case of polynomials with rational coefficients. As an example, consider the following system of three polynomials in the variables x and $\mathbf{y} = (y_1, y_2, y_3)$:

$$\begin{aligned} \mathcal{E} = \{ & y_1 y_3^4 + x^2 - y_2^2 + y_2, \quad -y_1^2 y_2 + x y_3 + y_1, \\ & -y_1^3 x y_3^5 + y_1^4 y_3^4 - y_1^2 x^3 y_3 + x^2 y_1^3 + x^2 y_2 y_3^2 - x^2 y_3^2 + y_1^2 - x y_3 - y_1 \}. \end{aligned} \quad (4.41)$$

Considered as a system of polynomials with coefficients in $\mathcal{C}(x)$, \mathcal{E} is not zero-dimensional — in fact, its dimension is 1^{10} . It is readily checked, though, that for $\mathbf{r}_0 = (1, 1, 1)$ we have $\mathcal{E}(0, \mathbf{r}_0) = 0$ and $\det((\nabla_{\mathbf{y}} \mathcal{E})(0, \mathbf{r}_0, \mathbf{r}_0)) = 12 \neq 0$. From Theorem 8, we conclude that the unique stream solution σ of \mathcal{E} satisfying $\sigma(0) = \mathbf{r}_0$ is algebraic.

4.7 Relations with the classical IFT

We now discuss a relation of our IFT with the classical IFT from calculus. We start with the following lemma. In the rest of the section, we fix $\mathbb{K} = \mathbb{R}$.

Lemma 12. *Let $p(x, \mathbf{y})$ be a polynomial, $\mathbf{r}_0 \in \mathbb{R}^n$, and y_i in \mathbf{y} . Consider the ordinary $\frac{\partial p}{\partial y_i}(x, \mathbf{y})$ and stream $\frac{\partial p}{\partial y_i}(x, \mathbf{y}_0, \mathbf{y})$ partial derivatives. Then $\frac{\partial p}{\partial y_i}(0, \mathbf{r}_0) = \frac{\partial p}{\partial y_i}(0, \mathbf{r}_0, \mathbf{r}_0)$.*

PROOF. Let $p = x \cdot p_0 + q$ where x does not occur in q . Write q as a sum of k monomials, $q = \sum_{j=1}^k \alpha_j y_i^{k_j}$, where both x and y_i do not occur in any of the

¹⁰ As checked with Maple's `lsZeroDimensional` function of the Groebner package.

monomials α_j . Moreover, let us write each α_j as $\alpha_j = \beta_j \cdot \gamma_j$, where β_j (resp. γ_j) contains all the y 's with index smaller (resp., greater) than i .

For the ordinary partial derivative, we have that

$$\frac{\partial p}{\partial y_i}(x, \mathbf{y}) = x \cdot \frac{\partial p_0}{\partial y_i} + \sum_{j=1}^k k_j \alpha_j y_i^{k_j-1}.$$

For the stream partial derivative, let us denote with $\mathbf{c}_{y_i}^h$ the quantity $\sum_{j=0}^h y_{0i}^j y_i^{h-j}$, with $\mathbf{c}_{y_i}^0 := 1$ and $\mathbf{c}_{y_i}^{-1} := 0$. Taking into account the rules for $\tilde{\partial}$ and writing $m(\mathbf{u})$ for the evaluation of a monomial m at $\mathbf{y} = \mathbf{u}$, we have

$$\frac{\tilde{\partial} p}{\tilde{\partial} y_i}(x, \mathbf{y}_0, \mathbf{y}) = \sum_{j=1}^k \beta_j(\mathbf{y}_0) (\mathbf{c}_{y_i}^{k_j-1}) \gamma_j(\mathbf{y}_0). \quad (4.42)$$

By denoting with $\mathbf{c}_{y_i}^h(r_1, r_2)$ the term $\mathbf{c}_{y_i}^h$ with $r_1 (\in \mathbb{R})$ in place of y_{0i} and $r_2 (\in \mathbb{R})$ in place of y_i , we have that $\mathbf{c}_{y_i}^h(r, r) = (h+1)r^h$, for any $r \in \mathbb{R}$. Upon evaluation of the above polynomials at $x = 0$, $\mathbf{y}_0 = \mathbf{r}_0$, $\mathbf{y} = \mathbf{r}_0$, we get

$$\begin{aligned} \frac{\partial p}{\partial y_i}(0, \mathbf{r}_0) &= \sum_{j=1}^k k_j \alpha_j(\mathbf{r}_0) r_{0i}^{k_j-1} \\ \frac{\tilde{\partial} p}{\tilde{\partial} y_i}(0, \mathbf{r}_0, \mathbf{r}_0) &= \sum_{j=1}^k \beta_j(\mathbf{r}_0) (\mathbf{c}_{y_i}^{k_j-1}(r_{0i}, r_{0i})) \gamma_j(\mathbf{r}_0) \\ &= \sum_{j=1}^k \beta_j(\mathbf{r}_0) (k_j r_{0i}^{k_j-1}) \gamma_j(\mathbf{r}_0) = \sum_{j=1}^k k_j \alpha_j(\mathbf{r}_0) r_{0i}^{k_j-1}. \end{aligned}$$

□

The above lemma implies that the classical and stream jacobian matrices evaluated at $x = 0$, $\mathbf{y} = \mathbf{r}_0$ are the same: $(\nabla_{\mathbf{y}} \mathcal{E})(0, \mathbf{r}_0) = (\nabla_{\mathbf{y}} \mathcal{E})(0, \mathbf{r}_0, \mathbf{r}_0)$. In particular, the first is invertible if and only if the latter is. Therefore, the classical and stream IFT can be applied exactly under the same hypotheses on \mathcal{E} and \mathbf{r}_0 . What is the relationship between the solutions provided by the two theorems? The next theorem precisely characterizes this relationship. In its statement and proof, we make use of the following concept. Consider the set \mathcal{A} of functions $\mathbb{R} \rightarrow \mathbb{R}$ that are real analytic around the origin, i.e., those functions that admit a Taylor expansion with a positive radius of convergence around $x = 0$. It is well-known that \mathcal{A} forms a \mathbb{R} -algebra. Now consider the function \mathcal{T} that sends each $f \in \mathcal{A}$ to the stream $\mathcal{T}[f]$ of its Taylor coefficients around 0, that is $\mathcal{T}[f](j) = f^{(j)}(0)/j!$ for each $j \geq 0$.¹¹ It is easy to check that \mathcal{T} acts as a \mathbb{R} -algebra homomorphism from \mathcal{A} to Σ ; in particular, by denoting with \cdot the (pointwise) product of functions, we have that $\mathcal{T}[f \cdot g] = \mathcal{T}[f] \times \mathcal{T}[g]$.

Theorem 9 (stream IFT, classical version). *Let $\mathbf{r}_0 \in \mathbb{R}^n$ be such that $\mathcal{E}(0, \mathbf{r}_0) = 0$ and $(\nabla_{\mathbf{y}} \mathcal{E})(0, \mathbf{r}_0)$ is invertible as a matrix in $\mathbb{R}^{n \times n}$. Then there is a unique stream solution σ of \mathcal{E} s.t. $\sigma(0) = \mathbf{r}_0$. In particular, $\sigma = \mathcal{T}[f]$, for $f : \mathbb{R} \rightarrow \mathbb{R}^n$ a real analytic*

¹¹Let Γ be the partial function that sends each stream σ to its ordinary generating function $\Gamma[\sigma](x) = \sum_{j \geq 0} \sigma(j) x^j$, provided the latter has a positive radius of convergence; then, $\mathcal{T} = \Gamma^{-1}$.

function at the origin, which is the unique solution around the origin of the following system of n rational ODES and initial conditions:

$$\frac{d}{dx}f(x) = -(\nabla_{\mathbf{y}} \mathcal{E})(x, f(x))^{-1} \cdot \left(\frac{\partial \mathcal{E}}{\partial x}(x, f(x)) \right)^T \quad f(0) = \mathbf{r}_0. \quad (4.43)$$

PROOF. Under the conditions on \mathcal{E} and \mathbf{r}_0 stated in the hypotheses, the classical IFT implies the existence of a unique real analytic function $f : \mathbb{R} \rightarrow \mathbb{R}^n$, say $f = (f_1, \dots, f_n)$, such that $f(0) = \mathbf{r}_0$ and $\mathcal{E}(x, f(x))$ is identically 0. Moreover, it tells us that f satisfies the system of (polynomial) nonlinear ODES and initial conditions in (4.43). Note that $\det((\nabla_{\mathbf{y}} \mathcal{E})(0, \mathbf{r}_0)) \neq 0$ and the continuity of $\det(\mathcal{E}(x, f(x)))$ around the origin, guaranteed by the IFT [152, Th.9.28], in turn guarantee that $(\nabla_{\mathbf{y}} \mathcal{E})(x, f(x))$ is nonsingular in a neighborhood of $x = 0$. Let $\sigma = (\sigma_1, \dots, \sigma_n)$ be the stream of the coefficients of the Taylor series of f expanded at $x = 0$, taken componentwise: $\sigma = \mathcal{T}[f] := (\mathcal{T}[f_1], \dots, \mathcal{T}[f_n])$. Now σ is a stream solution of \mathcal{E} , as a consequence of the fact that \mathcal{T} is a \mathbb{R} -algebra homomorphism between \mathcal{A} and Σ : indeed, for each $p(x, \mathbf{y}) \in \mathcal{E}$, $0 = p(x, f(x))$ implies $(0, 0, \dots) = \mathcal{T}[0] = \mathcal{T}[p(x, f(x))] = p(\mathcal{T}[x], \mathcal{T}[f]) = p(x, \sigma)$. Uniqueness of σ is guaranteed by Theorem 8, because $(\nabla_{\mathbf{y}} \mathcal{E})(0, \mathbf{r}_0) = (\nabla_{\mathbf{y}} \mathcal{E})(0, \mathbf{r}_0, \mathbf{r}_0)$ (see Lemma 12) and it is invertible by hypothesis. \square

A corollary of the above theorem is that one can obtain the unique stream solution of \mathcal{E} also by computing the Taylor coefficients of the solution f of (4.43). Such coefficients can be computed without having to explicitly solve the system of ODES. We will elaborate on this point in Section 4.9, where, we will compare in terms of efficiency the method based on SDEs with the method based on ODES, on two nontrivial polynomial systems. Here, we just consider the ODES method on a simple example.

Example 17. Consider again the system $\mathcal{E} = \{y - (1 + xy^2)\}$ in the single variable $y = y_1$, with the initial condition $r_0 = 1$, seen in Example 16. Since $(\nabla_y \mathcal{E})(x, y) = 1 - 2xy$ is nonzero at $(0, r_0)$, we can apply Theorem 9. The ODE and initial condition in (4.43) in this case are, letting $f = y$: $\frac{d}{dx}y(x) = \frac{y^2}{1-2xy}$ and $y(0) = 1$. This system can be solved explicitly. Alternatively, one can compute the coefficients of the Taylor expansion of the solution, e.g. by successive differentiation: $y(x) = \sum_{j \geq 0} \frac{y^{(j)}(0)}{j!} x^j = 1 + 1x + 2x^2 + 5x^3 + 14x^4 + 42x^5 + \dots$. Such coefficients form again the stream σ of Catalan numbers that is therefore the unique stream solution of \mathcal{E} with $\sigma(0) = r_0 = 1$.

4.8 An extended example: three-coloured trees

We consider a polynomial system \mathcal{E} implicitly defining the generating functions of ‘three-coloured trees’, Example 14 in [71, Sect.4]. For each of the three considered colours (variables), [71, Sect.4] shows how to reduce \mathcal{E} to a single nontrivial equation. This implies algebraicity of the series implicitly defined by \mathcal{E} : the reduction is conducted using results from elimination theory [57]. Here we will show how to directly transform \mathcal{E} into a system of polynomial SDEs and initial conditions, (\mathcal{D}, ρ) , as implied by the stream IFT (Theorem 8). As the coefficients in \mathcal{E} are rational, reduction to SDEs directly implies algebraicity (Remark 11), besides giving a method of calculating the

streams coefficients. We will also consider reduction of \mathcal{E} to a system of polynomial ODES, as implied by the classical version of the IFT (Theorem 9), and compare the obtained SDE and ODE systems.

Three-coloured trees are binary trees (plane and rooted) with nodes coloured by any of three colours, a, b, c , such that any two adjacent nodes have different colours and external nodes are coloured by the a -colour. Let $\mathcal{A}, \mathcal{B}, \mathcal{C}$ denote the sets of three-coloured trees with root of the a, b, c color respectively, and A, B, C the corresponding ordinary generating functions: the n -th coefficient of A is the number of trees with a -coloured root and n external nodes; similarly for B and C . Below, we report from [71, Sect.4,eq.(40)] the polynomial system \mathcal{E} ; to adhere to the notation of Section 4.2, we have replaced the variables (A, B, C) with $\mathbf{y} = (y_1, y_2, y_3)$.¹²

$$\mathcal{E} : \begin{cases} y_1 - x - (y_2 + y_3)^2 & = 0 \\ y_2 - (y_3 + y_1)^2 & = 0 \\ y_3 - (y_1 + y_2)^2 & = 0. \end{cases} \quad (4.44)$$

System (4.44) has been derived via the symbolic method [71], a powerful technique to translate formal definitions of combinatorial objects into equations on generating functions to count those objects. For instance, consider a three-coloured tree with an a -coloured root. It can either be single node, accounted by x in the first equation, or a root with two subtrees, each with root either of b - or of c -colour. Considering this structure, system (4.44) can be readily deduced.

Since the number of external nodes of any empty tree is 0, we set $\mathbf{r}_0 = (0, 0, 0)$. It is immediate to verify that $\mathcal{E}(0, \mathbf{r}_0) = 0$ and $(\nabla_{\mathbf{y}} \mathcal{E})(0, \mathbf{r}_0, \mathbf{r}_0) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = I$, that is obviously invertible, hence Theorem 8 holds, and we generate system (4.37) in Theorem 8. In particular, after applying Lemma 8, we get the following polynomial system of SDEs and initial conditions:

$$\left\{ \begin{array}{l} y_1' = 2wy_1y_2 + wy_2y_3 - w & y_1(0) = 0 \\ y_2' = -2wy_1^2 - 4wy_1y_2 - wy_1y_3 - wy_1 - 2wy_2y_3 & y_2(0) = 0 \\ y_3' = -wy_1y_2 - wy_1 - 2wy_2 & y_3(0) = 0 \\ w' = 4w^2y_1^2y_2^2 + 4w^2y_1^2y_2y_3 - 8w^2y_1^2y_3^2 - 6w^2y_1^2y_3 + 8w^2y_1y_2^3 + & w(0) = -1 \\ \quad 14w^2y_1y_2^2y_3 + 6w^2y_1y_2^2 - 10w^2y_1y_2y_3^2 - 8w^2y_1y_2y_3 - 2w^2 & \\ \quad y_1y_2 - 4w^2y_1y_3^3 - 7w^2y_1y_3^2 - 7w^2y_1y_3 + 4w^2y_2^3y_3 + 6w^2y_2^2 & \\ \quad y_3^2 + 3w^2y_2^2y_3 - 4w^2y_2^2 - 6w^2y_2y_3^3 - 3w^2y_2y_3^2 - 10w^2y_2y_3 - & \\ \quad 3w^2y_2 - 2w^2y_3^2 - 3w^2y_3. & \end{array} \right. \quad (4.45)$$

See Appendix C.2 for details of the derivation. By Theorem 8, the original system \mathcal{E} in (4.44) has a unique stream solution σ such that $\sigma(0) = \mathbf{r}_0$, and $(\sigma, \tau) = (\sigma_1, \sigma_2, \sigma_3, \tau)$, for a suitable τ , is the unique stream solution of (4.45). In particular, we have: $\sigma_1 = (0, 1, 0, 0, 4, 16, 56, 256, 1236, \dots)$. This matches the generating function expansion for y_1 in Example 14 of [71]: $g_1(z) = z + 4z^4 + 16z^5 + 56z^6 + 256z^7 + 1236z^8 + \dots$

¹²We note that there is a slip in the first equation appearing in [71], by which the term $(B + C)^2 = (y_2 + y_3)^2$ appears with the wrong sign. The correct sign is used here.

On the other hand, applying the classic IFT (Theorem 9) to system (4.44), there is a unique real analytic solution $f(x) = \mathbf{y}(x) = (y_1(x), y_2(x), y_3(x))$ of the ODE initial value problem (4.43) s.t. $\mathbf{y}(0) = \mathbf{r}_0 = (0, 0, 0)$. The system in question can be computed starting from the classical jacobian of \mathcal{E} , $(\nabla_{\mathbf{y}} \mathcal{E})(x, \mathbf{y}) = \begin{bmatrix} 1 & -2y_2 + -2y_3 & -2y_3 - 2y_2 \\ -2y_1 - 2y_3 & 1 & -2y_3 - 2y_1 \\ -2y_1 - 2y_2 & -2y_2 - 2y_1 & 1 \end{bmatrix}$.

Since $\frac{\partial \mathcal{E}}{\partial x}(x, \mathbf{y})|_{x=0, \mathbf{y}=\mathbf{r}_0} = (-1, 0, 0)$, (4.43) yields the following system of rational ODEs and initial conditions:

$$\frac{d}{dx} \mathbf{y}(x) = -(\nabla_{\mathbf{y}} \mathcal{E})^{-1} \cdot \left(\frac{\partial \mathcal{E}}{\partial x} \right)^T = \tilde{\Delta}^{-1} \cdot \begin{bmatrix} 4y_1^2 + 4y_1y_2 + 4y_1y_3 + 4y_2y_3 - 1 \\ -4y_1^2 - 4y_1y_3 - 4y_1y_2 - 2y_1 - 4y_2y_3 - 2y_3 \\ -4y_1^2 - 4y_1y_2 - 4y_1y_3 - 2y_1 - 4y_2y_3 - 2y_2 \end{bmatrix} \mathbf{y}(0) = \mathbf{r}_0. \quad (4.46)$$

where $\tilde{\Delta} := -16y_1^2y_2 - 16y_1^2y_3 - 4y_1^2 - 16y_1y_2^2 - 32y_1y_2y_3 - 12y_1y_2 - 16y_1y_3^2 - 12y_1y_3 - 16y_2^2y_3 - 4y_2^2 - 16y_2y_3^2 - 12y_2y_3 - 4y_3^3 + 1$ is the determinant of $\nabla_{\mathbf{y}} \mathcal{E}$.

Considering a series solution of the system, we obtain, for the first component of the solution f :

$$y_1(x) = x + 4x^4 + 16x^5 + 56x^6 + 256x^7 + 1236x^8 + 5808x^9 + O(x^{10})$$

whose coefficients match those of σ_1 for (4.45).

4.9 Classical vs. stream IFT: computational aspects

We compare the stream and the classical version of the IFT from a computational point of view. First, we discuss how the recurrence (4.5) can be effectively implemented for *any* polynomial SDE initial value problem of the form (4.3), not necessarily arising from an application of Theorem 8. The basic idea is to always reduce products involving more than two terms to binary products, for which the convolution formula (4.1) can be applied. In order to perform this reduction systematically, let us consider the set T of all subterms $t = t(x, \mathbf{y})$ that occur in the polynomials p_i in \mathcal{D} . We assume that T also includes all the constants appearing in \mathcal{D} , the constant 1, and all the variables y_0 ($:= x$), y_1, \dots, y_n . For each term t in T , a stream σ_t is introduced via the following recurrence relation that defines $\sigma_t(k)$. Formally, the definition goes by lexicographic induction on (k, t) , with the second elements ordered according the “*subterm of*” relation. For the sake of notation, below we let $p_0 = 1$, and let the case $t = c \cdot t_1$ for $c \in \mathbb{K}$ be subsumed by the last clause, where c is treated as the constant stream $(c, 0, 0, \dots)$. Finally, $k > 0$.

$$\begin{aligned} \sigma_t(0) &= t(0, \mathbf{r}_0) \\ \sigma_{y_i}(k) &= \sigma_{p_i}(k-1) && (i = 0, \dots, n) \\ \sigma_c(k) &= 0 && (c \in \mathbb{K}) \\ \sigma_{t_1+t_2}(k) &= \sigma_{t_1}(k) + \sigma_{t_2}(k) \\ \sigma_{t_1 \cdot t_2}(k) &= \sum_{j=0}^k \sigma_{t_1}(j) \cdot \sigma_{t_2}(k-j). \end{aligned} \quad (4.47)$$

This algorithm for turning a system of SDEs into a system of recurrence relations can be considered as folklore. It has been applied in e.g. [89, Sect.10], to the SDE for the Fibonacci numbers, which is linear. Here we explicitly describe it for the general case of polynomial SDEs. Its correctness, as stated by the next lemma, is obvious.

Lemma 13. Let $\sigma = (\sigma_1, \dots, \sigma_n)$ be the unique stream solution of a problem (\mathcal{D}, ρ) of the form (4.3). With the above definition of σ_t , we have $\sigma_i = \sigma_{y_i}$, for $i = 1, \dots, n$.

In a practical implementation of this scheme, one can avoid recursing over the structure of t , as follows. At the k -th iteration ($k > 0$), the values $\sigma_t(k)$ are computed and stored by examining the terms $t \in T$ according to a total order on T compatible with the “*subterm of*” relation. In this way, whenever either of the last two clauses is applied, one can access the required values $\sigma_{t_1}(j), \sigma_{t_2}(j)$ up to $j = k$ already computed and stored away in the current iteration. The computation of the k -th coefficient $\sigma(k)$, given the previous ones, requires therefore $O(Pk + S)$ multiplications and additions, where P and S are the number of overall occurrences in T of the product and sum operators, respectively. Overall, this means $O(Pk^2 + Sk)$ operations for the first k coefficients. This complexity is minimized by choosing a format of polynomial expressions that minimizes P : for example, a Horner scheme (note that Horner schemes exist also for multivariate polynomials). Memory occupation grows linearly as $O(k(P + S))$.

Another method to generate the coefficients of the stream solution is applying the classical version of the IFT (Theorem 9), and rely on the ODE initial value problem in (4.43). However, this choice appears to be computationally less convenient. Indeed, apart from the rare cases where (4.43) can be solved explicitly, one must obtain the coefficients of the solution by expanding it as a power series — indeed its Taylor series. Once the rational system (4.43) is reduced to a polynomial form, which is always possible by introducing one extra variable, the coefficients of this power series can be computed by a recurrence relation similar to that discussed in Lemma 13 for (4.5). The catch is that the size of the resulting set of terms T is *significantly larger* for the ODE system (4.43) than it is for the SDE system (4.37). To understand why, consider that, under the given hypotheses, the SDE system in (4.37) is equivalent to $\mathcal{E}' = 0$, while the ODE system in (4.43) is equivalent to $\frac{d}{dx}\mathcal{E} = 0$. Now, the terms appearing in \mathcal{E}' are approximately *half the size* of those appearing in $\frac{d}{dx}\mathcal{E}$. This is evident already when comparing with one another the stream and the ordinary derivatives of a bivariate polynomial $p(x, y) = q_m(y)x^m + \dots + q_1(y)x + q_0(y)$:

$$p(x, y)' = q_m(y)x^{m-1} + \dots + q_1(y) + q_0'(y)$$

$$\frac{d}{dx}p(x, y) = \left(q_m(y)mx^{m-1} + x^k \frac{d}{dx}q_m(y) \right) + \dots + \left(q_1(y) + x \frac{d}{dx}q_1(y) \right) + \frac{d}{dx}q_0(y).$$

A small experiment conducted with two different systems of polynomials, the three-coloured trees (4.44) and the one-dimensional system (4.41), is in agreement with these qualitative considerations. For each of these systems, we have computed a few hundreds coefficients of the solution, using both the methods in turn: SDEs via the recurrence relation of Lemma 13 (Theorem 8), and ODEs via a power series solution (Theorem 9). In the second case, we have used Maple’s `dsolve` command with the `series` option¹³. For both systems, we plot in Figure 4.2 the execution time as a function of the number of computed coefficients.

Remark 12 (Newton method). In terms of complexity w.r.t. k (number of computed coefficients), *Newton iteration* applied to formal power series [118, 169, 76, 36] does asymptotically better than the $O(k^2)$ algorithm outlined above. In particular, [36,

¹³Python and Maple code for this example available at <https://github.com/Luisa-unifi/IFT>

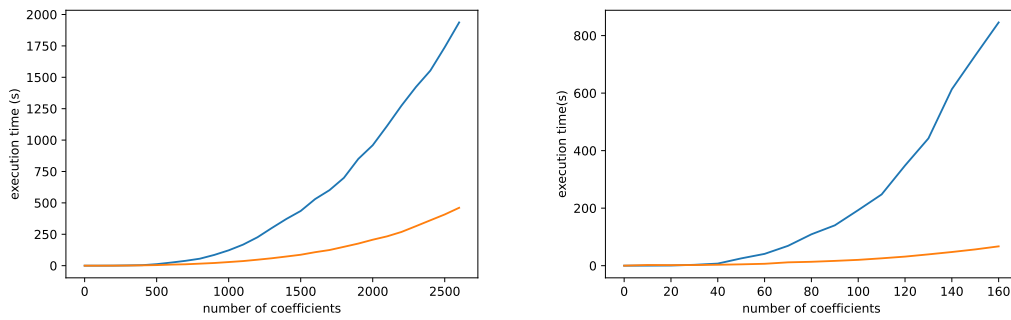


Figure 4.2: Execution time as a function of the number of computed coefficients for the stream solution of system (4.44) (left) and of system (4.41) (right). The orange (lower) curve is the recurrence relation (4.47) computed via Lemma 13 (authors’ Python code); the blue (upper) curve is the power series solution of (4.43) (Maple’s `dsolve`).

Th.3.12] shows that, under the same hypotheses of IFT, the first k coefficients of the solution of a system of algebraic equations can be computed by Newton iteration in time $O(k \log k)$; on the downside, each iteration of Newton involves in general finding the solution of a $n \times n$ linear system.

4.10 Conclusion

We have studied connections between polynomials, differential equations and streams, in terms of algebra and coalgebra. Our main result shows that, given any stream product that satisfies certain reasonable assumptions, there is a way to define a transition function on polynomials such that the induced unique coalgebra morphism into streams is a commutative \mathbb{K} -algebra homomorphism – and vice versa. An important application of this result is the design of algorithms for deciding polynomial stream equivalence, and for finding all polynomial equations of a given format. Moreover, we have presented an implicit function theorem for the stream calculus, a powerful set of tools for reasoning on infinite sequences of elements from a given field. Our theorem is directly inspired from the analogous one from classical calculus. We have shown that the stream IFT has clear computational advantages over the classical one.

As for future work, it would be interesting to see whether we can define new notions of products that respect the format we devised in this chapter. In the field of nonlinear dynamical systems [104], convolution of discrete sequences arises as a means to describe the composition of distinct signals or subsystems (e.g., a plant and a controller); we would like to understand if our approach can be useful to reason on such systems as well. The work on stream IFT can also be extended in several directions. First, we would like to explore the relations of our work with methods proposed in the realm of numerical analysis for efficient generation of the Taylor coefficients of ODE’ solutions [74]. Second, we would like to go beyond the polynomial format, and allow for systems of equation involving functions that are in turn defined via SDEs. Finally, we would like to extend the present results to the case of multivariate streams, that is consider a *vector* of independent variables, akin to the more general version of the classical IFT.

Chapter 5

Inference: Bayesian parameter estimation

5.1 Overview

In this chapter we consider parametric ordinary differential equations and we provide guaranteed estimates for their parameters values in terms of confidence intervals. More precisely, we investigate the more general problem of estimating posterior parameter distributions given an observation. The proposed framework encompasses a wide variety of systems, including ODEs with noisy state observations, and neural networks. We take a Bayesian standpoint, that is, we assume a known prior distribution on the unknown parameter values. Computationally, the most widespread approach to Bayesian posterior estimation relies on Monte Carlo (MC) simulation, and specifically on Markov Chain Monte Carlo (MCMC) [125, 91, 151] and on the particles-based Sequential Monte Carlo (SMC) [66]. The use of these techniques is justified by asymptotic results, saying that in the limit of an infinite number of simulation steps or particles, the samples produced by these methods converge in distribution to the exact posterior [151]. If the simulation is performed with only a finite number of steps or particles, which of course is always the case in practice, formal guarantees of correctness for the obtained samples are extremely hard to achieve.

We explore a hybrid approach, which combines imprecise probability in the form of *P-boxes* [68] (see below) with MC simulation. The goal is to obtain sharp estimates of posterior quantities, such as Cumulative Distribution Functions (CDFs), and their expectations and higher moments, equipped with formal guarantees of correctness. At the same time, we aim at reducing the computational effort required by the simulation phase, in comparison to the above mentioned classical MC methods. Instrumental in achieving these goals are the following three elements: (1) leveraging the power of *Interval Arithmetic* (IA) [131] in order to drastically reduce the parameter search space; (2) accepting a level of *controlled* uncertainty on the computed estimates, introduced by the MC simulation phase of our method; (3) switching from conditioning on an individual observation y^* to conditioning on a (small) *set* of potential observations S^* . We give a more detailed account of our approach below.

The proposed method consists of two phases. The first phase is entirely determin-

istic. We assume a functional relation among observations y , parameters θ and noise ψ , say $y = g(\theta, \psi) := f(\theta) + h(\psi)$, for known, real vector valued functions f, h . With this functional model, the problem of estimating the probability of θ given that $y \in S^*$ can be recast as a problem of volume — or better, probability measure — estimation (Subsection 5.2.1). Rather than going for a direct MC estimation of the involved measures, though, we first reduce the search space: we compute a tight overapproximation of the set of *feasible* parameters θ , those that can actually be mapped into S^* by g for some instance of the noise ψ . Such an overapproximation can be effectively computed relying on an *interval extension* of the function f : via IA, one can often determine at once if a whole rectangular region of the parameter space is unfeasible and discard it right away. This process can be repeated in a branch-and-bound fashion, and gives rise to a well-known refinement algorithm [100] (Section 5.2). The reduced parameter space obtained in this way can be significantly smaller than the original one. In any case, as discussed below, even a moderate volume reduction brings significant benefits in the subsequent phase of MC estimation. Moreover, the reduced space comes partitioned into axis-aligned hyper-rectangles, from which it is easy to draw samples.

In the second phase, we use a randomized algorithm \mathcal{A} to actually compute the confidence intervals of the wanted posterior quantities (Section 5.3). We start by building a pair of lower and upper approximations of the true posterior CDF: this pair is commonly referred to as a P-box [68], as the graphs of the approximate CDFs form an envelope for that of the exact CDF. Being the outcome of a randomized algorithm, unlike classical ones our P-boxes have a confidence level attached: such objects are known in the literature as *confidence bands* [108]. From confidence bands, confidence intervals for the posterior expectation and other moments can be easily built (Section 5.4). The core algorithm \mathcal{A} is very simple, and involves the extraction of a number of independent samples (θ, ψ) , with θ drawn from the reduced parameter space, and considering the fraction of these pairs that are mapped into S^* by g . Confidence levels for the resulting estimates can be established relying on an exponential tail inequality for the sum of independent random variables, Hoeffding’s bound [96]. We show that, by sampling from the reduced parameter space, the number of samples necessary to guarantee a given confidence level drops to a fraction μ^2 of the number necessary with the original space, where $\mu \in [0, 1]$ is the measure of the reduced space (Section 5.5).

We have put our algorithm at work on a few problems of parameter estimation from the literature (Section 5.6). Specifically, we have considered: the set of benchmarks for noisy ODE parameter estimation proposed in [52], where DSA, a method based on imprecise probability, is put forward; and a problem of feature relevance estimation for neural network classifiers proposed in [4]. For ODEs, we also offer a comparison with the results obtained with the state-of-the-art approaches (DSA, MCMC, SMC) from [52]. This comparison shows clearly the benefits of our method.

Related work We shall limit our discussion on related work to Bayesian inference, a framework where a prior distribution on parameters is presupposed. Bayesian parameter inference has found application in a variety of fields, ranging from biological models [80, 55, 179] to linear hybrid dynamical systems [75] and more recently probabilistic programming [83]; cf. the extensive literature review in [52, Sect.1]. As argued above, a problem of MCMC/SMC Bayesian inference methods is the difficulty of establishing formal guarantees for the obtained estimates. Moreover, these methods

are computationally demanding (cf. our Table 5.1 in Section 5.6) and require an explicit expression of the likelihood, the function mapping θ to the probability of obtaining a certain observation given θ ; this expression is often not available. An *Approximate Bayes Computation* (ABC) approach [117] has been proposed in recent years that also works in the absence of an explicit likelihood, and is more similar in spirit to ours. However, ABC shares the same difficulties as MCMC and SMC about formal guarantees, and is even more demanding from a computational point of view.

Closely related to ours is a method recently proposed by Chou and Sankaranarayanan [52] for ODE parameter inference. This method too is based on imprecise probability [64, 172, 70, 68], which is a way of dealing with uncertainty. Like in our case, the parameter space is divided into disjoint cells. Differently from our approach, first likelihoods bounds for each cells are computed analytically; then these bounds are normalized to obtain bounds on posterior probabilities. The method of [52] avoids MC estimation, hence the computed bounds on probabilities are certain. On the contrary, in our case the MC phase introduces a level of controlled aleatoric uncertainty, hence our bounds come equipped with confidence levels. As clearly shown by the comparison in Section 5.6, the MC phase allows us to trade off a small level of (un)certainty for greater efficiency and accuracy. Additional differences between [52] and our approach are discussed in Section 5.6.

An important computational ingredient of our approach is the SIVIA refinement algorithm [102, 100], which has been used in several works on parameter estimation. Notably, Jaulin in [101] proposes the use of IA for Bayesian estimation, in the following sense: given $\alpha \in [0, 1]$ and a posterior probability density function, compute a minimal volume region whose probability w.r.t. the density equals α . Note that this is very different from the problem considered here: we apply SIVIA to the *model* function f , not to the posterior density function. In fact, we do not even presuppose an explicit knowledge of the posterior density. Another recent proposal is the application of SIVIA to feature relevance in neural networks [4]; this is further discussed in Section 5.6.

A proposed method for obtaining confidence bands from empirical CDF functions relies on the Dvoretzky–Kiefer–Wolfowitz (DKW) inequality [67, 121, 108]. This is an exponential tail inequality that bounds the probability that the empirical CDF deviates from the exact CDF by more than a given ϵ . When compared to our measure-based approach, a serious drawback of empirical CDFs in the present setting is that they require *exact* sampling from the posterior, a nontrivial problem in itself. General exact sampling schemes, like rejection sampling [151], might turn out to be very expensive. We leave for future work an experimental comparison with the empirical CDF + DKW approach.

5.2 Preliminaries

5.2.1 Framework and problem statement

Let $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ and $h : \mathbb{R}^\ell \rightarrow \mathbb{R}^m$ be functions, with f continuous. Here, $y = g(\theta, \psi) := f(\theta) + h(\psi)$ will be interpreted as a functional relation among the *observations* y , the *parameters* θ and the (independent) *nuisance parameters* ψ . For

instance, the parameters ψ might represent additive noise: $\ell = m$, $h = \text{identity}$, hence $y = f(\theta) + \psi$. A probability measure μ on the space of all parameters $\mathbb{R}^n \times \mathbb{R}^\ell$ is given, and f, h are assumed to be measurable under μ . This induces a triple of random variables (Θ, Ψ, Y) where $Y := g(\Theta, \Psi) = f(\Theta) + h(\Psi)$. We will assume that $\mu(\cdot)$ factorizes as $\mu(A \times B) = \mu_0(A) \cdot \mu_1(B)$, where μ_0, μ_1 are probability measures over \mathbb{R}^n and \mathbb{R}^ℓ , respectively. In other words, Θ and Ψ are independent. Moreover, in all applications, we shall consider a μ_0 with a finite-diameter support¹. Given a (typically, small) measurable set of observations $S^* \subseteq \mathbb{R}^m$, for any $t \in (\mathbb{R} \cup \{+\infty\})^n$ one is interested in computing the quantity $F(t|S^*)$ defined below, which is the a posteriori CDF of Θ given $Y \in S^*$. Here, \leq on vectors is taken componentwise, $\mathbb{R}_{\leq t}^n := \{\theta \in \mathbb{R}^n : \theta \leq t\}$ and $P_t := \mathbb{R}_{\leq t}^n \times \mathbb{R}^\ell$. Provided $\Pr(Y \in S^*) > 0$, we define:

$$\begin{aligned} F(t|S^*) &:= \Pr(\Theta \leq t | Y \in S^*) \\ &= \frac{\mu(P_t \cap g^{-1}(S^*))}{\mu(g^{-1}(S^*))}. \end{aligned} \quad (5.1)$$

We introduce a notion of correctness for randomized algorithms that approximate $F(t|S^*)$.

Definition 14 (algorithms for confidence intervals). *Let f, h, g, μ and S^* be fixed as specified above. Consider a randomized algorithm \mathcal{A} that, taken as input a tuple $t \in (\mathbb{R} \cup \{+\infty\})^n$, returns as output a pair a real valued random variables, written $\mathcal{A}(t) = [\underline{\mathcal{A}}(t), \overline{\mathcal{A}}(t)]$. For any $\delta \geq 0$, we say that \mathcal{A} approximates $F(\cdot|S^*)$ with confidence $1 - \delta$ if for each $t \in (\mathbb{R} \cup \{+\infty\})^n$*

$$\Pr\left(\underline{\mathcal{A}}(t) \leq F(t|S^*) \leq \overline{\mathcal{A}}(t)\right) \geq 1 - \delta.$$

For each t , the probability $\Pr(\cdot)$ is taken only on the internal random choices in the execution of $\mathcal{A}(t)$.

In other words, for each t , $[\underline{\mathcal{A}}(t), \overline{\mathcal{A}}(t)]$ is a confidence interval for $F(t|S^*)$. In the above definition, we do not impose requirements on the accuracy of the approximation, that is on the width of the interval $[\underline{\mathcal{A}}(t), \overline{\mathcal{A}}(t)]$: this can only be judged a posteriori.

Based on an algorithm \mathcal{A} for $F(t|S^*)$, one can easily build P-boxes. We shall limit our discussion to the important special case of marginal CDFs. For $\lambda \in \mathbb{R}$, let $F(\lambda|S^*)$ abbreviate $F(t|S^*)$ with $t = (\lambda, +\infty, \dots, +\infty)$; similarly, let $\mathcal{A}(\lambda) := \mathcal{A}(t)$. Note that $F(\lambda|S^*) = \Pr(\Theta_1 \leq \lambda | Y \in S^*)$ is the first marginal posterior CDF of Θ (the same reasoning applies to the other marginals). Now choose $k+1 \geq 2$ node points on the real line, say $\lambda_0 < \lambda_1 < \dots < \lambda_k$, such that $F(\lambda_0|S^*) = 0$ and $F(\lambda_k|S^*) = 1$. Based on $\mathcal{A}(\lambda_1), \dots, \mathcal{A}(\lambda_{k-1})$, we define F^- and F^+ , stepwise lower and upper approximations of F , as follows. Below, for the sake of uniform notation, we convene that $\underline{\mathcal{A}}(\lambda_0)$ denotes 0, and that $\overline{\mathcal{A}}(\lambda_k)$ denotes 1. Moreover $j = 0, \dots, k-1$.

$$F^-(\lambda|S^*) := \begin{cases} 0 & \text{if } \lambda < \lambda_0 \\ \underline{\mathcal{A}}(\lambda_j) & \text{if } \lambda \in [\lambda_j, \lambda_{j+1}) \\ 1 & \text{if } \lambda \geq \lambda_k \end{cases} \quad F^+(\lambda|S^*) := \begin{cases} 0 & \text{if } \lambda < \lambda_0 \\ \overline{\mathcal{A}}(\lambda_{j+1}) & \text{if } \lambda \in [\lambda_j, \lambda_{j+1}) \\ 1 & \text{if } \lambda \geq \lambda_k. \end{cases} \quad (5.2)$$

¹supp(μ_0) is the smallest closed measurable set $T \subseteq \mathbb{R}^n$ s.t. $\mu_0(T) = 1$.

Note that F^- , F^+ are in turn random variables, depending on the random variables $\mathcal{A}(\lambda_1), \dots, \mathcal{A}(\lambda_{k-1})$. Importantly from the computational point of view, these $k - 1$ calls to \mathcal{A} are not required to be independent: as we shall see, there is a way of computing them at essentially the same cost of a single call (see Section 5.3, Remark 14). The next result says that, with high probability, the pair (F^-, F^+) is a P-box for the exact marginal posterior CDF $F(\lambda|S^*)$. Otherwise said, the pair (F^-, F^+) is a *confidence band* [108] for $F(\lambda|S^*)$. The proof of the proposition is an immediate consequence of the previous definition of \mathcal{A} and of a union bound on probabilities.

Proposition 6 (confidence bands). *Suppose \mathcal{A} approximates $F(\cdot|S^*)$ with confidence $1 - \delta$. Then, with probability at least $1 - (k - 1)\delta$, we have that for all $\lambda \in \mathbb{R}$:*

$$F^-(\lambda|S^*) \leq F(\lambda|S^*) \leq F^+(\lambda|S^*). \quad (5.3)$$

From the confidence band (F^-, F^+) , confidence intervals for a variety of statistics, including moments of the true posterior, can be easily computed. We will detail this point in Section 5.4. We will design a correct core algorithm \mathcal{A} under certain mathematical and computational assumptions, listed below. Mathematically, we assume the following.

1. $S^* = I_1 \times \dots \times I_m$ is an axis-aligned hyper-rectangle (from now on, *rectangle* for short), where each $I_j = [a_j, b_j]$ ($a_j < b_j$) is a closed interval of \mathbb{R} . In typical use cases, S^* might be a small rectangle centered at a given observation $y^* \in \mathbb{R}^m$;
2. there exists an *interval extension* \bar{f} of the function f , see the next section for the precise definition.

Computationally, we assume we have efficient algorithms to:

- (a) compute f, \bar{f} , and h ;
- (b) compute $\mu_0(R)$ for any rectangle $R \subseteq \mathbb{R}^n$;
- (c) for any non-zero measure rectangle $R \subseteq \mathbb{R}^n$, sample from the the random variable $\Theta|_R$ obtained by conditioning² Θ on the event $\Theta \in R$;
- (d) sample from Ψ .

In the next section, we will review in detail these prerequisites and explore some instances of the model where they are fulfilled.

5.2.2 Interval arithmetic, coverings, set inversion

Interval Arithmetic (IA) [131] offers a framework to compute rigorously with abstract versions of functions, where individual points are replaced by intervals. The abstract functions conservatively extend their concrete counterparts (see below). IA can be used, for instance, to compute certified bounds on the error of numerical operations. In what follows, we quickly introduce the terminology of IA we need.

²Explicitly, $\Theta|_R$ is the random variable induced by the measure on \mathbb{R}^n defined by $\mu_R(R') := \mu_0(R \cap R') / \mu_0(R)$.

Formally, an *interval* I is a finite, closed interval $[a, b] \subseteq \mathbb{R}$. A *rectangle* is a cartesian product of intervals, $R = I_1 \times \cdots \times I_k$. We let \mathbb{IR} denote the set of all intervals included in \mathbb{R} . For $J = (I_1, \dots, I_k) \in \mathbb{IR}^k$ a tuple of intervals, we define the rectangle $\overline{J} := I_1 \times \cdots \times I_k \subseteq \mathbb{R}^k$. An *interval extension* of $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ is a function $\overline{f} = (\overline{f}_1, \dots, \overline{f}_n) : \mathbb{IR}^n \rightarrow \mathbb{IR}^m$ that is compatible with f , that is:

1. whenever $x \in [I_1, \dots, I_n]$ then $f(x) \in [\overline{f}(I_1, \dots, I_n)]$; and
2. whenever $I_i \subseteq I'_i$ for $i = 1, \dots, n$ then $[\overline{f}(I_1, \dots, I_n)] \subseteq [\overline{f}(I'_1, \dots, I'_n)]$.

A basic fact about IA is that the set of interval extensions is closed under composition: if \overline{f} and \overline{g} are interval extensions of f, g respectively, where $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ and $g : \mathbb{R}^m \rightarrow \mathbb{R}^p$, then $\overline{g} \circ \overline{f}$ is an interval extension of $g \circ f$. By slight abuse of notation, for a rectangle $R = I_1 \times \cdots \times I_n$, we let $\overline{f}(R) := [\overline{f}(I_1, \dots, I_n)]$. The volume of the difference $\overline{f}(R) \setminus f(R)$ is a measure of how accurate the interval extension \overline{f} is with respect to the concrete function f . Functions most commonly found in applications, including all polynomials, exponentials and trigonometric functions, do possess accurate interval extensions [131]. Moreover, every monotonic function has an interval extension.

Given a set $A \subseteq \mathbb{R}^n$, a *covering* of A is a finite set of rectangles $\mathcal{C} = \{R_1, \dots, R_K\}$ such that: (a) $A \subseteq \bigcup_{i=1}^K R_i$, and (b) the rectangles R_1, \dots, R_K are *almost-disjoint* according to a fixed measure $\mu_0(\cdot)$ on \mathbb{R}^n , that is for each $1 \leq i < j \leq K$, $\mu_0(R_j \cap R_i) = 0$ ³. Given a function $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ and rectangles $S \subseteq \mathbb{R}^m$ and $R_0 \subseteq \mathbb{R}^n$, we will be interested in computing a covering of the set

$$f^{-1}(S) \cap R_0 = \{x \in \mathbb{R}^n : f(x) \in S\} \cap R_0.$$

This problem is referred to as *set inversion* in [102, 100], where a practical branch-and-bound algorithm based on IA is offered: SIVIA, standing for *Set Inversion Via Interval Analysis*. In our application of SIVIA, S will be a suitable superset of S^* , and R_0 a superset of the support of μ_0 . We give a pseudocode description of SIVIA as Algorithm 4. SIVIA maintains a set L of rectangles, each represented as a n -tuple of intervals, initially containing only R_0 . At each iteration, a rectangle R is extracted from L , and IA is used to check if the \overline{f} -image of R is: (a) entirely inside S (feasible), or (b) entirely outside S (unfeasible), or (c) indeterminate. In case (a), R is inserted into a set of *inner rectangles*, \mathcal{C}_{in} , and will not be reconsidered; in case (b), R is simply discarded; in case (c), R is bisected⁴ and the resulting halves R_1, R_2 are inserted into L for later consideration. An exception to the last rule is when the width of R is less than a given resolution threshold, ρ : in this case, R is inserted into a set of *outer rectangles*, \mathcal{C}_{out} and will not be reconsidered. Informally, outer rectangles are those found at the border of the covering. The output of the algorithm is the pair $(\mathcal{C}_{\text{in}}, \mathcal{C}_{\text{out}})$. Note that the set theoretic operations involving $\overline{f}(R)$ and S in lines 5 and 7 can be efficiently implemented, as both sets are rectangles.

³E.g. R_i and R_j might share part of a face.

⁴Explicitly, if $R = I_1 \times \cdots \times I_n$ and $I_j = [a, b]$ is the largest among the intervals involved in the product, we have: $\text{width}(R) := b - a$, $R_1 := I_1 \times \cdots \times [a, c] \times \cdots \times I_n$ and $R_2 = I_1 \times \cdots \times [c, b] \times \cdots \times I_n$, where $c = \frac{a+b}{2}$.

Algorithm 4 SIVIA [100]

Input: $S \subseteq \mathbb{R}^m$, a rectangle to be inverted; $R_0 \in \mathbb{R}^n$, a rectangle; $\bar{f} : \mathbb{R}^n \rightarrow \mathbb{R}^m$, an interval extension of function f ; $\rho > 0$, a resolution threshold.

Output: $(\mathcal{C}_{\text{in}}, \mathcal{C}_{\text{out}})$, sets of rectangles such that $\mathcal{C}_{\text{in}} \cup \mathcal{C}_{\text{out}}$ covers $f^{-1}(S) \cap R_0$.

```

1:  $\mathcal{C}_{\text{in}}, \mathcal{C}_{\text{out}} \leftarrow \emptyset$ 
2:  $L \leftarrow \{R_0\}$ 
3: while  $L \neq \emptyset$  do
4:    $R \leftarrow \text{remove}(L)$  ▷ extract a rectangle from  $L$ 
5:   if  $\bar{f}(R) \subseteq S$  then ▷ if the rectangle is feasible
6:      $\mathcal{C}_{\text{in}} \leftarrow \mathcal{C}_{\text{in}} \cup \{R\}$  ▷ then insert it into  $\mathcal{C}_{\text{in}}$ 
7:   else if  $\bar{f}(R) \cap S \neq \emptyset$  then ▷ otherwise if the rectangle is indeterminate
8:     if  $\text{width}(R) < \rho$  then ▷ if its width less than resolution
9:        $\mathcal{C}_{\text{out}} \leftarrow \mathcal{C}_{\text{out}} \cup \{R\}$  ▷ then insert it into  $\mathcal{C}_{\text{out}}$ 
10:    else
11:       $R_1, R_2 \leftarrow \text{Bisect}(R)$  ▷ otherwise bisect the rectangle
12:       $L \leftarrow L \cup \{R_1, R_2\}$  ▷ insert the resulting halves into  $L$ 
13:    end if
14:  end if
15: end while
16: return  $(\mathcal{C}_{\text{in}}, \mathcal{C}_{\text{out}})$ 

```

Lemma 14. *Algorithm 4 always terminates returning a pair $(\mathcal{C}_{\text{in}}, \mathcal{C}_{\text{out}})$ of sets of rectangles. Moreover, (1) $\mathcal{C}_{\text{in}} \cup \mathcal{C}_{\text{out}}$ is a covering of $f^{-1}(S) \cap R_0$; (2) $\bigcup \mathcal{C}_{\text{in}} \subseteq f^{-1}(S) \cap R_0$.*

Remark 13 (complexity). The worst case time complexity of SIVIA is exponential in n [100], not surprisingly given its branch-and-bound structure. This theoretical complexity is less of a concern for our purposes than it may seem at first glance, for two reasons. First, we can set a relatively large resolution threshold ρ , as the subsequent Monte Carlo estimation of $\mu_0(f^{-1}(S))$ can greatly benefit even from a conservative covering; this point will be made precise in Section 5.5. Second, in our application of SIVIA, the set S will be typically quite small: as a consequence, one may expect that in roughly half of the iterations the rectangle R will be unfeasible hence will not lead to further bisections.

5.2.3 Discretized ODES and neural networks

Ordinary differential equations and neural networks are models that naturally fit in the framework introduced in Subsection 5.2.1.

Let us consider ODES first. Let $z = (z_1, \dots, z_p)$ be a vector of distinct variables. Consider an initial value problem defined by: a system of p (nonlinear) ODES that also depend on a tuple of n parameters θ , written $\dot{z}(t) = \phi(z(t), \theta)$, and a fixed initial condition $z(0) = z_0 \in \mathbb{R}^p$. Under suitable regularity assumptions on ϕ , for any $\theta^* \in \mathbb{R}^n$ a unique solution $z(t; z_0; \theta^*)$ to this problem exists in a time interval containing 0. Assuming ϕ , z_0 and a prior probability distribution $\mu_0(\theta)$ are known, the

goal is to estimate the posterior distribution of θ given a vector y^* of k observations, obtained by a measurement of the solution $z(t; z_0; \theta^*)$ at fixed time points $t_1 < \dots < t_k$, say $y^* = (y_1^*, \dots, y_k^*) \in \mathbb{R}^m$, with $y_i^* \in \mathbb{R}^p$ and $m = k \cdot p$. Such measurements will be assumed to be affected by an additive noise $\psi \in \mathbb{R}^m$, generated by an independent random variable Ψ , induced by a known probability measure μ_1 over \mathbb{R}^m . To recast this in the setting of Subsection 5.2.1, let us fix a tolerance threshold $\gamma > 0$, let $\theta \in \mathbb{R}^n$, $\psi \in \mathbb{R}^m$ and define:

$$\begin{aligned} S^* &= \prod_{i=1}^k \prod_{j=1}^p [y_{i,j}^* - \gamma, y_{i,j}^* + \gamma] \\ f(\theta) &= (z(t_1; z_0; \theta), \dots, z(t_k; z_0; \theta)) \\ g(\theta, \psi) &= f(\theta) + \psi. \end{aligned}$$

An interval version of $f(\theta)$ can be computed via set reachability techniques for ODEs, see e.g. [47, 8]; but this is quite expensive. In our experiments, from the outset we will replace the original model with an accurate discretized version of the ODE, obtained by applying Euler's scheme, which we now quickly introduce. For a fixed z_0 and time step $\tau > 0$, consider the recurrence relation ($s \geq 0$): $\tilde{z}_0 := z_0$ and $\tilde{z}_{s+1} = \tilde{z}_s + \tau \cdot \phi(\tilde{z}_s, \theta)$. It can be seen that $\tilde{z}_s \approx z(s \cdot \tau; z_0; \theta)$, and this approximation can be made arbitrarily accurate by choosing τ sufficiently small. Making the dependence on θ explicit in the notation, let us denote by $\tilde{z}_s(\theta)$ the elements of this sequence. Assuming that each $s_i = t_i/\tau$ is an integer for $i = 1, \dots, k$, we will replace the above $f(\theta)$ with the following

$$f(\theta) := (\tilde{z}_{s_1}(\theta), \dots, \tilde{z}_{s_k}(\theta)).$$

If ϕ has an interval extension $\bar{\phi}$, it is easy to compute \bar{f} , an interval extension of f .

Example 18 (simple ball/1). We use a toy model also considered in [52] as a running example. The vertical motion of a ball obeys the following ODE, where in $z = (z_1, z_2)$, z_1 is the position, z_2 is the velocity, and the parameter θ , on which we want to make inference, is gravity acceleration: $\dot{z} = (\dot{z}_1, \dot{z}_2) = \phi(z, \theta) := (z_2, -\theta)$. We take $z_0 := (0, -4)$ as the initial condition. Although this ODE is trivial to solve analytically, for the purpose of illustration we will consider its Euler's discretization. We choose $\tau = 0.1$ and consider the recurrence relation for $\tilde{z}_s = (\tilde{z}_{1,s}, \tilde{z}_{2,s})$ given by: $\tilde{z}_0 := z_0$ and $\tilde{z}_{s+1} := \tilde{z}_s + \tau \cdot (\tilde{z}_{2,s}, -\theta)$. Making the dependence on the parameter θ explicit, let us write this as $\tilde{z}_s(\theta)$. We choose to observe the system once at time $t = 1$, hence set $f(\theta) := \tilde{z}_{10}(\theta)$. For $\theta^* = 9.8$, one has $f(\theta^*) = (-8.4, -13.8)$: we choose this as our observation y^* and fix the tolerance $\gamma = 0.5$, hence $S^* = [-8.9, -7.9] \times [-14.3, -13.3]$. We assume a noise vector $\psi \in [-1, 1]^2$ and let h be the identity: consequently, $g(\theta, \psi) := f(\theta) + \psi$ is the functional description of our model. To complete the description, we have to specify the probability measures μ_0 and μ_1 : we choose μ_0 to be the uniform distribution on the finite support $R_0 := [7, 12]$, and μ_1 to consist of a pair of independent truncated gaussian distributions on $[-1, 1]$ of standard deviation 0.1. Application of SIVIA to this example is postponed to Example 19.

Let us now consider neural networks. Generally speaking, a trained feedforward neural network with k hidden layers [82] implements a function $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$

defined as

$$f(\theta) = (f_{k+1} \circ f_k \circ \cdots \circ f_0)(\theta)$$

where each layer f_j has the structure $f_j(\xi_j) = \alpha_j(W_j \cdot \xi_j + b_j)$: here, ξ_j is a column vector, W_j, b_j are fixed and known weight matrix and bias vector of appropriate dimensions, respectively, while α_j is an activation function, applied componentwise to $W_j \cdot \xi_j + b_j$; here $\xi_0 = \theta$ is seen as a column vector. Provided each of the activation functions α_j possesses an interval extension $\bar{\alpha}_j$, an interval extension \bar{f} exists and can be easily computed. Commonly encountered activation functions, such as various versions of Linear Unit (LU), hyperbolic tangent, and in general monotonic activation functions, do possess interval extensions. Recasting this in the framework of Subsection 5.2.1, one can be interested in inferring a posterior distribution of θ given the result of an application of the function, $y^* = f(\theta^*)$. Noise is not considered in this setting. Moreover, one is often interested in the restriction of f to few selected components of θ . This is the case in the application to feature relevance, that will be discussed in detail in Section 5.6.

5.3 The core algorithm \mathcal{A}

Consider equality (5.1). We will first discuss how to estimate the denominator $\mu(g^{-1}(S^*))$. Let $R_0 \subseteq \mathbb{R}^n$ be a rectangle, $R_0 \supseteq \text{supp}(\mu_0)$, the support of μ_0 : in the discussion in Section 5.1, R_0 corresponds to the *original space*, the one from which one would sample θ in the absence of further information. We take the *feasible space* to be

$$\mathcal{F} := \text{pr}_{1..n} \left(g^{-1}(S^*) \right) \cap R_0.$$

This set⁵ contains all θ 's that can be sampled and mapped into S^* for some choice of ψ . Now assume we have a covering $\mathcal{C} = \{R_1, \dots, R_K\}$ of \mathcal{F} : the union of the rectangles in \mathcal{C} forms the *reduced space*. We will discuss later in the section how to compute \mathcal{C} .

Lemma 15. $\mu(g^{-1}(S^*)) = \mu \left(g^{-1}(S^*) \cap (R_1 \times \mathbb{R}^\ell) \right) + \cdots + \mu \left(g^{-1}(S^*) \cap (R_K \times \mathbb{R}^\ell) \right)$.

PROOF. Let $A = R_1 \cup \cdots \cup R_K$. We have $g^{-1}(S^*) \cap (R_0 \times \mathbb{R}^\ell) = g^{-1}(S^*) \cap (A \times \mathbb{R}^\ell)$, because for every $\theta \in R_0 \setminus A \subseteq \mathcal{F}^c$ we have $(\theta, \psi) \notin g^{-1}(S^*)$ for any $\psi \in \mathbb{R}^\ell$. Then by elementary set-theoretic reasoning

$$g^{-1}(S^*) \cap (R_0 \times \mathbb{R}^\ell) = \left(g^{-1}(S^*) \cap (R_1 \times \mathbb{R}^\ell) \right) \cup \cdots \cup \left(g^{-1}(S^*) \cap (R_K \times \mathbb{R}^\ell) \right).$$

By assumption, the rectangles R_1, \dots, R_K are almost disjoint w.r.t. μ_0 , which implies the above union is almost disjoint w.r.t. μ . Moreover, $R_0 \times \mathbb{R}^\ell \supseteq \text{supp}(\mu)$. Consequently:

$$\begin{aligned} \mu(g^{-1}(S^*)) &= \mu \left(g^{-1}(S^*) \cap (R_0 \times \mathbb{R}^\ell) \right) \\ &= \mu \left(g^{-1}(S^*) \cap (R_1 \times \mathbb{R}^\ell) \right) + \cdots + \mu \left(g^{-1}(S^*) \cap (R_K \times \mathbb{R}^\ell) \right). \end{aligned}$$

⁵ $\text{pr}_{1..n}$ is projection on the first n coordinates. Elements of \mathcal{F} outside the support of μ_0 play no role.

□

Now each of the summands in Lemma 15, $\mu\left(g^{-1}(S^*) \cap (R_i \times \mathbb{R}^\ell)\right)$ for $i = 1, \dots, K$, can be estimated via a MC simulation: informally speaking, one draws a number N_i of samples (θ, ψ) from $R_i \times \mathbb{R}^\ell$ and computes the fraction r_i of them such that $g(\theta, \psi) = f(\theta) + h(\psi) \in S^*$. Then

$$\begin{aligned} \mu\left(g^{-1}(S^*) \cap (R_i \times \mathbb{R}^\ell)\right) &\approx r_i \cdot \mu(R_i \times \mathbb{R}^\ell) \\ &= r_i \cdot \mu_0(R_i) \end{aligned} \quad (5.4)$$

(the approximate equality above will be rendered rigorously below). Note that, by our assumption of independence, (θ, ψ) can be sampled by separately drawing θ from \mathbb{R}^n via $\mu_0|_{R_i}$, and ψ from \mathbb{R}^ℓ via μ_1 , which we assume we know how to do. Overall, the more tightly the union of the R_i 's overapproximates \mathcal{F} , the more efficient this process is: this will be made precise in Section 5.5.

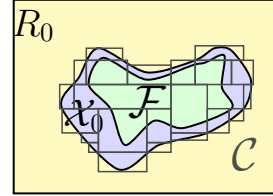
Concerning the actual computation of a covering \mathcal{C} of the feasible space, we proceed as follows. Let \underline{h}, \bar{h} be the vectors of inf's and sup's values of h over \mathbb{R}^ℓ , taken componentwise, possibly equal to $\pm\infty$. That is, for $i = 1, \dots, m$

$$\underline{h}_i := \inf_{\psi \in \mathbb{R}^\ell} h_i(\psi) \quad \bar{h}_i := \sup_{\psi \in \mathbb{R}^\ell} h_i(\psi).$$

The following lemma is an easy consequence of the definition of \mathcal{F} . We let $A + B$ denote the Minkowski sum of two subsets of \mathbb{R}^m , $A + B := \{a + b : a \in A \text{ and } b \in B\}$, with the sum taken componentwise, and $[-\bar{h}, -\underline{h}] := \prod_{i=1}^m [-\bar{h}_i, -\underline{h}_i]$.

Lemma 16. $\mathcal{F} \subseteq \mathcal{X}_0 := f^{-1}(S^* + [-\bar{h}, -\underline{h}]) \cap R_0$.

Therefore a covering \mathcal{C} of \mathcal{X}_0 is also a covering of \mathcal{F} , cf. the figure on the right. Accordingly, we will consider coverings of \mathcal{X}_0 from now on. Moreover, we will always consider cases where both \underline{h} and \bar{h} are finite. Note that too large values of $|\underline{h}|, |\bar{h}|$ will tend to make \mathcal{X}_0 coincide with R_0 , trivializing the proposed method. A covering \mathcal{C} of \mathcal{X}_0 can be computed via the SIVIA algorithm presented in Section 5.2: $\mathcal{C} := \mathcal{C}_{\text{in}} \cup \mathcal{C}_{\text{out}}$, where $(\mathcal{C}_{\text{in}}, \mathcal{C}_{\text{out}}) = \text{SIVIA}(S, R_0, \bar{f}, \rho)$, with $S := S^* + [-\bar{h}, -\underline{h}]$ and $\rho > 0$ a chosen resolution threshold. Note that S too is a rectangle in \mathbb{R}^m , since S^* is by assumption.



The estimation of the numerator $\mu(g^{-1}(S^*) \cap P_t)$ in (5.1) proceeds similarly, but $g^{-1}(S^*)$ must be replaced with $g^{-1}(S^*) \cap P_t$, where $P_t = \mathbb{R}_{\leq t}^n \times \mathbb{R}^\ell$. Also, one must ensure that \mathcal{C} refines $\mathbb{R}_{\leq t}^n$ (see details in Algorithm 5, step 1). The summation in Lemma 15 is replaced by one that involves only the rectangles contained in $\mathbb{R}_{\leq t}^n$.

Having identified the basic ingredients, we proceed now to a formal presentation of the core algorithm \mathcal{A} : see Algorithm 5. The algorithm consists of three steps. Step 1 is entirely deterministic, and just consists in the refinement of \mathcal{C} , if required. Step 2 introduces the basic random variables. Step 3 introduces the random variables that correspond to the actual simulation part, consisting in an overall N independent samplings of the random variables defined in the previous step, and in the construction of the actual confidence interval. Here $\epsilon > 0$ represents an error threshold, which has an impact on the width of the returned confidence interval. Note that the quantity $r_i \cdot \mu_0(R_i)$

Algorithm 5 core algorithm \mathcal{A}

Input: $t \in (\mathbb{R} \cup \{+\infty\})^n$, a n -tuple of real numbers or $+\infty$.

Output: $\mathcal{A}(t) = [\underline{\mathcal{A}}(t), \overline{\mathcal{A}}(t)]$, a pair of random variables defining a confidence interval for $F(t|S^*)$.

Fixed parameters: S^*, f, h, μ_0, μ_1 , as in Subsection 5.2.1; $\mathcal{C} = \{R_1, \dots, R_K\}$, a covering of \mathcal{X}_0 s.t. $\mu_0(R_i) > 0$ for each $i = 1, \dots, K$; $N = N_1 + \dots + N_K$ ($N_i \geq 1$), an integer number of samples to draw in the simulation step (budget); $\epsilon > 0$, an error threshold.

- 1: If necessary, split the rectangles of \mathcal{C} to make it a *refinement* of $\mathbb{R}_{\leq t}^n$, that is: for each $R \in \mathcal{C}$, either $R \subseteq \mathbb{R}_{\leq t}^n$ or $\mu_0(R \cap \mathbb{R}_{\leq t}^n) = 0$. Let $H_t := \{j : 1 \leq j \leq K \text{ and } R_j \subseteq \mathbb{R}_{\leq t}^n\}$.
- 2: For each $i = 1, \dots, K$, recalling that $\Theta_{|R_i}$ is drawn from R_i according to $\mu_{|R_i}(\cdot)$, define the random variable

$$X_i := \mu_0(R_i) \cdot 1_{\{g(\Theta_{|R_i}, \Psi) \in S^*\}}.$$

- 3: For each $i = 1, \dots, K$, let X_{i1}, \dots, X_{iN_i} be N_i i.i.d. copies of X_i . Let $X := \sum_{i=1}^K \frac{1}{N_i} \sum_{j=1}^{N_i} X_{ij}$ and $X_t := \sum_{h \in H_t} \frac{1}{N_h} \sum_{j=1}^{N_h} X_{hj}$. Return the following, where the right endpoint by convention is 1 if $X - \epsilon < 0$.

$$\mathcal{A}(t) = [\underline{\mathcal{A}}(t), \overline{\mathcal{A}}(t)] := \left[\frac{X_t - \epsilon}{X + \epsilon}, \frac{X_t + \epsilon}{X - \epsilon} \right].$$

in (5.4) of the informal derivation above, corresponds in step 3 of the algorithm to a realization of the random variable $\sum_{j=1}^{N_i} \frac{1}{N_i} X_{ij} = \left(\frac{1}{N_i} \sum_{j=1}^{N_i} 1_{\{g(\Theta_{|R_i}^{(ij)}, \Psi^{(ij)}) \in S^*\}} \right) \cdot \mu_0(R_i)$, with the superscript (ij) used here to denote different i.i.d. copies of a random variable. As part of the parameters, we presuppose a partition of the sampling budget over the K rectangles of the covering \mathcal{C} , $N = \sum_{i=1}^K N_i$: an optimal way of determining this partition will be discussed in Section 5.5.

We proceed to prove the correctness of \mathcal{A} , which is based on the following well-known result. Note that the random variables considered in the statement are required to be independent, but need not be identically distributed.

Lemma 17 (Hoeffding's bound [96]). *Let Z_1, \dots, Z_k be independent random variables such that $a_i \leq Z_i \leq b_i$ for $i = 1, \dots, k$. Let $Z := \sum_{i=1}^k Z_i$ and $\epsilon > 0$. Then $\Pr(|Z - E[Z]| > \epsilon) \leq 2 \exp\left(-\frac{2\epsilon^2}{\sum_{i=1}^k (b_i - a_i)^2}\right)$.*

Theorem 10 (correctness of \mathcal{A}). *For any $t \in (\mathbb{R} \cup \{+\infty\})^n$, let $\mathcal{C}, N, N_i, X, X_t, \epsilon$ and $\mathcal{A}(t)$ be as defined in Algorithm 5.*

1. $E[X] = \mu(g^{-1}(S^*))$ and $\Pr(|X - E[X]| > \epsilon) \leq \delta_0 := 2 \exp\left(-\frac{2\epsilon^2}{\sum_{i=1}^K \frac{\mu_0(R_i)^2}{N_i}}\right)$.
2. $E[X_t] = \mu(P_t \cap g^{-1}(S^*))$ and $\Pr(|X_t - E[X_t]| > \epsilon) \leq \delta_1 := 2 \exp\left(-\frac{2\epsilon^2}{\sum_{h \in H_t} \frac{\mu_0(R_h)^2}{N_h}}\right)$.
3. $F(t|S^*) = \frac{E[X_t]}{E[X]} \in [\underline{\mathcal{A}}(t), \overline{\mathcal{A}}(t)]$ with probability at least $1 - \delta$, where $\delta = \delta_0 + \delta_1 \leq 2\delta_0$. In other words, \mathcal{A} approximates $F(\cdot|S^*)$ with confidence $1 - \delta$.

PROOF. We consider the three parts separately.

1. Let $i \in \{1, \dots, K\}$ and consider the definition of X_i in step 2, $X_i = \mu_0(R_i) \cdot 1_{\{g(\Theta_{|R_i}, \Psi) \in S^*\}}$. Now $Z := 1_{\{g(\Theta_{|R_i}, \Psi) \in S^*\}}$ is a Bernoulli random variable with success (1) probability p equal to

$$p = E[Z] = \Pr(g(\Theta_{|R_i}, \Psi) \in S^*) = \frac{\mu(g^{-1}(S^*) \cap R_i \times \mathbb{R}^\ell)}{\mu_0(R_i)}.$$

Hence, for each i, j : $E[X_{ij}] = E[X_i] = \mu_0(R_i)E[Z] = \mu(g^{-1}(S^*) \cap R_i \times \mathbb{R}^\ell)$. Applying the linearity of expectation, we have:

$$E[X] = \sum_{i=1}^K \frac{1}{N_i} \sum_{j=1}^{N_i} E[X_{ij}] = \sum_{i=1}^K \frac{1}{N_i} \cdot N_i \cdot \mu(g^{-1}(S^*) \cap R_i \times \mathbb{R}^\ell) \quad (5.5)$$

$$= \sum_{i=1}^K \mu(g^{-1}(S^*) \cap R_i \times \mathbb{R}^\ell) = \mu(g^{-1}(S^*)) \quad (5.6)$$

where the last step stems from Lemma 15. The upper bound on $\Pr(|X - E[X]| > \epsilon)$ is obtained by applying Hoeffding's bound (Lemma 17) to X , seen as the sum of the N independent random variables $\frac{X_{ij}}{N_i}$, for $i = 1, \dots, K$ and $j = 1, \dots, N_i$. Here we take into account the fact that, for each such i, j we have $0 \leq \frac{X_{ij}}{N_i} \leq \frac{\mu_0(R_i)}{N_i}$.

2. The derivation for $E[X_t]$ is similar to the previous case, but only the variables X_{hj} for $h \in H_t$, corresponding to rectangles contained in $\mathbb{R}_{\leq t}^n$, contribute to the summation. Therefore in place of (5.5)-(5.6), we have

$$\begin{aligned} E[X_t] &= \sum_{h \in H_t} \frac{1}{N_h} \sum_{j=1}^{N_h} E[X_{hj}] = \sum_{h \in H_t} \frac{1}{N_h} \cdot N_h \cdot \mu(g^{-1}(S^*) \cap R_h \times \mathbb{R}^\ell) \\ &= \sum_{h \in H_t} \mu(g^{-1}(S^*) \cap R_h \times \mathbb{R}^\ell) = \mu(P_t \cap g^{-1}(S^*)). \end{aligned}$$

3. The previous two parts imply that $F(t|S^*) = \frac{\mathcal{E}[X_t]}{\mathcal{E}[X]}$ (note that by assumption $E[X] = \mu(g^{-1}(S^*)) > 0$). The event $\frac{\mathcal{E}[X_t]}{\mathcal{E}[X]} \notin [\underline{\mathcal{A}}(t), \overline{\mathcal{A}}(t)]$ can be decomposed as: $(\frac{\mathcal{E}[X_t]}{\mathcal{E}[X]} < \underline{\mathcal{A}}(t))$ or $(\frac{\mathcal{E}[X_t]}{\mathcal{E}[X]} > \overline{\mathcal{A}}(t))$. We analyse these two events separately.

- (a) $\frac{\mathcal{E}[X_t]}{\mathcal{E}[X]} < \underline{\mathcal{A}}(t) = \frac{X_t - \epsilon}{X + \epsilon}$ implies $\mathcal{E}[X_t] < X_t - \epsilon$ or $\mathcal{E}[X] > X + \epsilon$, given the positivity of $X + \epsilon$. In turn, this implies $|\mathcal{E}[X_t] - X_t| > \epsilon$ or $|\mathcal{E}[X] - X| > \epsilon$.
- (b) $\frac{\mathcal{E}[X_t]}{\mathcal{E}[X]} > \overline{\mathcal{A}}(t)$ implies $X - \epsilon > 0$ and $\frac{\mathcal{E}[X_t]}{\mathcal{E}[X]} > \overline{\mathcal{A}}(t) = \frac{X_t + \epsilon}{X - \epsilon}$ (note that $X - \epsilon \leq 0$ would imply $\overline{\mathcal{A}}(t) = 1$ by definition of \mathcal{A} , but $\frac{\mathcal{E}[X_t]}{\mathcal{E}[X]} \leq 1$). Given the positivity of $X - \epsilon$, this in turn implies $\mathcal{E}[X_t] > X_t + \epsilon$ or $\mathcal{E}[X] < X - \epsilon$. In turn, this implies $|\mathcal{E}[X_t] - X_t| > \epsilon$ or $|\mathcal{E}[X] - X| > \epsilon$.

We have therefore proved that

$$F(t|S^*) \notin [\underline{\mathcal{A}}(t), \overline{\mathcal{A}}(t)] \text{ implies } (|\mathcal{E}[X_t] - X_t| > \epsilon \text{ or } |\mathcal{E}[X] - X| > \epsilon).$$

In terms of probability, by applying the bounds obtained in part 1 and 2 and a union bound, we obtain:

$$\begin{aligned} \Pr\left(F(t|S^*) \notin [\underline{\mathcal{A}}(t), \overline{\mathcal{A}}(t)]\right) &\leq \Pr(|\mathcal{E}[X_t] - X_t| > \epsilon) + \Pr(|\mathcal{E}[X] - X| > \epsilon) \\ &\leq \delta_1 + \delta_0 = \delta. \end{aligned}$$

From this inequality, the thesis for this part immediately follows. \square

Example 19 (simple ball/2). Consider the model defined in Example 18. For the additive noise Ψ , we have the range $\underline{h} = (-1, -1)$ and $\overline{h} = (1, 1)$, hence, in the notation discussed in this section, we can set $S = [-9.9, -6.9] \times [-15.3, -12.3]$ and $\mathcal{X}_0 = f^{-1}(S) \cap R_0$. A covering \mathcal{C} of \mathcal{X}_0 can be computed calling SIVIA($S, R_0, \overline{f}, \rho$), where we set the resolution to $\rho = 0.2 \times 5$, that is the 20% of the width of R_0 . After five bisections, we obtain a covering of \mathcal{X}_0 composed of four rectangles, specifically: $\mathcal{C} = \{[8.25, 8.875], [8.875, 9.5], [9.5, 10.75], [10.75, 11.375]\}$; we have $\mu_0(\cup \mathcal{C}) = 0.625$. Now, suppose we want to estimate $F(t|S^*)$ for $t = 8.8$. We set $\epsilon = 0.01$, $N = 15000$, and the N_i 's proportional to $\mu_0(R_i)$, getting $\delta_0 < 0.001$ hence $\delta < 0.002$. We run $\mathcal{A}(t)$. In step 1, we refine \mathcal{C} by splitting $[8.25, 8.875]$ into $[8.25, 8.8]$, $[8.8, 8.875]$, thus obtaining five rectangles R_1, \dots, R_5 . In step 2, we define the basic r.v.'s X_i . In step 3, we run the actual simulation, in which a value for X and one for X_t are computed. In detail, X will take on the value $\sum_{i=1}^5 r_i \cdot \mu_0(R_i)$, where r_i the fraction of the N_i i.i.d. samples (θ, ψ) with $\theta \in R_i$ that are mapped into S^* ; and X_t will take on the value $r_1 \cdot \mu_0(R_1)$ where $R_1 = [8.25, 8.8]$. In a specific simulation, we have found $X = 0.592$ and $X_t = 0.095$, so that, taking into account $\epsilon = 0.01$, the r.v. $\mathcal{A}(t) = [\underline{\mathcal{A}}(t), \overline{\mathcal{A}}(t)]$ takes on confidence interval $[0.141, 0.181]$.

Remark 14 (enhancements of \mathcal{A}). We outline two straightforward enhancements of the core algorithm \mathcal{A} .

1. It is sometimes possible to identify rectangles $R \in \mathcal{C}$ such that $g(R \times [h, \overline{h}]) \subseteq S^*$. In case $\mathcal{C} = \mathcal{C}_{\text{in}} \cup \mathcal{C}_{\text{out}}$ is a set of rectangles obtained with SIVIA, one can check the rectangles $R \in \mathcal{C}_{\text{in}}$ using an interval version of g , $\overline{g} := \overline{f} + [h, \overline{h}]$. In any case, let \mathcal{C}_0 be the set of identified rectangles that satisfy this property. Letting $v_0 := \mu_0(\cup \mathcal{C}_0)$ and $v_{0,t} := \mu_0(\cup \{R \in \mathcal{C}_0 : R \subseteq \mathbb{R}_{\leq t}^n\})$, one defines $\tilde{X} := v_0 + \sum_{R_i \in \mathcal{C} \setminus \mathcal{C}_0} \sum_{j=1}^N X_{ij}$ and $\tilde{X}_t := v_{0,t} + \sum_{i \in H_{t \text{ s.t. } R_i \in \mathcal{C} \setminus \mathcal{C}_0}} \sum_{j=1}^N X_{ij}$. A tighter interval confidence $\tilde{\mathcal{A}}(t)$ can then be obtained using \tilde{X}, \tilde{X}_t in place of X, X_t . The confidence $1 - \delta$ itself is modified accordingly, and gets sharper. We omit the rather obvious details.
2. Suppose one must compute $\mathcal{A}(t)$ for $t \in \{t_1, \dots, t_k\}$, rather than for a single point. By a slight modification of \mathcal{A} , it is possible to return confidence intervals for each of the $F(t_i|S^*)$ in a single run. The required modifications of the core algorithm \mathcal{A} are: first, the covering \mathcal{C} is ensured to refine all the $\mathbb{R}_{\leq t_i}^n$'s; second, in step 3, all the variables X_{t_1}, \dots, X_{t_k} are defined, and the corresponding k confidence intervals are computed accordingly. We denote by $\mathcal{A}(t_1, \dots, t_k)$ a call to this modified algorithm. By a union bound, the probability that $F(t_i|S^*) \in \mathcal{A}(t_i)$ for all $i = 1, \dots, k$ is at least $1 - k\delta$. This algorithm can be used to compute the confidence bands described in Subsection 5.2.1.

5.4 Confidence intervals for posterior moments

We concentrate on the first marginal of the posterior distribution, denoted by $F(\lambda|S^*)$ according to the notation introduced in Subsection 5.2.1. The same arguments, of course, apply to the other marginals. In Subsection 5.2.1, we have seen how to compute stepwise approximations of $F(\lambda|S^*)$, starting from confidence intervals for $F(\lambda|S^*)$ for a set of node points, say $\mathcal{A}(\lambda_1), \dots, \mathcal{A}(\lambda_k)$, see (5.2). The computed approximation forms a confidence band, (F^-, F^+) . Starting from this band, we can compute confidence intervals for various statistics of $F(\lambda|S^*)$. We first examine in detail the case of the expected value, $E[\Theta_1|S^*]$. In the result below, the hypothesis $\lambda_0 \geq 0$ can be removed by resorting to a slightly more complicated formula.

Corollary 3. *Let (F^-, F^+) be the confidence band defined in (5.2). Assume $\lambda_0 \geq 0$. Then $E[\Theta_1|S^*] \in [v^-, v^+]$ with probability at least $1 - (k - 1)\delta$, where:*

$$v^- := \lambda_0 + \sum_{j=0}^{k-1} (1 - F^+(\lambda_j|S^*))(\lambda_{j+1} - \lambda_j) \quad v^+ := \lambda_0 + \sum_{j=0}^{k-1} (1 - F^-(\lambda_j|S^*))(\lambda_{j+1} - \lambda_j).$$

PROOF. According to a well known formula [69], for a positively supported random variable Z with CDF F , one has $E[Z] = \int_0^{+\infty} (1 - F(z))dz$. Below, we apply this formula to $F(\lambda|S^*)$. We have

$$\begin{aligned} E[\Theta_1|S^*] &= \int_0^{+\infty} (1 - F(z|S^*))dz && \leq \int_0^{+\infty} (1 - F^-(z|S^*))dz \\ &= \int_0^{\lambda_0} dz + \int_{\lambda_0}^{\lambda_k} (1 - F^-(z|S^*))dz && = \lambda_0 + \sum_{j=0}^{k-1} \int_{\lambda_j}^{\lambda_{j+1}} (1 - F^-(z|S^*))dz \\ &= \lambda_0 + \sum_{j=0}^{k-1} \int_{\lambda_j}^{\lambda_{j+1}} (1 - F^-(\lambda_j|S^*))dz && = \lambda_0 + \sum_{j=0}^{k-1} (1 - F^-(\lambda_j|S^*))(\lambda_{j+1} - \lambda_j) \\ &= v^+ \end{aligned}$$

where, thanks to Proposition 6, the \leq in the first row above holds true with probability at least $1 - (k - 1)\delta$. The other inequality is proven similarly. \square

Example 20 (simple ball/3). Consider again the simple ball model introduced in Example 18. We build a confidence band (F^-, F^+) as specified in (5.2). We choose $k = 25$ evenly spaced points in R_0 : $\lambda_0 = 7, \lambda_1, \dots, \lambda_{25} = 12$. Then we compute $\mathcal{A}(\lambda_1), \dots, \mathcal{A}(\lambda_{24})$ as specified in Remark 14(2), this time setting SIVIA's resolution to $\rho = 0.1 \times 5$, and choosing ϵ, N in such a way that $(k - 1)\delta \leq 0.001$. The obtained F^- and F^+ are plotted in Figure 5.1, left. Relying on (F^-, F^+) , we apply Corollary 3, and obtain the confidence interval $[v^-, v^+] = [9.65, 9.85]$ for $E[\Theta|S^*]$.

A similar confidence interval can be obtained for the variance $\sigma^2 = \text{Var}[\Theta_1|S^*]$, relying on the formula $E[Z^2] = 2 \int_0^{+\infty} z(1 - F(z))dz$, which again holds for a nonnegative Z . One gets $\sigma^2 \in [\sigma^{2-}, \sigma^{2+}]$ with probability $\geq 1 - 2(k - 1)\delta$, where $\sigma^{2-} = 2[\lambda_0 + \sum_{j=0}^{k-1} \lambda_j(1 - F^+(\lambda_j|S^*))(\lambda_{j+1} - \lambda_j)] - (v^+)^2$ and $\sigma^{2+} = 2[\lambda_0 + \sum_{j=0}^{k-1} \lambda_{j+1}(1 - F^-(\lambda_j|S^*))(\lambda_{j+1} - \lambda_j)] - (v^-)^2$.

5.5 Optimal allocation of computational resources

Generally speaking, optimality in this section should be intended in the sense of algorithmic choices that minimize the expression of the Hoeffding bound. In the following analysis, we refer to the core algorithm of Section 5.3. Suppose that a covering \mathcal{C} of \mathcal{X}_0 is given. The next result says how to optimally allocate a budget of N samples among the K rectangles of \mathcal{C} , that is a strategy that minimizes the quantity δ_0 defined in Theorem 10(1), which bounds (up to a factor of 2) the probability of error of \mathcal{A} .

Theorem 11. *Let $\mathcal{C} = \{R_1, \dots, R_K\}$ be a covering of \mathcal{X}_0 , $A = \bigcup_{i=1}^K R_i$ with $\mu_0(A) > 0$, $N = N_1 + \dots + N_K$ ($N_i \geq 1$) and δ_0 as in Theorem 10(1). Then*

$$\delta_0 \geq 2 \exp\left(-\frac{2N\epsilon^2}{\mu_0(A)^2}\right). \quad (5.7)$$

Equality in (5.7) holds if $N_i = N \frac{\mu_0(R_i)}{\mu_0(A)}$ for $i = 1, \dots, K$.

PROOF. Recall that $\delta_0 = 2 \exp\left(-\frac{2\epsilon^2}{\sum_{i=1}^K \frac{\mu_0(R_i)^2}{N_i}}\right)$. Consider the denominator inside the exponential. We have

$$\begin{aligned} \sum_{i=1}^K \frac{\mu_0(R_i)^2}{N_i} &= N \sum_{i=1}^K \frac{N_i}{N} \left(\frac{\mu_0(R_i)}{N_i}\right)^2 \geq N \left(\sum_{i=1}^K \frac{N_i}{N} \frac{\mu_0(R_i)}{N_i}\right)^2 \\ &= N \left(\sum_{i=1}^K \frac{1}{N} \mu_0(R_i)\right)^2 = \frac{\mu_0(A)^2}{N} \end{aligned}$$

where in the second step we have applied Jensen's inequality to the convex function $\lambda \mapsto \lambda^2$. Now (5.7) is a direct consequence of the inequality just proved. Finally, by inspection, equality in (5.7) holds true under the stated condition. \square

With reference to the expression of δ_0 in Theorem 10, the preceding theorem elucidates two important facts. First, an optimal allocation of a budget of N samples can be obtained by drawing $N_i = N \frac{\mu_0(R_i)}{\mu_0(A)}$ samples from each rectangle R_i of the covering. As we have assumed we can sample efficiently from rectangles, we will adopt this strategy. Note that we will actually draw $N_i := \lceil N \frac{\mu_0(R_i)}{\mu_0(A)} \rceil$ samples per rectangle, leading to an actual number of samples slightly larger than the allocated budget of N , but still less than $N + K$. With the actual sampling strategy, we will still have

$$\delta_0 \leq 2 \exp\left(-\frac{2N\epsilon^2}{\mu_0(A)^2}\right). \quad (5.8)$$

Second, with the above optimal strategy, δ_0 will only depend on the volume of the set A that encloses \mathcal{X}_0 : hence coverings that yield tighter enclosures A of \mathcal{X}_0 should be preferred. Letting $\delta = 2\delta_0$ (cf. Theorem 10(3)) and holding ϵ fixed, the number $N = \frac{\ln(1/2\delta)}{2\epsilon^2} \mu_0(A)^2$ of samples necessary to guarantee a confidence level δ decreases quadratically as $\mu_0(A)$ decreases: see the plot in Figure 5.1. This part of the result also

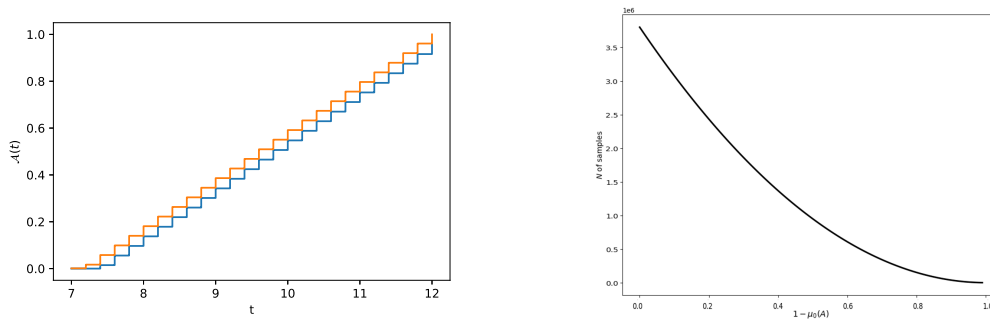


Figure 5.1: **Left:** confidence band ($\delta \leq 0.001$) for the posterior CDF of the simple ball model in Example 20. **Right:** number N of samples necessary to achieve $\delta = 0.001$ as a function of $1 - \mu_0(A)$. Here $\epsilon = 0.001$.

explains why a covering-based based algorithm is, typically, by far more convenient than a crude MC sampling from a rectangle R_0 containing \mathcal{X}_0 : switching from R_0 to A , the number of samples drops from $N_0 = \frac{\ln(1/2\delta)}{2\epsilon^2}$ (recall that $\mu_0(R_0) = 1$) to $N_0 \cdot \mu_0(A)^2$.

The above discussion on budget allocation presupposes that a covering \mathcal{C} is given. What if the cost of building \mathcal{C} must explicitly be taken into account? The costs of refinement and of simulation are not easily comparable on the same scale, mainly because, depending on the function f , computing $\bar{f}(R)$ can be much more expensive than drawing a sample θ and computing $f(\theta)$. A practical strategy might be to allocate a time budget for the construction of \mathcal{C} and stop the iterations of SIVIA as soon as this time expires, returning the current $\mathcal{C} = \mathcal{C}_{\text{in}} \cup \mathcal{C}_{\text{out}} \cup L$ as a covering. If the extraction policy in step 4 of SIVIA privileges rectangles with the largest width, this \mathcal{C} is, practically speaking, the best covering that can be obtained with the allocated time budget.

5.6 Experiments

We have put a proof-of-concept implementation⁶ of \mathcal{A} at work on a number of examples concerning ODES models and neural network classifiers.

5.6.1 Discretized ODES

We have put \mathcal{A} at work on the benchmarks⁷ in [52]. We describe our experimental setting with reference to the notation introduced in subsection 5.2.3. In all cases, we apply \mathcal{A} to an Euler-discretized version of the ODE. The timestep τ is chosen small enough to guarantee that, over the considered time horizon, the discretized solution is in very good agreement with the solution obtained via a traditional numerical

⁶Python code and examples available at https://github.com/Luisa-unifi/Posterior_estimates.

⁷With one exception, the ANALOG model, which has switching control features not easy to represent in our framework.

| Benchmark | n_z, n_θ | θ^* | $E_{\mathcal{A}}$ confidence interval | $t_{\mathcal{A}}$ | γ | E_{DSA} | t_{DSA} | E_{MCMC} | t_{MCMC} | E_{SMC} |
|-----------------|-----------------|--------------------|--|-------------------|--------------|--------------------|------------------|--------------------|-------------------|--------------------|
| FITZHUGH-NAGUMO | | | | | | | | | | |
| <i>model-1</i> | 2, 2 | 0.3 | $0.308 \pm 7.0e-03$ | 11.46 | 0.05 | 0.295 | 38 | 0.29 | 2011 | 0.29 |
| | | 0.15 | $0.153 \pm 6.7e-03$ | | | 0.16 | | 0.16 | | 0.16 |
| <i>model-2</i> | 2, 3 | 0.30 | $0.294 \pm 2.4e-02$ | 15.90 | 0.05 | 0.26 | 1077 | 0.26 | 11410 | 0.26 |
| | | 0.15 | $0.152 \pm 1.0e-03$ | | | 0.10 | | 0.09 | | 0.09 |
| | | 0.5 | $0.569 \pm 2.5e-02$ | | | 0.41 | | 0.39 | | 0.40 |
| LAUB-LOOMIS | | | | | | | | | | |
| <i>model-1</i> | 7, 3 | 1.8 | $1.810 \pm 2.8e-02$ | 16.81 | 0.05 | 1.83 | 2612 | 1.52 | 1282 | 1.92 |
| | | 0.8 | $0.797 \pm 2.3e-02$ | | | 0.80 | | 0.80 | | 0.80 |
| | | 0.3 | $0.297 \pm 2.3e-02$ | | | 0.34 | | 0.39 | | 0.33 |
| <i>model-2</i> | 7, 3 | 0.9 | $0.912 \pm 5.8e-03$ | 24.55 | 0.05 | 0.93 | 848 | 0.90 | 1251 | 0.89 |
| | | 0.8 | $0.799 \pm 2.3e-02$ | | | 0.77 | | 0.82 | | 0.82 |
| | | 0.3 | $0.297 \pm 2.3e-02$ | | | 0.27 | | 0.28 | | 0.28 |
| <i>model-3</i> | 7, 4 | 1.8 | $1.811 \pm 2.81e-02$ | 28.13 | 0.07 | 1.85 | 557 | 1.64 | 4008 | 1.39 |
| | | 0.8 | $0.797 \pm 2.3e-02$ | | | 0.79 | | 0.79 | | 0.78 |
| | | 0.3 | $0.298 \pm 2.3e-02$ | | | 0.29 | | 0.33 | | 0.50 |
| | | 2.5 | $2.556 \pm 2.9e-02$ | | | 2.49 | | 2.54 | | 2.58 |
| <i>model-4</i> | 7, 4 | 0.9 | $0.912 \pm 5.8e-03$ | 41.93 | 0.05 | 0.94 | 1794 | 0.94 | 3828 | 0.95 |
| | | 0.8 | $0.797 \pm 2.3e-02$ | | | 0.78 | | 0.80 | | 0.80 |
| | | 0.3 | $0.297 \pm 2.3e-02$ | | | 0.26 | | 0.28 | | 0.26 |
| | | 1.4 | $1.457 \pm 5.6e-03$ | | | 1.46 | | 1.47 | | 1.48 |
| <i>model-5</i> | 7, 5 | 0.9 | $0.90 \pm 5.7e-03$ | 118.25 | 0.05 | 0.89 | 3974 | 0.86 | 4213 | 0.87 |
| | | 0.8 | $0.839 \pm 4.0e-02$ | | | 0.78 | | 0.85 | | 0.82 |
| | | 0.3 | $0.339 \pm 4.0e-02$ | | | 0.29 | | 0.28 | | 0.29 |
| | | 2.5 | $2.594 \pm 4.6e-02$ | | | 2.63 | | 2.52 | | 2.59 |
| | | 1.3 | $1.297 \pm 2.3e-02$ | | | 1.27 | | 1.26 | | 1.27 |
| <i>model-6</i> | 7, 5 | 1.8 | $1.847 \pm 5.1e-02$ | 185.66 | 0.05 | 1.82 | 5239 | 1.62 | 3811 | 1.92 |
| | | 0.8 | $0.839 \pm 4.0e-02$ | | | 0.76 | | 0.77 | | 0.77 |
| | | 0.3 | $0.296 \pm 2.3e-02$ | | | 0.31 | | 0.37 | | 0.32 |
| | | 2.5 | $2.529 \pm 8.7e-03$ | | | 2.67 | | 2.66 | | 2.67 |
| | | 1.3 | $1.297 \pm 2.3e-02$ | | | 1.26 | | 1.29 | | 1.29 |
| <i>model-7</i> | 7, 6 | 0.9 | $0.909 \pm 5.79e-03$ | 386.70 | 0.05 | 0.89 | 75166 | 0.85 | 4189 | 0.84 |
| | | 0.8 | $0.797 \pm 2.3e-02$ | | | 0.78 | | 0.84 | | 0.87 |
| | | 0.3 | $0.297 \pm 2.3e-02$ | | | 0.34 | | 0.34 | | 0.34 |
| | | 2.5 | $2.509 \pm 6.4e-03$ | | | 2.68 | | 2.78 | | 2.67 |
| | | 1.3 | $1.308 \pm 2.3e-02$ | | | 1.28 | | 1.30 | | 1.3 |
| | | 1.8 | $1.799 \pm 2.8e-02$ | | | 1.99 | | 2.14 | | 2.04 |
| P53 | 6, 2 | 9.0e-04 9.9e-06 | $9.0e-04 \pm 5.2e-07$ $9.6e-06 \pm 4.9e-08$ | 50.28 | 0.04 0.04 | 8.9e-04 9.9e-06 | 111 | 8.9e-04 9.8e-06 | 15600 | 8.9e-04 9.8e-06 |
| ROSSLER | 3, 2 | 0.1 0.1 | $0.150 \pm 2.6e-02$ $0.170 \pm 6.7e-02$ | 16.66 | 0.05 0.05 | 0.12 0.09 | 34 | 0.11 0.09 | 1386 | 0.11 0.10 |
| GENETIC | 9, 2 | 50.0 50.0 | $50.013 \pm 1.1e-02$ $50.042 \pm 4.8e-01$ | 10.26 | 0.05 0.05 | 50 49.1 | 155 | 50 49.1 | 3120 | 50.0 49.0 |
| DALLA-MAN | 10, 2 | 0.0581 0.0871 | $0.060 \pm 1.31e-03$ $0.088 \pm 9.75e-04$ | 11.47 | 0.05 0.05 | 0.05 0.082 | 9119 | 0.055 0.081 | 24000 | 0.055 0.082 |

Table 5.1: Comparison of \mathcal{A} , DSA and MCMC on the benchmarks from [52]. **Legend.** n_z : number of state variables of the ODE; n_θ : number of parameters of the ODE; θ^* : values of θ used to generate the observed data; $E_{\mathcal{A}}$, **confidence interval**: confidence interval for the posterior expectations $E[\Theta_i | S^*]$ ($i = 1, \dots, n_\theta$), built via \mathcal{A} (Corollary 3); $t_{\mathcal{A}}$: execution time (s) of \mathcal{A} , including SIVIA; γ : half-side of S^* (tolerance) for \mathcal{A} ; E_{DSA} , E_{MCMC} , E_{SMC} : estimates of posterior expectations obtained via DSA, MCMC and SMC; t_{DSA} , t_{MCMC} : execution times (s) of DSA and MCMC.

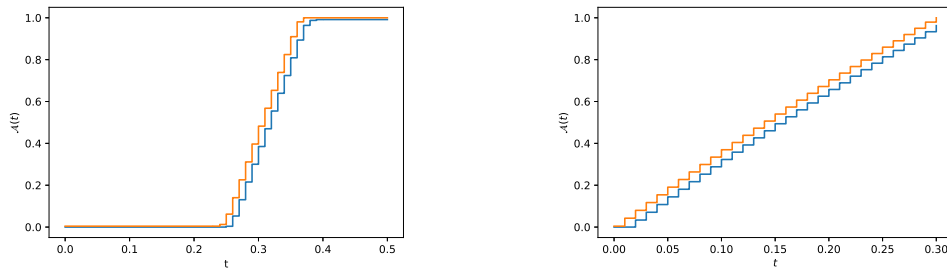


Figure 5.2: confidence bands ($\delta \leq 0.001$) for the marginal CDFs of the two parameter Fitzhugh-Nagumo ODE. **Left:** θ_1 . **Right:** θ_2 .

ODE integrator — specifically, Python’s `odeint()`. In all cases, the observed data y^* consists of a single measurement of the trajectory, taken at the end of the time horizon: multiple measurements do not bring any advantage in terms of accuracy, and introduce unnecessary computational burden. We consider an additive gaussian noise Ψ centered at the origin, with standard deviation as specified in [52, 53], but truncated at ≥ 5 standard deviations (only in one case, 3) to the left and to the right of the origin. The set S^* is a hypercube centered at y^* of side 2γ chosen experimentally, with small volumes ranging from 10^{-1} to 10^{-10} . Initial conditions and true parameter values are fixed as specified in [52, 53]. In all cases, the sampling budget N , the error threshold ϵ and the number $k + 1$ of node points in the construction of the confidence bands via \mathcal{A} (Proposition 6) are chosen so as to ensure a confidence $1 - \delta \geq 0.999$. We focus on computing confidence intervals for the posterior’s marginal expectations, as per Corollary 3. The obtained results are reported in Table 5.1. By way of example, in Fig. 5.2 we also report the confidence bands for the marginal CDFs of an instance of the Fitzhugh-Nagumo model.

For comparison, we have also reported the results of the DSA algorithm of [52] and of classical MCMC and SMC estimation; all these figures are taken from [52, Table 1]. In comparing these results, one should be aware of the differences between the theoretical and experimental settings here and in [52]. First, while considering non discretized ODEs, [52] relies on sensitivity analysis for the estimation of rectangle measures, which can be regarded as a form of — in general, unsound — discretization of the parameter space; on the contrary, we start from the outset with a discretized ODE model, and do not introduce further levels of discretization or unsoundness. Second, [52] provides an estimate of the posterior density given an individual observation y^* ; we consider the posterior probability distribution given an observation set S^* of small but positive measure. Third, the estimate intervals provided by [52] are certain — modulo the unsoundness discussed above; in our case, confidence intervals by definition introduce an extra level of aleatoric uncertainty, however small. The case of MCMC/SMC is still different, because no formal guarantee is provided. Despite these differences, we note that the estimates of the posteriors’ expected value produced by the three approaches are overall remarkably similar. The execution time of \mathcal{A} is up to *three orders of magnitude smaller* than DSA’s and MCMC’s⁸. A final caveat concerns the comparison with the true value θ^* used to generate the observation y^* . Although

⁸[52] does not report execution times for SMC, but we expect them to be in line with MCMC’s.

θ^* is often taken as a proxy of the exact posterior expectation $E[\Theta|S^*]$, one should remark that these two values need not coincide. Indeed, in a few experiments, it is observed that θ^* lies outside the returned confidence interval: this is not an indication that the computed interval is ‘wrong’.

5.6.2 Feature relevance in neural network classifiers

Our algorithm can be applied also to the quantification of feature relevance in neural network classifiers. We illustrate this methodology in the case of a classifier for images of the MNIST dataset [115].

Consider a classifier that maps items x consisting of p real valued features, say $x = (x_1, \dots, x_p)$, into one of s categories. Possibly after normalization of domain and range, without loss of generality one can regard such a classifier as a function $C : [0, 1]^p \rightarrow [0, 1]^s$, where $C = (C_1, \dots, C_s)$. Here it is understood that $x \in [0, 1]^p$ is classified as $i \in \{1, \dots, s\}$ if and only if $i = \operatorname{argmax}_{j=1, \dots, s} C_j(x)$ and $C_i(x) > l_0$, for a chosen threshold $1 > l_0 \geq 1/2$ — in particular, ties are ruled out. Assume a given $x^* = (x_1^*, \dots, x_p^*) \in [0, 1]^p$ is classified as i by C . One is often interested in assessing the relative importance of a certain feature, or set thereof, in obtaining such a classification: one speaks of *feature relevance*. Here we follow a recent proposal in [4]. Consider the k -th feature ($k \in \{1, \dots, p\}$) and the function $C_{i,k} : [0, 1] \rightarrow [0, 1]$ defined by

$$C_{i,k}(\theta) := C_i(x_1^*, \dots, x_{k-1}^*, \theta, x_{k+1}^*, \dots, x_p^*).$$

Let $\mu(\cdot)$ denote the uniform probability measure on $[0, 1]$. The relevance of the k -th feature in classifying x^* to i is defined as:

$$\eta_k(x^*) := 1 - \mu\left(C_{i,k}^{-1}([l_0, 1])\right).$$

Note that $\eta_k(x^*) \in [0, 1)$. According to [4], the value $\eta_k(x^*)$ reflects how sensitive the classification of x^* is to changes in the k -th feature, other features held constant: values closer to 1 indicate higher sensitivity.

We recast the problem of estimating $\eta_k(x^*)$ with guarantees in the framework of Subsection 5.2.1. In the notation of that section, we let $f = C_{i,k}$, hence let $n = m = 1$, and $\ell = 0$, that is no noise, which implies $f = g$ and $\mu = \mu_0$; moreover, we let $S^* = [l_0, 1]$. Then $\mu\left(C_{i,k}^{-1}([l_0, 1])\right) = \mu\left(f^{-1}(S^*)\right)$. We assume non noise both for ease of implementation, but also to be as close as possible to the configuration used in [4], where noise is not considered. Letting $R_0 = [0, 1]$, we define the variable X as in Algorithm 5 in Section 5.3. Then apply the first item of Theorem 10 to find that $\mu\left(C_{i,k}^{-1}([l_0, 1])\right) = E[X]$. The same theorem says that X approximates $E[X]$ with confidence $1 - \delta_0$. As a consequence, $1 - X$ approximates $\eta_k(x^*)$ with confidence $1 - \delta_0$.

We have applied the above outlined methodology to the estimation of the feature relevance for a classifier of the MNIST image dataset [115]. The considered classifier is implemented by a feedforward neural network with two hidden layers of respectively 30 and 10 neurons, using eLU (exponential LU) as an activation function for all layers. We use the Batch Gradient Descent as an optimization method with batch size of 10, and the dataset is shuffled for each epoch. The trained network exhibits an accuracy

of around 96% on a test subset of the MNIST dataset. For a few selected pictures from the dataset, we have computed the relevance of each of the $28 \times 28 = 784$ features (pixels) composing the image, applying the above methodology. Each pixel is represented by a grayscale value in the interval $[0, 1]$. This way, for each selected picture, we have obtained a 28×28 matrix of feature relevance values in $[0, 1]$. We have set $\delta_0 \leq 0.001$. The average computation time for each pixel is about 6 seconds. For a better visualization and interpretation of the results, a feature relevance matrix can be converted to a colormap, called *relevance map* in [4]: see Fig. 5.3. While, as expected, most pixels have relevance 0, there are a few clusters of highly relevant pixels, located approximately in the void zones of the original image.

These empirical results differ slightly from those reported in [4], where highly relevant features tend to reproduce the contours of the represented digit. This difference can be imputed to the experimental settings here and in [4] being not exactly comparable, as in [4] a neural network with only one hidden layer of neurons is considered. Despite this discrepancy, also in [4] certain pixels located at a significant distance from the contours appear to influence significantly the classification decision of the network (cf. [[4], Fig.10]). The authors of [4] leave a more in-depth discussion of this behaviour as future work, and we do the same here. Let us also point out that the shallow neural networks employed here and in [4] are not known to be particularly appropriate for the MNIST benchmark; but our aim here is just to show a potential future application of the proposed approach.

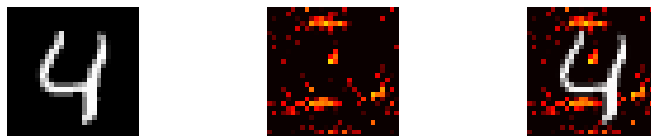


Figure 5.3: Example of relevance map. **Left:** original image from the MNIST dataset, correctly classified as ‘4’. **Center:** relevance map, with brighter colours representing more relevant features (black=0, white=1). **Right:** overlay of the first and second image. Clusters of highly relevant pixels tend to occupy void zones.

5.7 Conclusion

Assuming a functional relation between observations and parameters, we describe a method to estimate the Bayesian posterior parameters distribution, given that the observation belongs to a small set. Guarantees for the estimated a posteriori quantities, including CDFs and moments thereof, are given in the form of confidence bands or confidence intervals. We leverage IA to drastically reduce the computational cost of MC simulation. In terms of accuracy and execution time, the method compares very favourably to state-of-the-art techniques on benchmarks for noisy ODE parameter estimation. An application to relevance feature in neural networks has also been proposed.

As for future research, we would like to further investigate the scalability of the method, studying applications to more complex models and possibly to probabilistic

programming. Another direction is the relation with Approximate Bayesian Computation (ABC) [117] and Importance Sampling [151], which, among well-established techniques, appear to be closest in spirit to our approach.

Chapter 6

Inference, a more general view: Probabilistic Programming

6.1 Overview

Probabilistic Programming (PP) is a powerful programming paradigm, but also an efficient statistical modeling tool that enables inference to be performed automatically. In the field of Probabilistic Programming, the denotational approach dating back to Kozen’s work is well established, see [111, 85, 15] and references therein. Equivalent operational approaches based on sequential composition and/or ‘big step’ semantics also have been considered, see e.g. [137]. Here we depart from this tradition and consider an action-based probabilistic programming language with a ‘small-step’ operational semantics: we want to argue that this brings benefits both in terms of clarity of presentation and in pragmatical terms of effective implementation. The language we consider (Section 6.3) is a calculus à la Milner’s CCS [127], where an action can be either a random sampling or an assignment. Continuous and discrete distributions can be freely mixed, conditionals and unbounded loops are allowed. In order to account for conditioning/observation, we also consider a fail statement indicating that the current computation must be rejected.

The small-step operational semantics is formalized in terms of a (self) *product* of a Markov kernel (Sections 6.2, 6.3), and directly leads to a sampling method, as we discuss below. On top of the operational semantics, an observational semantics (Section 6.4) is introduced, based on a probability space of infinite sequences of states. To this purpose, a standard cylindrical sigma-field construction [12, Ch.2] is leveraged. In fact, we are more flexible than this, and define the observational semantics in terms of expectation of measurable functions over the considered probability space, conditioned on non-failure. The main result here is an approximation theorem, which provides lower and upper bounds of the exact semantics (expectation), based on the semantics of the program truncated at a chosen finite execution length t : the finite semantics is effectively computable, or at least relatively easy to estimate with guarantees.

Indeed, one of the benefits and motivations of our approach is that the small-step semantics directly translates into an *exact* sampling algorithm, that can be efficiently parallelized (Section 6.5). The returned independent samples can be used for *Monte*

Carlo (MC) inference, with formal statistical guarantees, expressed in terms of confidence intervals computed via exponential tail inequalities, such as Hoeffding’s [96]. Specifically, we exploit a form of *Single Instruction Multiple Data (SIMD)* parallelism that arises naturally from the definition of our Markov kernel: the basic idea is that, on each transition, the kernel is applied to a whole vector of independent store-program pairs at once. Here SIMD should be contrasted with *MIMD* (M=Multiple), a form of thread-level parallelism that is a commonplace in MC methods¹. Being able to exploit SIMD parallelism is practically relevant: indeed, modern CPUs and programming languages offer extensive support for massive SIMD parallelism in terms of *vectorization*, that is instructions operating on whole arrays at once, that can result in huge gains in performance. If desired, the proposed sampling scheme can be effectively combined with Importance Sampling (IS) to alleviate the problem of rejections, arising in the presence of conditioning/fail actions.

Preliminary experiments conducted with a proof-of-concept implementation based on TensorFlow [1], and comparison with some state-of-the-art tools, show promising results (Section 6.6). Some concluding remarks are drawn in the final section (Section 6.7).

All in all, we can summarize our two main contributions as follows: (1) A simple yet rigorous, measure-theoretic small-step operational semantics, that directly translates into an effective SIMD-parallelisable inference algorithm. (2) A clean observational semantics and a finite approximation theorem, which provides a formal basis for MC estimation with guarantees.

Related work Book [15] provides an introduction to and a survey of recent literature on PP; see also the review article [85]. With few notable exceptions that we will review below, most work on the semantics of probabilistic programs still follows the denotational approach initiated by Kozen [111]. This includes, among others, the work of Borgström, Gordon et al. see e.g. [20], and the work of Staton see e.g. [174, 175], who consider a measure-theoretic and/or categorical point of view. In this line of work emphasis is, for instance, on providing conditions under which a *density* for a program-induced random variable exists. We do not consider such aspects in our framework, as a (cumulative) *distribution*, which always exists, is all that is needed.

As mentioned earlier, a few works depart from the traditional denotational setting, and are closer in spirit to ours. The work of Aditya et al. on Markov Chain Monte Carlo (MCMC) for R2 [137] considers a big-step sampling semantics. It is unclear how a big-step semantics would translate into a SIMD-parallel algorithm. Also, no approximation results in terms of finite execution paths is provided. In [45] the authors apply program analysis to IS. Here we too consider IS, but with a complementary concern: integrating IS into a clean measure-theoretic semantics, rather than devising syntax-driven transformations, as they do.

The work of Jasen et al. on model checking PP [99] also follows an operational approach. The main difference from us is that [99] only considers discrete distributions, for which techniques of discrete-space Markov chains [13] apply; they also consider nondeterminism, which is not a concern in our work. Similar remarks apply to Gretz et al. [87], who adopt a nondeterministic Dijkstra guarded command language extended

¹E.g. when multiple threads execute independent chains in a Markov Chain Monte Carlo method.

with a discrete probabilistic choice (pGCL), basically corresponding to sampling from Bernoulli distributions. Again, the distributions definable by the programs are discrete, and discrete-space Markov Decision Processes (MDPs) are sufficient for operational modeling of the programs. They propose reasoning techniques based on probabilistic versions of weakest preconditions and invariants: these techniques allow in some cases to compute expected values of programs exactly, but they are not as generally applicable as the guaranteed MC estimation, based on finite approximation, we consider here.

Several works describe tools and algorithms for inference in PP, based on MCMC or variations thereof, like R2 [137] or WebPPL [83], among the many. In Section 6.6 we compare our sampling algorithm with these tools mainly in terms of efficiency. Not being MCMC an exact sampling technique (as the drawn samples are not truly independent), these algorithms cannot offer formal guarantees of accuracy. On the side of formal guarantees, symbolic tools like Hakaru [134] and Psi-solver [77] are based on formal manipulations of integrals, that can return explicit, exact answers in a limited set of cases. More similar in spirit to ours is the work of Sankaranarayanan et al. [165], who provide formal guarantees by only analyzing a subset of adequately chosen execution paths of a program. The analysed programs are loop-free queries. We leave an experimental comparison with [165] for future work.

It is common for programming languages, including the above mentioned webPPL and R2, to provide programmers with vectorization primitives. This language-level vectorization should not be confused with leveraging vectorization at the level of the *inference algorithm*, which is our concern here. Indeed, on nontrivial probabilistic programs and inference tasks, the difference in terms of execution time and accuracy between our approach and others shows up clearly, sometimes dramatically; see the experiments in Section 6.6.

6.2 Preliminaries

We review a few basic concepts of measure theory following closely the presentation in the first two chapters of [12], which is a reference for whatever is not explicitly described below. Given a nonempty set Ω , a *sigma-field* \mathcal{F} on Ω is a collection of subsets of Ω that contains Ω , and is closed under complement and under countable disjoint union. The pair (Ω, \mathcal{F}) is called a *measurable space*. A (total) function $f : \Omega_1 \rightarrow \Omega_2$ is *measurable* w.r.t. the sigma-fields $(\Omega_1, \mathcal{F}_1)$ and $(\Omega_2, \mathcal{F}_2)$ if whenever $A \in \mathcal{F}_2$ then $f^{-1}(A) \in \mathcal{F}_1$. We let $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, +\infty\}$ be the set of extended reals, assuming the standard arithmetic for $\pm\infty$ (cf. [12, Sect.1.5.2]), and $\overline{\mathbb{R}}^+$ the set of nonnegative reals including $+\infty$. The *Borel sigma-field* \mathcal{F} on $\Omega = \overline{\mathbb{R}}^m$ is the minimal sigma-field that contains all rectangles of the form $[a_1, b_1] \times \cdots \times [a_n, b_n]$, with $a_i, b_i \in \overline{\mathbb{R}}$. We shall mostly work with measurable spaces (Ω, \mathcal{F}) where $\Omega = \overline{\mathbb{R}}^m$ for some $m \geq 1$ and \mathcal{F} is the Borel sigma-field over Ω . Throughout this chapter, “*measurable*” means “*Borel measurable*”, both for sets and for functions. In particular, function f is Borel measurable if the preimage of any Borel set under f is a Borel set. On functions, Borel measurability is preserved by composition and other elementary operations on functions; continuous real functions are Borel measurable.

A *measure* over a measurable space (Ω, \mathcal{F}) is a function $\mu : \mathcal{F} \rightarrow \overline{\mathbb{R}}^+$ that is countably additive, that is $\mu(\cup_{j \geq 1} A_j) = \sum_{j \geq 1} \mu(A_j)$ whenever A_j ’s are pairwise

disjoint sets in \mathcal{F} . The *Lebesgue integral* of a Borel measurable function f w.r.t. a measure μ [12, Ch.1.5], both defined over a measurable space (Ω, \mathcal{F}) , is denoted by $\int_{\Omega} \mu(d\omega) f(\omega)$, with the subscript Ω omitted when clear from the context. When μ is the standard Lebesgue measure, we may omit μ and write the integral as $\int_{\Omega} d\omega f(\omega)$. For $A \in \mathcal{F}$, $\int_A \mu(d\omega) f(\omega)$ denotes $\int_{\Omega} \mu(d\omega) f(\omega) 1_A(\omega)$, where $1_A(\cdot)$ is the indicator function of the set A . We let δ_v denote Dirac's measure concentrated on v : for each set A in an appropriate sigma-field, $\delta_v(A) = 1$ if $v \in A$, $\delta_v(A) = 0$ otherwise. Otherwise said, $\delta_v(A) = 1_A(v)$. Another measure that arises (in connections with discrete distributions) is the counting measure, $\mu_C(A) := |A|$. In particular, for a nonnegative f , we have the equality $\int_A \mu_C(d\omega) f(\omega) = \sum_{\omega \in A} f(\omega)$.

Let h be a nonnegative measurable function and defined on (Ω, \mathcal{F}) , and μ be a measure on \mathcal{F} . The function $\nu(A) := \int_A \mu(d\omega) h(\omega)$ ($A \in \mathcal{F}$) defines a new measure on \mathcal{F} . The function h is called a *density* of ν w.r.t. μ . A basic fact about densities is that, for each measurable f , $\int \nu(d\omega) f(\omega) = \int \mu(d\omega) h(\omega) \cdot f(\omega)$, in the sense that if one of the integrals exists so does the other, and the two are equal: this is called *chain rule* for densities; see [12, Ch.2.2,Pr.4].

A *probability measure* is a measure μ defined on Ω such that $\int \mu(d\omega) = 1$. For a given nonnegative measurable function f defined over Ω , its *expectation* w.r.t. a probability measure ν is just its integral: $E_{\nu}[f] = \int \nu(d\omega) f(\omega)$. The following definition is central.

Definition 12 (Markov kernel). Let $(\Omega_1, \mathcal{F}_1)$ and $(\Omega_2, \mathcal{F}_2)$ be measurable spaces. A function $K : \Omega_1 \times \mathcal{F}_2 \rightarrow \overline{\mathbb{R}}^+$ is a Markov kernel from Ω_1 to Ω_2 if it satisfies the following properties:

1. for each $\omega \in \Omega_1$, the function $K(\omega, \cdot) : \mathcal{F}_2 \rightarrow \overline{\mathbb{R}}^+$ is a probability measure on $(\Omega_2, \mathcal{F}_2)$;
2. for each $A \in \mathcal{F}_2$, the function $K(\cdot, A) : \Omega_1 \rightarrow \overline{\mathbb{R}}^+$ is measurable.

We will mostly be concerned with the case $\Omega_1 = \Omega_2$, $\mathcal{F}_1 = \mathcal{F}_2$. The following is a standard result about the construction of finite product of measures over a space $\Omega^t = \Omega \times \cdots \times \Omega$ (t times) for $t \geq 1$ an integer. The formulation below is a specialization of [12, Th.2.6.7] to Markov kernels and nonnegative functions. In particular, part (a) gives a way to construct a measure on the product space Ω^t , starting from an initial measure μ^1 and $t - 1$ Markov kernels. The product space is, intuitively, the sample space of the paths of length t of a Markov chain. In particular, a path of length $t = 1$ consists of just an initial state — no transition has been fired. Part (b) is a generalization of Fubini theorem, which allows one to express an integral over the product space w.r.t. the measure of part (a) in terms of iterated integrals over the component spaces. Below, we will let ω^t range over Ω^t .

Theorem 13 (product of measures). Let $t \geq 1$ be an integer. Let μ^1 be a probability measure on Ω and K_2, \dots, K_t be $t - 1$ (not necessarily distinct) Markov kernels from Ω to Ω .

- (a) There is a unique probability measure μ^t defined on $(\Omega^t, \mathcal{F}^t)$ such that for every $A_1 \times \cdots \times A_t \in \mathcal{F}^t$ we have:

$$\mu^t(A_1 \times \cdots \times A_t) = \int_{A_1} \mu^1(d\omega_1) \int_{A_2} K_2(\omega_1)(d\omega_2) \cdots \int_{A_t} K_t(\omega_{t-1})(d\omega_t). \quad (6.1)$$

(b) (Fubini) Let f be a nonnegative measurable function defined on Ω^t . Then, letting $\omega^t = (\omega_1, \dots, \omega_t)$, we have

$$\int \mu^t(\omega^t) f(\omega^t) = \int \mu^1(d\omega_1) \int K_2(\omega_1)(d\omega_2) \cdots \int K_t(\omega_{t-1})(d\omega_t) f(\omega^t). \quad (6.2)$$

In particular, on the right-hand side, for each $j = 1, \dots, t-1$ and $(\omega_1, \dots, \omega_{j-1})$, the function $\omega_j \mapsto \int K_{j+1}(\omega_j)(d\omega_{j+1}) \cdots \int K_t(\omega_{t-1})(d\omega_t) f(\omega^t)$ is measurable over Ω .

It is customary to denote the measure μ^t defined by part (a) of the theorem also as $\mu^1 \otimes K_2 \otimes \cdots \otimes K_t$.

6.3 Probabilistic programs

When writing programs, one will have to rely on a repertoire of basic distributions and functions. We introduce the necessary terminology below. Then we introduce the syntax and operational semantics of the language, given in terms of a Markov kernel.

Basic elements Both continuous and discrete basic distributions will be considered. Each basic distribution is assumed to admit a density w.r.t. a suitable measure. This enables efficient sampling, but does not imply that measures corresponding to whole programs (cf. Section 6.4) have a density. A crucial point for the expressiveness of the language is that a basic density may depend on *parameters*, whose value at runtime is determined by the state of the program. To ensure that the resulting programs define measurable functions, it is important that the dependence between the parameters and the density be in turn of measurable type. We formalize this with the concept of parametric density, introduced below. In the definition, $v \in \overline{\mathbb{R}}^m$ represent the parameters, and $G(v, \cdot)$ a density over the reals, determined by the parameters v .

Definition 14 (parametric density). Let $G : \overline{\mathbb{R}}^m \times \overline{\mathbb{R}} \rightarrow \overline{\mathbb{R}}^+$ be a function. Assume G is measurable over $\overline{\mathbb{R}}^{m+1}$. We say G is a parametric density w.r.t. the measure μ_G over $\overline{\mathbb{R}}$ if the function $(v, A) \mapsto \int_A \mu_G(dr) G(v, r)$ is a Markov kernel from $\overline{\mathbb{R}}^m$ to $\overline{\mathbb{R}}$.

Example 21. Consider $\rho_U(v, r) = 1_{(0,1)}(r)$ (the argument v is ignored): this is the density of the uniform distribution on $(0, 1)$ w.r.t. the Lebesgue measure. For $h = (h_1, h_2)$ measurable functions, with h_2 positive, consider $\rho_{G,h}(v, r) := \frac{1}{h_2(v)\sqrt{2\pi}} \exp\left(-\frac{1}{2} \left(\frac{r-h_1(v)}{h_2(v)}\right)^2\right)$: this is a Normal (Gaussian) distribution of mean $h_1(v)$ and standard deviation $h_2(v)$ w.r.t. the Lebesgue measure. For an example of discrete distribution, consider a measurable function h which takes values in $[0, 1]$; then $\rho_{B,h}(v, r) = (1-h(v)) \cdot 1_{\{0\}}(r) + h(v) \cdot 1_{\{1\}}(r)$ is the density of a Bernoulli distribution with success probability $h(v)$ w.r.t. the counting measure.

Let $m \geq 1$ be fixed throughout the following definitions: m will represent the number of variables in the program. For the actual syntax of our language, we fix three sets of functions.

- *Update functions*: a countable set of measurable functions $g : \overline{\mathbb{R}}^m \rightarrow \overline{\mathbb{R}}$.

- *Parametric densities*: a countable set of parametric densities $G : \overline{\mathbb{R}}^m \times \overline{\mathbb{R}} \rightarrow \overline{\mathbb{R}}^+$. For simplicity, we will stick to the case where μ_G is either the standard Lebesgue measure (for continuous distributions) or the counting measure (for discrete ones).
- *Predicates*: a countable set of measurable functions $\phi : \overline{\mathbb{R}}^m \rightarrow \{0, 1\}$. By $\overline{\phi}$ we denote the predicate $1 - \phi$. An Iverson bracket style notation will be often employed, e.g.: $[x_1 \geq 1]$ is the predicate that on input v yields 1 if $v_1 \geq 1$, 0 otherwise.

Syntax Fix a tuple of m distinct variables (symbols), $x = (x_1, \dots, x_m)$. An *action* α is either an assignment or a random extraction²

$$\alpha ::= x_i = g \mid x_i \sim G \quad (6.3)$$

for any $1 \leq i \leq m$. Moreover, we fix a countable set of statement variables ranged over by Y, Y', \dots . The syntax of *program statements* S, S', \dots is given by the following grammar:

$$S ::= Y \mid \text{nil} \mid \text{fail} \mid \alpha.S \mid \text{if } \phi \text{ then } S_1 \text{ else } S_2 \mid \text{rec } Y.S \quad (6.4)$$

with the convention that $\text{rec } Y$ binds Y in the last clause above, and that different occurrences of rec bind different variables. We shall only consider *closed* program statements with *guarded* recursion, a standard restriction in process calculi [127]: in each recursive subterm $\text{rec } Y.S$, all occurrences of Y in S are within the scope of some action $\alpha.(\cdot)$ operator. We let \mathcal{P} denote the set of closed and guarded program statements. Note that \mathcal{P} is, by construction, a countable set. In this syntax, nil stands for (correct) termination and fail for failure. The other clauses are self-explanatory. We shall use the following abbreviation

$$\text{obs}(\phi).S := \text{if } \phi \text{ then } S \text{ else fail}.$$

Example 22 (random walk). The following program S models a random walk via a an unbounded loop. The initial value of variables, (i, r, y) in this example, is always assumed to be zero. First, r is drawn at random³ in $(0, 1)$. Then, at each iteration y is updated according to a Normal distribution ρ_G of mean y and s.d. $2r$, until $|y| \geq 1$.

$$S = r \sim \rho_U. \text{rec } Y. (\text{if } |y| < 1 \text{ then } (y \sim \rho_G(y, 2 \cdot r).i = i + 1.Y) \text{ else nil}) . \quad (6.5)$$

As a variation of the above, one might be interested in observing only those computations that terminate in at least 3 iterations:

$$S' = r \sim \rho_U. \text{rec } Y. (\text{if } |y| < 1 \text{ then } (y \sim \rho_G(y, 2 \cdot r).i = i + 1.Y) \text{ else obs}(i \geq 3)) . \quad (6.6)$$

²Variables x_i are only introduced as syntactic sugar: they will not get instantiated, and might be dispensed with using a terser notation, like $@i = g$ and $@i \sim G$.

³Technically, Definition 14 requires that we specify a density $\rho_G((c, 2r), \cdot)$ also when $r \leq 0$: this can be fixed arbitrarily for our purposes.

Operational semantics The operational semantics of \mathcal{P} will be given by a Markov kernel $\mathbb{K}(\cdot, \cdot)$. Some additional notation is in order. First, we assume a bijection between \mathcal{P} and \mathbb{N} , so that without loss of generality we can regard \mathcal{P} as a subset of \mathbb{R} . In particular, we will consider a set of *state-program pairs* $\overline{\mathbb{R}}^m \times \mathcal{P} \subseteq \overline{\mathbb{R}}^{m+1}$. Henceforth, we fix our state space and sigma-field as follows:

$$\Omega := \overline{\mathbb{R}}^{m+1} \quad \mathcal{F} := \text{Borel sigma-field over } \overline{\mathbb{R}}^{m+1}$$

while reserving the symbol \mathcal{F}_k for the Borel sigma-field over $\overline{\mathbb{R}}^k$, for any $k \geq 1$. For any $S \in \mathcal{P}$ and $A \in \mathcal{F}$, we let $A_S := \{v \in \overline{\mathbb{R}}^m : (v, S) \in A\}$ be the *section* of A at S . Note that $A_S \in \mathcal{F}_m$, as sections of measurable sets are measurable, see [12, Th.2.6.2, proof(1)].

For $v = (v_1, \dots, v_m) \in \overline{\mathbb{R}}^m$, $r \in \overline{\mathbb{R}}$ and $1 \leq i \leq m$, we let $v[r_{\#i}] := (v_1, \dots, r, \dots, v_m)$ denote the tuple where v_i has been replaced by r . Moreover, we let $v_{\cdot i} \in \overline{\mathbb{R}}^{m-1}$ be the vector obtained from v by removing its i -th component, and let $A_{(v_{\cdot i}, S)} = \{r \in \overline{\mathbb{R}} : (v[r_{\#i}], S) \in A\} \subseteq \overline{\mathbb{R}}$ be the section of A at $(v_{\cdot i}, S)$, which is a measurable set in \mathcal{F}_1 .

Now we define a function $\mathbb{K} : \Omega \times \mathcal{F} \rightarrow \overline{\mathbb{R}}^+$. Due to the presence of *rec*, a plain inductive definition on the structure of S would not go through. Nevertheless, it is still possible to give a syntax-driven definition, as follows. The syntactic *depth* of S , written $\text{dp}(S)$, is inductively defined by plain induction on S , as expected: $\text{dp}(Y) = \text{dp}(\text{nil}) = \text{dp}(\text{fail}) = 0$, $\text{dp}(\alpha.S) = 1 + \text{dp}(S)$, $\text{dp}(\text{if } \phi \text{ then } S_1 \text{ else } S_2) = 1 + \max\{\text{dp}(S_1), \text{dp}(S_2)\}$, and $\text{dp}(\text{rec } Y.S) = 1 + \text{dp}(S)$. Let the *unfolding* of S , written $\text{u}(S)$, be the term obtained from S by replacing each subterm $\text{rec } Y.S'$ that is *not* in the scope of a $\text{rec}(\cdot)$ or $\alpha(\cdot)$, by $S'[\text{rec } Y.S'/Y]$. As an example, if $S = \text{rec } Y.\text{rec } Y'. \text{if } \phi \text{ then } \alpha.Y \text{ else } \alpha'.Y'$ then

$$\text{u}(S) = \text{rec } Y'. \text{if } \phi \text{ then } \alpha.S \text{ else } \alpha'.Y'$$

We call S *stable* if $S = \text{u}(S)$. Define the k -th unfolding of S , $\text{u}^{(k)}(S)$, for $k \geq 0$, by induction on k as expected. We let the *stability depth* be $\text{sdp}(S) := \min\{k \geq 0 : \text{u}^{(k)}(S) \text{ is stable}\}$. This parameter is well defined and finite for programs $S \in \mathcal{P}$. In particular, if Y occurs in S , then $\text{sdp}(S[\text{rec } Y.S/Y]) < \text{sdp}(\text{rec } Y.S)$ for each term $\text{rec } Y.S \in \mathcal{P}$. In what follows, we shall refer to the induction on the pairs $\text{p}(S) := (\text{sdp}(S), \text{dp}(S))$ ordered lexicographically as to *structural induction on S* . Note that in particular $\text{p}(S[\text{rec } Y.S/Y]) < \text{p}(\text{rec } Y.S)$. Below, we shall write $\mathbb{K}(\omega, A)$ as $\mathbb{K}(\omega)(A)$, for any $A \in \mathcal{F}$.

Definition 15 (Markov kernel over Ω). We let $\mathbb{K} : \Omega \times \mathcal{F} \rightarrow \overline{\mathbb{R}}^+$ be the function defined by structural induction on S as follows. Below, $v \in \overline{\mathbb{R}}^m$, $r \in \overline{\mathbb{R}}$ and $A \in \mathcal{F}$.

$$\begin{aligned} \mathbb{K}(\omega)(A) &= \delta_\omega(A) \quad (\omega \notin \overline{\mathbb{R}}^m \times \mathcal{P}) \\ \mathbb{K}(v, \text{nil})(A) &= \delta_{(v, \text{nil})}(A) \\ \mathbb{K}(v, \text{fail})(A) &= \delta_{(v, \text{fail})}(A) \\ \mathbb{K}(v, x_i = g.S)(A) &= \delta_{(v[g(v_{\#i}], S)}(A) \\ \mathbb{K}(v, x_i \sim G.S)(A) &= \int_{A_{(v_{\cdot i}, S)}} \mu_G(dr) G(v, r) \\ \mathbb{K}(v, \text{if } \phi \text{ then } S_1 \text{ else } S_2)(A) &= \phi(v)\mathbb{K}(v, S_1)(A) + \bar{\phi}(v)\mathbb{K}(v, S_2)(A) \\ \mathbb{K}(v, \text{rec } Y.S)(A) &= \mathbb{K}(v, S[\text{rec } Y.S/Y])(A). \end{aligned}$$

Note that the first clause ensures that $\kappa(\cdot)(A)$ is defined on all elements of Ω , even those that do not represent valid state-program pairs.

Lemma 18. *The function \mathbb{K} is a Markov kernel from Ω to Ω .*

The proof that \mathbb{K} is a Markov kernel is a bit laborious but not difficult. We show separately that for each fixed $\omega \in \Omega$, the function $A \mapsto \mathbb{K}(\omega)(A)$ is a probability measure on \mathcal{F} , and that for each fixed $A \in \mathcal{F}$, the function $\omega \mapsto \mathbb{K}(\omega)(A)$ is measurable on Ω . This allow us to conclude, relying on some basic facts from measure theory (e.g. continuity implies measurability).

Example 23. Consider the program S (random walk) in (6.5). Here we have $m = 3$ variables, (i, r, y) . Assume for simplicity that $A = R \times \mathcal{P}$, for some rectangle $R = [a_1, b_1] \times [a_2, b_2] \times [a_3, b_3] \subseteq [0, 1]^3$. Also call S_1, S_2, S_3 the continuations of S after the first, second and third action, respectively. Let $\rho_G((c, s), r)$ be the Normal parametric density of mean c and standard deviation s , and assume $s \in (0, 1)$ below. Then one can check:

$$\begin{aligned} \mathbb{K}(((0, 0, 0), S)(A) &= \delta_1([a_1, b_1]) \\ \mathbb{K}(((1, 0, 0), S_1)(A) &= b_2 - a_2 \\ \mathbb{K}(((1, s, 0), S_2)(A) &= \int_{a_3}^{b_3} \rho_G((0, 2s), r) dr. \end{aligned}$$

6.4 Observable semantics

For any $t \geq 1$, we call Ω^t the set of *paths of length t* . Consider now the set of paths of infinite length, Ω^∞ , that is the set of infinite sequences $\tilde{\omega} = (\omega_1, \omega_2, \dots)$ with $\omega_i \in \overline{\mathbb{R}}^{m+1}$. For any $\omega^t \in \Omega^t$ and $\tilde{\omega} \in \Omega^\infty$, we identify the pair $(\omega^t, \tilde{\omega})$ with the element of Ω^∞ in which the prefix ω^t is followed by $\tilde{\omega}$. For $t \geq 1$ and a measurable $B^t \subseteq \Omega^t$, we let $B_t := B^t \times \Omega^\infty \subseteq \Omega^\infty$ be the *measurable cylinder* generated by B^t . We let \mathcal{C} be the minimal sigma-field over Ω^∞ generated by all measurable cylinders. Under the same assumptions of Theorem 13 on the measure μ^1 and on the kernels K_2, K_3, \dots there exists a unique measure μ^∞ on \mathcal{C} such that for each $t \geq 1$ and each measurable cylinder B_t , it holds that $\mu^\infty(B_t) = \mu^t(B^t)$: see [12, Th.2.7.2], also known as the *Ionescu-Tulcea theorem*. In the definition below, we let $0 = (0, \dots, 0)$ (m times) and consider $\delta_{(0,S)}$, the Dirac's measure on Ω that concentrates all the probability mass in $(0, S)$.

Definition 16 (probability measure induced by S). *Let $S \in \mathcal{P}$. For each integer $t \geq 1$, we let μ_S^t be the probability measure over Ω^t uniquely defined by Theorem 13(a) by letting $\mu^1 = \delta_{(0,S)}$ and $K_2 = \dots = K_t = \mathbb{K}$. We let μ_S^∞ be the unique probability measure on \mathcal{C} induced by μ_1 and $K_2 = \dots = K_t = \dots = \mathbb{K}$, as determined by the Tulcea-Ionescu theorem.*

In other words, $\mu_S^t = \delta_{(0,S)} \otimes \mathbb{K} \otimes \dots \otimes \mathbb{K}$ ($t - 1$ times \mathbb{K}). By convention, if $t = 1$, $\mu_S^t = \delta_{(0,S)}$. The measure μ_S^∞ can be informally interpreted as the limit of the measures μ_S^t and represents the semantics of S . We will be interested in the following events of Ω^t and Ω^∞ , for $t \geq 1$. We start from the following three sets that form a partition of Ω : $L := \overline{\mathbb{R}}^m \times (\overline{\mathbb{R}} \setminus \{\text{nil}, \text{fail}\})$, $T := \overline{\mathbb{R}}^m \times \{\text{nil}\}$, $F := \overline{\mathbb{R}}^m \times \{\text{fail}\}$.

- $F_t := L^{t-1} \times F \times \Omega^\infty$, the paths that *fail* at time t ;
- $T_t := L^{t-1} \times T \times \Omega^\infty$, the paths that *terminate* at time t ;
- $F_\infty := \bigcup_{t \geq 1} F_t$, the paths that *eventually fail*;
- $T_\infty := \bigcup_{t \geq 1} T_t$, the paths that *eventually terminate*.

We are interested in evaluating a program *only on non failed paths*: that is, we will work by conditioning on the event $(F_\infty)^c$. Recall that the *support* of an (extended) real valued function f is the set $\text{supp}(f) := \{z : f(z) \neq 0\}$. In what follows, we shall concentrate on nonnegative measurable functions f to avoid unnecessary complications with the existence of integrals. The general case can be dealt with by the usual trick of decomposing f as $f = f^+ - f^-$, where for each z one defines $f^+(z) := \max(0, f(z))$ and $f^-(z) := -\min(0, f(z))$, and then dealing separately with f^+ and f^- .

Definition 17 (observable semantics). *Let f be a nonnegative measurable function defined on Ω^∞ . We let the unnormalized semantics of S and f be $[S]f := E_{\mu_S^\infty}[f]$ ($= \int \mu_S^\infty(d\tilde{\omega})f(\tilde{\omega})$). Let the support of f be contained in $(F_\infty)^c$, then we let*

$$\llbracket S \rrbracket f := \frac{[S]f}{[S]1_{(F_\infty)^c}} \quad (6.7)$$

provided the denominator is not zero; otherwise $\llbracket S \rrbracket f$ is undefined.

We are mainly interested in $\llbracket S \rrbracket f$ in cases where, informally speaking, the value of f does not depend on what happens after termination, that is it is determined by the first terminated state: we call these functions *termination based*, and define them formally below. We also define *lifting*, a natural way of defining termination based functions from functions on Ω that only look at the value of variables in (correctly) terminated states.

Definition 18 (termination based f , lifting). *Let $f : \Omega^\infty \rightarrow \overline{\mathbb{R}}^+$ be a measurable function. We say f is termination based if $\text{supp}(f) \subseteq T_\infty$ and for each $t \geq 1$ and $\omega^t \in L^{t-1} \times T$, we have that f is constant on $\{\omega^t\} \times \Omega^\infty$. For any such f and $t \geq 1$, we let $f_t : \Omega^t \rightarrow \overline{\mathbb{R}}^+$ be defined by $f_t(\omega^t) := f(\omega^t, *^\infty)$, for an arbitrarily fixed $* \in L$.*

Given a nonnegative measurable g on Ω with $\text{supp}(g) \subseteq \overline{\mathbb{R}}^m \times \{\text{nil}\}$, we define the lifting of g to Ω^∞ as the function f defined as follows: for $\tilde{\omega} = (\omega_1, \omega_2, \dots)$, $f(\tilde{\omega}) := g(\omega_t)$ if $\tilde{\omega} \in T_t$ for some $t \geq 1$; otherwise, $f(\tilde{\omega}) := 0$. The lifting of g to Ω^t is f_t , where f is the lifting of g to Ω^∞ .

Most functions of interest can be defined via lifting.

Example 24. $f = 1_{T_\infty}$, the indicator function of the termination event, is termination based, and is just the lifting of the predicate $g(v, S) = [S = \text{nil}]$ on Ω . Another example are lifting of functions g that just return the final values of the program's variables, or functions thereof, in terminated states. Outside the class of lifted functions, a function that, for instance, counts the number of times a given event $E \subseteq \Omega$ is observed until termination, is termination based.

Informally speaking, the functions f_t approximate f over paths of length t . It is not difficult to check that, for any t , f_t is measurable over Ω^t , starting from the obvious case where f is the indicator function. The next result shows how to approximate $\llbracket S \rrbracket f$ with quantities defined *only in terms of f_t and μ_S^t* , which is the basis for the MC approximation algorithm in the next section. Formally, for $t \geq 1$ and a measurable function $h : \Omega^t \rightarrow \overline{\mathbb{R}}^+$, we let

$$[S]^t h := E_{\mu_S^t}[h] (= \int \mu_S^t(d\omega^t) h(\omega^t)).$$

We also use the abbreviation $\mathsf{T}^{\leq t} := \cup_{j=1}^t \mathsf{L}^{j-1} \times \mathsf{T} \times \Omega^{t-j} \subseteq \Omega^t$. The intuitive content of the result is as follows. At a given time t , the probability mass over paths of length t is divided among terminated ones ($\mathsf{T}^{\leq t}$), live ones (L^t) and failed ones ($(\mathsf{T}^{\leq t} \cup \mathsf{L}^t)^c$). As t grows, the live probability mass can be distributed over terminated and failed paths. Then considering the numerator in (6.7), at any given time t we obtain an upper bound by moving all the live probability mass to terminated paths, while taking into account a bound M on f ; and a lower bound by zeroing the live probability mass. We can argue similarly for the denominator.

Theorem 19 (finite approximation). *Assume that, for some $t \geq 1$, $\mu_S^t(\mathsf{T}^{\leq t}) > 0$. Then for any termination based function f , we have that $\llbracket S \rrbracket f$ is well defined. Moreover, given a uniform upper bound $f_t \leq M$ ($M \in \overline{\mathbb{R}}^+$), for each t large enough:*

$$\frac{[S]^t f_t}{[S]^t 1_{\mathsf{T}^{\leq t}} + [S]^t 1_{\mathsf{L}^t}} \leq \llbracket S \rrbracket f \leq \frac{[S]^t f_t + M \cdot [S]^t 1_{\mathsf{L}^t}}{[S]^t 1_{\mathsf{T}^{\leq t}}}. \quad (6.8)$$

In particular, if for some t we have $[S]^t 1_{\mathsf{L}^t} = 0$, then

$$\llbracket S \rrbracket f = \frac{[S]^t f_t}{[S]^t 1_{\mathsf{T}^{\leq t}}}. \quad (6.9)$$

When f is an indicator function, $f = 1_A$, we can of course take $M = 1$ in the theorem above.

Example 25. Let us consider again the program S on the variables (i, r, y) in (6.5). Let f be the lifting of the indicator function g defined on Ω that characterizes the final states of the program in which $i \geq 3$, formally: $g((a, b, c), z) = 1$ if $a \geq 3$ and $z = \text{nil}$, and 0 elsewhere. So $\llbracket S \rrbracket f$ is the probability that the execution of S terminates with at least 3 iterations. Fix $t = 120$. Drawing a large number of independent samples⁴ from the program, we can estimate the expected values in (6.8) as arithmetic means, and can check that in this case the bounds yield:

$$0.45 \leq \llbracket S \rrbracket f \leq 0.56.$$

Also, $[S]_{1_{\mathsf{T}^{\leq t}}}^t \geq 0.93$, that is the probability of terminating within time $t = 120$ is at least 0.93. Further details on the actual computation of these figures are postponed until Section 6.6.

⁴For this example, we have used 10^5 samples. In fact, the stated bounds hold with very high probability: this will be made rigorous in the next section in terms of confidence intervals.

Remark 15. Concerning the probability of termination, that is the $\mu_S^\infty(\mathbb{T}_\infty) = \llbracket S \rrbracket f$ with $f = 1_{\mathbb{T}_\infty}$, we note only the *lower* bound in (6.8) can be useful. Indeed, in this case $f_t = 1_{\mathbb{T}^{\leq t}}$, so that the upper bound's numerator in (6.8) is always at least as large as the denominator.

Remark 16 (from expectations to distributions). While our semantics is given in terms of expectations, it is fairly general. In particular, for any nonnegative measurable h defined on Ω^∞ , all the relevant information about h is conveyed by its *cumulative distribution function*, conditioned on non failure, defined for any $r \in \overline{\mathbb{R}}$ as

$$F(r) := \mu_S^\infty \left(h^{-1}([-\infty, r]) \cap (\mathbb{F}_\infty)^c \right) / \mu_S^\infty \left((\mathbb{F}_\infty)^c \right).$$

Now $F(r)$ can be recovered by letting $f = 1_{h \leq r} \cdot 1_{(\mathbb{F}_\infty)^c}$ in Definition 17, that is $F(r) = \llbracket S \rrbracket (1_{h \leq r} \cdot 1_{(\mathbb{F}_\infty)^c})$. To see this, note that this f is just the indicator function of $h^{-1}([-\infty, r]) \cap (\mathbb{F}_\infty)^c$. We also note that this f is *not* termination based, so Theorem 19 as is does not apply to it. We think that at least for finite (rec-free) S , an extension of the theorem to such f is easy, but leave the details for future work.

6.5 Inference via vectorized Monte Carlo sampling

Monte Carlo estimation It is convenient to extend our notation for the measure μ_S^t to the case of an arbitrary initial vector $v \in \overline{\mathbb{R}}^m$, by letting $\mu_{S,v}^t := \delta_{(v,S)} \otimes \mathbb{K} \otimes \cdots \otimes \mathbb{K}$ ($t - 1$ times \mathbb{K}), and $[S]_v^t f := \int \mu_{S,v}^t(\omega^t) f(\omega^t)$.

Proposition 20. *Let $f : \Omega^t \rightarrow \overline{\mathbb{R}}^+$ be a nonnegative measurable function and $t \geq 1$. Then, for $\omega_1 := (v, S) \in \Omega$*

$$[S]_v^t f = \int \mathbb{K}(\omega_1)(d\omega_2) \cdots \int \mathbb{K}(\omega_{t-1})(d\omega_t) f(\omega_1, \omega_2, \dots, \omega_t). \quad (6.10)$$

Moreover, the function $v \mapsto [S]_v^t f$ is measurable.

PROOF. We have

$$\begin{aligned} [S]_v^t f &= \int \mu_{v,S}^t(\omega^t) f(\omega^t) \\ &= \int \delta_{(v,S)}(d\omega_1) \int \mathbb{K}(\omega_1)(d\omega_2) \cdots \int \mathbb{K}(\omega_{t-1})(d\omega_t) f(\omega^t) \\ &= \int \mathbb{K}(v, S)(d\omega_2) \cdots \int \mathbb{K}(\omega_{t-1})(d\omega_t) f((v, S), \omega_2, \dots, \omega_t) \end{aligned}$$

where second equality follows from Theorem 13(b) (Fubini), and the third equality follows from the definition of Dirac's measure. The expression in the last line is (6.10), which equals (6.11) by definition. As to measurability, call $h(\omega)$ the expression obtained from (6.10) by replacing (v, S) with a generic $\omega = (v, y) \in \Omega$. The resulting function of ω is measurable, as implied by the last part of Theorem 13(b) (Fubini). Hence also the function obtained from h by fixing $y = S$, that is $v \mapsto [S]_v^t f$, is measurable: it is obtained by composing a measurable function (h) with the continuous hence measurable function $v \mapsto (v, S)$. \square

By virtue of Proposition 20, any expectation (integral) involved in (6.7) or in (6.8) can be expressed as $t - 1$ iterated expectations (integrals). We shall sometimes abbreviate this as follows, letting $\omega_1 = (0, S)$:

$$[S]_v^t f = \mathbb{E}_{\mu_S^t}[f] = \mathbb{E}_{\omega_2 \sim \mathbb{K}(\omega_1), \omega_3 \sim \mathbb{K}(\omega_2), \dots, \omega_t \sim \mathbb{K}(\omega_{t-1})}[f(\omega_1, \dots, \omega_t)]. \quad (6.11)$$

Equation (6.11) is the basis for implementing a Monte Carlo sampling inference algorithm. Indeed, the Law of Large Numbers implies that expectation can be approximated by arithmetic mean (see further below for a rigorous statement):

$$[S]^t f \approx \frac{1}{N} \sum_{i=1}^N f(Z_i) \quad (6.12)$$

where Z_1, Z_2, \dots, Z_N are N i.i.d. random variables, with Z_i distributed according to μ_S^t . Explicitly, for each i , as prescribed by (6.11):

$$Z_i = (\omega_{i,1}, \dots, \omega_{i,t}), \text{ where: } \omega_{i,1} = (0, S), \omega_{i,2} \sim \mathbb{K}(\omega_{i,1}), \dots, \omega_{i,t} \sim \mathbb{K}(\omega_{i,t-1}). \quad (6.13)$$

If f is of bounded variation, that is $f(\omega^t) \leq M < +\infty$ for each $\omega^t \in \Omega^t$, the quality of the approximation (6.12) can be controlled via the Hoeffding inequality [96], which implies that, for each $\epsilon > 0$

$$\Pr \left(\left| [S]^t f - \frac{1}{N} \sum_{i=1}^N f(Z_i) \right| > \epsilon \right) \leq \delta := 2e^{-2\frac{N\epsilon^2}{M^2}}. \quad (6.14)$$

From confidence intervals of width ϵ for expectations $[S]^t f$, it is easy to build confidence intervals for the ratios representing the lower and upper bounds in (6.8) via simple algebraic manipulations. These resulting bounds tend to be quite loose in some situations, e.g. when f returns small values. To obtain less naive bounds, consider the subset of the N samples that are in $\mathbb{T}^{\leq t} \cup \mathbb{L}^t$, say $\tilde{Z}_1, \dots, \tilde{Z}_{N_a}$ for some $0 \leq N_a \leq N$; we call these the *accepted* samples. Out of these, there will be $0 \leq N_t \leq N_a$ samples in $\mathbb{T}^{\leq t}$ (terminated), and $N_a - N_t$ in \mathbb{L}^t (live). Assuming $N_a > 0$, for $\epsilon > 0$, a tighter confidence interval is

$$\frac{\sum_{i=1}^{N_a} f_t(\tilde{Z}_i)}{N_a} - \epsilon \leq \llbracket S \rrbracket f \leq \frac{\sum_{i=1}^{N_a} f_t(\tilde{Z}_i)}{N_t} + \epsilon + M \cdot \left(\frac{N_a}{\max\{0, N_t - \epsilon N_a\}} - 1 \right) \quad (6.15)$$

where, by applying Hoeffding's inequality, one can prove that the both the above bounds hold with confidence at least $1 - (2\delta_a + \delta_t)$, with $\delta_a = e^{-2N_a\epsilon^2/M^2}$ and $\delta_t = e^{-2N_a\epsilon^2}$. The convention $r/0 = +\infty$ for $r \geq 0$ applies above. When it is known that $\mu_S^t(\mathbb{L}^t) = 0$, the term $M \cdot (\dots)$ can be omitted, and we can set $N_t = N_a$. Let us also mention that there is life beyond Hoeffding: in particular, empirical versions of Bernstein inequality [122, 133] are convenient when the samples exhibit a low empirical variance.

A vectorized algorithm While the above is standard, we argue that, thanks to the uniform formulation of the Markov kernel $\mathbb{K}(\cdot)$, (6.12) can be efficiently implemented in vectorized form. This is outlined in Figure 6.1, where we use the following notation and conventions. We use a pair of arrays (V, P) to store N (independent) instances of pairs (program store, control pointer). In particular, $V \in \overline{\mathbb{R}}^{N \times m}$ and $P \in \mathcal{P}^{N \times 1}$ store, respectively: N program's stores $v \in \overline{\mathbb{R}}^m$; and N control pointers, each represented as a statement $S \in \mathcal{P}$. The basic operations we rely upon are the following.

1. *Boolean masking*: for any two arrays $W \in \overline{\mathbb{R}}^{K \times m}$ and $\beta \in \overline{\mathbb{R}}^{K \times 1}$, we let $W[\beta]$ denote the sub-array of W formed by all rows of W at indices i s.t. $\beta(i) \neq 0$.

| | |
|---|--|
| $\tilde{\mathbb{K}}(V, P)$ 1: $V', P' \leftarrow$ empty arrays of the same shape as V, P 2: $B \leftarrow \{(S, 1_{\{S\}}(P)) : S \in \text{set}(P)\}$ 3: for $(S, \beta) \in B$ do 4: switch S do 5: case nil or fail 6: $V'[\beta] \leftarrow V[\beta], P'[\beta] \leftarrow S$ 7: case $x_i = g.S'$ 8: $V'[\beta, i] \leftarrow g(V[\beta]), P'[\beta] \leftarrow S'$ 9: case $x_i \sim G.S'$ 10: $V'[\beta, i] \leftarrow R, P'[\beta] \leftarrow S'$, where $R \sim D_G(V[\beta])$ 11: case if ϕ then S_1 else S_2 12: $\beta_1 = \phi(V), \beta_2 = 1 - \beta_1$ 13: $V'[\beta \cdot \beta_j], P'[\beta \cdot \beta_j] \leftarrow \tilde{\mathbb{K}}(V[\beta \cdot \beta_j], S_j), j = 1, 2$ 14: case rec $Y.S'$ 15: $V'[\beta], P'[\beta] \leftarrow \tilde{\mathbb{K}}(V[\beta], S'[S/Y])$ 16: end for 17: return (V', P') | $\mathcal{A}(S, h, t, N)$ 1: $V \leftarrow N \times m$ array, filled with 0 2: $P \leftarrow N \times 1$ array, filled with S 3: do $t - 1$ times 4: $V, P \leftarrow \tilde{\mathbb{K}}(V, P)$ 5: end do 6: return $\frac{1}{N} \sum h(V, P)$ |
|---|--|

Figure 6.1: Vectorized Monte-Carlo algorithm for estimating $[S]^t f$, with $f : \Omega^t \rightarrow \overline{\mathbb{R}}$. **Left:** Algorithm $\tilde{\mathbb{K}}$: given $(V, P) = ((v_j)_{j=1}^N, (S_j)_{j=1}^N)$, the returned (V', P') stores N independent drawings from $\mathbb{K}(v_1, S_1), \dots, \mathbb{K}(v_N, S_N)$. **Right:** Algorithm \mathcal{A} , Monte Carlo estimation (6.12) of $[S]^t f$. Here we assume f is the lifting to Ω^t of a function h defined on Ω . **Notation.** $\text{set}(P) \subseteq \mathcal{P}$ is the set of elements that appear in P ; $V[\beta, i]$ is the i -th column of $V[\beta]$; $\sum h(V, P) := \sum_{j=1}^N h(v_j, S_j)$, where v_j, S_j denote the j -th row of V, P , respectively. Broadcast of individual values to whole arrays is automatically performed as needed: $W \leftarrow w$ means filling the array W with a specific value w , and so on. $\beta \cdot \beta'$ denotes the element-wise product of β and β' .

2. *Vectorization:* for any function $h : \overline{\mathbb{R}}^m \rightarrow \overline{\mathbb{R}}$, possibly a predicate, and array $W \in \overline{\mathbb{R}}^{K \times m}$, we let $h(W)$ be the $K \times m$ array obtained by applying h elementwise on W , as expected.

Vectorization also applies to random sampling. Denote by $D_G(v)$ the distribution on $\overline{\mathbb{R}}$ induced by the parametric density $G(v, r)$, taking parameters $v \in \overline{\mathbb{R}}^m$: in $R \sim D_G(W)$, R is a $K \times 1$ array obtained by K independent extractions, drawn according to the distributions $D_G(w)$ for w in W . Modern CPUs offer extensive support for performing boolean masked assignments such as $W[\beta] \leftarrow W'[\beta]$ as *single instructions* (to some extent, depending on the size of W): this effectively leverages SIMD parallelism, eliminating levels of looping and iterations, and potentially leading to significant runtime speed-up. Vectorization is also supported by many high-level programming languages⁵. Note that, in the **for** loop of algorithm $\tilde{\mathbb{K}}$ at line 3, the number of iterations is bounded by $|\text{set}(P)|$, which is of the order of the syntactic size of S and independent of the number of samples N to be taken. An optimization of this scheme is possible by considering a suitable restricted language of program statements; this will be explored in Section 6.6.

⁵For instance, in Python via the Numpy or TensorFlow packages.

Importance sampling Consider the program $S = x_1 \sim \rho_U(0,1).x_2 \sim \rho_G(x_1,1).\text{obs}(|x_2 - x_1| > 3).\text{nil}$. The `obs` statement demands that x_2 fall more than three standard deviations away from x_1 . Suppose we draw N samples from this program, relying in the algorithm \mathcal{A} in the previous section. Recalling that $\text{obs}(\phi).\text{nil} = \text{if } \phi \text{ then nil else fail}$, a large fraction of the resulting computations — on average about 99.7% — will end into fail, that is will be *rejected*. If the purpose of this procedure is obtaining samples of the terminated states of the program, this is huge waste of computational resources. According to the principles of *importance sampling* (IS), it is better if in the second sampling step in S one draws directly from the distribution $\rho_G((x_1,1), \cdot)$ restricted to the set $A_{x_1} = (-\infty, x_1 - 3) \cup (x_1 + 3, +\infty)$: call $\tilde{\rho}_G((x_1,1), \cdot)$ this new distribution. This change of sampling distribution must be compensated by weighing each sample with an *importance weight* $W(x_1) = \int_{A_{x_1}} \mu(dr) \rho_G((x_1,1), r)$. Formally, we apply Proposition 20 with $t = 3$, take into account the chain rule for densities and, assuming f is the lifting of some g , compute as follows (in the second step, we take into account that $g(r_1, r_2, \text{fail}) = 0$; in the third step, the definition of the restricted density $\tilde{\rho}_G(r_1, 1; r_2)$):

$$\begin{aligned}
[S]^t f &= \int \mu(dr_1) \rho_U(0,1;r_1) \cdot \int \mu(dr_2) \rho_G((r_1,1), r_2) \cdot [|r_2 - r_1| > 3] \cdot g(r_1, r_2, \text{nil}) + \\
&\quad \int \mu(dr_1) \rho_U(0,1;r_1) \cdot \int \mu(dr_2) \rho_G((r_1,1), r_2) \cdot [|r_2 - r_1| \leq 3] \cdot g(r_1, r_2, \text{fail}) \\
&= \int \mu(dr_1) \rho_U(0,1;r_1) \cdot \int \mu(dr_2) \left(\frac{\rho_G(r_1,1;r_2) \cdot [|r_2 - r_1| > 3]}{W(r_1)} \right) W(r_1) g(r_1, r_2, \text{nil}) \\
&= \int \mu(dr_1) \rho_U(0,1;r_1) \cdot \int \mu(dr_2) \tilde{\rho}_G((r_1,1), r_2) W(r_1) g(r_1, r_2, \text{nil}) \\
&= [\tilde{S}]^t \tilde{f}
\end{aligned} \tag{6.16}$$

where $\tilde{S} = x_1 \sim \rho_U(0,1).x_2 \sim \tilde{\rho}_G(x_1,1).\text{nil}$ and \tilde{f} is the lifting of $W \cdot g$. Now, provided we know how to draw efficiently samples from $\tilde{\rho}_G$ and how to compute $W(x_1)$ (which is in fact easy), evaluating $[\tilde{S}]^t \tilde{f}$ by applying \mathcal{A} to \tilde{S} and \tilde{f} will be much more efficient than applying \mathcal{A} directly to S and f . In this example, when applying \mathcal{A} to \tilde{S} we will have no rejections. Moreover, from the point of view of the application of Hoeffding's bound (6.14), we can replace M with the much smaller $M \cdot \sup_{x_1} W(x_1)$, obtaining to a significative sharpening of the bound. In fact, in this case $W(x_1)$ is a constant, $W(x_1) \approx 0.003$. The above example can be generalized to a systematic procedure, the interested reader can find the details in Appendix D.2. Let us remark here that applying this procedure can be costly, as, in the general case, the number of terms that may appear in an expansion like (6.16) can be exponential in the size of S . In our experiments, IS turned out to be less convenient than a pure vectorized Monte Carlo estimation.

6.6 Implementation and evaluation

We will specialize the sampling algorithm \mathcal{A} of Section 6.5 to a sublanguage of \mathcal{P} , and present an implementation based on TensorFlow (TF) [1], an open source library enabling efficient and SIMD-parallel manipulation of tensors (multidimensional arrays). We will then evaluate this implementation on a few probabilistic programs taken from the literature. For loop-free examples, we will also compare our results with those obtained by applying two state-of-the-art tools for probabilistic programming.

6.6.1 TSI: a TensorFlow based implementation

We focus on a sublanguage $\mathcal{P}_0 \subseteq \mathcal{P}$, and present a TF-based implementation of a specialized version of algorithm \mathcal{A} . The syntax of \mathcal{P}_0 is obtained by imposing that every `rec` subterm has the following form:

$$\text{rec } Y.(\text{ if } \phi \text{ then } S_1 \text{ else } S_2)$$

such that S_1 does not contain `rec` nor `nil`, $S_1 \neq \text{fail}$ and S_2 does not contain Y . So nested `rec`'s are not allowed, and exiting a loop is possible only either via a `fail` statement inside S_1 or by making ϕ false, in the last case proceeding then to S_2 . Accordingly, in the rest of the section we will also use the following abbreviation, for any `rec`-free statement $S_1 \neq \text{fail}$ where every `nil` is guarded by an action prefix, and S_2 does not contain Y :

$$\text{while } \phi \text{ } S_1, S_2 := \text{rec } Y.(\text{ if } \phi \text{ then } S_1[Y/\text{nil}] \text{ else } S_2)) \quad (6.17)$$

The sublanguage \mathcal{P}_0 has been tailored so as to be mapped easily into TF, as explained below; yet it is still expressive enough to allow for the description of nearly all the examples in the literature we have examined. Our translation leverages the following linguistic features of TF.

1. Vectorized **assignment** `xi=g(x)`, where `xi`, `x` can be either scalar or tensor variables, or tuples thereof. Functions and predicates on scalars are automatically extended to tensors as expected. Likewise, vectorized **sampling** from a distribution with parameters `x` will be written `G(x).sample()`: when the parameter `x` is a tensor, this will yield a tensor of independent samples, one for each component of `x`.
2. Vectorized **if-then-else** expression `tf.where(cond, a, b)`. Here `cond` is a tensor of booleans that acts as a mask: the result of `tf.where` is a tensor of the same shape as `a`, where the value of each component is the corresponding element from tensor `a` (if the corresponding element from `cond` is `True`) or from tensor `b` (otherwise). These operations are executed independently on the elements of the involved tensors `cond, a, b`, in a vectorized fashion.
3. Vectorized **while-loop** expression `tf.while_loop(fcond, fbody, vars)`. Here both `fcond` and `fbody` are functions that can be called with the `vars` tensor variables as arguments: `fcond` returns a boolean scalar, and `fbody` returns a tuple of tensors that can be assigned to variables `vars`. The semantics dictates that the assignment `vars=fbody(vars)` be repeated as long as `fcond(vars)` yields `True`; upon termination, the current value of `vars` is returned. Therefore, although the termination condition `fcond(vars)` is global, different instances of `fbody` are independently executed on the elements of tensors `vars`, in a vectorized fashion.

An outline of our translation of \mathcal{P}_0 into TF is presented in Figure 6.2. A statement $S \in \mathcal{P}_0$ is translated into a TF-Python function f_S , introduced by the declaration $\mathbb{T}(S)$. When invoked, f_S will return N independent samples, each drawn according to (6.13).

In more detail, the command $x, \text{mask}, \text{t} = f_S(x, \text{mask}, \text{t})$ executes N independent instances of the initial statement S , applying successive transitions, until each instance has reached at least a prescribed time (=path length) of t_0 . The tuple of tensors $x = x_1, \dots, x_m$, one for each program variable, initially filled with 0, will store the drawn N independent samples. The control pointer vector V of algorithm \mathcal{A} is replaced here by a pair of tensors mask, t . The tensor mask will keep track of which of the N instances is actually terminated (`True`), failed (`False`) or live (`NaN`) at the end of the execution. The tensor t is a counter, that will record the actual time at which each instance will reach `nil` or `fail`, if ever. Note that “stuttering” transitions from `nil` or `fail` do not increment t ; but transitions from `if-then-else`’s where one of the two branches is a `nil` or `fail` do increment t (cf. lines 19 and 31). The use of t is necessary because the N instances are not necessarily time-aligned at the end of the execution of f_S . The information contained in mask, t will be used at the end to discern which instances are actually terminated/failed/live at time t_0 , and the corresponding samples from x (like in lines 42,43). Taken together, these form the wanted N i.i.d. samples from μ^{t_0} drawn according⁶ to (6.13).

The clauses of the translation function $T(\cdot)$ should be self-explanatory, but possibly for the last one. In the case of `while`, a complication is that, within the given time limit of t_0 , the N independently executed instances may exit the loop at different times, or even not exit the loop at all. In the corresponding code (lines 23-34), the execution of an instance is suspended as soon as it reaches either an exit condition or time t_0 (second branch of `where` in the definition of `fbody`, line 26); the `while_loop` stops when all instances have reached this situation (condition in `fcond`, lines 28,29). Next, the execution of all instances is resumed with S_2 (lines 30,31), but this will have an effect only for instances that had exited the loop normally, that is by invalidating ϕ before time t_0 without failing: the other instances are deemed either live or failed at time t_0 (lines 32, 33), and S_2 will have no effect for them. Further details are provided in the caption of Figure 6.2. A simplification of this translation is possible for statements $S \in \mathcal{P}_0$ that rule out either `while` or `if-then-else` (this in fact covers all the examples we will see); see [34].

Technically, the TF execution model is based on *data flow graphs*: under the hood [130], the code is expressed as a computational graph, a data structure where nodes represent operations, and edges represent data, arranged as tensors, that flow through the graph between operations. A graph execution model is defined that leverages fine-grained, massive SIMD-parallelism. Better performances are obtained on GPU-based hardware.

We will refer to the framework resulting from our translation of \mathcal{P}_0 into TF as *TSI*, for *Tensor-based Sampling and Inference*.

6.6.2 Evaluation of TSI

In the first subsection, we present experimental results obtained by putting TSI at work on four loop-free probabilistic programs taken from the literature: TrueSkill, Pearl’s Burglar Alarm, ClickGraph and MontyHall. At the same time, we compare TSI with

⁶Like for algorithm \mathcal{A} , since f is assumed to be the lifting of some h defined on Ω , we only keep the last element of each sequence, ω_{i,t_0} in the notation of (6.13).

two state-of-the-art probabilistic languages (and inference tools), webPPL [83] and R2 [137]: we have chosen them because the corresponding languages are easy to compare with ours, in particular both feature observe or equivalent statements for conditioning. webPPL incorporates the principles of Church [84], a LISP based PPL with an elegant semantics. For the time being, we have left out of our comparison languages and tools that are more tailored to fitting parameters to data and do not feature observe statements, like Stan [43], or that lack a formally defined semantics.

Translation function

```

38 T(nil) = return x, TRUE, t
39
40 T(fail) = return x, FALSE, t
41
42 T( $x_i = g.S$ ) =
43     t=t+1
44     xi=g(x)
45     T(S)
46
47 T( $x_i \sim G.S$ ) =
48     t=t+1
49     xi=G(x).sample()
50     T(S)
51
52 T( if  $\phi$  then  $S_1$  else  $S_2$ ) =
53     T( $S_1$ )
54     T( $S_2$ )
55     x,mask,t'=tf.where(phi(x), {f $_{S_1}$ (x,mask,t)}, {f $_{S_2}$ (x,mask,t)})
56     t=tf.where(t==t', t+1, t')
57     return x,mask,t
58
59 T(while  $\phi$   $S_1, S_2$ ) =
60     T( $S_1$ )
61     T( $S_2$ )
62     def fbody(x,mask,t):
63         x,mask,t=tf.where(phi(x)&(mask!=False)&(t<t0),{f $_{S_1}$ (x,mask,t)},{x,mask,t})
64         return x,mask,t
65     fcond=lambda x,mask,t: Any(phi(x)&(mask!=False)&(t<t0))
66     x,mask,t =tf.while_loop(fcond, fbody, (x,mask,t))
67     x',mask',t'=f $_{S_2}$ (x,mask,t)
68     t'=tf.where(t==t', t+1, t')
69     x,mask',t=tf.where(not(phi(x))&(t<t0)&(mask!=False),{x',mask',t'},{x,NAN,t})
70     mask=tf.where(mask!=False,mask',False)
71     return x,mask,t
72
73 T(S)=def f $_S$ (x,mask,t):
74     T(S)

```

Workflow

Example assuming $m = 1$, hence x is x_1 . Compute est , the MC estimation (6.12) of $[S]^{t_0} f_{t_0}$, assuming f is the lifting of h .

```

38 x1=tf.fill(0,shape=(1,N))
39 mask=tf.fill(NaN,shape=(1,N))
40 t=tf.fill(1,shape=(1,N))
41 T(S)
42 x1,mask,t=f_S(x1,mask,t)
43 term = (mask==True) & (t<=t0) # mask selecting terminated instances at time t0
44 est = sum(h(x1[term]))/N

```

Figure 6.2: Outline of the translation from \mathcal{P}_0 into Python with TensorFlow. **Top, translation function.** Definition of functions $\mathbb{T}(S)$ and auxiliary $T(S)$, by structural induction on S . **Bottom, workflow.** The function f_S defined by $\mathbb{T}(S)$ is called with input arguments x, mask, t where $x = x_1, \dots, x_m$: here x_i, mask, t are tensors of shape $(1, N)$, filled with 0, NaN and 1, respectively. $x, \text{mask}, t = f_S(x, \text{mask}, t)$ computes N independent samplings stored in x , and masks mask and t , to be used for selecting instances that are actually live/terminated/failed at the specified time t_0 , and calculate estimations of $[S]^{t_0} f_{t_0}$ and $\llbracket S \rrbracket f$. **Notation.** TRUE (resp. FALSE, NAN) denote the $(1, N)$ tensor filled with True (resp. False, NaN). We use $\{\dots\}$ to denote concatenation of tensors $x, \text{mask}, t = x_1, \dots, x_m, \text{mask}, t$ along the axis 0: assuming each of x_i, mask, t is a tensor of shape $(1, N)$, then $\{x, \text{mask}, t\}$ returns a tensor of shape $(m+2, N)$. Likewise, the left-hand side of any assignment, $x, m, t = \text{tf.where}(\dots)$ actually denotes slicing of the tensor resulting from $\text{tf.where}(\dots)$ into $m+2$ $(1, N)$ -tensors, which are then assigned to the variables $x_1, \dots, x_m, \text{mask}, t$. In actual TF syntax, $\{\dots\}$ corresponds to using $\text{tf.concat}()$, and $x, m, t = \text{tf.where}(\dots)$ to suitable tensor slicing operations. Any(b), for b a tensor of booleans, is the scalar boolean value obtained by OR-ing all elements in b . The usual Python's notation and rules to broadcast scalars to whole tensors apply, as well as those on boolean masking of tensors.

In the second subsection, we apply TSI to probabilistic programs involving unbounded loops, and discuss the obtained results mainly from the point of view of accuracy. Code and examples for these experiments⁷ are available online [34].

Loop-free programs and comparison with WebPPL and R2 We consider TrueSkill, Pearl's Burglar Alarm, ClickGraph and Monty Hall, four nontrivial probabilistic programs with obs statements. On these models, we compare TSI with both WebPPL, a programming language based on JavaScript, and R2, a probabilistic programming system based on C#. In Table 6.1, we report the execution time required by the different approaches on the four programs, as the number of samples increases, together with the estimated expected value for their output variables. In each case, the value of t for TSI has been chosen large enough to guarantee termination or failure of all instances (no live terms). N is the number of samples generated by TSI, including non-accepted (failed) ones. Out of these, a number $N_a \leq N$ in each case is accepted: this N_a is the number of samples that the other tools are required to generate in each

⁷Our PC configuration is as follows. OS: Windows 10; CPU: 2.8 GHz Intel Core i7; GPU: Nvidia T500, driver v. 522.06; TF: v. 2.10.1; CUDA Toolkit v. 11.8; cuDNN SDK v. 8.6.0.

case. In all cases TSI is significantly faster — up to three orders of magnitude — than the other tools. In the table, we only report the central value of TSI’s confidence interval of width 2ϵ , computed by (6.15) with $N_t = N_a$ and the rightmost term elided. We see that TSI also performs very well in terms of accuracy: e.g. for $N = 10^6$, the reported value coincides with the exact expectation, in all the analyzed models. The confidence is also good: for $N \geq 5 \times 10^6$ and $\epsilon = 0.005$, it always exceeds 0.99. Note that not being MCMC an exact sampling techniques, in the case of webPPL-MCMC and R2 the generated samples cannot be used to give formal guarantees. In conclusion, over the considered examples, TSI shows a significant advantage in terms of both execution time and accuracy.

| | | TS (skillA=1.057) | | | | BA (burg=0.029) | | | | CG (simAll=0.570) | | | | MH (win=0.666) | | | |
|---------------------|----------------|-------------------|----------------|----------------|--------------|-----------------|----------------|----------------|--------------|-------------------|----------------|----------------|--------------|----------------|----------------|----------------|--------------|
| | | TSI | webppl rej. | webppl mcmc | R2 | TSI | webppl rej. | webppl mcmc | R2 | TSI | webppl rej. | webppl mcmc | R2 | TSI | webppl rej. | webppl mcmc | R2 |
| $N = 10^4$ | <i>runtime</i> | 0.001 | 0.051 | 0.041 | 0.063 | 0.003 | 0.036 | 0.016 | 0.046 | 0.006 | 0.032 | 0.011 | 0.031 | 0.011 | 0.104 | 0.097 | 0.015 |
| | <i>exp.</i> | 1.057 | 1.056 | 1.052 | 1.063 | 0.029 | 0.035 | 0.037 | 0.035 | 0.560 | 0.541 | 0.630 | 0.513 | 0.671 | 0.663 | 0.675 | 0.668 |
| $N = 10^5$ | <i>runtime</i> | 0.001 | 0.305 | 0.147 | 0.640 | 0.005 | 0.191 | 0.065 | 0.296 | 0.008 | 0.119 | 0.052 | 0.141 | 0.015 | 0.692 | 0.523 | 0.141 |
| | <i>exp.</i> | 1.056 | 1.056 | 1.057 | 1.063 | 0.029 | 0.030 | 0.027 | 0.026 | 0.561 | 0.554 | 0.551 | 0.530 | 0.666 | 0.666 | 0.664 | 0.664 |
| $N = 10^6$ | <i>runtime</i> | 0.002 | 2.765 | 1.061 | 6.370 | 0.040 | 1.154 | 0.344 | 2.893 | 0.031 | 0.954 | 0.223 | 1.309 | 0.071 | 6.426 | 4.925 | 1.454 |
| | <i>exp.</i> | 1.057 | 1.057 | 1.056 | 1.057 | 0.029 | 0.029 | 0.027 | 0.029 | 0.570 | 0.547 | 0.542 | 0.567 | 0.666 | 0.666 | 0.665 | 0.666 |
| $N = 5 \times 10^6$ | <i>runtime</i> | 0.002 | 14.345 | 5.215 | 32.567 | 0.042 | 8.198 | 1.636 | 14.610 | 0.125 | 4.791 | 0.824 | 6.283 | 0.344 | 33.562 | 25.200 | 7.332 |
| | <i>exp.</i> | 1.057 | 1.057 | 1.058 | 1.057 | 0.029 | 0.029 | 0.028 | 0.029 | 0.570 | 0.547 | 0.551 | 0.570 | 0.666 | 0.666 | 0.666 | 0.666 |

Table 6.1: Vectorized Comparison of TSI, webPPL and R2 on TrueSkill (TS), Burglar Alarm (BA), ClickGraph (CG) and Monty Hall (MH) programs. *runtime*: execution time; *exp.*: estimated expected value; N : number of i.i.d. samples generated by TSI. For N_a = number of i.i.d. samples accepted by TSI, the estimated acceptance rates N_a/N for the four programs are: 0.14, 0.20, 0.025 and 1.

Unbounded loops and random walks Consider the programs S and S' defined respectively in (6.5) and (6.6) in Example 22, describing two random walks. Both programs use unbounded loops in an essential way that neither webPPL nor R2 are able to express easily. Moreover, both programs fall in sublanguage \mathcal{P}_0 we have considered in this section. Indeed, using the abbreviation introduced in (6.17), S and S' can be equivalently rewritten as

$$S = r \sim \rho_U. \text{ while } (|y| < 1) (y \sim \rho_G(y, 2 \cdot r).i = i + 1. \text{nil}), \text{ nil}$$

$$S' = r \sim \rho_U. \text{ while } (|y| < 1) (y \sim \rho_G(y, 2 \cdot r).i = i + 1. \text{nil}), \text{ obs}(i \geq 3).$$

Continuing now with Example 22, consider $\llbracket S \rrbracket f$, that is the probability that S terminates correctly in at least 3 iterations. In the case of S' , we consider $\llbracket S' \rrbracket f'$, where f' (is the lifting of the function that) looks at the final value of variable r : that is, r conditioned on the observation that $i \geq 3$. We can compute upper and a lower bounds of $\llbracket S \rrbracket f$ and $\llbracket S' \rrbracket f'$ by applying (6.8), for any fixed value of $t \geq 1$, the length of execution paths. As the exact expected values involved in (6.8) are not available, we proceed to their estimation via TSI, with confidence intervals provided by Hoeffding inequality (6.14), as discussed in Section 6.5. In detail, we fix an error threshold of $\epsilon = 0.005$ and $N = 10^6$ samples, and apply the bounds (6.15). We report the obtained

results in Table 6.2, for different values of t . As t gets larger, TSI provides tighter and tighter confidence intervals for $\llbracket \cdot \rrbracket$. For example, at $t = 202$, for $\llbracket S \rrbracket f$ an interval of width 0.04 is computed in less than 0.5 seconds.

| t | $\llbracket S \rrbracket f$ | term. prob. | runtime | t | $\llbracket S' \rrbracket f'$ | term. prob. | runtime |
|-----|-----------------------------|-------------|---------|-----|-------------------------------|-------------|---------|
| 12 | 0.623 ± 0.112 | 0.714 | 0.016 | 12 | 0.663 ± 0.337 | 0.445 | 0.016 |
| 22 | 0.576 ± 0.065 | 0.817 | 0.021 | 22 | 0.663 ± 0.336 | 0.647 | 0.022 |
| 102 | 0.538 ± 0.027 | 0.927 | 0.042 | 102 | 0.442 ± 0.115 | 0.861 | 0.062 |
| 202 | 0.530 ± 0.020 | 0.950 | 0.343 | 202 | 0.406 ± 0.079 | 0.904 | 0.268 |

Table 6.2: Vectorized Confidence intervals for $\llbracket S \rrbracket f$ (left) and $\llbracket S' \rrbracket f'$ (right), termination probability on paths of length $\leq t$ and corresponding execution time, as a function of t . The chosen path lengths t correspond to the execution of K iterations of the while loop, for $K = 5, 10, 50, 100$.

6.7 Conclusion

We have presented a simple yet rigorous measure-theoretic small-step semantics for a probabilistic programming language, that lends itself to a direct, SIMD-parallel implementation. An approximation theorem reduces the effective estimation, with guarantees, of the program semantics (expectation) to the analysis of finite execution paths of a chosen length. TSI, a prototype implementation based on TensorFlow, has shown encouraging results.

Chapter 7

Conclusion

This thesis is centered on Formal Methods for the analysis of continuous dynamical systems, especially of those specified in terms of ordinary differential equations. General goals of Formal Methods for dynamical system include designing methods for proving correctness in terms of Safety, and more generally of reachability, and for guaranteed parameter inference. In these respects, our main contributions can be summarized as follows.

- *Invariants and conservation laws.* We present a method to compute polynomial conservation laws for systems of partial differential equations. The method is effective, being based mainly on linear algebraic operations, and complete, in the sense that it can find all the polynomial conservation laws up to a specified order of derivatives and degree.
- *Linearization and reachability.* We study conditions and methods to compute reduced, linear approximations of nonlinear ODEs that are accurate also non locally. We leverage the approximations generated in this way for reachability analysis. In particular, we describe CKR, an algorithm to overapproximate the set of reachable states of a given nonlinear ODEs system over a finite time horizon.
- *SDEs, a (co)algebraic approach.* We study connections among polynomials, differential equations and streams in terms of algebra and coalgebra. We describe a method to find all invariant polynomials that fit a user-specified polynomial template, for a given ODEs system. This generalizes to systems of Streams Differential Equations (SDEs). We also propose a stream version of the Implicit Function Theorem (IFT) for algebraic systems of equations.
- *Inference: Bayesian parameter estimation.* We provide a method to compute guaranteed estimates for ODEs parameters, relying on noisy observations from their trajectories. To this effect, we combine Monte Carlo simulation, uncertain probability and Interval Arithmetic. Guarantees come in the form of confidence intervals for the posterior expectations of the parameters.
- *Inference, a more general view: Probabilistic Programming.* We present an action based probabilistic programming language equipped with a small-step

operational semantics, formalized in terms of a product of Markov kernels. This approach provides benefits in terms of both clarity and effective implementation and directly leads to an exact sampling algorithm that can be efficiently SIMD-parallelized. In particular, the algorithm can be applied to compute estimates of (discretized) ODEs parameters.

In each case, we have experimented with a proof-of-concept implementation of the proposed method, and shown that it compares favourably to state-of-the-art tools or techniques, in terms of accuracy and/or performance.

7.1 Future work

This work is open to further expansion in several directions. For instance, regarding reachability analysis, it is well-known that states sets representation is crucial for both accuracy and efficiency of the considered reachability method. In order to further improve the scalability of `ckr` algorithm, we might change the states sets representation and rely on zonotopes rather on polytopes. In this regard, some preliminary experiments have shown very encouraging results. Moreover, it would be interesting to apply `ckr` to Runtime Verification and Model Predictive Control to see how it performs and whether it can be effectively useful.

Turning specifically to safety of ODEs, it would be interesting to investigate a new family of deep neural network models that has been introduced recently in [49]: Neural Ordinary Differential Equations (NODEs). Essentially, NODEs are ODEs in which the dynamics are specified by a neural network. From a more deep-learning perspective, NODEs are deep neural network models in which the derivative of the hidden state is parameterized using a neural network, rather than specifying a discrete sequence of hidden layers. More generally, it would be interesting to deepen Data-Driven approaches for ODEs to exploit the large amount of data that is available nowadays and try to prove safety properties or to provide guarantees of correctness for them.

Concerning stream differential equations, we would like to verify if new notions of products can be defined respecting the product format presented in Chapter 4. We would like also to extend the stream Implicit Function Theorem presented in Chapter 4 to the case of multivariate streams, that is considering a vector of independent variables.

Regarding parameter inference, it would be interesting to apply the hybrid approach to provide guaranteed estimates for ODEs parameters developed in Chapter 5 to more complex models, and further analyze its application to feature relevance in neural network classifiers. Finally, the sampling scheme based on probabilistic programming proposed in Chapter 6 can be improved considering more sophisticated inference algorithms based for example on particles filters as Sequential Monte Carlo [63]. In this way rejected computations are replaced via resampling rather than being discarded. At the same time, it would be interesting to establish a more formal relation between inference algorithms and formal semantics for probabilistic programming languages. Probabilistic programming has been heavily applied in different fields: to do sensitivity analysis, but also for support in medical diagnostics or computer vision. It would be interesting to consider some of these applications in detail.

Bibliography

- [1] M. Abadi, P.B.J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D.G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, X. Zheng. TensorFlow: A System for Large-Scale Machine Learning. *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*. *arXiv:1605.08695*, 2016.
- [2] M.J. Ablowitz, P.A. Clarkson. *Solitons, Nonlinear Evolution Equations and Inverse Scattering*. Cambridge University Press, Cambridge, 1991.
- [3] E. Ábrahám. Modeling and Analysis of Hybrid Systems. *Lecture Notes*, 2012.
- [4] S.P. Adam, A.C. Likas. A Set Membership Approach to Discovering Feature Relevance and Explaining Neural Classifier Decisions. In *arXiv:2204.02241*, 2022.
- [5] A. A. Alahmadi, J. A. Flegg, D. G. Cochrane, C. C. Drovandi and J. M. Keith. A comparison of approximate versus exact techniques for Bayesian parameter inference in nonlinear ordinary differential equation models. *R. Soc. open sci.* 7(3):19131, 2020.
- [6] M. Althoff. Build Your Own Reachability Analyzer with CORA. In: *Post-proceedings of the Summer School Marktoberdorf: Safety and Security of Software Systems-Logics, Proofs, Applications*, 2020.
- [7] M. Althoff. Reachability analysis of nonlinear systems using conservative polynomialization and non-convex sets. *Proceedings of the 16th international conference on Hybrid systems: computation and control (HSCC'13)*, pp. 173-182 <https://doi.org/10.1145/2461328.2461358>, ACM, 2013.
- [8] M. Althoff. An introduction to CORA 2015. In *Proc. of the Workshop on Applied Verification for Continuous and Hybrid Systems*, pages 120-151, 2015.
- [9] M. Althoff, G. Frehse, A. Girard. Set Propagation Techniques for Reachability Analysis. *Annual Review of Control, Robotics, and Autonomous Systems*, Annual Reviews 2021, 4 (1), 10.1146/annurev-control-071420-081941, hal-03048155, 2021.
- [10] A. Amini, Q. Sun, N. Motee. Error bounds for Carleman Linearization of General Nonlinear Systems. In *Proceedings of the Conference on Control and its Applications*, SIAM, 2021.

- [11] S. Anco, G. Bluman. Direct construction method for conservation laws of partial differential equations. Part II: General treatment. *Europ. J. Appl. Math.* 13, 567–585, 2002.
- [12] R.B. Ash. *Real Analysis and Probability*. Academic Press, Inc., New York, NY, USA, 1972.
- [13] C. Baier and J. P. Katoen. *Principles of model checking*. MIT press, 2008.
- [14] G. Barthe, J.P. Katoen, A. Silva: Foundations of Probabilistic Programming. *Cambridge University Press*, 2020.
- [15] G. Barthe, J.P. Katoen, and A. Silva. *Foundations of Probabilistic Programming*. Cambridge University Press, 2020.
- [16] H. Basold, M. Bonsangue, H. Hansen, and J. Rutten. (co)algebraic characterizations of signal flow graphs. In *Horizons of the Mind. A Tribute to Prakash Panangaden - Essays Dedicated to Prakash Panangaden on the Occasion of His 60th Birthday*, volume 8464 of *LNCS*, pages 124–145. Springer, 2014.
- [17] H. Basold, H. Hansen, J. Pin, and J. Rutten. Newton series, coinductively: a comparative study of composition. *Math. Struct. Comput. Sci.* 29, 1, pp. 38–66, 2019.
- [18] R. Bellman, J.M. Richardson. On some questions arising in the approximate solution of nonlinear differential equations. *Quarterly of Applied Mathematics* 20(4):333-339, January, 1963.
- [19] P. Bhaduri. Coalgebras for Bisimulation of Weighted Automata over Semirings. *Log. Methods Comput. Sci.* 19, 1, 38–66, 2023.
- [20] S. Bhat, J. Borgström, A.D. Gordon, and C. Russo. Deriving probability density functions from probabilistic functional programs. *Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2013)*, 2013.
- [21] F. Bonchi, M. Bonsangue, M. Boreale, J. Rutten, and Alexandra Silva. A coalgebraic perspective on linear weighted automata. *Inf. Comput.*, 211:77–105, 2012.
- [22] F. Bonchi, D. Petrisan, D. Pous, and J. Rot. Coinduction up-to in a fibrational setting. In *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14*, pages 20:1–20:9. ACM, 2014.
- [23] F. Bonchi, P. Sobocinski, and F. Zanasi. Full Abstraction for Signal Flow Graphs. *Proc. of POPL*, pages 515–526. ACM, 2015.
- [24] F. Bonchi, P. Sobocinski, and F. Zanasi. The Calculus of Signal Flow Diagrams I: Linear relations on streams. *Inf. Comput.*, 252: 2–29. Elsevier, 2017.

- [25] M. Bonsangue, H. Hansen, A. Kurze, and J. Rot. Presenting Distributive Laws. *Log. Methods Comput. Sci.*, 11, 3, 2015.
- [26] M. Boreale and D. Gorla. Algebra and Coalgebra of Stream Products. *32nd International Conference on Concurrency Theory, CONCUR 20*, Virtual Conference, LIPIcs 203, 2021.
- [27] M. Boreale. Algebra, coalgebra, and minimization in polynomial differential equations. *Log. Methods Comput. Sci.* 15, 1, 2019.
- [28] M. Boreale. Algorithms for exact and approximate linear abstractions of polynomial continuous systems. *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (part of CPS Week), HSCC'18*, ACM, 2018.
- [29] M. Boreale. Complete algorithms for algebraic strongest postconditions and weakest preconditions in polynomial ODE's. *Sci. Comput. Program.* 193, Elsevier, 2020. doi 10.1016/j.scico.2020.102441. Short version in *SOFSEM 2018*, LNCS 10706:442-455, Springer, 2018.
- [30] M. Boreale. Weighted Bisimulation in Linear Algebraic Form. In *Proceedings of CONCUR 2009*, volume 5710 of Lecture Notes in Computer Science, Springer (2009), 163-177.
- [31] M. Boreale. On the Coalgebra of Partial Differential Equations. *MFCS 2019*, LIPIcs 138: 24:1-24:13, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. <https://drops.dagstuhl.de/opus/volltexte/2019/10968/pdf/LIPIcs-MFCS-2019-24.pdf>
- [32] M. Boreale. Automatic pre- and postconditions for partial differential equations. In Marco Gribaudo, David N. Jansen, and Anne Remke, editors, *Quantitative Evaluation of Systems - 17th International Conference, QEST 2020, Vienna, Austria, August 31 - September 3, 2020, Proceedings*, volume 12289 of *Lecture Notes in Computer Science*, pages 193–210. Springer, 2020.
- [33] M. Boreale, L. Collodi. Linearization, Model Reduction and Reachability in Nonlinear odes. *Reachability Problems 2022 (RP22)*. LNCS 13608: 49-66, Springer, 2022.
- [34] M. Boreale, and L. Collodi. Python code for the experiments described in the present paper. https://github.com/Luisa-unifi/probabilistic_programming, 2023.
- [35] J. Borgström, A. Gordon, M. Greenberg, J. Margetson, J. Van Gael. Measure Transformer Semantics for Bayesian Machine Learning. Pages 77–96 of: *Programming Languages and Systems*. Springer, 2011.
- [36] A. Bostan, F. Chyzak, M. Giusti, R. Lebreton, G. Lecerf, B. Salvy, É. Schost. *Algorithmes Efficaces en Calcul Formel*. Palaiseau, France, 2018. URL: <https://hal.archives-ouvertes.fr/AECF>.

- [37] A. Bueno-Orovio, M. Cherry, and F. Fenton. Minimal model for human ventricular action potentials in tissue. *Journal of Theoretical Biology* 253, pp. 544–560, 2008.
- [38] L. Cardelli, G. Squillace, M. Tribastone, M. Tschaikowski, A. Vandin. Formal lumping of polynomial differential equations through approximate equivalences. *JLAMP 134*: 100876 (2023) – with extends a QEST’18 publication.
- [39] L. Cardelli, M. Tribastone, M. Tschaikowski, A. Vandin. ERODE: A tool for the evaluation and reduction of ordinary differential equations. Proceedings of the 22nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), *Lecture Notes in Computer Science* (Springer, Berlin), Vol 10206, pp 310–328, 2017.
- [40] L. Cardelli, M. Tribastone, M. Tschaikowski, A. Vandin. Symbolic computation of differential equivalences. *Theor. Comput. Sci.* 777: 132-154, 2019.
- [41] L. Cardelli, M. Tribastone, M. Tschaikowski, A. Vandin. Comparing Chemical Reaction Networks: A Categorical and Algorithmic Perspective. *LICS 2016*: 485-494.
- [42] L. Cardelli, M. Tribastone, M. Tschaikowski, A. Vandin. Symbolic computation of differential equivalences. *POPL 2016*: 137-150.
- [43] B. Carpenter, A. Gelman, M. D. Hoffman, D. Lee, B. Goodrich, M. Betancourt, M. Brubaker, J. Guo, P. Li, and A. Riddell. Stan: A probabilistic programming language. *Journal of Statistical Software* 76(1), 2017. 10.18637/jss.v076.i01
- [44] G. Casella and L. R. Berger. *Statistical Inference*. Second Edition. Duxbury Press, 2002.
- [45] A. Chaganty, A. Nori, and S. Rajamani. Efficiently sampling probabilistic programs via program analysis. In *Artificial Intelligence and Statistics*. 153-160, 2013
- [46] X. Chen, E. Abraham, and S. Sankaranarayanan. Taylor Model Flowpipe Construction for Non-linear Hybrid Systems. In *Real-Time Systems Symposium (RTSS)*, pp. 183-192, IEEE, 2012.
- [47] X. Chen, E. Abraham, and S. Sankaranarayanan. Flow*: An Analyzer for Non-linear Hybrid Systems. In *Computer Aided Verification (CAV)*, LNCS 8044, pp. 258-263, Springer, 2013.
- [48] X. Chen, E. Abraham, and S. Sankaranarayanan. Flow* benchmarks. <https://flowstar.org/benchmarks/>.
- [49] T. Q. Chen, Y. Rubanova, J. Bettencourt, D. K. Duvenaud. Neural ordinary differential equations. *Adv. Neural Inf. Process. Syst.* 31, 6571–6583 (2018).
- [50] A. Cheviakov. Computation of fluxes of conservation laws. *Journal of Engineering Mathematics* 66, 153-173, 2010.

- [51] C. Chicone. *Ordinary Differential Equations with Applications*, 2/e. Texts in Applied Mathematics, Springer-Verlag, 2006.
- [52] Y. Chou, S. Sankaranarayanan. Bayesian Parameter Estimation for Nonlinear Dynamics Using Sensitivity Analysis. In *International Joint Conferences on Artificial Intelligence Organization*, pp 5708-5714, 2019.
- [53] Y. Chou. Detailed experimental set up of [52]. Personal communication, 2022.
- [54] A. Chutinan, B. H. Krogh. Computational techniques for hybrid system verification. *IEEE Trans. Autom. Control*. 48(1): 64-75, 2003.
- [55] F. Coelho, C. Codeco, M. Gabriela, M. Gomes. A bayesian framework for parameter estimation in dynamical models. In *PLoS ONE*, 6, 2011.
- [56] M. A. Colón. Polynomial approximations of the relational semantics of imperative programs. *Science of Computer Programming*, 64(1):76 – 96, 2007.
- [57] D. Cox, J. Little, and D. O’Shea. *Ideals, Varieties, and Algorithms An Introduction to Computational Algebraic Geometry and Commutative Algebra*. Undergraduate Texts in Mathematics, Springer, 2015.
- [58] F. Dahlqvist and T. Schmid. How to write a coequation ((co) algebraic pearls). *9th Conference on Algebra and Coalgebra in Computer Science (CALCO 2021)*, Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.
- [59] V. Danos, J. Feret, W. Fontana, R. Harmer, and J. Krivine. Abstracting the differential semantics of rule-based models: Exact and automated model reduction. In *LICS*, pages 362–381, 2010.
- [60] V. Danos, and T. Ehrhard. Probabilistic coherence spaces as a model of higher-order probabilistic computation. *Inf. Comput.*, 209(6), 966–991, 2011.
- [61] V. Danos, and R. Harmer. Probabilistic game semantics. *ACM Trans. Comput. Log.*, 3(3), 359–382, 2002.
- [62] SC. Dass, J. Lee, K. Lee, J. Park. Laplace based approximate posterior inference for differential equation models. *Stat. Comput.* 27, 679–698, 2017.
- [63] P. Del Moral, A. Doucet, and A. Jasra. Sequential Monte Carlo Samplers. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, 68(3), 411–436, 2006.
- [64] A.P. Dempster. Upper and lower probabilities induced by a multivalued mapping. In *The Annals of Mathematical Statistics*, 38(2):325–339, 1967.
- [65] A. Donzé and O. Maler. Systematic simulations using sensitivity analysis. In *Hybrid Systems: Computation and Control*, LNCS 4416, pages 174–189. Springer, 2007.
- [66] A. Doucet, N. De Freitas, N. Gordon. *Sequential Monte Carlo Methods in Practice*. Springer, 2001.

- [67] A. Dvoretzky, J. Kiefer, J. Wolfowitz. Asymptotic minimax character of the sample distribution function and of the classical multinomial estimator. *Annals of Mathematical Statistics*, 27 (3): 642–669, doi:10.1214/aoms/1177728174, 1956.
- [68] M.G.R. Faes, M. Daub, S. Marelli, E. Patelli, M. Beer. Engineering Analysis With Probability Boxes: A Review on Computational Methods. In *Struct. Safety* 93, paper 102092, 2021.
- [69] W. Feller. *An introduction to probability theory and its applications. Volume II* (Second edition of 1966 original ed.). New York–London–Sydney: John Wiley & Sons, Inc. MR 0270403, 1971.
- [70] S. Ferson, V. Kreinovich, L. Ginzburg, D.S. Myers, K. Sentz. Constructing probability boxes and Dempster-Shafer structures. In *Technical Report*, SAND2002-4015, Sandia Laboratories, January 2003.
- [71] P. Flajolet and R. Sedgewick. Analytic combinatorics: functional equations, rational and algebraic functions. *Research Report RR-4103*, INRIA, 2001.
- [72] M. Forets, A. Pouly. Explicit Error Bounds for Carleman Linearization. *arXiv:1711.02552*, 2017.
- [73] M. Forets, C. Schilling. Reachability of Weakly Nonlinear Systems Using Carleman linearization. In *Proc. of RP 2021*, LNCS 13035:85–99, 2021.
- [74] B. Fornberg and J. A. C. Weideman. A Numerical Methodology for the Painlevé Equations. *Journal of Computational Physics*, 230(15):5957–73, 2011.
- [75] E.B. Fox. *Bayesian Nonparametric Learning of Complex Dynamical Phenomena*. PhD thesis, MIT, 2009.
- [76] A. Galántai. *The theory of Newton’s method*. Journal of Computational and Applied Mathematics, 124(1-2):25–44, 2000.
- [77] T. Gehr, S. Misailovic, and M.T. Vechev. Psi: Exact symbolic inference for probabilistic programs. *Proceedings of the 28th International Conference in Computer Aided Verification (CAV 2016), Toronto*, page 62–83, 2016.
- [78] K. Ghorbal, A. Platzer. Characterizing Algebraic Invariants by Differential Radical Invariants. *TACAS 2014* : 279-294, 2014. Extended version available from <http://reports-archive.adm.cs.cmu.edu/anon/2013/CMU-CS-13-129.pdf>.
- [79] A. Girard and G. Pappas. Verification using simulation. In *Hybrid Systems: Computation and Control*, LNCS 3927, pages 272–286. Springer, 2006.
- [80] M. Girolami. Bayesian inference for differential equations. In *Theoretical Computer Science*, 408(1):4-16, 2008.
- [81] A. Goldsztejn, A. Neumaier. On the Exponentiation of Interval Matrices. 2009. *hal-00411330v1*, 2009.

- [82] I. J. Goodfellow, Y. Bengio, A. Courville. *Deep Learning*. MIT Press, 2016.
- [83] N.D. Goodman and Andreas Stuhlmüller. The design and implementation of probabilistic programming languages. Retrieved 2023-8-31 from <http://dippl.org>.
- [84] N. Goodman, V. Mansinghka, D. Roy, K. Bonawitz, and J.T. Church: a language for generative models. *Proc. Uncertainty in Artificial Intelligence*. 2008.
- [85] A.D. Gordon, T.A. Henzinger, A.V. Nori, S.K Rajamani. Probabilistic programming. *Proceedings of Future of Software Engineering Proceedings (FOSE2014)* pp. 167–181, 2014.
- [86] E. Goubault, J.-H. Jourdan, S. Putot, S. Sankaranarayanan. Finding non-polynomial positive invariants and lyapunov functions for polynomial systems through Darboux polynomials. *ACC 2014: 3571-3578*, 2014.
- [87] F. Gretz, J.P. Katoen, and A. McIver. Operational versus weakest pre-expectation semantics for the probabilistic guarded command language. *Performance Evaluation* 73, 110 – 132., 2014
- [88] R. Grosu, G. Batt, F. Fenton, J. Glimm, C. Le Guernic, S. Smolka, E. Bartocci. From cardiac cells to genetic regulatory networks Proc. of CAV 2011: The 23rd International Conference on Computer Aided Verification, *Lecture Notes in Comput. Sci.*, vol. 6806, Springer, Berlin/Heidelberg, pp. 396-411,2011.
- [89] H. Hansen, C. Kupke, and J. Rutten. Stream differential equations: Specification formats and solution methods. *Log. Methods Comput. Sci.*, 13(1), 2017.
- [90] H. A. Harrington, R. A. Van Gorder. Reduction of dimension for nonlinear dynamical systems. *Nonlinear Dynamics*, 88(1), 715–734, 2017.
- [91] W. Hastings. Monte Carlo sampling methods using Markov chains and their applications. In *Biometrika* 57: 97-109, 1970.
- [92] T. A. Henzinger. The Theory of Hybrid Automata. *Hybrid Systems: Computation and Control*. HSCC 2004. In: Proc.of LICS'96. IEEE Computer Society Press, pp. 278–292, 1996.
- [93] W. Hereman, P.J. Adams, H.L. Eklund, M.S. Hickman, B.M. Herbst. Direct methods and symbolic software for conservation laws of nonlinear equations. In: Yan, Z. (Ed.), *Advances in Nonlinear Waves and Symbolic Computation*. Nova Science Publishers, New York, pp. 19–79, 2009.
- [94] W. Hereman, M. Colagrosso, R. Sayers, A. Ringler, B. Deconinck, M. Nivala, M.S. Hickman. Continuous and discrete homotopy operators and the computation of conservation laws. In: Wang, D., Zheng, Z. (Eds.), *Differential Equations with Symbolic Computation*. Birkhäuser, Basel, pp. 249–285, 2005.
- [95] C. Heunen, O. Kammar, S. Staton, H. Yang. A convenient category for higher-order probability theory. Pages 1–12 of: *Proc. of LICS 2017*.

- [96] W. Hoeffding. Probability inequalities for sums of bounded random variables. In *Journal of the American Statistical Association* 58 (301), 1963.
- [97] P. D. Hoff. *A First Course in Bayesian Statistical Methods*. New York, NY: Springer Science & Business Media, 2009.
- [98] G. Iacobelli, M. Tribastone, and A. Vandin. Differential bisimulation for a Markovian process algebra. In *MFCS*, 2015.
- [99] N. Jansen, C. Dehnert, B.L. Kaminski, J.P. Katoen, and L. Westhofen. Bounded model checking for probabilistic programs. In *Proceedings of the International Symposium on Automated Technology for Verification and Analysis (ATVA 2016)*, 9938:68–85, 2016.
- [100] L. Jaulin, E. Walter. Set inversion via interval analysis for nonlinear bounded-error estimation. In *Automatica*, 29(4):10531064, 1993.
- [101] L. Jaulin. Computing minimal-volume credible sets using interval analysis; application to bayesian estimation. *IEEE Transactions on Signal Processing*, vol. 54, no. 9, pp. 3632-3636, doi: 10.1109/TSP.2006.877676, 2006.
- [102] L. Jaulin, E. Walter. Guaranteed nonlinear parameter estimation via interval computations. In *Interval Computation*, pp. 61–75, 1993.
- [103] R. M. Jungers, P. Tabuada. Non-local Linearization of Nonlinear Differential Equations via Polyflows. *2019 American Control Conference (ACC)*, pp. 1-6, doi: 10.23919/ACC.2019.8814337, 2019.
- [104] H. Khalil. *Nonlinear Systems*, 3/e. Prentice-Hall, 2002.
- [105] B. Klin. Bialgebras for structural operational semantics: An introduction. *Theoretical Computer Science*, 412(38):5043–5069, 2011.
- [106] H. Kong, F. He, X. Song, W. Hung, M. Gu. Exponential-Condition-Based Barrier Certificate Generation for Safety Verification of Hybrid Systems. In: Sharygina N., Veith H. (eds) *Computer Aided Verification. CAV 2013*. LNCS 8044. Springer, Berlin, Heidelberg. <https://doi.org/10.1007/978-3-642-39799-8-17>
- [107] H. Kong, S. Bogomolov, C. Schilling, Y. Jiang, T. A. Henzinger. Invariant Clusters for Hybrid Systems. *HSCC 2017*:163-172, ACM, 2017.
- [108] M. R. Kosorok. *Introduction to Empirical Processes and Semiparametric Inference*. Springer, 2006.
- [109] K. Kowalski, H. Steeb. *Nonlinear Dynamical Systems and Carleman Linearization*. World Scientific, 1991.
- [110] K. Kowalski, W.-H. Steeb. *Nonlinear Dynamical Systems and Carleman Linearization*. World Scientific, 1991.

- [111] D. Kozen. Semantics of probabilistic programs. *Journal of Computer and System Sciences* 22(3), 328–350 (1981)
- [112] S.G. Krantz, H.R. Parks. *Implicit function theorem: history, theory, and applications*. Undergraduate Texts in Mathematics. Springer, 2012.
- [113] W. Kuich and A. Salomaa. *Semirings, Automata, Languages*. Monographs in Theoretical Computer Science: An *EATCS Series*. Springer, 1986.
- [114] F. Lemaire. Les classements les plus généraux assurant l’analyticité des solutions des systèmes orthonomes pour des conditions initiales analytiques. In *Computer Algebra in Scientific Computing, CASC 2002. Proceedings of the Fifth International Workshop on Computer Algebra in Scientific Computing, Yalta, Ukraine*. V.G. Ganzha, E.W.Mayr, E.V.Vorozhtsov (eds.). Technische Universität München, Germany, 2002. <http://www.orcca.on.ca/~lemaire/publications/Casc2002.ps.gz>.
- [115] Y. LeCun, C. Cortes, C. Burges. MNIST handwritten digit database: <http://yann.lecun.com/exdb/mnist/>.
- [116] R.J. LeVeque. *Numerical Methods for Conservation Laws*. Birkhäuser, 1992.
- [117] J. Lintusaari, M.U. Gutmann, R. Dutta, S. Kaski, and J. Corander. Fundamentals and recent developments in approximate bayesian computation. In *Systematic Biology*, 66(1): 66–82, 2017.
- [118] J. D. Lipson. Newton’s method: A great algebraic algorithm. In *Proc. of Symp. on Symbolic and Algebraic Computation*. ACM, 1976.
- [119] C. Mallinger Algorithmic manipulations and transformations of univariate holonomic functions and sequences. *Diplomarbeit, Johannes Kepler Universität Linz*, 1996.
- [120] K. Makino and M. Berz. Rigorous integration of flows and ODEs using Taylor models. In *Proc. of Symbolic-Numeric Computation*, pages 79–84, <https://doi.org/10.1145/1577190.1577206>, ACM, 2009.
- [121] P. Massart. The tight constant in the Dvoretzky–Kiefer–Wolfowitz inequality. *Annals of Probability* 18 (3): 1269–1283, doi:10.1214/aop/1176990746, 1990.
- [122] A. Maurer and M. Pontil. Empirical bernstein bounds and sample-variance penalization. In *Proceedings of the 22nd Conference on Learning Theory (COLT 2009)*, 2009.
- [123] A. Mauroy, I. Mezic. Global Stability Analysis Using the Eigenfunctions of the Koopman Operator. In *IEEE Transactions on Automatic Control*, vol. 61, no. 11, pp. 3356–3369, doi: 10.1109/TAC.2016.2518918, 2016
- [124] A. Mauroy, I. Mezic, Y. Susuki (eds.) *The Koopman Operator in Systems and Control: Concepts, Methodologies, and Applications*. Springer, 2020.

- [125] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, E. Teller. Equations of state by fast computing machines. *J. Chem. Phys.* 21, 1087–1092, 1953.
- [126] S. Milius. A Sound and Complete Calculus for Finite Stream Circuits. In *Proc. of LICS*, pages 421–30. IEEE, 2010.
- [127] R. Milner. *Communication and Concurrency*. Prentice Hall, International Series in Computer Science, 1989.
- [128] I. M. Mitchell, A. M. Bayen, and C. J. Tomlin. A time-dependent Hamilton–Jacobi formulation of reachable sets for continuous dynamic games. *IEEE Transactions on Automatic Control*, 50:947–957, 2005.
- [129] S. Mitsch, A. Platzer. ModelPlex: verified runtime validation of verified cyber-physical system models. *Form. Methods Syst. Des.* 49:33–74, <https://doi.org/10.1007/s10703-016-0241-z>, Springer, 2016.
- [130] D. Moldovan. *Autograph documentation*. <https://github.com/tensorflow/tensorflow/blob/master/tensorflow/python/autograph/g3doc/reference/index.md>.
- [131] R.E. Moore. *Interval Analysis*. Prentice-Hall, Englewood Cliffs, N. J., 1966.
- [132] B. Mor, S. Garhwal, A. Kumar. A Systematic Review of Hidden Markov Models and Their Applications. *Arch Computat Methods Eng* 28, 1429–1448, 2021.
- [133] R. Munos, J. Audibert and C. Szepesvári. Exploration-exploitation tradeoff using variance estimates in multi-armed bandits. *Theoretical Computer Science*, 410.19:1876–1902, 2009.
- [134] P. Narayanan, J. Carette, C. Wren Romano, and R. Zinkov. Probabilistic inference by program transformation in Hakaru (system description). *Proceedings of the 13th International Symposium on Functional and Logic Programming (FLOPS 2016)*, pages 62–79, 2016.
- [135] R. Naz, F.M. Mahomed, D.P. Mason. Comparison of different approaches to conservation laws for some partial differential equations in fluid mechanics. *Appl. Math. Comput.* 205, 212–230, 2008.
- [136] N. S. Nedialkov, K. R. Jackson, G. F. Corliss. Validated solutions of initial value problems for ordinary differential equations. *Appl. Math. Comput.* 105(1): 21–68, Elsevier, 1999.
- [137] A. Nori, C. Hur, S. Rajamani, and S. Samuel. R2: An efficient mcmc sampler for probabilistic programs. *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, page 2476–2482, July 2014.
- [138] D. Novikov and S. Yakovenko. Trajectories of polynomial vector fields and ascending chains of polynomial ideals. *Annales de l’institut Fourier*, 49(2)(1999)563–609, 2011.
- [139] The on-line encyclopedia of integer sequences. <https://oeis.org>.

- [140] P.J. Olver. *Applications of Lie Groups to Differential Equations*, 2/E. Graduate Texts in Mathematics. Springer, 1993.
- [141] D. Pavlovic and M. Escardó. Calculus in coinductive form. In *Proc. of LICS*, pages 408–417. IEEE, 1998.
- [142] A. Platzer. *Dynamic logics of dynamical systems*. CoRR, abs/1205.4788, 2012.
- [143] A. Platzer. Differential-algebraic dynamic logic for differential-algebraic programs. In *Real-Time Systems Symposium (RTSS)*, pp. 183-192, IEEE, 2012. *Log. Comput.* 20, 1, 309–352. (IEEE Cat. No.03CH37475), pp. 2884-2889 Vol.3, 2003.
- [144] D. Poole, W. Hereman. Symbolic computation of conservation laws for nonlinear partial differential equations in multiple space dimensions. *Journal of Symbolic Computation*, 46 (12), pp. 1355 - 1377, 2011.
- [145] D. Pous and J. Rot. Companions, Codensity, and Causality. In *Proc. of FOSSACS*, volume 10203 of LNCS, pages 106–123. Springer, 2017.
- [146] S. Prajna. Barrier certificates for nonlinear model validation. *42nd IEEE International Conference on Decision and Control (IEEE Cat. No.03CH37475)*, pp. 2884-2889 Vol.3, doi: 10.1109/CDC.2003.1273063, 2003.
- [147] S. Prajna, A. Jadbabaie. Safety Verification of Hybrid Systems Using Barrier Certificates. *Hybrid Systems: Computation and Control*. HSCC 2004.
- [148] S. Prajna, A. Papachristodoulou, P. Seiler, P. Parrilo. SOSTOOLS and its control applications. *Positive polynomials in control*, pp. 580- 580, 2005.
- [149] D. Riley, X. Koutsoukos, K. Riley. Reachability analysis of stochastic hybrid systems: A biodiesel production system. *European Journal on Control* 16, 6, 609–623, 2010.
- [150] C. Riquier. *Les systèmes d'équations aux dérivées partielles*. Gauthiers-Villars, Paris, 1910.
- [151] C.P. Robert, G.Casella. *Monte Carlo Statistical Methods*. Springer, 1999.
- [152] W. Rudin. *Principles of mathematical analysis*. International series in pure and applied mathematics, McGraw-Hill, 1976.
- [153] C.J. Rust, G.J. Reid, A.D. Wittkopf. Existence and Uniqueness Theorems for Formal Power Series Solutions of Analytic Differential Systems. *ISSAC 1999*: 105-112, 1999.
- [154] J.J.M.M. Rutten. Elements of Stream Calculus (An Extensive Exercise in Coinduction). *Proc. of MFPS*, volume 45 of ENTCS, pages 358–423. Elsevier, 2001.
- [155] J.J.M.M. Rutten. Behavioural differential equations: a coinductive calculus of streams, automata, and power series. *Theor. Comput. Sci.*, 308(1-3):1–53, 2003.

- [156] J.J.M.M. Rutten. *The Method of Coalgebra: Exercises in Coinduction*. CWI, Amsterdam, The Netherlands, 2019.
- [157] J.J.M.M. Rutten. A coinductive calculus of streams. *Math. Struct. Comput. Sci.*, 15(1):93–147, 2005.
- [158] J.J.M.M. Rutten. An Application of Stream Calculus to Signal Flow Graphs. Proc. of *FMCO*, volume 3188 in *LNCS*, pages 276–291. Springer, 2003.
- [159] J.J.M.M. Rutten. A tutorial on coinductive stream calculus and signal flow graphs. *Theor. Comput. Sci.*, 343(3):443–481. Elsevier, 2005.
- [160] Y. Saad. *Iterative methods for sparse linear systems*. SIAM, 2003.
- [161] C. Sánchez, G. Schneider, W. Ahrendt, W. et al. A survey of challenges for runtime verification from advanced application domains (beyond software). *Form. Methods Syst. Des.* 54, 279–335. <https://doi.org/10.1007/s10703-019-00337-w>, Springer, 2019.
- [162] D. Sangiorgi. *Introduction to Bisimulation and Coinduction*. CUP, 2011.
- [163] S. Sankaranarayanan. Automatic abstraction of non-linear systems using change of bases transformations. *HSCC 2011*: 143-152.
- [164] S. Sankaranarayanan, H. B. Sipma, and Z. Manna. Non-linear Loop Invariant Generation using Gröbner Bases. In *Proceedings of the ACM Symposium on Principles of Programming Languages (POPL)*, 2004.
- [165] S. Sankaranarayanan, A. Chakarov, and S. Gulwani. Static analysis for probabilistic programs: Inferring whole program properties from finitely many paths. *Proceedings of the 34th ACM SIGPLAN conference on Programming language design and implementation.*, 2013.
- [166] S. Sankaranarayanan. Reachability analysis using message passing over tree decompositions. In: Lahiri, S.K., Wang, C. (eds.) *CAV 2020*. LNCS, vol. 12224, pp. 604–628. Springer, 2020.
- [167] S. Sankaranarayanan. Change-Of-Bases Abstractions for Non-Linear Systems. *CoRR abs/1204.4347*, 2012.
- [168] G. M. Scarpello, and D. Ritelli. A Historical Outline of the Theorem of Implicit Functions. *Divulgaciones Matemáticas*, 10(2): 171–180, 2022.
- [169] A. Schönhage. *The fundamental theorem of algebra in terms of computational complexity*. Preliminary Report. Mathematisches Institut der Universität Tübingen, 1982.
- [170] S. Schupp, E. Ábrahám, X. Chen, I. Ben Makhoul, G. Frehse, S. Sankaranarayanan, and S. Kowalewski. Current Challenges in the Verification of Hybrid Systems. In: Proc. of CyPhy’15. Vol. 9361. *Information Systems and Applications*, incl. Internet/Web, and HCI. Springer, pp. 8–24, 2015.

- [171] S. Schupp, E. Ábrahám, I. Ben Makhlof, and S. Kowalewski. HyPro: A C++ Library for State Set Representations for Hybrid Systems Reachability Analysis. In: *Proc. of NFM'17*. Vol. 10227. LNCS. Springer, pp. 288–294, 2017.
- [172] G. Shafer. *A Mathematical Theory of Evidence*. Princeton University Press, 1976.
- [173] R. Stanley. *Enumerative Combinatorics, 2nd edition*. CUP, 2012.
- [174] S. Staton. Commutative semantics for probabilistic programming. In *Proceedings of the 26th European Symposium on Programming (ESOP2017)*, Uppsala, Sweden, 2017.
- [175] S. Staton, F. Wood, H. Yang, C. Heunen, and O. Kammar. Semantics for probabilistic programming: higher-order functions, continuous distributions, and soft constraints. *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, 2016.
- [176] A. Tiwari, G. Khanna. Nonlinear systems: Approximating reach sets. In *HSCC 2004*:600-614, ACM, 2004.
- [177] M. Tribastone and A. Vandin. Speeding up stochastic and deterministic simulation by aggregation: an advanced tutorial. *Winter Simulation Conference (WSC)*, Gothenburg, Sweden, 2018 pp. 336-350, doi: 10.1109/WSC.2018.8632364.
- [178] B. Van Der Pol. The nonlinear theory of electric oscillations. *Proceedings of the Institute of Radio Engineers*, 22: 1051-1086, 1934.
- [179] J. Vanlier, C.A. Tiemann, P.A.J. Hilbers, N.A.W. van Riel. Parameter uncertainty in biochemical models described by ordinary differential equations. In *Mathematical Biosciences*, 246(2):305 – 314, 2013.
- [180] M. Wetzlinger. Personal communication. June 2022.
- [181] Wikipedia. Angle of attack. https://en.wikipedia.org/wiki/Angle_of_attack, 11 October 2023.
- [182] J. Winter. *Coalgebraic Characterizations of Automata-Theoretic Classes*. PhD thesis, Radboud Universiteit Nijmegen, 2014.
- [183] T. Wolf. A comparison of four approaches to the calculation of conservation laws. *Europ. J. Appl. Math.* 13, pp. 129–152, 2002.
- [184] B. Xue, M. Fränzle, N. Zhan. Under-Approximating Reach Sets for Polynomial Continuous Systems. *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (part of CPS Week), HSCC'18*, pp. 51–60, <https://doi.org/10.1145/3178126.3178133>, ACM, 2018.
- [185] T. Zhang, Q. Yin, B. Caffo, Y. Sun, and D. Boatman-Reich. Bayesian Inference of High-Dimensional, Cluster-Structured Ordinary Differential Equation Models With Applications to Brain Connectivity Studies. *The Annals of Applied Statistics* 11, 868–897, 2017.

Appendix A

Proofs of Chapter 2

A.1 Proof of Corollary 1

We can conservatively extend any given monomial order \prec on $(X \cup D)^\otimes$ by introducing an *elimination order* \prec_{el} on $(a \cup X \cup D)^\otimes$, as follows. First, order the elements in a as $a_1 < \dots < a_s$. Then, for $\alpha, \alpha' \in a^\otimes$ and $\beta, \beta' \in (X \cup D)^\otimes$, we let $\alpha\beta \prec_{\text{el}} \alpha'\beta'$ if and only if either $\alpha \prec_{\text{lex}} \alpha'$ or $(\alpha = \alpha' \text{ and } \beta \prec \beta')$. In other words, monomials are first compared lexicographically in their a -part and in case of ties, in their $(X \cup D)$ -part according to the original order. Any $\Delta \subseteq \mathbb{R}[X \cup D]$ that is a Groebner basis in the ring $\mathcal{P} = \mathbb{R}[X \cup D]$ w.r.t. the original order is also a Groebner basis in the larger ring $\mathbb{R}[a \cup X \cup D]$ w.r.t. the elimination order: this is an immediate consequence of Buchberger's Criterion [57, Ch.2, §6, Th.6], as the S-polynomials of Δ w.r.t. the original order and the elimination order coincide. **PROOF OF COROLLARY 1.** First, by

the linearity of total derivative, it is easily seen that for each $x_i \in X$, template π and $v \in \mathbb{R}^s$: $D_{x_i}(\pi[v]) = (D_{x_i}\pi)[v]$. Now, in order to prove the inclusion in the statement, consider any $v \in V$. We have:

$$\begin{aligned}
 S(\mathbf{div}(\Pi[v])) &= S\left(\sum_i D_{x_i}(\pi_i[v])\right) \\
 &= S\left(\left(\sum_i D_{x_i}\pi_i\right)[v]\right) \\
 &= S((\mathbf{div}\Pi)[v]) \\
 &= (S(\mathbf{div}\Pi))[v] && \text{(A.1)} \\
 &= 0 && \text{(A.2)}
 \end{aligned}$$

where (A.1) follows from Lemma 3 and (A.2) from the definition of V . Therefore $\mathbf{div}(\Pi[v]) \in \langle \Sigma \rangle$ and, by Lemma 1, $\mathbf{div}(\Pi[v]) \in \mathcal{I}nv(\Sigma)$. Assume now that $\langle \Sigma \rangle = \mathcal{I}(\mathcal{V}(\Sigma))$, we prove the reverse inclusion. Let $(p_1, \dots, p_n) \in \text{CL}(\Sigma, Z)$. By assumption, there is $v \in \mathbb{R}^s$ such that $(p_1, \dots, p_n) = (\pi_1[v], \dots, \pi_n[v]) = \Pi[v]$. We

prove that $v \in V$, that is $(S(\operatorname{div} \Pi))[v] = 0$. Indeed

$$(S(\operatorname{div} \Pi))[v] = S((\operatorname{div} \Pi)[v]) \quad (\text{A.3})$$

$$= S\left(\sum_i D_{x_i} \pi_i[v]\right)$$

$$= S\left(\sum_i D_{x_i} (\pi_i[v])\right)$$

$$= S(\operatorname{div} (\Pi[v]))$$

$$= 0 \quad (\text{A.4})$$

where (A.3) follows again from Lemma 3 and (A.4) from the fact that by hypothesis $\operatorname{div} (\Pi[v]) \in \mathcal{I}nv(\Sigma)$, hence $\operatorname{div} (\Pi[v]) \in \langle \Sigma \rangle$ by Lemma 1. \square

A.2 Proof of Proposition 2

We first introduce Riquier's analyticity theorem. An *initial data specification* $\rho : \mathcal{P}a(\Sigma) \rightarrow \mathbb{R}$ is said to be *analytic at* $x^0 \in \mathbb{R}^n$ if, for each $u \in \mathcal{U}$, the following power series¹ defines a real analytic function of x in a neighborhood of x^0

$$w^\rho(x) \triangleq \sum_{u_\zeta \in \mathcal{P}a(\Sigma)} \frac{\rho(u_\zeta)}{\zeta!} (x - x^0)^\zeta. \quad (\text{A.5})$$

We report below Riquier's analyticity theorem, seen as a special case of the version given by Lemaire [114, Th.1].

Theorem A.2.1 (Riquier's analyticity theorem). *Let Σ be passive orthonomic w.r.t. a ranking which is Riquier and weakly orderly. For any initial data specification ρ analytic at x^0 there is a unique analytic solution U of Σ around x^0 such that $\frac{\partial^{|\tau|}}{\partial \tau} U^i(x^0) = \rho(u_\tau^i)$ for each $u_\tau^i \in \mathcal{P}a(\Sigma)$.*

PROOF OF PROPOSITION 2. We show that Σ is D -locally solvable, analytically. Partition D as $D = D_0 \cup D_1$, where D_1 are the leading derivatives of Σ , and D_0 are the remaining derivatives, which must therefore be parametric. Let $(x^0, u_D^0) = (x^0, u_{D_0}^0, u_{D_1}^0) \in \mathcal{V}(\Sigma)$. Below, we will denote the component of u_D^0 corresponding to $u_\tau^i \in D$ by $u_\tau^{0,i}$. Define an initial data specification $\rho : \mathcal{P}a(\Sigma) \rightarrow \mathbb{R}$ as follows:

$$\rho(u_\tau^i) \triangleq \begin{cases} u_\tau^{0,i} & \text{if } u_\tau^i \in D_0 \\ 0 & \text{otherwise.} \end{cases}$$

For each $u \in \mathcal{U}$, the corresponding function $w^\rho(x)$ defined in (A.5) is clearly analytic around x^0 — in fact, a polynomial — hence ρ is analytic at x^0 . Riquier's Theorem A.2.1 ensures therefore the existence of an analytic solution of the corresponding initial value problem, that is an analytic function $U = (U^1, \dots, U^m)$ around x^0 such that $\frac{\partial^{|\tau|}}{\partial \tau} U^i(x^0) = \rho(u_\tau^i)$ for each $u_\tau^i \in \mathcal{P}a(\Sigma)$. In particular, for each $u_\tau^i \in D_0$, we have $\frac{\partial^{|\tau|}}{\partial \tau} U^i(x^0) = u_\tau^{0,i}$. In other words, $U_{D_0}(x^0) = u_{D_0}^0$.

¹Here for $\zeta = x_1^{k_1} \cdots x_n^{k_n}$ we let $\zeta! \triangleq k_1! \cdots k_n!$.

Consider now any leading derivative $u_\tau^i \in D_1$, with $u_\tau^i + f \in \Sigma$. By assumption, $f = f(x, u_{D_0})$, that is f does not depend on u_{D_1} . Then we have

$$\begin{aligned} \frac{\partial^{|\tau|}}{\partial \tau} U^i(x^0) &= -f(x^0, U_{D_0}(x^0)) \\ &= -f(x^0, u_{D_0}^0) \\ &= u_\tau^{0,i} \end{aligned}$$

where the first equality follows because U is a solution of Σ , the second one from $U_{D_0}(x^0) = u_{D_0}^0$ and the last one because $(x^0, u_D^0) \in \mathcal{V}(\Sigma)$, hence $u_\tau^{0,i} + f(x^0, u_D^0) = 0$. Overall, we have shown that $U_D(x^0) = u_D^0$. Since $(x^0, u_D^0) \in \mathcal{V}(\Sigma)$ is arbitrary, this proves that Σ is D -locally solvable, analytically.

The rest of the proof is exactly like that of Proposition 1. □

Appendix B

Proofs and additional details of Chapter 3

B.1 On-the-fly computation of Arnoldi iteration

Let us describe a convenient way of building the subspace \mathcal{K}_m and the matrices V, B, H_m .

1. For brevity, let $L := A^T$. Note that, as $L^{j+1}v = L(\mathbf{L}^j v)$, all that is needed to build the set spanning \mathcal{K}_m in (3.13) is the vector v and access to the matrix-vector multiplication function $u \mapsto Lu$. In fact, there exist effective and numerically stable algorithms that, given a handle to such a function, will build both the orthonormal basis V of \mathcal{K}_m and the matrix $H_m = V^T L V$. For example, one has the *Arnoldi algorithm* [160, Ch.6]. In particular, the matrix H_m is built by Arnoldi incrementally, along with the vectors v_j as they are generated, without having to perform the matrix multiplications $V^T L V$ explicitly. The resulting matrix H_m has the additional nice property of being upper Hessenberg: that is, upper triangular except possibly for the presence of nonzero elements in the first sub-diagonal. The matrix V , on the other hand, need not be sparse: in fact, it is typically observed that the vectors v_j tend to get dense as j approaches m . Moreover, these vectors must be kept stored as they are generated until the termination of the algorithm. This is an unpleasant characteristic of Arnoldi; mitigations are possible, see [160].
2. Concerning the the function $u \mapsto A^T u$, note that, from (3.10)

$$\mathcal{L}(u^T \alpha) = \alpha^T A^T u + \psi^T B^T u. \quad (\text{B.1})$$

This implies that, for each $u \in \mathbb{R}^M$, the vector $A^T u$ can be obtained by first taking the Lie derivative of the function $u^T \alpha$ and then collecting the nonzero coefficients of the α_i 's ($1 \leq i \leq M$) in this derivative. As a consequence, the matrix-vector multiplication function $u \mapsto A^T u$ can be computed “on the fly”, without building the whole A explicitly.

We also briefly outline a method for computing the remainder function $h(x)$. The vectors $v_m^T B$ and r_m appearing in the definition (3.22) of $h(x)$ can be computed relying

on Lie derivatives only, basically as discussed in the previous subsection. In particular: (a) $v_m^T B$ is built by collecting the nonzero coefficients of the ψ_i functions in the Lie derivative $\mathcal{L}(v_m^T \alpha)$. As such derivative is already invoked in the last iteration ($j = m$) of the Arnoldi algorithm to build $A^T v_m$, the vector $v_m^T B$ comes at no additional cost; (b) r_m can be obtained by running an extra iteration of Arnoldi ($j = m + 1$) and letting $r_m = w_{m+1}$.

B.2 Proofs

PROOF OF THEOREM 3. Let us first note that every vector v_j of the basis V can be expressed as a product of v with a certain polynomial of the matrix A^T of degree not greater than $j - 1$. More precisely, for each $j = 1, \dots, m$, there is a univariate polynomial p_j such that $v_j = p_j(A^T)v$ and $\deg(p_j) \leq j - 1$: this can be easily proven by induction on j .

Now, let $R = [r_1 | \dots | r_m]$ be the $M \times m$ matrix whose j -th column r_j is the projection of $A^T v_j$ onto \mathcal{K}_m^\perp : explicitly, $r_j := A^T v_j - VV^T A^T v_j$. Note that $r_j = r_m$ had been introduced earlier on, when defining $h(x)$; cf. (3.22). The proof of [28, Th.4] (pp. 9-10) shows that

$$\epsilon_g(t; x_0) = \|v\|_2 \int_0^t e_1^{(t-\tau)H_m^T} \left(V^T B \psi(x(\tau; x_0)) + R^T \alpha(x(\tau; x_0)) \right) d\tau \quad (\text{B.2})$$

where $e_1^{(t-\tau)H_m^T}$ is the first row of the exponential matrix $e^{(t-\tau)H_m^T}$ (note that our basis α is denoted ‘ ϕ ’ in the notation of [28]). Now we use our previous observation that $v_j = p_j(A^T)v$, for a certain polynomial p_j of degree $\leq j - 1$ to establish two facts. (a) Applying also (3.12), we have that, for each $1 \leq j \leq m - 1$: $v_j^T B = (p_j(A^T)v)^T B = v^T p_j(A)B = 0$. In other words, the first $m - 1$ rows of $V^T B$ are zero. (b) At the same time, also $r_1 = r_2 = \dots = r_{m-1} = 0$, that is the first $m - 1$ rows of R^T are zero. To see this, for $1 \leq j \leq m - 1$, recall that $\deg(p_j) \leq j - 1$ so that $A^T v_j = A^T p_j(A^T)v$ can be expressed as a linear combination of the $(A^T)^i v$ for $i = 0, \dots, m - 1$: as such $A^T v_j \in \mathcal{K}_m$ hence its orthogonal projection r_j onto \mathcal{K}_m^\perp is zero. From these facts (a) and (b), it follows that (B.2) reduces to

$$\begin{aligned} \epsilon_g(t; x_0) &= \|v\|_2 \int_0^t e_{1m}^{(t-\tau)H_m^T} \left(v_m^T B \psi(x(\tau; x_0)) + r_m^T \alpha(x(\tau; x_0)) \right) d\tau \\ &= \|v\|_2 \int_0^t e_{1m}^{(t-\tau)H_m^T} h(x(\tau; x_0)) d\tau. \end{aligned}$$

From the above equality, (3.23) follows by applying basic properties of integrals and norms.

Concerning (3.24), note that $|e_{1m}^{(t-\tau)H_m^T}| \leq \|e^{(t-\tau)H_m^T}\| \leq D e^{-(t-\tau)\lambda}$, for a suitable positive D and nonnegative λ s.t. $-\lambda$ is \geq than the real part of any eigenvalue of H_m , see e.g. [104, Th.4.5]. The thesis follows by noting that $e^{-(t-\tau)\lambda} \leq 1$. \square

We refer the reader to [104] for the definition of *exponentially stable* system at the origin.

Corollary B.2.1 (error on infinite time horizon). *Suppose the functions in α and ψ are polynomials, and that $h(0) = 0$. Suppose H_m is stable, and that the system (3.3)*

is exponentially stable at the origin. Then there are positive constants G and λ and a region of exponential stability at the origin such that for each x_0 in this region and each t large enough

$$|\epsilon_g(t; x_0)| \leq G \frac{(1 - e^{-\lambda t})}{\lambda}. \quad (\text{B.3})$$

Therefore $|\epsilon_g(t; x_0)|$ is bounded as $t \rightarrow +\infty$.

PROOF. Since $h(0) = v_m^T B \psi(0) + r_m \alpha(0) = 0$, the function $h(x)$ in (3.22) is a polynomial missing the constant term, that is $h(x) = \sum_{i=1}^k \mu_i \gamma_i$ for some coefficients $\mu_i \in \mathbb{R}$ and monomials $\gamma_i \neq 1, i = 1, \dots, k$. Consider the ∞ -norm of $x(t; x_0)$: by exponential stability, there are positive constants E, λ such that for each t large enough we must have $\|x(t; x_0)\|_\infty \leq E e^{-\lambda t} < 1$, hence $|h(x(t; x_0))| \leq C(t) := (\sum_k |\mu_k|) E e^{-\lambda t}$. Let $G := \|v\|_2 D(\sum_k |\mu_k|) E$. Then by (3.24), $\epsilon_g(t; x_0) \leq G \int_0^t e^{-\lambda \tau} d\tau = G(1 - e^{-\lambda t})/\lambda$. \square

The constants G and λ can be readily calculated from the Jordan decomposition of H_m , which is in turn easy to obtain for a relatively small m ; see [104, pp.135-136]. A region of exponential stability can be determined by applying methods based on Lyapunov functions [104, Ch.4].

PROOF OF PROPOSITION 3. Let Λ be the $m \times m$ diagonal matrix with $\lambda_1, \dots, \lambda_m$ in the main diagonal. Then definition (3.27) can be re-written as follows, where we let $\tilde{\phi} := (\tilde{\phi}_1, \dots, \tilde{\phi}_m)^T = U^T \alpha$ denote the vector of approximate Koopman eigenfunctions.

$$\begin{aligned} g_K(t; x_0) &:= \sum_{i=1}^m \tilde{v}_i e^{\lambda_i t} \tilde{\phi}_i(x_0) \\ &= \tilde{v}^T e^{\Lambda t} \tilde{\phi}(x_0) \\ &= (v^T W)(e^{\Lambda t})(U^T \alpha(x_0)). \end{aligned} \quad (\text{B.4})$$

Now $W e^{\Lambda t} = W(I + \Lambda t + \dots + \frac{(\Lambda t)^k}{k!} + \dots) = (W + W \Lambda t + \dots + \frac{W(\Lambda t)^k}{k!} + \dots)$. Each term of the last summation can be written as $\frac{((\Lambda t)^k W^T)^T}{k!} = \frac{(W^T (\Lambda t)^k)^T}{k!} = \frac{(\Lambda t)^k W}{k!}$. Hence $W e^{\Lambda t} = e^{\Lambda t} W$. Now $e^{\Lambda t} (W U^T \alpha(x_0))$ is precisely the unique solution of the system identified in the statement. From this and from (B.4) the first part of the thesis follows. Finally, if A is diagonalisable all of its eigenvalues are non-defective, hence $m = M$ and U, W are invertible square matrices. From $W^T U = I$ we get $(W^T U)^T = U^T W = I$ hence $(U^T W)^{-1} = W^{-1} (U^T)^{-1} = I$. Therefore $W U^T = W (W^{-1} (U^T)^{-1}) U^T = I$, which implies $z_0 = \alpha(x_0)$. \square

PROOF OF PROPOSITION 4. Let Λ be the diagonal matrix with the eigenvalues $\hat{\lambda}_1, \dots, \hat{\lambda}_m$ of H in the main diagonal. Then (3.33) can be re-written as follows.

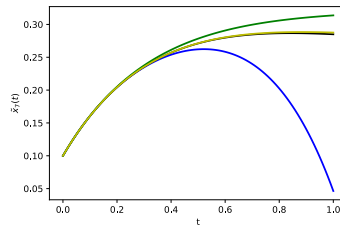
$$\begin{aligned} \hat{g}_K(t; x_0) &:= \sum_{i=1}^m \hat{v}_i e^{\hat{\lambda}_i t} \hat{\phi}_i(x_0) \\ &= \hat{v}^T e^{\Lambda t} \hat{\phi}(x_0) \\ &= (v^T V \hat{W})(e^{\Lambda t})(\hat{U}^T V^T \alpha(x_0)). \end{aligned} \quad (\text{B.5})$$

Just as in the proof of Proposition 3, one sees that $W e^{\Lambda t} = e^{H^T t} W$. Moreover, from the identity $\hat{W}^T \hat{U} = I$ implies $\hat{U}^T \hat{W} = I$, which in turn implies

$\widehat{W}^{-1}(\widehat{U}^T)^{-1} = I_m$. Hence $\widehat{W}\widehat{U}^T = \widehat{W}(\widehat{W}^{-1}(\widehat{U}^T)^{-1})\widehat{U}^T = I_m$. By definition, $y(t) := We^{\Lambda t}\widehat{U}^TV^T\alpha(x_0) = e^{H^T t}V^T\alpha(x_0)$ is the unique solution of the system $\dot{y} = H^Ty$ with the initial condition $y(0) = y_0 = V^T\alpha(x_0)$. Hence $\widehat{y}(t)$ coincides with the $y(t; y_0)$ in (3.34). Then from (B.5), $\widehat{g}_K(t; x_0) = v^T V\widehat{y}(t) = v^T V y(t; y_0) = \widehat{g}(t; y_0)$. \square

B.3 Further details on experiments

Plots for x_7 of system LL



Reachsets computation In this section we provide some additional details on the experiments we performed in order to compare `CKR` with `CORA` and `Flow*`. Regarding Table 3.1, in addition to the considered time horizon, we also report m , the approximation order used respectively by the three algorithms. However, it is worth pointing out that m has slightly different meanings for each of them. For `Flow*`, m is the order of the Taylor expansion of the flow map, computed via Lie derivatives [47]. Instead, for `CORA`, it represents the number of terms of the Taylor series of the matrix exponential that expresses the solution of the linear system used to approximate the original nonlinear dynamics. Finally, in `CKR`, m is the approximation order in the sense of Theorem 2, which coincides with the dimension of the reduced system (3.15).

For `CKR`, we use $m = 4$ or $m = 5$ in all runs, as these values are sufficient to guarantee fairly accurate results. In the case of `Flow*` and `CORA`, we set $m = 4, 8, 10$ and check how the results vary in terms of accuracy. Only `Flow*` computes less accurate reachsets with $m = 4$ (for system (3.41)(b)); in the remaining cases there seems to be no significant difference. Anyway, for both `Flow*` and `CORA` it is also possible to tune m semi-automatically.

Another relevant parameter to be set in our experiments for reachsets computation is *stepsize* Δ . For both `Flow*` and `CORA`, the stepsize has been empirically chosen in the range $[0.001, 0.05]$ so as to maximize the breakdown time — except in the case of the Lotka-Volterra system (3.41)(b), where for `CORA` we accepted a slightly shorter breakdown time in exchange of a considerable improvement in accuracy. Instead, for `CKR` we fixed the initial timestep to $\Delta = 0.05$, although in this case the timestep is adaptive, thus it may change during execution.

One must also note that, although it never happens in the experiments of Table 3.1, `CKR` breaks down as well. For instance, in the case of `VdP`, `CKR` breaks down at $T = 7.88$ seconds when setting the initial timestep to 0.05 and $m = 5$. In the case of system (3.41)(b), with initial timestep 0.01 and $m = 5$, it breaks down at $T = 6.58$. In other cases, as the execution proceeds, it may happen that the algorithm does not break down, yet the stepsize is reduced more and more slowing down the execution.

Finally, in few cases, `CKR` seems to never break down no matter how large is the time horizon. This happens typically with stable systems, like (3.43).

Concerning the configuration of the parameters of `CORA`, in the experiments reported in Table 3.1 we have used the default representation of reachsets (zonotopes) for both system (3.41)(b) and system (3.43). Only in the case of `VdP`, the use of the more sophisticated *polynomial zonotopes* gave significant benefits, allowing the execution to be completed up to $T = 5$. We must note that, in this case, the average area of `CORA`'s reachsets has been computed by first converting them into zonotopes; as a result this area might be slightly overestimated. More generally, the bad results (i.e., large over-approximations) for `CORA` are mainly due to the size of the initial set. Indeed, for the employed linearization algorithm, the dynamics within the set differ too much, so that at some point in time the abstraction error becomes too large and the analysis terminates prematurely [180].

Appendix C

Proofs and additional details of Chapter 4

C.1 Proof

Here we show that the shuffle product is well-behaved; the other products are dealt with similarly. To this aim, we need a preliminary lemma.

Lemma C.1.1. *For every $m_1, m_2 \in \mathcal{M}$, it holds that $F_{\otimes}[m_1; m_2] = \delta_{\otimes}(m_1 m_2)$.*

PROOF. By induction on m_1 . The base is $m = 1$: by (4.15), $F_{\otimes}[1; m_2] = \delta_{\otimes}(m_2) = \delta_{\otimes}(1 m_2)$. For the inductive step, assume $m_1 \neq 1$ and let x_i be the variable with the lowest index in $m_1 m_2$; assume $m_2 = x_i m'_2$ (the proof can be done similarly if $m_1 = x_i m'_1$). Then:

$$\begin{aligned}
 F_{\otimes}[m_1; m_2] &= \delta_{\otimes}(m_1) \cdot m_2 + m_1 \cdot \delta_{\otimes}(x_i m'_2) && \text{by def. of } F_{\otimes} \\
 &= \delta_{\otimes}(m_1) \cdot m_2 + m_1 \cdot F_{\otimes}[x_i; m'_2] && \text{by (4.13)} \\
 &= \delta_{\otimes}(m_1) \cdot x_i m'_2 + m_1 \cdot (\delta_{\otimes}(x_i) \cdot m'_2 + x_i \cdot \delta_{\otimes}(m'_2)) && \text{by def. of } F_{\otimes} \\
 &= \delta_{\otimes}(m_1) \cdot x_i m'_2 + \delta_{\otimes}(x_i) \cdot m_1 m'_2 + x_i m_1 \cdot \delta_{\otimes}(m'_2) \\
 &= \delta_{\otimes}(x_i) \cdot m_1 m'_2 + x_i \cdot (\delta_{\otimes}(m_1) \cdot m'_2 + m_1 \cdot \delta_{\otimes}(m'_2)) \\
 &= \delta_{\otimes}(x_i) \cdot m_1 m'_2 + x_i \cdot F_{\otimes}[m_1; m'_2] && \text{by def. of } F_{\otimes} \\
 &= \delta_{\otimes}(x_i) \cdot m_1 m'_2 + x_i \cdot \delta_{\otimes}(m_1 m'_2) && \text{by induction} \\
 &= F_{\otimes}[x_i; m_1 m'_2] && \text{by def. of } F_{\otimes} \\
 &= \delta_{\otimes}(x_i m_1 m'_2) && \text{by (4.13)} \\
 &= \delta_{\otimes}(m_1 m_2) && \square
 \end{aligned}$$

Proposition C.1.1. \otimes is well-behaved.

PROOF. Recall from Example 8 that $F_{\otimes} = y_2 y_3 + y_1 y_4$. Then:

- property (4.15) is satisfied, since $F_{\otimes}[1; q] = 0 \cdot q + 1 \cdot \delta_{\otimes}(q) = \delta_{\otimes}(q)$;

- property (4.16) follows from Lemma C.1.1. Indeed, $F_{\otimes}[x_i m_1; m_2] = \delta_{\otimes}(x_i m_1 m_2) = \delta_{\otimes}(m_1 x_i m_2) = F_{\otimes}[m_1; x_i m_2]$;

- property (4.17) holds. Indeed:

$$\begin{aligned}
F_{\otimes}[\sum_{i \in I} r_i m_i; q] &= \delta_{\otimes}(\sum_{i \in I} r_i m_i) \cdot q + (\sum_{i \in I} r_i m_i) \cdot \delta_{\otimes}(q) && \text{by def. of } F_{\otimes} \\
&= (\sum_{i \in I} r_i \delta_{\otimes}(m_i)) \cdot q + (\sum_{i \in I} r_i m_i) \cdot \delta_{\otimes}(q) && \text{by (4.14)} \\
&= \sum_{i \in I} r_i (\delta_{\otimes}(m_i) \cdot q) + \sum_{i \in I} r_i (m_i \cdot \delta_{\otimes}(q)) \\
&= \sum_{i \in I} r_i (\delta_{\otimes}(m_i) \cdot q + m_i \cdot \delta_{\otimes}(q)) \\
&= \sum_{i \in I} r_i F_{\otimes}[m_i; q] && \text{by def. of } F_{\otimes}
\end{aligned}$$

- property (4.18) trivially holds. □

PROOF OF LEMMA 5. We proceed by structural induction on p . We have two base cases:

1. $p = 0$:

$$\begin{aligned} \delta_\pi(p \cdot q) &= \delta_\pi(0) && \text{by def. of product in } \mathcal{P} \\ &= 0 && \text{by (4.14)} \\ &= F_\pi[0; q] && \text{by (4.17)}. \end{aligned}$$
2. $p = m \in \mathcal{M}$: If $m = 1$, we trivially conclude by (4.15), since $p \cdot q = q$. Otherwise, we consider a second structural induction on q . The non-trivial base case of this second induction is for $q = m' \in \mathcal{M}$. Let x_i be the variable with the smallest index in $m \cdot m'$ and m'' be $m \cdot m'$ with one occurrence of x_i removed. Then $\delta_\pi(m \cdot m') = \delta_\pi(x_i \cdot m'')$, by commutativity and associativity in \mathcal{M} . Now
 - if $m'' = 1$, then $m' = 1$ (i.e., that $q = 1$) and $m = x_i$; then $\delta_\pi(x_i \cdot m'') = \delta_\pi(x_i) = F_\pi[m''; x_i] = F_\pi[m; m']$, by identity of the product, (4.15) and (4.18);
 - otherwise, $\delta_\pi(x_i \cdot m'') = F_\pi[x_i; m''] = F_\pi[m; m']$, by (4.13) and (4.16) (applied $|m| - 1$ times).

For the second inductive step, let $q = \sum_{j \in J} r_j m_j$, for $|J| > 0$. We have

$$\begin{aligned} \delta_\pi(p \cdot q) &= \delta_\pi(\sum_{j \in J} r_j (m \cdot m_j)) && \text{by distributivity in } \mathcal{P} \\ &= \sum_{j \in J} r_j \delta_\pi(m \cdot m_j) && \text{by (4.14)} \\ &= \sum_{j \in J} r_j F_\pi[m; m_j] && \text{by the base case for } q \text{ (} q \text{ a monomial)} \\ &= \sum_{j \in J} r_j F_\pi[m_j; m] && \text{by (4.18)} \\ &= F_\pi[q; m] && \text{by (4.17)} \\ &= F_\pi[p; q] && \text{by (4.18)}. \end{aligned}$$

For the inductive step, let $p = \sum_{i \in I} r_i m_i$, for $|I| > 0$. Then:

$$\begin{aligned} \delta_\pi(p \cdot q) &= \delta_\pi(\sum_{i \in I} r_i (m_i \cdot q)) && \text{by distributivity in } \mathcal{P} \\ &= \sum_{i \in I} r_i \delta_\pi(m_i \cdot q) && \text{by (4.14)} \\ &= \sum_{i \in I} r_i F_\pi[m_i; q] && \text{by the second base case for } p \text{ (} p \text{ a monomial)} \\ &= F_\pi[p; q] && \text{by (4.17)}. \end{aligned}$$

□

Let us recall the notation introduced in the proof of Theorem 4. Given a polynomial substitution (a map from variables to polynomials) ζ , and a monomial $m = x_{i_1} \cdots x_{i_k}$, we let $m\zeta$ denote the polynomial $\zeta(x_{i_1}) \cdots \zeta(x_{i_k})$. Similarly, given a stream substitution (a map from variables to streams) ξ , we let $m\xi$ denote the stream $\xi(x_{i_1}) \pi \cdots \pi \xi(x_{i_k})$.

PROOF OF PROPOSITION 5. We have to prove two properties for ν , for every $p \in \mathcal{P}$.

1. $o_\rho(p) = \nu(p)(0)$. To this aim, let us first prove that $o_\rho(m) = \nu(m)(0)$, for every $m \in \mathcal{M}$; the proof is by induction on m .

- **Base** ($m = 1$). $o_\rho(1) = 1 = 1_\pi(0) = \nu(1)(0)$, where the first equality holds by definition, the second one by Def. 8(2), and the last one by homomorphism of ν .
- **Induction** ($m = x_i m_1$). $\nu(x_i m_1)(0) = (\nu(x_i) \pi \nu(m_1))(0) = (\nu(x_i)(0))(\nu(m_1)(0)) = \rho(x_i) o_\rho(m_1) = o_\rho(m)$, where the first equality holds by homomorphism, the second one by Def. 8(1), the third one since ν respects ρ and by induction, and the last one by definition.

Now, let $p = \sum_i r_i m_i$. Then, $o_\rho(p) = \sum_i r_i o_\rho(m_i) = \sum_i r_i (\nu(m_i)(0)) = \nu(p)(0)$, where the first equality holds by definition of o_ρ , the second one by this claim for monomials, and the third one by algebra homomorphism of ν .

2. $\nu(\delta_\pi(p)) = \nu(p)'$. We proceed by cases on p .

$p = 0$. In this case, $\nu(\delta_\pi(0)) = \nu(0) = 0 = \nu(0)'$, where the first equality holds by (4.14), the second one by homomorphism, and the third one by definition of 0 and homomorphism of ν .

$p = m \in \mathcal{M}$. The proof is by induction on m .

- $m = 1$. In this case, let G be $\sum_i r_i m_i$, where the m_i 's are monomials in y_1 . Then, by letting ζ be the substitution that maps y_1 to 1, we have that:

$$\begin{aligned}
 \nu(\delta_\pi(1)) &= \nu(G(1)) && \text{by (4.11)} \\
 &= \sum_i r_i \nu(m_i \zeta) && \text{by def. of } G \text{ and homomorphism of } \nu \\
 &= \sum_i r_i \nu(1) && \text{since every } m_i \text{ is a monomial only in } y_1 \\
 &= \sum_i r_i 1_\pi && \text{by homomorphism of } \nu \\
 &= G(1_\pi) && \text{by def. of } G \text{ and the fact that } m_i \text{'s are monomials only in } y_1 \\
 &= 1'_\pi && \text{by Def. 8(2)} \\
 &= \nu(1)' && \text{by homomorphism of } \nu.
 \end{aligned}$$

- $m = x_i$. In this case, $\nu(\delta_\pi(x_i)) = \nu(\mathcal{D}(x_i)) = \nu(x_i)'$, where the first equality holds by (4.12), and the second one since ν respects \mathcal{D} .
- $m = x_i \bar{m}$, for $\bar{m} \neq 1$ and x_i the variable with smallest index in m . Let F be $\sum_j r_j m_j$, where the m_j 's are monomials in the variables x, y_1, \dots, y_4 . For every such a monomial m_j , let us denote by $e_{j,k}$ the exponent of variable y_k in m_j , for $k = 0, \dots, 4$, where we let $y_0 = x$; i.e. $m_j = x^{e_{j,0}} y_1^{e_{j,1}} y_2^{e_{j,2}} y_3^{e_{j,3}} y_4^{e_{j,4}}$. Furthermore, let us define the substitutions ζ and ξ as follows:

$$\begin{array}{ll}
 \zeta : & x \mapsto x & \xi : & x \mapsto \nu(x) \\
 & y_1 \mapsto x_i & & y_1 \mapsto \nu(x_i) \\
 & y_2 \mapsto \mathcal{D}(x_i) & & y_2 \mapsto \nu(x_i)' \\
 & y_3 \mapsto \bar{m} & & y_3 \mapsto \nu(\bar{m}) \\
 & y_4 \mapsto \delta_\pi(\bar{m}) & & y_4 \mapsto \nu(\bar{m})'.
 \end{array}$$

By assumption on ν (homomorphism and respect of (\mathcal{D}, ρ)) and by induction on \bar{m} , which is smaller than m , we have the following, where $(\nu(u))^e$ stands for $\nu(u) \pi \cdots \pi \nu(u)$ (e times):

$$\begin{aligned} \nu(m_j \zeta) &= (\nu(x))^{e_{j,0}} \pi (\nu(x_i))^{e_{j,1}} \pi (\nu(\mathcal{D}(x_i)))^{e_{j,2}} \pi (\nu(\bar{m}))^{e_{j,3}} \pi (\nu(\delta_\pi(\bar{m})))^{e_{j,4}} \\ &= (\nu(x))^{e_{j,0}} \pi (\nu(x_i))^{e_{j,1}} \pi (\nu(x_i)')^{e_{j,2}} \pi (\nu(\bar{m}))^{e_{j,3}} \pi (\nu(\bar{m})')^{e_{j,4}} \\ &= m_j \xi. \end{aligned} \tag{C.1}$$

Putting all together, we obtain the desired result for m :

$$\begin{aligned} \nu(\delta_\pi(x_i \bar{m})) &= \nu(F(x_i, \mathcal{D}(x_i), \bar{m}, \delta_\pi(\bar{m}))) && \text{by (4.13) and (4.12)} \\ &= \nu(\sum_j r_j m_j \zeta) && \text{by definition of } F \text{ and } \zeta \\ &= \sum_j r_j \nu(m_j \zeta) && \text{by homomorphism of } \nu \\ &= \sum_j r_j m_j \xi && \text{by (C.1)} \\ &= F(\nu(x_i), \nu(x_i)', \nu(\bar{m}), \nu(\bar{m})') && \text{by definition of } F \text{ and } \xi \\ &= (\nu(x_i) \pi \nu(\bar{m}))' && \text{by Def. 8(1)} \\ &= \nu(x_i \bar{m})' && \text{by homomorphism of } \nu. \end{aligned}$$

$p = \sum_{i \in I} r_i m_i$, for $|I| > 0$.

$$\begin{aligned} \nu(\delta_\pi(p)) &= \nu(\sum_{i \in I} r_i \delta_\pi(m_i)) && \text{by (4.14)} \\ &= \sum_{i \in I} r_i \nu(\delta_\pi(m_i)) && \text{since } \nu \text{ is an algebra homomorphism} \\ &= \sum_{i \in I} r_i (\nu(m_i)') && \text{by the case for } p \text{ a monomial} \\ &= \nu(p)' && \text{by linearity of stream derivatives. } \quad \square \end{aligned}$$

C.2 Three-coloured trees example: details

In order to generate the rational system (4.37), rather than explicitly determining the inverse of the jacobian $\nabla_{\mathbf{y}} \mathcal{E}$, it is practically convenient firstly to form the equivalent system (4.35) and then solve for \mathbf{y}' . To this purpose, we apply the syntactic stream derivative operator to the polynomial equations of system (4.44), obtaining:

$$\begin{cases} y'_1 - 1 - y'_2 y_2 - y'_3 y_3 - 2y'_2 y_3 &= 0 \\ y'_2 - y'_3 y_3 - y'_1 y_1 - 2y'_3 y_1 &= 0 \\ y'_3 - y'_1 y_1 - y'_2 y_2 - 2y'_1 y_2 &= 0. \end{cases} \tag{C.2}$$

Then we note that (C.2) is a linear system in the variables $\mathbf{y}' = (y'_1, y'_2, y'_3)^T$ of the form $A \mathbf{y}' = \mathbf{b}$, where $A = (\nabla_{\mathbf{y}} \mathcal{E})(0, \mathbf{r}_0, \mathbf{y}) = \begin{bmatrix} 1 & -y_2 - 2y_3 & -y_3 \\ -y_1 & 1 & -y_3 - 2y_1 \\ -y_1 - 2y_2 & -y_2 & 1 \end{bmatrix}$ and $\mathbf{b} = -(\nabla_x \mathcal{E})(0, \mathbf{y})^T = (1, 0, 0)^T$. Note that this is another way of writing system (4.35). Now, we solve $A \mathbf{y}' = \mathbf{b}$ for \mathbf{y}' . Denoting the determinant of $\nabla_{\mathbf{y}} \mathcal{E}$ as

$$\Delta = 2y_1^2 y_2 + 4y_1^2 y_3 + 4y_1 y_2^2 + 10y_1 y_2 y_3 + 3y_1 y_2 + 2y_1 y_3^2 + 3y_1 y_3 + 2y_2^2 y_3 + 4y_2 y_3^2 + 3y_2 y_3 - 1$$

and taking into account the initial condition, we arrive at (4.37):

$$\begin{cases} y_1' = \Delta^{-1} \cdot (2y_1y_2 + y_2y_3 - 1) & y_1(0) = 0 \\ y_2' = \Delta^{-1} \cdot (-2y_1^2 - 4y_1y_2 - y_1y_3 - y_1 - 2y_2y_3) & y_2(0) = 0 \\ y_3' = \Delta^{-1} \cdot (-y_1y_2 - y_1 - 2y_2) & y_3(0) = 0. \end{cases} \quad (\text{C.3})$$

In order to convert the above rational SDE initial value problem to a polynomial one, we can apply Lemma 8. In practice, we replace Δ^{-1} with a new variable w , then we add the corresponding equation to system (C.3). In order to derive a SDE for the variable w , we recall that the multiplicative inverse σ^{-1} of a stream σ s.t. $\sigma(0) \neq 0$ satisfies the SDE and initial condition (4.8). In our case, starting from $w' = w(0)\Delta'w$, and calling p_1, p_2, p_3 the right-hand sides of the SDEs in (C.3), we get

$$\begin{cases} w' &= w(0) \cdot (2y_1'y_1y_2 + 4y_1'y_1y_3 + 4y_1'y_2^2 + 10y_1'y_2y_3 + 3y_1'y_2 + 2y_1'y_3^2 + 3y_1' + y_3 \\ &\quad + 2y_2'y_2y_3 + 4y_2'y_3^2 + 3y_2'y_3)w \\ &= -(2p_1y_1y_2 + 4p_1y_1y_3 + 4p_1y_2^2 + 10p_1y_2y_3 + 3p_1y_2 + 2p_1y_3^2 + 3p_1 + y_3 \\ &\quad + 2p_2y_2y_3 + 4p_2y_3^2 + 3p_2y_3)w \\ w(0) &= -1 \end{cases}$$

where the initial condition $w(0) = -1$ is implied by $\Delta(0, \mathbf{r}_0) = -1$. Finally, expanding the p_i 's and putting everything together, we obtain the following polynomial system of SDEs and initial conditions:

$$\begin{cases} y_1' = 2wy_1y_2 + wy_2y_3 - w & y_1(0) = 0 \\ y_2' = -2wy_1^2 - 4wy_1y_2 - wy_1y_3 - wy_1 - 2wy_2y_3 & y_2(0) = 0 \\ y_3' = -wy_1y_2 - wy_1 - 2wy_2 & y_3(0) = 0 \\ w' = 4w^2y_1^2y_2^2 + 4w^2y_1^2y_2y_3 - 8w^2y_1^2y_3^2 - 6w^2y_1^2y_3 + 8w^2y_1y_2^3 + & w(0) = -1 \\ & 14w^2y_1y_2^2y_3 + 6w^2y_1y_2^2 - 10w^2y_1y_2y_3^2 - 8w^2y_1y_2y_3 - 2w^2 \\ & y_1y_2 - 4w^2y_1y_3^3 - 7w^2y_1y_3^2 - 7w^2y_1y_3 + 4w^2y_2^3y_3 + 6w^2y_2^2 \\ & y_3^2 + 3w^2y_2^2y_3 - 4w^2y_2^2 - 6w^2y_2y_3^3 - 3w^2y_2y_3^2 - 10w^2y_2y_3 - \\ & 3w^2y_2 - 2w^2y_3^2 - 3w^2y_3. \end{cases} \quad (\text{C.4})$$

Appendix D

Proofs and additional details of Chapter 6

D.1 Proofs

PROOF OF LEMMA 18. The proof consists of the following three steps.

- (a) For each fixed $\omega \in \Omega$, the function $A \mapsto \mathbb{K}(\omega)(A)$ is a probability measure on \mathcal{F} . For $\omega \notin \overline{\mathbb{R}}^m \times \mathcal{P}$, this is obvious as δ_ω is a probability measure on \mathcal{F} . For $\omega = (v, S)$ with $S \in \mathcal{P}$, the proof goes by structural induction on S . The base cases when $S = \text{nil}$, $S = \text{fail}$ and $S = (x_i = g.S')$ are obvious, and the wanted probability measure is again δ_ω in the first two cases, and $\delta_{(v[g(v)@i], S')}$ in the third case. Let us now consider the base case when $S = x_i \sim G.S'$. We will show that $\mathbb{K}(v, S)$ satisfies the additivity property on countable disjoint unions. Let a sequence of disjoint measurable sets $A_1, A_2, \dots \in \mathcal{F}$ be given and let $A = \bigcup_{j \geq 1} A_j$. Note that, by elementary set-theoretic reasoning, $A_{(v_i, S)} = \bigcup_{j \geq 1} A_{j, (v_i, S)}$, where the union is disjoint. Then $\mathbb{K}(v, S) = \int_{A_{(v_i, S')}} \mu_G(dr) G(v, r) = \sum_{j \geq 1} \int_{A_{j, (v_i, S')}} \mu_G(dr) G(v, r) = \sum_{j \geq 1} \mathbb{K}(v, S)(A_j)$, where the second equality follows from basic property of integrals of measurable functions, in particular [12, Cor.1.6.4]. This proves that $\mathbb{K}(v, S)$ is a measure on the sigma-field \mathcal{F} . Moreover, it is a probability measure, as $\mathbb{K}(v, S)(\Omega) = 1$, because $\Omega_{(v_i, S)} = \overline{\mathbb{R}}$ and by definition of parametric density G . This completes the proof for the case $S = x_i \sim G.S'$. For the remaining two cases (inductive step), the thesis follows immediately from the induction hypothesis.
- (b) For each fixed $A \in \mathcal{F}$ and fixed $S \in \mathcal{P}$, the function $v \mapsto \mathbb{K}(v, S)(A)$ is measurable on $\overline{\mathbb{R}}^m$. The proof goes again by structural induction on S . For $S = \text{nil}$, we note that $\delta_{(v, \text{nil})}(A) = 1_{A_{\text{nil}}}(v)$, which as an indicator function is measurable. The proof for $S = \text{fail}$ is essentially the same. Similarly, for $S = (x_i = g.S')$, we have $\delta_{(v[g(v)@i], S')}(A) = 1_{A_{S'}}(v[g(v)@i])$, which, as a composition of measurable functions of v , is measurable. The most delicate case is $S = x_i \sim G.S'$. Consider the function $\zeta(v, A) := \int_A \mu_G(dr) G(v, r)$ for $v \in \overline{\mathbb{R}}^m$ and $A \in \mathcal{F}_1$, the measurable sets of $\overline{\mathbb{R}}$. By definition of parametric density G , ζ is a Markov kernel from $\overline{\mathbb{R}}^m$ to $\overline{\mathbb{R}}$. Let us write a generic $v \in \overline{\mathbb{R}}^m$

as $v = (v_{-i}, v_i)$. The function that we want to prove measurable is $g(v_{-i}, v_i) = \zeta((v_{-i}, v_i), A_{(v_{-i}, S')})$, where A and S' are fixed. We will show measurability for the two arguments separately, v_{-i} and v_i , from which measurability of g as a function of v will follow. First note that, for each fixed v_i , also $\zeta_{v_i} : (v_{-i}, A) \mapsto \zeta((v_{-i}, v_i), A)$ is a Markov kernel from $\overline{\mathbb{R}}^{m-1}$ to $\overline{\mathbb{R}}$; and similarly for each fixed v_{-i} , also $\zeta_{v_{-i}} : (v_i, A) \mapsto \zeta((v_{-i}, v_i), A)$ is a Markov kernel from $\overline{\mathbb{R}}$ to $\overline{\mathbb{R}}$. Now consider the two statements separately. (1) for each fixed v_i , the function $v_{-i} \mapsto g(v_{-i}, v_i) = \zeta_{v_i}(v_{-i}, A_{(v_{-i}, S')})$ is measurable: indeed, $A_{(v_{-i}, S')}$ is the section of $A_{S'}$ at v_{-i} , and we can apply a general fact on kernels and sections in [12, Th.2.6.2, proof(2)] to the kernel ζ_{v_i} . (2) For each fixed v_{-i} , the function $v_i \mapsto g(v_{-i}, v_i) = \zeta_{v_{-i}}(v_i, A_{(v_{-i}, S')})$ is measurable: this follows directly from the fact that $\zeta_{v_{-i}}$ is a Markov kernel, and that $A_{(v_{-i}, S')}$ is a fixed measurable set that does not depend on the first argument v_i of $\zeta_{v_{-i}}$. The remaining two cases follow easily from the induction hypothesis.

- (c) For each fixed $A \in \mathcal{F}$, the function $\omega \mapsto \mathbb{K}(\omega)(A)$ is measurable on Ω . For each $\omega = (v, z) \in \Omega$ and $A \in \mathcal{F}$, we can write:

$$\mathbb{K}(v, z)(A) = 1_{\overline{\mathbb{R}}^m \times \mathcal{P}^c}(v, z) \cdot 1_A(v, z) + \sum_{S \in \mathcal{P}} 1_{\overline{\mathbb{R}}^m \times \{S\}}(v, z) \cdot \mathbb{K}(v, S)(A).$$

Each term of the above summation is a measurable function of (v, z) . In particular, for each fixed term S , we have shown in (b) above that $\mathbb{K}(v, S)(A)$ is a measurable function of v alone, hence $\mathbb{K}(v, S)(A)$ is also a measurable function of (v, z) (i.e. the function obtained by composing the projection $(v, z) \mapsto v$ with $v \mapsto \mathbb{K}(v, S)(A)$). A countable summation of nonnegative measurable functions of ω is still measurable a measurable function of ω (e.g. because it is the pointwise limit of the partial sum functions, cf e.g. [12, Th.1.5.4]). This completes the proof that \mathbb{K} is a Markov kernel. □

We now turn to the proof of Theorem 19. We first need a general lemma about measurability of functions in a cylindrical sigma-field.

Lemma D.1.1. *Let $h : \Omega^t \rightarrow \overline{\mathbb{R}}^+$ a nonnegative measurable function, and define $\tilde{h} : \Omega^\infty \rightarrow \overline{\mathbb{R}}^+$ as $\tilde{h}(\tilde{\omega}) := h(\tilde{\omega}_{1:t})$. Then \tilde{h} is measurable, and for each measurable cylinder $B_t \subseteq \Omega^\infty$, we have $\int_{B_t} \mu_S^\infty(d\tilde{\omega})\tilde{h}(\tilde{\omega}) = \int_{B_t} \mu_S^t(d\omega^t)h(\omega^t)$.*

PROOF. First, consider the case of indicator functions $h = 1_{A^t}$, for a measurable $A^t \subseteq \Omega^t$. Then $\tilde{h} = 1_{A_t}$, the indicator function of the measurable cylinder generated by A^t , and the statement is obvious, because \tilde{h} is measurable, and $\int_{B_t} \mu_S^\infty(d\tilde{\omega})\tilde{h}(\tilde{\omega}) = \int \mu_S^\infty(d\tilde{\omega})\tilde{h}(\tilde{\omega})1_{B_t}(\tilde{\omega}) = \mu_S^\infty(B_t \cap A_t) = \mu_S^t(B^t \cap A^t) = \int_{B^t} \mu_S^t(d\omega^t)h(\omega^t)$. The statement for the general case of h follows then by standard measure-theoretic arguments (linearity, dominated convergence). □

Lemma D.1.2. *Let S be a program, f termination based and $t \geq 1$. Then f_t is measurable. Moreover, $\int_{\mathbb{T}_t} \mu_S^\infty(d\tilde{\omega})f(\tilde{\omega}) = \int_{\mathbb{I}^{t-1} \times \mathbb{T}} \mu_S^t(d\omega^t)f_t(\omega^t)$.*

PROOF. Let us first show that f_t is measurable in Ω^t . Let A be any measurable set in $\overline{\mathbb{R}}$. By definition, $f^{-1}(A) = \{(\omega^t, \tilde{\omega}) : f(\omega^t, \tilde{\omega}) \in A\}$ is measurable. The section at

$\tilde{\omega} = *^\infty$ of this set is precisely $f_t^{-1}(A)$: a proof analogous to that in [12, Th.2.6.2(1)] shows that sections of elements in \mathcal{C} are measurable. Let f be termination based and consider Lemma D.1.1 with $h = f_t$. Note that f coincides with \tilde{f}_t on \mathbb{T}_t . Indeed, for any $\omega^t \in \mathbb{L}^{t-1} \times \mathbb{T}$, we have that f is constant on $\{\omega^t\} \times \Omega^\infty$ by definition of termination based function; hence, for any $(\omega^t, \tilde{\omega}) \in \mathbb{T}_t$, we have $f(\omega^t, \tilde{\omega}) = f(\omega^t, *^\infty) = f_t(\omega^t) = f_t((\omega^t, \tilde{\omega})_{1:t}) = \tilde{f}_t(\omega^t, \tilde{\omega})$. Therefore $\int_{\mathbb{T}_t} \mu_S^\infty(d\tilde{\omega})f(\tilde{\omega}) = \int_{\mathbb{T}_t} \mu_S^\infty(d\tilde{\omega})\tilde{f}_t(\tilde{\omega})$. Now we apply Lemma D.1.1 with $h = f_t$ and $B_t = \mathbb{T}_t$, hence $B^t = \mathbb{L}^{t-1} \times \mathbb{T}$, and the thesis follows. \square

Lemma D.1.3. *Let f be termination based, S a program and $t \geq 1$. Then we have:*

- (a) $\int \mu_S^\infty(d\tilde{\omega})f(\tilde{\omega}) = \sum_{t \geq 1} \int_{\mathbb{L}^{t-1} \times \mathbb{T}} \mu_S^t(d\omega^t)f_t(\omega^t)$;
- (b) $\sum_{1 \leq j \leq t} \int_{\mathbb{L}^{j-1} \times \mathbb{T}} \mu_S^j(d\omega^j)f_j(\omega^j) = \int \mu_S^t(d\omega^t)f_t(\omega^t)$;
- (c) *Let $f(\tilde{\omega}) \leq M$ for each $\tilde{\omega}$, possibly $M = +\infty$. Then $\int \mu_S^t(d\omega^t)f_t(\omega^t) \leq \int \mu_S^\infty(d\tilde{\omega})f(\tilde{\omega}) \leq \int \mu_S^t(d\omega^t)f_t(\omega^t) + M \cdot \mu_S^t(\mathbb{L}^t)$.*

PROOF.

- (a) As $\text{supp}(f) \subseteq \mathbb{T}_\infty = \bigcup_{t \geq 1} \mathbb{T}_t$ (disjoint union), it follows that $f = \sum_{t \geq 1} f \cdot 1_{\mathbb{T}_t}$, hence by commutativity of \int with series of nonnegative functions [12, Cor.1.6.4], we have $\int \mu_S^\infty(d\tilde{\omega})f(\tilde{\omega}) = \sum_{t \geq 1} \int_{\mathbb{T}_t} \mu_S^\infty(d\tilde{\omega})f(\tilde{\omega})$. Now apply Lemma D.1.2 to each summand $\int_{\mathbb{T}_t} \mu_S^\infty(d\tilde{\omega})f(\tilde{\omega})$.
- (b) We have that $\sum_{1 \leq j \leq t} \int_{\mathbb{L}^{j-1} \times \mathbb{T}} \mu_S^j(d\omega^j)f_j(\omega^j) = \sum_{1 \leq j \leq t} \int_{\mathbb{T}_j} \mu_S^\infty(d\tilde{\omega})f(\tilde{\omega})$ by Lemma D.1.2. Now each cylinder \mathbb{T}_j , $1 \leq j \leq t$, can be written as $\mathbb{T}_j = \mathbb{L}^{j-1} \times \mathbb{T} \times \Omega^\infty = (\mathbb{L}^{j-1} \times \mathbb{T} \times \Omega^{t-j}) \times \Omega^\infty$. Therefore: $\sum_{1 \leq j \leq t} \int_{\mathbb{T}_j} \mu_S^\infty(d\tilde{\omega})f(\tilde{\omega}) = \sum_{1 \leq j \leq t} \int_{(\mathbb{L}^{j-1} \times \mathbb{T} \times \Omega^{t-j}) \times \Omega^\infty} \mu_S^\infty(d\tilde{\omega})f(\tilde{\omega}) = \sum_{1 \leq j \leq t} \int_{\mathbb{L}^{j-1} \times \mathbb{T} \times \Omega^{t-j}} \mu_S^t(d\omega^t)f_t(\omega^t)$, where in the last step we have applied Lemma D.1.1 with $B = \mathbb{L}^{j-1} \times \mathbb{T} \times \Omega^{t-j}$ and $h = f_t$ (hence $\tilde{h} = f$) to each summand. Now $\Omega^t = \bigcup_{j=1}^t (\mathbb{L}^{j-1} \times \mathbb{T} \times \Omega^{t-j}) \cup A$, for some measurable A , such that all the unions are disjoint, and $A \cap \text{supp}(f_t) = \emptyset$. As a result, $\sum_{1 \leq j \leq t} \int_{\mathbb{L}^{j-1} \times \mathbb{T} \times \Omega^{t-j}} \mu_S^t(d\omega^t)f_t(\omega^t) = \int_{\Omega^t} \mu_S^t(d\omega^t)f_t(\omega^t)$, which completes the proof of this part.
- (c) The lower bound is obvious, from the previous two parts. As to the upper bound, first note that the support of f is included in \mathbb{T}_∞ , which can be decomposed as $\mathbb{T}_\infty = \bigcup_{j=1}^t \mathbb{T}_j \cup \bigcup_{j>t} \mathbb{T}_j$ (disjoint unions). Hence

$$\begin{aligned} \int \mu_S^\infty(d\tilde{\omega})f(\tilde{\omega}) &= \sum_{j=1}^t \int_{\mathbb{T}_j} \mu_S^\infty(d\tilde{\omega})f(\tilde{\omega}) + \int_{\bigcup_{j>t} \mathbb{T}_j} \mu_S^\infty(d\tilde{\omega})f(\tilde{\omega}) \\ &\leq \sum_{j=1}^t \int_{\mathbb{L}^{j-1} \times \mathbb{T}} \mu_S^j(d\omega^j)f_j(\omega^j) + M \int_{\bigcup_{j>t} \mathbb{T}_j} \mu_S^\infty(d\tilde{\omega}) \end{aligned}$$

where, in the last step, we have applied Lemma D.1.2 to the first summation and the assumption on M to bound the second summation. Now $\bigcup_{j>t} \mathbb{T}_j \subseteq \mathbb{L}^t \times \Omega^\infty$, hence $\int_{\bigcup_{j>t} \mathbb{T}_j} \mu_S^\infty(d\tilde{\omega}) = \mu_S^\infty(\bigcup_{j>t} \mathbb{T}_j) \leq \mu_S^\infty(\mathbb{L}^t \times \Omega^\infty) = \mu^t(\mathbb{L}^t)$.

To sum up: $\int \mu_S^\infty(d\tilde{\omega})f(\tilde{\omega}) \leq \sum_{j=1}^t \int_{\mathbb{L}^{j-1} \times \mathbb{T}} \mu_S^j(d\omega^j)f_j(\omega^j) + M\mu^t(\mathbb{L}^t)$. Now the first summation equals $\int \mu_S^t(d\omega^t)f_t(\omega^t)$ by part (b), from which the thesis follows.

□

PROOF OF THEOREM 19. The proof goes by bounding the numerator and the denominator in definition (6.7). As for the numerator, one uses directly the bounds in part (c) of Lemma D.1.3. As for the denominator, one first notes that $\bigcup_{j=1}^t \mathbb{T}_j \subseteq (\mathbb{F}_\infty)^c \subseteq \bigcup_{j=1}^t \mathbb{T}_j \cup (\mathbb{L}^t \times \Omega^\infty)$, where all unions are disjoint. Therefore we have $\sum_{j=1}^t \mu_S^\infty(\mathbb{T}_j) \leq \mu_S^\infty((\mathbb{F}_\infty)^c) \leq \sum_{j=1}^t \mu_S^\infty(\mathbb{T}_j) + \mu_S^\infty(\mathbb{L}^t \times \Omega^\infty)$. Now, for each j , by definition $\mu_S^\infty(\mathbb{T}_j) = \mu_S^j(\mathbb{L}^{j-1} \times \mathbb{T})$, hence $\sum_{j=1}^t \mu_S^\infty(\mathbb{T}_j) = \sum_{j=1}^t \mu_S^j(\mathbb{L}^{j-1} \times \mathbb{T})$. Now we apply Lemma D.1.3(b) to the last summation, with $f = 1_{\mathbb{T}_\infty}$, and also note that $f_t = 1_{\mathbb{T}^{\leq t}}$ to get:

$$\begin{aligned} \sum_{j=1}^t \mu_S^j(\mathbb{L}^{j-1} \times \mathbb{T}) &= \sum_{j=1}^t \int_{\mathbb{L}^{j-1} \times \mathbb{T}} \mu_S^j(d\omega^j) (1_{\mathbb{T}_\infty})_j(\omega^j) \\ &= \int \mu_S^t(d\omega^t) (1_{\mathbb{T}_\infty})_t(\omega^t) \\ &= \int \mu_S^t(d\omega^t) 1_{\mathbb{T}^{\leq t}}(\omega^t) \\ &= \mu_S^t(\mathbb{T}^{\leq t}). \end{aligned}$$

We have proven the following intermediate equality, that we record for use further below:

$$\mu_S^\infty(\bigcup_{j=1}^t \mathbb{T}_j) = \mu_S^t(\mathbb{T}^{\leq t}). \quad (\text{D.1})$$

Next, we note that by definition $\mu_S^\infty(\mathbb{L}^t \times \Omega^\infty) = \mu_S^t(\mathbb{L}^t)$. To sum up, for any $t \geq 1$, we have the following bounds for the numerator and denominator in (6.7) expressed in the $[S](\cdot)$ notation:

$$\begin{aligned} [S]^t f_t &\leq [S]f \leq [S]^t f_t + M \cdot [S]^t 1_{\mathbb{L}^t} \\ [S]^t 1_{\mathbb{T}^{\leq t}} &\leq [S]1_{\mathbb{F}^c} \leq [S]^t 1_{\mathbb{T}^{\leq t}} + [S]^t 1_{\mathbb{L}^t}. \end{aligned} \quad (\text{D.2})$$

Now consider t such that $\mu_S^t(\mathbb{T}^{\leq t}) > 0$, which exists by hypothesis. Then $[S]1_{\mathbb{F}^c} \geq \mu_S^\infty(\bigcup_{j=1}^t \mathbb{T}_j) = \mu_S^t(\mathbb{T}^{\leq t}) > 0$, where the last equality is (D.1). Therefore $[S]f$ is well defined for any termination based f . Moreover, $\mu_S^{t'}(\mathbb{T}^{\leq t'}) > 0$ for every $t' > t$. Hence, for every t large enough, the ratios in (6.8) exist and, as a consequence of (D.2), the bounds (6.8) themselves hold true. □

Next, we prove the bounds (6.15), giving confidence intervals for the MC estimation procedure. We need an extension of Hoeffding inequality to conditional expectations.

Some notation is in order. Below, we let ν be any probability measure over a measurable space Ω . We let $E_\nu[h]$ denote the expectation (integral) of h taken according to ν . Let A be a measurable event such that $\nu(A) = E[1_A] > 0$; then $E_\nu[h|A]$ denotes the conditional expectation of h given A , that is: $E_\nu[h|A] := E_\nu[h \cdot 1_A] / E_\nu[1_A]$. This is the same as the expectation of h taken according to the probability measure $\tilde{\nu}(C) := \nu(C \cap A) / \nu(A)$, as can be readily proved by e.g. applying the chain rule for densities (as for each C , $\tilde{\nu}(C) = \int_C \nu(d\omega) (\frac{1_A}{\nu(A)})(\omega)$). For a sequence of random variables all defined on Ω , say $X = (X_1, \dots, X_N)$, for $A \subseteq \Omega$ an event, and for $b \in \{0, 1\}^N$, we let $A_b \subseteq \Omega^N$ denote the event $\bigcap_{i=1}^N (1_A(X_i) = b_i)$. For any $k \geq 1$, ν^k will denote the product measure on Ω^k .

Lemma D.1.4. *Let A be a measurable event such that $\nu(A) = E_\nu[1_A] > 0$ and $N > 0$ an integer. Let f be a bounded nonnegative measurable function on \mathbb{R} , with*

$f \leq M$. Let X_1, \dots, X_N be i.i.d. random variables distributed according to ν . Define $\widehat{N} := \sum_{i=1}^N 1_A(X_i)$ and

$$Y := \frac{1}{\widehat{N}} \sum_{i=1}^N f(X_i) 1_A(X_i)$$

with the convention that $(1/0) \cdot 0 = 0$. For each $\epsilon > 0$, and $b \in \{0, 1\}^N$ with $|b|_H = n$:

$$\Pr_{\nu^N} \left(Y - \mathbb{E}_\nu[f|A] > \epsilon \mid A_b \right) \leq e^{-2\frac{n\epsilon^2}{M^2}}. \quad (\text{D.3})$$

PROOF. Consider $b \in \{0, 1\}^N$ and $n = |b|_H > 0$, as for $n = 0$ there is nothing to prove. Let i_1, \dots, i_n be the indices in $1..N$ such that $b_{i_j} = 1$. We have

$$\Pr_{\nu^N} \left(Y - \mathbb{E}_\nu[f|A] > \epsilon \mid A_b \right) = \Pr_{\nu^N} \left(\frac{1}{n} \sum_{j=1}^n f(X_{i_j}) - \mathbb{E}_\nu[f|A] > \epsilon \mid A_b \right) \quad (\text{D.4})$$

$$= \Pr_{\tilde{\nu}^n} \left(\frac{1}{n} \sum_{j=1}^n f(X_{i_j}) - \mathbb{E}_\nu[f|A] > \epsilon \right) \quad (\text{D.5})$$

where $\tilde{\nu}(C) := \nu(A \cap C)/\nu(A)$ for any measurable $C \subseteq \Omega$. Here, equality (D.4) holds by definition of A_b . Equality (D.5) can be proved via the Fubini theorem, as follows. We shall use the following notation. As a subset of Ω^N , A_b can be written as: $A_b = A_1 \times \dots \times A_N$, with $A_i = A$ if $b_i = 1$ and $A_i = A^c$ if $b_i = 0$. We shall write $\omega^N = (\omega^n, \omega^{N-n})$ to indicate that the tuple $\omega^N = (\omega_1, \dots, \omega_N)$ is partitioned into two sub-tuples: ω^n , consisting of the components of ω^N of indices (i_1, \dots, i_n) ; and ω^{N-n} , consisting of the remaining components, say (j_1, \dots, j_{N-n}) . Also, for brevity we denote by $B \subseteq \Omega^N$ the event to the left of $|$ in expression in (D.4), and we let $\hat{B} := \{\omega^n : (\omega^n, \omega^{N-n}) \in B\}$ — that is, the projection of B onto the components of Ω^N of indices (i_1, \dots, i_n) . With this notation in place, we first note that $1_B(\omega^N)$ does not depend on components in ω^{N-n} , that is $1_B(\omega^N) = 1_{\hat{B}}(\omega^n)$. This allows us to factorize the indicator function $1_{B \cap A_b}$ as follows: $1_{B \cap A_b}(\omega^N) = 1_{(A^c)^{N-n}}(\omega^{N-n}) \cdot 1_{A^n}(\omega^n) \cdot 1_{\hat{B}}(\omega^n) = 1_{(A^c)^{N-n}}(\omega^{N-n}) \cdot 1_{\hat{B} \cap A^n}(\omega^n) = 1_{A^c}(\omega_{j_1}) \cdots 1_{A^c}(\omega_{j_{N-n}}) \cdot 1_{\hat{B} \cap A^n}(\omega^n)$. Then

we have:

$$\Pr_{\nu^N}(B | A_b) = \frac{1}{\nu^N(A_b)} \int \nu^N(d\omega^N) 1_{B \cap A_b}(\omega^N) \quad (\text{D.6})$$

$$= \frac{1}{\nu^N(A_b)} \int \nu(d\omega_1) \int \cdots \int \nu(d\omega_N) (1_{A^c}(\omega_{j_1}) \cdots 1_{A^c}(\omega_{j_{N-n}})) \cdot 1_{\hat{B} \cap A^n}(\omega^n) \quad (\text{D.7})$$

$$= \frac{\nu(A^c)^{N-n}}{\nu^N(A_b)} \int \nu(d\omega_{i_1}) \int \cdots \int \nu(d\omega_{i_n}) 1_{\hat{B} \cap A^n}(\omega^n) \quad (\text{D.8})$$

$$= \frac{\nu(A^c)^{N-n}}{\nu(A^c)^{N-n} \nu(A)^n} \int \nu(d\omega_{i_1}) \int \cdots \int \nu(d\omega_{i_n}) 1_{\hat{B} \cap A^n}(\omega^n) \quad (\text{D.9})$$

$$= \int \nu(d\omega_{i_1}) \int \cdots \int \nu(d\omega_{i_n}) \left(\frac{1_{\hat{B} \cap A^n}}{\nu(A)^n} \right) (\omega^n) \quad (\text{D.10})$$

$$= \int \nu^n(d\omega^n) \left(\frac{1_{\hat{B} \cap A^n}}{\nu(A)^n} \right) (\omega^n) \quad (\text{D.11})$$

$$= \int \nu^n(d\omega^n) \left(\frac{1_{A^n}}{\nu(A)^n} \cdot 1_{\hat{B}} \right) (\omega^n) \quad (\text{D.12})$$

$$= \int \tilde{\nu}^n(d\omega^n) 1_{\hat{B}}(\omega^n) \quad (\text{D.13})$$

$$= \Pr_{\tilde{\nu}^n} \left(\frac{1}{n} \sum_{j=1}^n f(X_{i_j}) - \mathbb{E}_\nu[f|A] > \epsilon \right) \quad (\text{D.14})$$

where: (D.6) follows from the definition of $\Pr_{\nu^N}(\cdot|\cdot)$; (D.7) from applying Fubini (Theorem 13(b)) and from the above discussed factorization of $1_{B \cap A_b}(\omega^N)$; (D.8) from recalling that $\int \nu(d\omega_j) 1_C(\omega_j) h(\omega^n) = \nu(C) h(\omega^n)$ whenever ω_j is not in ω^n ; (D.9) from $\nu^N(A_b) = \nu(A_1) \cdots \nu(A_N)$ and from the definition of the A_i 's; (D.10) from algebraic manipulations and basic properties of integrals (linearity); (D.11) again from applying Fubini (Theorem 13(b)); (D.12) from a basic property of indicator functions for intersection; (D.13) from the chain rule for densities and from the definition of $\tilde{\nu}$, recalling that we can write $\tilde{\nu}^n(C) = \int_C \nu^n(d\omega^n) (1_{A^n} / \nu^n(A^n))(\omega^n)$; finally (D.14) from definition of $\Pr_{\tilde{\nu}^n}(\cdot)$.

From elementary properties of expectation, $\mathbb{E}_{\tilde{\nu}^n}[\frac{1}{n} \sum_{j=1}^n f(X_{i_j})] = \mathbb{E}_{\tilde{\nu}^n}[f(X_1)] = \mathbb{E}_{\tilde{\nu}}[f(X_1)] = \mathbb{E}_\nu[f|A]$. Moreover, according to the product measure $\tilde{\nu}^n$ the variables $\tilde{X} = X_{i_1}, \dots, X_{i_n}$ are by definition i.i.d. Therefore we can apply the one sided version of Hoeffding inequality to $\tilde{\nu}^n$ and \tilde{X} to deduce that $\Pr_{\tilde{\nu}^n} \left(\frac{1}{n} \sum_{j=1}^n f(X_{i_j}) - \mathbb{E}_\nu[f|A] > \epsilon \right) \leq e^{-2\frac{n\epsilon^2}{M^2}}$. This fact and (D.5) imply the thesis, that is inequality (D.3). \square

D.2 Importance Sampling

In order to generalize the example in Section 6.5, we introduce some new notation. For any integer $j \geq 0$, we let $r^j = (r_1, \dots, r_j)$ range over $\overline{\mathbb{R}}^j$ (when $j = 0$, r^j is the empty tuple). A *drawing sequence* is sequence of triples (index, parametric density, predicate),

$$\sigma = (i_1, \rho_1, \psi_1), \dots, (i_\ell, \rho_\ell, \psi_\ell)$$

where $1 \leq i_j \leq m$. Relatively to a given drawing sequence σ , for $v = (v_1, \dots, v_m) \in \mathbb{R}^m$ and $0 \leq j \leq \ell$, we let $v[r_1, \dots, r_j] = v[r^j] := v[r_1 \circ i_1][r_2 \circ i_2] \cdots [r_j \circ i_j]$. Moreover, we define (below, μ_j is the measure associated with the density ρ_j , and g is any a nonnegative measurable function defined on \mathbb{R}^ℓ and $0 \in \mathbb{R}^m$):

$$\begin{aligned} J(\sigma, g) &:= \int \mu_1(dr_1) \rho_1(0, r_1) \psi_1(0[r_1]) \cdot \int \mu_2(dr_2) \rho_2(0[r_1], r_2) \psi_2(0[r_1, r_2]) \cdot \\ &\quad \cdots \cdot \int \mu_\ell(dr_\ell) \rho_\ell(0[r^\ell]) \psi_j(0[r^\ell]) \cdot g(r^\ell) \\ W_j(r^{j-1}) &:= \int \mu_j(dr_j) \rho_j(0[r^{j-1}], r_j) \psi_j(0[r^j]) \quad \tilde{\rho}_j(v, r) := \frac{\rho_j(v, r) \psi_j(v[r \circ i_j])}{W_j(v_{i_1}, \dots, v_{i_{j-1}})} \\ \tilde{J}(\sigma, g) &:= \int \mu_1(dr_1) \tilde{\rho}_1(0, r_1) \cdot \int \mu_2(dr_2) \tilde{\rho}_2(0[r^1], r_2) \cdot \\ &\quad \cdots \cdot \int \mu_\ell(dr_\ell) \tilde{\rho}_\ell(0[r^\ell]) \cdot W_1 \cdot W_2(r_1) \cdots W_\ell(r^{\ell-1}) \cdot g(r^\ell). \end{aligned} \tag{D.15}$$

where the definitions of $\tilde{\rho}_j, \tilde{J}$ apply only if all the W_j 's are everywhere strictly positive (the measurability of the functions involved in the integrals is dealt with in the next proposition). Next, let us call programs like \tilde{S} above, consisting solely of a sequence of drawings actions $x_i \sim G$ terminated by nil, *straight line* programs [45]. The general strategy consists in expanding $[S]^t f$ into a sum of iterated integrals $J(\sigma, g)$, like the one in (6.16). Basically, each drawing sequence σ corresponds to an integral $J(\sigma, g)$, which in turn corresponds to a straight line program (under a technical condition). We state the formal result below.

Theorem D.2.1 (importance sampling). *Let $S \in \mathcal{P}$ and $t \geq 1$.*

(a) *There exist $k \geq 0$ drawing sequences $\sigma_1, \dots, \sigma_k$ s.t. for each measurable function $f : \Omega^t \rightarrow \mathbb{R}^+$*

$$[S]^t f = \sum_{i=1}^k J(\sigma_i, g_i) \tag{D.16}$$

for suitable nonnegative measurable functions g_1, \dots, g_k (that depend also on f).

(b) *For each σ_i , the functions W_j and the functions under each integral in (D.15) are measurable. If, additionally, all the W_j 's are strictly positive, then there exist a straight line program \tilde{S}_i such that for $t_i = |\sigma_i| + 1$*

$$J(\sigma_i, g_i) = \tilde{J}(\sigma_i, g_i) = [\tilde{S}_i]^{t_i} \tilde{f}_i \tag{D.17}$$

for some measurable function \tilde{f}_i (that depends also on g_i) defined on Ω^{t_i} .

Before discussing the proof of this result, let us make some remarks on its actual application. It is possible to effectively compute the expansion in (D.16) given S and t , as well as the straight line programs in (D.17) along with expressions for \tilde{f}_i (see Proposition D.2.1 below). Under the assumption of positivity of the W_j 's, for each i we can use (D.17) as the basis for a Monte Carlo estimation of $J_i(\sigma_i, g_i)$ via the algorithm \mathcal{A} in Section 6.5. There will be no rejection in the corresponding sampling. The variance of the integrand $W_1 \cdot W_2(r_1) \cdots W_\ell(r^{\ell-1}) \cdot g(r^\ell)$ in $\tilde{J}_i(\sigma_i, g_i)$ will be much lower than in the pure Monte Carlo version without importance sampling. Moreover, if a nontrivial upper bound $b < 1$ on the product of the W_j 's is available, it can be exploited to improve the Hoeffding bound (6.14) by replacing M with $M \cdot b$.

Concerning the positivity assumptions on the weight functions W_j , this can sometimes be ensured by simple program transformations, like those considered in [45].

For instance, for ρ_D a uniform discrete density on a set of integers, one might transform $S = x_1 \sim \rho_D. x_2 \sim \rho_D. x_3 \sim \rho_D. \text{obs}(x_1 \leq x_3 \leq x_2). \text{nil}$ into the (pragmatically equivalent) $S' = x_1 \sim \rho_D. x_2 \sim \rho_D. \text{obs}(x_2 \geq x_1). x_3 \sim \rho_D. \text{obs}(x_1 \leq x_3 \leq x_2). \text{nil}$. The expansion (D.16) of S' will result in a sum of terms with drawing sequences having positive weight functions. For continuous distributions this kind of transformations is trickier, as one must tackle properly zero measure sets.

Finally, let us note that computing the expansion (D.16) can be costly, as k can be exponential in the size of S , which typically happens when there are many nested if-then-else's, or loops with complicated exit conditions. In programs where there are no fail statements, hence no obs, there will be no rejections, and IS might be less convenient than the pure Monte Carlo estimation of Section 6.5. We now turn to the actual proof of the above theorem. We need a preliminary result.

Proposition D.2.1 (expanded form). *Let $t \geq 1$, $f : \Omega^t \rightarrow \overline{\mathbb{R}}^+$ be a nonnegative measurable function and $S \in \mathcal{P}$. Then there are $k \geq 0$ measurable nonnegative functions over $\overline{\mathbb{R}}^m$, say J_1, \dots, J_k , such that for each $v \in \overline{\mathbb{R}}^m$*

$$[S]_v^t f = \sum_{i=1}^k J_i(v). \quad (\text{D.18})$$

Moreover, each J_i has the following form, for some functions μ_j, ρ_j, ψ_j ($1 \leq j \leq \ell$) and γ and sequence of indices $i_1, \dots, i_\ell \in \{1, \dots, m\}$ depending on i but not on f :

$$\begin{aligned} J_i(v) &= \int \mu_1(dr_1) \rho_1(v, r_1) \psi_1(v, r_1) \cdot \int \mu_2(dr_2) \rho_2(v[r_1], r_2) \psi_1(v[r_1], r_2) \cdot \\ &\quad \cdots \cdot \int \mu_\ell(dr_\ell) \rho_\ell(v[r^{\ell-1}], r_\ell) \psi_\ell(v[r^\ell]) f(\gamma(v[r^\ell])). \end{aligned} \quad (\text{D.19})$$

In particular, for each such J_i , we have the following.

(a) For $j = 1, \dots, \ell$, the function $\rho_j(\cdot, \cdot)$ is a parametric density from $\overline{\mathbb{R}}^m$ to $\overline{\mathbb{R}}$ w.r.t. a (Lebesgue or counting) measure μ_j over $\overline{\mathbb{R}}$ appearing in S , and $\psi_j(\cdot)$ is a predicate. Moreover, $\gamma : \overline{\mathbb{R}}^m \rightarrow \Omega^t$ is a measurable function.

(b) For $j = 1, \dots, \ell$, each v and $r^j = (r_1, \dots, r_j)$, the function

$$\begin{aligned} r_j &\mapsto \rho_j(v[r^{j-1}], r_j) \psi_\ell(v[r^j]) \cdot \int \mu_{j+1}(dr_{j+1}) \rho_j(v[r^j], r_{j+1}) \psi_{j+1}(v[r^{j+1}]) \cdot \\ &\quad \cdots \cdot \int \mu_\ell(dr_\ell) \rho_\ell(v[r^{\ell-1}], r_\ell) \psi_\ell(v[r^\ell]) f(\gamma(v[r^\ell])) \end{aligned} \quad (\text{D.20})$$

is measurable over $\overline{\mathbb{R}}$.

PROOF. We proceed by structural induction on S . We only cover in detail the case $S = x_i \sim G.S'$, as the other cases are easier and/or follows immediately from the induction hypothesis. For any v , let $f_v : \Omega^{t-1} \rightarrow \overline{\mathbb{R}}^+$ be defined by $f_v(\omega^{t-1}) := f((v, S), \omega^{t-1})$; this is a measurable function. Applying Proposition 20, equality (6.10) in the present case can be written as

$$[S]_v^t f = \int \mu_G(dr_1) G(v, r_1) [S']_{v[r_1 \circ i]}^{t-1} f_v. \quad (\text{D.21})$$

By induction hypothesis, there are functions of v , say $J'_i(v)$'s, s.t. for each v , $[S']_{v[r_1 \circ i]}^{t-1} f_v = \sum_i J'_i(v[r_1 \circ i])$, and each $J'_i(v)$ is a measurable function of the form (D.18), say:

$$\begin{aligned} J'_i(v) &= \int \mu_2(dr_2) \rho_2(v, r_2) \psi_2(v[r_2]) \cdot \int \mu_3(dr_3) \rho_3(v[r_2], r_3) \psi_3(v[r_2, r_3]) \cdot \\ &\quad \cdots \cdot \int \mu_\ell(dr_\ell) \rho_\ell(v[r_2, \dots, r_{\ell-1}], r_\ell) \psi_\ell(v[r_2, \dots, r_\ell], r_\ell) f_v(\gamma'(v[r_2, \dots, r_\ell])). \end{aligned} \quad (\text{D.22})$$

Now let $J_i(v) := \int \mu_1(dr_1) \rho_1(v, r_1) \cdot J'_i(v[r_1 \circledast i])$, where $\mu_1 := \mu_G$ and $\rho_1 := G$. Distributing the integral $\int \mu_1(dr_1) \rho_1(v, r_1) (\cdot)$ over the sum $[S']_{v[r_1 \circledast i]}^{t-1} f = \sum_i J'_i(v[r_1 \circledast i])$ in (D.21), we have $[S]_v^t f = \sum_i J_i(v)$. Moreover, we can check that each $J_i(v)$ is the form required by (D.18), with $i_1 := i$, $\psi_1(v, r_1) := 1$ and $\gamma(v, r_1, r_2, \dots, r_\ell) := ((v, S), \gamma'(v[r_1 \circledast i], r_2, \dots, r_\ell))$; indeed from (D.22):

$$\begin{aligned} J_i(v) &= \int \mu_1(dr_1) \rho_1(v, r_1) \cdot J'_i(v[r_1 \circledast i]) \\ &= \int \mu_1(dr_1) \rho_1(v, r_1) \cdot \int \mu_2(dr_2) \rho_2(v[r_1], r_2) \psi_2(v[r_1, r_2]) \cdot \\ &\quad \cdots \cdot \int \mu_\ell(dr_\ell) \rho_\ell(v[r_1, r_2, \dots, r_{\ell-1}], r_\ell) \psi_\ell(v[r_1, r_2, \dots, r_\ell]) f_v(\gamma'(v[r_1, r_2, \dots, r_\ell])) \\ &= \int \mu_1(dr_1) \rho_1(v, r_1) \psi_1(v, r_1) \cdot \int \mu_2(dr_2) \rho_2(v[r_1], r_2) \psi_2(v[r_1, r_2]) \cdot \\ &\quad \cdots \cdot \int \mu_\ell(dr_\ell) \rho_\ell(v[r_1, r_2, \dots, r_{\ell-1}], r_\ell) \psi_\ell(v[r_1, r_2, \dots, r_\ell]) f(\gamma(v[r_1, r_2, \dots, r_\ell])) \end{aligned}$$

where the last equality stems from the definition of f_v . We now check the conditions (a) and (b) on ρ_j, ψ_j and γ .

- (a) For $j = 1, \dots, \ell$, the function ρ_j is a parametric density. For $j = 1$, this holds by definition of G and $\mu_1 = \mu_G$. For $j > 1$, this holds by induction hypothesis. Measurability of ϕ_j and γ follows similarly, by induction hypothesis, and noting (in the case of γ) that the composition of measurable functions is measurable.
- (b) For $j = 1, \dots, \ell$, for each v , the function in (D.20) is measurable. For $2 \leq j \leq \ell$, this follows immediately from the induction hypothesis (b) applied to J'_i : indeed, with the above positions, the $\ell - 1$ functions in the present case are the same as in the induction hypotheses. For $j = 1$, the considered function can be written as $r_1 \mapsto \rho_1(v, r_1) J'_i(v[r_1 \circledast i])$. As ρ_1 and J'_i are measurable as functions of (v, r_1) , so are the functions obtained from each of them by fixing all arguments but one: in particular, $r_1 \mapsto \rho_1(v, r_1)$ and $r_1 \mapsto J'_i(v[r_1 \circledast i])$. Finally, the product of measurable functions is measurable.

We are left with checking that each $J_i(v)$ is measurable as a function of v . Let $\tilde{\mu}(v, \cdot)$ denote the measure $A \mapsto \int_A \mu_1(dr_1) \rho_1(v, r_1)$: by assumption, $(v, A) \mapsto \tilde{\mu}(v, A)$ is a Markov kernel.

Moreover, the chain rule implies that $\int \tilde{\mu}(v, dr_1) J'_i(v[r_1 \circledast i]) = J_i(v)$. The measurability of $J_i(v)$ then follows from the last part of Theorem 13(b) (Fubini, in the case $t = 2, j = 2$), applied to the mentioned Markov kernel and to the function $r_1 \mapsto J'_i(v[r_1 \circledast i])$. \square

PROOF OF THEOREM D.2.1. Part (a) follows immediately from Proposition D.2.1, letting $v = 0$ in equalities (D.18) and (D.19); in particular, from (D.19), for each i we have $J_i(0) = J(\sigma_i, g_i)$ where $\sigma_i = (i_1, \rho_1, \psi_1), \dots, (i_\ell, \rho_\ell, \psi_\ell)$ and $g_i: r^\ell \mapsto f(\gamma(0[r^\ell]))$, as specified in (D.19).

Consider part (b). The measurability of the functions appearing in the iterated integrals in $J(\sigma_i, g_i)$ is stated in Proposition D.2.1 (b). We consider now the measurability of W_j . As ρ_j is a parametric density, the map $(w, A) \mapsto \int_A \mu_j(dr) \rho_j(w, r)$ is a Markov kernel from $\overline{\mathbb{R}}^m$ to $\overline{\mathbb{R}}$. Consider the function $h(w) := \int \mu_j(dr_j) \rho_j(w, r_j) \psi_j(w[r_j])$. By the chain rule for densities, $h(w)$ is just the integral of the function $r_j \mapsto \psi_j(w[r_j])$ w.r.t. the measure obtained when fixing the first argument of this Markov kernel to w . The measurability of h then follows then from the last part of Theorem 13(b) (Fubini, in the case $t = 2$), applied to the mentioned Markov kernel and to the function $r_j \mapsto \psi_j(w[r_j])$. Now $r^{j-1} \mapsto W_j(r^{j-1}) = h(0[r^{j-1}])$ can be expressed as a composition of measurable functions, from which the measurability of W_j as a function of

r^{j-1} follows. This also implies measurability of $W_j(r^{j-1})$ as a function defined on $\overline{\mathbb{R}}^m$ (where some of the arguments are ignored).

Finally, consider (D.17). Consider the expression for $J(\sigma_i, g_i) = J_i(0)$, given by Proposition D.2.1 in (D.19) with $v = 0$. Under the assumption that the functions W_j are all (strictly) positive, multiply and divide by $W_j(r^{j-1})$ each factor $\rho_j(0[r^{j-1}], r_j)\psi_j(0[r^j])$, then move the factors $W_j(r^{j-1})$ inside the innermost integral, which is possible by linearity of the integral. Letting $x = (x_1, \dots, x_m)$, by definition $\tilde{\rho}_j(x, r) = \rho_j(x, r) \cdot \psi_j(x[r_{\otimes j}]) / W_j(x_{i_1}, \dots, x_{i_{j-1}})$, and we can write:

$$\begin{aligned} J(\sigma_i, g_i) &= J_i(0) \\ &= \int \mu_1(dr_1) \tilde{\rho}_1(0, r_1) \cdot \int \mu_2(dr_2) \tilde{\rho}_2(0[r_1], r_2) \cdot \\ &\quad \cdots \cdot \int \mu_\ell(dr_\ell) \tilde{\rho}_\ell(0[r^\ell]) \cdot (W_1 \cdot W_2(r_1) \cdot \cdots \cdot W_\ell(r^{\ell-1}) \cdot f(\gamma(0[r^\ell]))) \\ &= \tilde{J}(\sigma_i, g_i). \end{aligned}$$

Using the positivity assumption on the W_j 's, from Proposition D.2.1(a) it easily follows that each $\tilde{\rho}_j(x, r)$ is a parametric density w.r.t. the measure μ_j . Consider now the straight line program $\tilde{S}_i := x_{i_1} \sim \tilde{\rho}_1 \cdot \cdots \cdot x_{i_\ell} \sim \tilde{\rho}_\ell \cdot \text{nil}$. Consider the function g on Ω defined by $g(x, S) := W_1 \cdot W_2(x_{i_1}) \cdot \cdots \cdot W_\ell(x_{i_1}, \dots, x_{i_\ell}) \cdot f(\gamma(0[x_{i_1}, \dots, x_{i_\ell}])) \cdot 1_{\{\text{nil}\}}(S)$ and let \tilde{f}_i be the lifting of g to Ω^{t_i} , where $t_i = \ell + 1$. Relying on Proposition 20, one can check that $[\tilde{S}_i]^{t_i} \tilde{f}_i = J_i(0)$. From this fact and from Proposition D.2.1 equation (D.18) the thesis follows. \square

D.3 Models

We consider the following probabilistic models described for example in [85]. For convenience, the programs are described in the language of [85], which is based on sequential composition, but they are easy to translate into our language.

```

1: float skillA, skillB,skillC;
2: float perfA1,perfB1,perfB2;
3: float perfC2,perfA3,perfC3;
4:
5: skillA = Gaussian(100,10);
6: skillB = Gaussian(100,10);
7: skillC = Gaussian(100,10);
8:
9: // first game:A vs B, A won
10: perfA1 = Gaussian(skillA,15);
11: perfB1 = Gaussian(skillB,15);
12: observe(perfA1 > perfB1);
13:
14: // second game:B vs C, B won
15: perfB2 = Gaussian(skillB,15);
16: perfC2 = Gaussian(skillC,15);
17: observe(perfB2 > perfC2);
18:
19: // third game:A vs C, A won
20: perfA3 = Gaussian(skillA,15);
21: perfC3 = Gaussian(skillC,15);
22: observe(perfA3 > perfC3);
23: return skillA;

```

(a) TrueSkill model.

```

1: simAll = Uniform (0 ,1);
2: clicksA = [1, 1, 0];
3: clicksB = [1, 1, 0];
4: for i in [0..3] {
5: sim = Bernoulli ( simAll );
6: if (sim) {
7: p1 = Uniform (0 ,1);
8: p2 = p1;
9: } else {
10: p1 = Uniform (0 ,1);
11: p2 = Uniform (0 ,1);
12: }
13: clickA = Bernoulli (p1 );
14: clickB = Bernoulli (p2 );
15: observe ( clickA == clicksA [i ]);
16: observe ( clickB == clicksB [i ]);}
17: return simAll ;
18:
19:

```

(c) Reduced ClickGraph model.

```

1: bool earthquake, burglary, alarm;
2: bool phoneWorking, maryWakes,called;
3: earthquake = Bernoulli(0.001);
4: burglary = Bernoulli(0.01);
5:
6: alarm = earthquake || burglary;
7:
8: if(earthquake){
9: phoneWorking = Bernoulli(0.6);
10: }else{
11: phoneWorking = Bernoulli(0.99);}
12: if(alarm && earthquake){
13: maryWakes = Bernoulli(0.8);
14: }else if(alarm){
15: maryWakes = Bernoulli(0.6);
16: }else{
17: maryWakes = Bernoulli(0.2);
18: }
19: called = maryWakes && phoneWorking;
20: observe(called);
21: return burglary;
22:
23:

```

(b) Pearl's burglar alarm model.

```

1: prob3 = [1/3, 1/3, 1/3];
2: prob2 = [1/2, 1/2];
3: carDoor = Categorical.Sample(prob3);
4: chosenDoor = Categorical.Sample(prob3);
5: if (carDoor == chosenDoor){
6: j = 0;
7: for i in [0..3]{
8: if (i != carDoor){
9: possibleOpenDoors[j] = i;
10: j=j+1;}}
11: j=Categorical.Sample(prob2);
12: openedDoor = possibleOpenDoors[j];
13: }else{
14: for i in [0..3]{
15: if ((i != carDoor) && (i != chosenDoor)){
16: openedDoor = i;}}
17: win = false;
18: for i in [0..3]{
19: if ((i != chosenDoor) && (i != openedDoor)){
20: win = (i == carDoor);}
21: return win;}

```

(d) Monty Hall model.