



Discrete-Time and Continuous-Time Systems over the Reals: Relating Complexity with Robustness, Length and Precision

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

École doctorale n°626 École doctorale de l'Institut Polytechnique de Paris (EDIPP)
Spécialité de doctorat : Informatique, données, IA

Thèse présentée et soutenue à Palaiseau, le 28 mai 2025, par

MANON BLANC

Composition du Jury :

Pierre Valarcher Professeur des universités, Université Paris-Est Créteil (LACL)	Président
Enrico Formenti Professeur, Université Côte-d'Azur (I3S)	Rapporteur
Elvira Mayordomo Professeure des universités, Universidad de Zaragoza (I3A)	Rapporteuse
Luis Cristóbal Rojas González Professeur, Pontificia Universidad Católica de Chile (IMC)	Rapporteur
Nathalie Aubrun Directrice de recherche, Université Paris-Saclay (LISN)	Co-directrice de thèse
Olivier Bournez Professeur, École Polytechnique (LIX)	Co-directeur de thèse
Juha Kontinen Professeur, Helsingin Yliopisto (Department of Mathematics and Statistics)	Examineur

Présentation en français

Cette thèse a été réalisée en co-direction entre l'École Polytechnique et l'Université Paris-Saclay.



Le monde physique qui nous entoure est continu et décrit par des équations différentielles ordinaires (EDO): par exemple, nos ordinateurs sont constitués de composants électroniques analogiques, décrits naturellement en physique par certaines EDO. Mais, dans les ordinateurs d'aujourd'hui, ce n'est pas le modèle utilisé: les composants sont contraints de vivre dans un régime vrai/faux. Ainsi, les modèles mathématiques de calcul, comme les machines de Turing, sont discrets: ils fonctionnent sur des mots d'un alphabet fini, avec des pas, c'est-à-dire un temps discret. Cependant, nous avons aussi des modèles mathématiques pour les machines analogiques, et les ordinateurs analogiques, comme le "General Purpose Analog Computer" proposé par Claude Shannon, qui peuvent être vus comme le pendant des machines de Turing pour le monde analogique.

Ma contribution porte sur la puissance de calcul de tels modèles : comprendre si nous pourrions calculer plus et plus vite avec eux. La calculabilité et la théorie de la complexité sont un moyen de classer la difficulté des problèmes. La calculabilité vise à diviser les problèmes entre ce que nous pouvons résoudre avec un modèle de calcul (décidable ou calculable), et ce que nous ne pouvons pas. Parmi les problèmes résolubles, il est intéressant de savoir combien de ressources sont utilisées: c'est là qu'intervient la théorie de la complexité. Selon le temps et la mémoire (on parle plutôt *d'espace* que de mémoire dans ce contexte) nécessaires, ces problèmes appartiennent à des classes différentes. Le modèle de calcul que l'informatique utilise habituellement pour définir ces classes est celui des machines de Turing. Elles sont bien définies pour les problèmes traitant uniquement d'entiers, car elles ont une représentation finie, autrement dit, pour les modèles discrets, et les calculs binaires. Pour comparer les approches, nous avons besoin d'un moyen de mesurer le coût des calculs dans des contextes continus. *Nous avons développé et prouvé des caractérisations de classes de complexité pour les calculs utilisant des nombres réels et des modèles analogiques.*

O. Bournez et A. Durand ont établi une caractérisation du temps polynomial (PTIME) dans le monde discret et nous avons étendu ce résultat aux paramètres continus: c'est-à-dire aux fonctions sur les réels. *Nous avons prouvé une caractérisation algébrique de suites réelles calculables en temps polynomial* dans notre publication à MCU 2022 (Best Student Paper Award) et l'International Journal of Foundations of Computer Science. Nous avons ensuite généralisé ce résultat et prouvé des caractérisations algébriques de PTIME et de l'espace polynomial (PSPACE) pour des fonctions sur les réels. Ces travaux sont publiés à la conférence MFCS 2023 (Best Paper Award). Avec ces caractérisations algébriques, nous avons prouvé que *le calcul d'une fonction en temps polynomial ou en espace polynomial équivaut à résoudre une EDO discrète* (aussi appelée différences finies). Nous lions le modèle basé sur des équations différentielles à l'analyse calculable.

Maintenant que nous avons des caractérisations avec des EDO discrètes, nous avons

voulu utiliser des EDO continues, pour couvrir le modèle de Shannon. A. Pouly a prouvé le lien avec le temps, qui correspond à la longueur de la dynamique représentée par l'EDO. Nous avons ici prouvé un lien avec l'espace, qui est associé à la précision du calcul. *Nous sommes maintenant capables de mesurer l'espace pour un système physique.* En utilisant ces résultats, nous avons réussi à prouver une caractérisation algébrique de PSPACE avec une classe d'EDO continues, prolongeant ainsi le travail d'A. Kawamura. Nous avons proposé une manière originale et assez inhabituelle de résoudre les EDO: nous les résolvons de manière récursive, elle s'inspire de la preuve du théorème de Savitch, et cela permet de garantir l'optimalité en mémoire (espace). *Nous avons prouvé le lien entre l'espace polynomial et la précision polynomiale dans la résolution de l'EDO.* L'avantage est que les EDO continues pourraient être plus naturelles que les EDO discrètes, car elles sont étudiées dans davantage de domaines scientifiques. Cela a donné lieu à un autre article publié à la conférence ICALP 2024.

Un autre axe de recherche et point de vue que j'ai étudié pendant ma thèse porte sur la robustesse des systèmes dynamiques et les applications en théorie de la complexité et en théorie des pavages. Il est bien connu que le raisonnement sur les systèmes dynamiques évoluant sur les réels conduit à l'indécidabilité. Cela vient principalement du fait que l'accessibilité des systèmes dynamiques sur les réels est fondamentalement indécidable, car les machines de Turing peuvent être intégrées dans des systèmes dynamiques (même très simples). Cependant, divers résultats dans la littérature ont montré que des procédures de décision existent lorsque on se limite à des systèmes robustes, avec une notion de robustesse convenablement choisie. En particulier, dans le domaine de la vérification, si l'accessibilité des états n'est pas sensible aux petites perturbations, alors des procédures de décision pour l'accessibilité des états existent. Dans le contexte des théories logiques sur les réels, il a été établi que des procédures de décision existent en se concentrant sur des propriétés insensibles à des perturbations arbitrairement petites.

Nous proposons d'abord une théorie unifiée expliquant dans un cadre uniforme ces énoncés, qui ont été établis dans des contextes différents. Plus fondamentalement, bien que tous ces énoncés ne concernent que des questions de calculabilité, nous considérons également des aspects de la théorie de la complexité. *Nous prouvons que la robustesse à une certaine précision est intrinsèquement liée à la complexité de la procédure de décision.* Lorsqu'un système est robuste, il est logique de quantifier à quel niveau de perturbation il se trouve. Nous prouvons que supposer une robustesse à une perturbation polynomiale sur la précision conduit à une caractérisation de PSPACE. Nous prouvons que supposer une robustesse à une perturbation polynomiale sur le temps ou la longueur conduit à des énoncés similaires pour PTIME. En d'autres termes, *nous avons prouvé que la précision des calculs est intrinsèquement liée à la complexité spatiale, tandis que la longueur ou le temps des trajectoires est intrinsèquement lié à la complexité temporelle.* Cela a donné lieu à une publication à la conférence CSL 2024. Nous avons étendu ces résultats à d'autres types de systèmes dynamiques: les pavages, donnant lieu à une soumission en conférence. Nous soulignons *le rôle joué par des concepts venant de la topologie dans les calculs informatiques, en lien avec tous les résultats précédents.*

Contents

Acknowledgements	6
Introduction	8
1 Mathematical background	14
1.1 Some Analysis and Topology	14
1.1.1 Topology	14
1.1.2 A classical statement from numerical analysis	17
1.2 Concepts of Ordinary Differential Equations	18
1.2.1 <i>Continuous</i> Ordinary Differential Equations	18
1.2.2 <i>Discrete</i> Ordinary Differential Equations	20
1.3 Computability, Complexity, Computable Analysis	23
1.3.1 Some notions of <i>classical</i> computability and complexity theory	23
1.3.2 On computable analysis: Computability	32
1.3.3 On computable analysis: Complexity	35
1.3.4 Reductions in continuous settings	38
1.4 Dynamical Systems	41
1.5 Tiling theory	44
1.5.1 Tilings as dynamical systems	44
1.5.2 Minimality and Essential Minimality	46
1.5.3 Tiles and Tilings	47
1.5.4 Transducers and meta-transducers	49
1.6 Notations Tabulars	52
2 State of the art	54
2.1 Solving efficiently ODEs: what is known	54
2.2 A variant definition of Turing machines	57
2.3 Dynamical systems and associated complexity issues	57
2.3.1 Embedding TMs into dynamical systems	57
2.3.2 Some complexity results on graphs	60
2.4 Implicit complexity	62
2.5 Using Discrete ODEs in complexity	64
2.5.1 Length Ordinary Differential Equations	64
2.5.2 Implicit definition of polynomial time	68
2.6 A previous characterisation of FPSPACE	74
2.7 Programs and Turing machines	75

3	Algebraic Characterisations of \mathbf{FPTIME} with Discrete ODEs	78
3.1	Algebraic characterisation of the real sequences computable in polynomial time	79
3.1.1	Towards functions from integers to the reals: simulation with Turing machines	81
3.1.2	Proof of Theorem 3.1.1: $f \in \mathbf{FPTIME}$ is equivalent to the existence of some $\tilde{f} \in \mathbb{LDL}^\bullet$	83
3.1.3	Proving Theorems 3.1.5 and 3.1.6: $\overline{\mathbb{LDL}^\bullet}$ characterises \mathbf{FPTIME} for real sequences	86
3.2	Algebraic characterisation of \mathbf{FPTIME} for functions over the reals	89
3.2.1	Preliminary results about continuous functions	90
3.2.2	Simulating Turing machines with functions of \mathbb{LDL}°	101
3.2.3	Converting integers and dyadics to words, and conversely	104
3.2.4	Proof of Theorem 3.2.3: $\overline{\mathbb{LDL}^\circ}$ characterises \mathbf{FPTIME} for real functions	105
3.3	Chapter Conclusion	109
4	Robustness and applications	110
4.1	Perturbed TMs	111
4.2	Discrete-Time Dynamical Systems	116
4.2.1	The case of rational systems	116
4.2.2	The case of computable systems	124
4.3	Relating robustness to drawability	127
4.4	Continuous-time systems	129
4.5	Other type of perturbations	131
4.6	Properties of provably robust TM	134
4.7	Chapter Conclusion	136
5	Algebraic Characterisations of $\mathbf{FPSPACE}$	137
5.1	Algebraic characterisation of \mathbf{PSPACE} for functions over the reals with discrete ODEs	138
5.1.1	Characterisation of $\mathbf{FPSPACE}$ and generalisations	139
5.2	Characterisation with Continuous ODEs	140
5.2.1	Solving efficiently ODEs: a space efficient method	141
5.2.2	Simulating a discrete ODE using a continuous ODE	143
5.2.3	Revisiting some previous constructions	146
5.2.4	Constructing the missing functions	150
5.2.5	Working with all steps of a Turing machines	153
5.2.6	Proof of Theorem 5.2.1: the algebra also characterises $\mathbf{FPSPACE}$ in discrete settings	154
5.2.7	Proof of Theorem 5.2.2: $\overline{\mathbb{RCD}}$ characterises $\mathbf{FPSPACE}$ for real functions	155
5.3	A completeness result on the reachability of dynamical systems	155
5.4	Chapter Conclusion	157

6	Robustness in Tilings and Subshifts	159
6.1	Some representation aspects	160
6.1.1	About representation of involved sets	160
6.1.2	About allowed inputs	161
6.2	Viability and Invariance Kernels	161
6.2.1	Some vocabulary from verification, control and dynamical systems theory	162
6.2.2	Robustness of dynamical systems	164
6.2.3	Space-time of dynamical systems and related problems	168
6.2.4	Relating robustness to reachability	169
6.2.5	Reachability and Witness of non-reachability	171
6.2.6	A specific case	173
6.3	Robustness of Turing machines	173
6.3.1	Space-time diagrams of a Turing machine	173
6.4	Robust Subshifts	174
6.4.1	Computability and uncomputability issues for subshifts	174
6.4.2	Robustness of subshifts	176
6.5	The Robinson tileset is polynomially robust	176
6.5.1	Robustness and invariants	179
6.6	Application to other tilesets	181
6.6.1	Extension: Tiling equivalence	181
6.6.2	Jeandel-Rao's tileset	181
6.6.3	Kari's tilings	182
6.7	Other types of robustness	184
6.7.1	Robinson's tilings described by a recurrence relation	186
6.7.2	Semantically robust tilesets	190
6.7.3	Properties	190
6.7.4	Provably robust tilesets	191
6.8	Chapter Conclusion	193
	Conclusion	194
	Bibliography	197

Acknowledgements

Hygge er en tilstand man oplever, hvis man har fred med sin selv, sin ægtfælle, skattevæsnet og sine indre organer.

Tove Ditlevsen

First of all, I would like to thank my reviewers, Enrico Formenti, Elvira Mayordomo and Cristóbal Rojas, for accepting to read my (rather long) manuscript and their useful comments. Thank you for your reports and and your encouragements.

I thank Enrico Formenti, Juha Kontinen, Elvira Mayordomo, Cristóbal Rojas and Pierre Valarcher for accepting to be in my jury.

Je suis très reconnaissante envers Nathalie Aubrun et Olivier Bournez, mes directeur-ices de thèse, pour leur soutien au cours de ces 3 années. Ce travail aurait été impossible sans leurs confiances et leurs précieux conseils.

Je remercie mes laboratoires, le LIX et le LISN, pour m'avoir accueillie pendant 3 ans, plus précisément les équipes AICo, COSYNUS, Grace¹ et GALaC. Plus particulièrement merci à mes co-doctorant-es Bernardo, Émile, Mathilde, Pierre, Nicolas, Valentin et Nuwa. Merci à Johanne d'avoir officieusement relu mes dossiers de candidature et m'avoir fait répéter ma soutenance.

Merci aux gestionnaires et anciennes gestionnaires de mon équipe et du laboratoire: Vanessa, Bahareh, Héléna, Fatima et Jackie. C'est grâce à vous que tout tourne bien et que nous pouvons partir en mission sereinement.

Merci à Arnaud Durand et Pierre Valarcher pour avoir été un super comité de suivi de thèse. Merci à eux de m'avoir écoutée et encouragée.

Merci à mes relecteurices de manuscrit, Olivier, Nathalie, Louis, Bernardo² et Dread. J'espère que j'ai implémenté tous vos conseils.

Je tiens à remercier Louis pour m'avoir hébergée chez lui à Edinburgh pendant 1 mois pour rédiger mon manuscrit (et fait en sorte que je survive aux randonnées écossaises de février). Dziękuję bardzo, pour ça et pour tout le reste.

Mes études supérieures n'ont pas toujours été faciles pour pleins de raison différentes. Et certaines personnes m'ont fortement influencée à continuer et à ne pas lâcher. En particulier, je remercie Frédéric Simon, Laurent Sartre et Benoît Eloseguy, mes professeurs de mathématiques, informatique et physique en première année de prépa. A niveau de l'ENS Cachan³, je remercie Serge Haddad pour sa confiance. Et surtout, j'ai énormément de gratitude envers Hubert Comon-Lundh pour son soutien pendant ma scolarité à l'ENS. Merci de m'avoir redonné confiance en mes capacités⁴.

Pendant ma thèse, il n'y a évidemment pas eu que de la science. Je tiens à remercier Marie, Adrien, Louis et Elric pour nos soirées Escape Game (on a toujours réussi à sortir, et ce malgré les caprices des actrices). Je remercie Dread et Louis pour les concerts et les festivals, les découvertes de nouveaux groupes et les circle pits. Merci à Bernardo pour la découverte du cinéma brésilien. Merci à Bastien pour les soirées tisanes. Merci à Blandine pour les balades parisiennes. Merci à Alex et Rosemonde pour le cat-sitting.

¹ Kong

² obrigado para você

³ ou Paris-Saclay, on peut dire les deux

⁴ Je sais qu'il en a assez que je le remercie pour cela, mais ce sont mes remerciement de thèse, et je suis têtue

Merci à Zéphyr pour nos conversations toujours intéressantes. Merci à Baptiste et Simon pour les soirées après les conférences et pour m'avoir montrée les mille nuances du pastis. Merci à ma soeur, Margaux, pour les matchs de rugby et les sorties au bar.

Jeg takker mine venner fra den Danske Skole, Tom, Thomas, Clémence, Olivier, Joséphine, Armande, Frederike, Véronique and Kyrian, og vores vidunderlig lærer, Lotte Nør-Larsen. Det var meget godt at lave og lære andre ting end videnskab. Tak for vores ugentlige møder.

Commencer des études supérieures n'aurait évidemment pas été possible sans le soutien de mes parents, Fabienne et Serge.

Pendant assez longtemps, je n'ai pas eu l'impression de souffrir du manque de femmes en sciences⁵. Après de nombreuses discussions, je me suis aperçue de tous les biais dont on était pétri-es, et d'à quel point le sexisme (ordinaire) est présent, même dans nos laboratoires. Il faut que ça change. Il faut que nous prenions collectivement conscience du problème et faire mieux. Je remercie donc toutes les chercheuses qui m'ont inspirée, notamment Johanne Cohen, Nathalie Aubrun et Marie Kerjean, pour m'avoir montré que c'était normal de ne pas se laisser faire. C'est une posture que je n'oublierai pas.

Je dédie ma thèse à vous toutes

⁵Je pointe ici sur un problème parmi tant d'autres

Introduction

C'est faisable, il faut juste le faire.
Olivier Bournez

Our physical world is continuous and described by ordinary differential equations (ODEs). Our computers are made of analogue electronic components, naturally described in physics by some ODEs. In today's computers, everything is true or false: components are forced to live in a true/false regime. Consequently, the classical mathematical models of computation, such as Turing machines, are discrete: they work on words over a finite alphabet, with steps, i.e. a discrete time. At the same time, we also have mathematical models for analogue machines with discrete- and continuous-time, and analogue computers, such as the General Purpose Analog Computer (GPAC) proposed by Shannon, that can be seen as the counterpart of Turing machines for the analogue world.

We are interested in computability and complexity over the reals. There are various models of computation for real numbers in computer science: some are in discrete-time, others are in continuous-time. For studying those models, we will mostly focus on the framework of *computable analysis*: it is based on Turing machines, which are very natural to consider in computer science.

From a complexity point of view, we will study complexity classes doing computations over real numbers. There are several ways to define complexity classes. Usually, we use Turing machines to do so. However, this model is a rather heavy framework. Moreover, it is not widespread in other scientific communities, and being able to do efficient computations is critical in many areas of fundamental and applied science, for example in physics or biology. Also, those domains use continuous settings. We want to abstract from Turing machines to have a more natural model of computation for every community. We aim to give understandable translations of complexity classes over the reals. As a side-effect, we prove that it is possible to program with ODEs and ensure that we can do it efficiently. ODEs also relate mathematics, computer science, and physics, paving the way to a common research goal.

Since the continuous framework is rather difficult to deal with, we will start by simplifying it. We study discrete-time systems on continuous space (in Chapters 3, 4, 5 and 6) and even discrete-time systems on discrete spaces (in Chapters 3 and 6). We generalise these frameworks afterwards to all-continuous settings.

Also, inspired by complexity, we study computability questions. Many simply-stated problems turn out to be undecidable. We show that, by slightly twisting some of them, they can become decidable. We aim to draw more precisely the border between what is computable and what is not.

We are specifically interested in studying the complexity properties of dynamical systems. As many systems in our world are naturally modelled by dynamical systems over

the reals (or by so-called hybrid systems, mixing continuous and discrete aspects), verification of safety properties on these systems is inherently related to state reachability for dynamical systems. Roughly speaking, a system is safe if the subset of “*bad states*” (i.e. those not satisfying some property) cannot be reached from the subset of the initial states of the system. Unfortunately, it is well-known that such questions are undecidable, even for elementary dynamical systems. For example, it is undecidable even for systems with piecewise affine functions [KCG94, Moo91] over the compact domain $[0, 1]^2$. Also, the computational power of dynamical systems is at least as powerful as Turing machines. Thus, studying dynamical systems becomes a good way to study the questions we are interested in.

Such undecidability statements are proved by showing that a Turing machine, a particular discrete-time dynamical system, evolving with time over configurations, can be embedded into dynamical systems. This requires mapping the infinite set of possible configurations of a Turing machine into a real domain. Hence, the simulation of a Turing machine requires an infinite precision encoding when the system is defined over a compact domain. This establishes that verification is undecidable for systems with infinite precisions. However, as many authors observe (see e.g. [AB01]), this does not prove that undecidability holds for systems needing infinite precision computations.

In that spirit, while several undecidability results were stated for hybrid systems, such as Linear Hybrid Automata [HKPV98], Piecewise Constant Derivative systems [AMP95], an informal conjecture (that we will call the “*robustness conjecture*”) appeared in the field of verification of hybrid and continuous systems by various authors. It states that *undecidability is due to non-stability, non-robustness and sensitivity to the initial values of the systems*. There were several attempts to formalise and prove this, including [Frä99, AB01].

Nota Bene (NB) 0.0.1

The “robustness conjecture” is an acknowledged informal conjecture, as it is related to the mathematical notion of robustness. It holds for some mathematical concepts of robustness, such as the ones considered in this document. It is provably false for some other frameworks: see e.g. [HR00].

The mathematical formalisation of robustness, or the question of the various “natural” concepts of robustness, is related to philosophical questions about the limits of mathematical models, handled by computability theory. It is important to highlight that a Turing machine is an ideal model, as well as dynamical systems over the reals are often idealisations of models. We do not aim to review these kinds of discussions, even if our results shed some light on these questions, such as the fact that complexity theory is associated with quantifying the accepted level of robustness.

A long-standing open problem was how to measure time complexity in continuous time models. It was recently proved [BGP17] that the length of the solution curves provides a measure equivalent to time for digital models. In the spirit of avoiding the model of Turing machines and measuring *time* as the *length* of a curve, we will develop in Chapter 3 alternative characterisations of polynomial time, as finite sets of functions stable under finite sets of operators. We call such characterisations *algebraic*.

NB 0.0.2

Turing machines is an operationnal model: the computations are made very explicitly, step-by-step. An algebraic characterisation, like the ones we will state and

prove in Chapter 3 and also in Chapter 5, provides a denotational point of view: it works on the mathematical representation.

We consider in Chapter 4 the approach perturbation of [AB01], where classes of dynamical systems are considered. A notion of perturbed dynamics by a small ε is associated with each system. We define a perturbed reachability relation as the intersection of all reachability relations obtained by ε -perturbations. The authors of [AB01] showed that, for many models, the perturbed reachability relation is co-computably enumerable (co-c.e., Π_1) and any co-c.e. relation can be defined as the perturbed reachability relation of such models. Consequently, it follows from basic computability arguments, namely that a computably enumerable and co-computably enumerable set is decidable that, *if robustness is defined as the stability of the reachability relation under infinitesimal perturbation*, then robust systems have a decidable reachability relation and hence a decidable verification (i.e. the *robustness conjecture* holds).

In the context of decision procedures for logic over the reals, the authors of [GAC12] observed that it is well-known that some logics, such as real arithmetic, are decidable. However, decidability does not hold for simple extensions of real arithmetic. Indeed, even the set of Σ_1 -sentences in a language extending real arithmetic with the sine function is already undecidable. If a relaxed and more “robust” notion of correctness is considered (one asks to answer true when a given formula ϕ is true and to return false when it is δ -robustly wrong) the truth of a formula becomes algorithmically solvable. In other words, undecidability intrinsically comes from the fact that the truth of a sentence might depend on infinitesimally small variations of its interpretation.

Recently, the author of [Rat23] proposed a first-order predicate language for reasoning about multi-dimensional smooth real-valued functions. They proved the specification of an algorithm solving formulas robustly satisfiable, up to some metrics. The proof of decidability can also be interpreted using an argument similar to [AB01, GAC12].

The question of a natural measure for space complexity remained open, despite some very recent characterisations of **(F)PSPACE** using ODEs [BGGP23]. We will solve it in Chapter 5, by proposing a measure and a rather simple characterisation of polynomial space. The theory developed in Chapter 4 and Chapter 5 comes from an attempt to get to simpler (algebraic) characterisations of **(F)PSPACE**, with continuous ODEs. It also allows us to relate the concept of *space* to the *precision* of the computation. It can be related to [BGP16b, BGP17]: the authors of these articles provide a characterisation of **PTIME** with *continuous* ODEs, establishing that time complexity corresponds to the length of the involved curve, i.e. the motto **time complexity = length**. Here, we get a maxim of the form **space complexity = precision**: we characterise space for continuous-time systems. The proofs of the algebraic characterisations will rely on continuously approximating non-continuous functions, such as the fractional part, and encoding the execution of the underlying Turing machine into the algebras. We obtained this theory initially with the idea that getting to **(F)PSPACE** requires a way to forbid undecidability. This led us to develop this theory based on these notions of robustness, guaranteeing computability.

Let us review more thoroughly the contributions of each chapter:

To give some context, we start by introducing, in Chapter 1, concepts of mathematics and computer science necessary to understand the rest of the document. We will also describe some previous, somewhat non-classical, works on the subjects we tackle in Chapter 2.

Chapter 3 We give characterisations of polynomial time over the reals with discrete ODEs. First, we characterise real sequences (Theorem 3.1.6), then functions over the reals (Theorem 3.2.3). The core of the proofs relies on encoding the execution of Turing machines into particular discrete ODEs, on which we know or prove complexity properties. Characterisations of polynomial time with discrete ODEs existed before, but only for decision problems. Here we characterise sequences and functions over the reals using *robust* discrete ODEs.

Chapter 4 Using various arguments from computability and computable analysis, we prove that being robust for a dynamical system implies that its reachability relation is decidable. Also, we directly relate the level of robustness to the complexity of associated decision problems.

More precisely, we establish the following:

- We consider various classes of dynamical systems: in turn, Turing machines, then discrete-time dynamical systems preserving the rationals, then general discrete-time dynamical systems and eventually continuous-time dynamical systems. For all of them, we define robustness as non-sensitivity to infinitesimal perturbations of the associated reachability relation, in the spirit of [AB01].
 - We prove that for this natural concept of robustness, the “*robustness conjecture*” holds: verification or reachability relation (and hence safety verification) is decidable.
 - We characterise and relate robustness to decidability by proving that the converse holds if some property is added (Corollary 4.2.19). This means that there is a form of completeness of the above statement: when decidability holds, establishing the robustness of the corresponding system can be done for a suitable notion of perturbation or metric.
 - We relate this notion of robustness of Turing machines to a programming languages and logic point of view based on provability and Hoare logic.
- Furthermore, we relate this approach, inspired by [AB01] from verification, to the concept of δ -decision of [GAC12], from the context of decision procedures in logic. A system is robust iff its reachability relation is either true or ε -far from being true (Proposition 4.2.14).
- We also prove that robustness can be seen as having a reachability relation depicted by a pixelated image. It is a simple and elegant geometric property (Corollary 4.3.4 and 4.3.5).
- More fundamentally, in the whole chapter, while the above results are about decidability, we also discuss *complexity* issues. Indeed, when a system is robust, it is natural to quantify the perturbation level we allow.
 - We show that considering a perturbation polynomially small relates in a very intuitive way to the complexity of the associated verification or decision problem (Theorem 4.1.11, Theorem 4.2.24).

- More precisely, polynomial space computability is related to precision (Theorem 4.1.11, Theorem 4.2.24). Polynomial time computability is related to the time or length of the trajectory (Theorem 4.5.10).

Chapter 5 Now that we have characterisations for polynomial time with discrete ODEs in Chapter 3, it makes sense to characterise space complexity. Furthermore, we wanted to use continuous ODEs instead of discrete ODEs to get closer to models such as the GPAC. Amaury Pouly in [Pou15] proved their link with time: it corresponds to the length of the dynamic represented by the ODE. Here, we proved a link with space, which is associated with the precision of the computation.

In this chapter, we solve the problem of having a proper way to measure space complexity in continuous settings. The theory developed here provides arguments to state that, over a compact domain, space corresponds to the precision of the computations. More precisely, it corresponds to the logarithm of the size of some graphs for systems over more general domains. Meanwhile, this idea has been used to provide a simple characterisation of **FSPACE** with discrete ordinary differential equations (Theorem 5.1.6).

We are now able to measure space of an actual physical system. Using these results, we managed to prove an algebraic characterisation of **PSPACE** with a class of continuous ODEs (Theorem 5.2.2), also using the work of Akitoshi Kawamura, to be seen. As a side-effect, we propose a quite unusual way to solve ODEs: we solve them recursively. It is inspired by the proof of the Savitch theorem (Theorem 1.3.30). We proved the link between polynomial space and polynomial precision in the solution of the ODE. The advantage is that continuous ODEs might be more natural than discrete ODEs, as they are studied in more fields of science.

Defining a robust and well-defined computation theory for continuous-time computations was an open problem and not an easy one: see [BP21] for the most recent survey. In short, the problem with time complexity is that considering the time variable as a measure of time is not robust: a curve can always be re-parameterised using a change of variable. The problem with space complexity is similar: reparameterisation corresponds to a change of time variable, but also of space-variable, introducing space and time contractions: see e.g. [BP21, BGP17]. Furthermore, many problems for simple dynamical systems are known to be undecidable, hence forbid **PSPACE**-completeness: see [GZB06] and [GZ18].

Chapter 6 We extend the concepts of robustness of Chapter 4 regarding reachability to more general settings, namely invariance properties. Another main difference is that, we adopt here a more topological point of view. We study the specific case of tilings and subshifts. We prove that it provides explanations of phenomena observed in the literature.

In this chapter, we adapt tools, concepts and arguments from the continuum to discrete objects. While concepts from computer science lead naturally to reachability questions (e.g. “does some Turing machine reach a halting state”), the dual view leads naturally to viability questions (e.g. “does some non-deterministic Turing machine run forever” or invariance questions (e.g. “what are the inputs on which the machine will run forever”). Some of these questions can be reduced from one to another (e.g. “a machine with no rejecting state never halts on some input if it does not reach a halting state”). Those problems remain natural and of clear practical interest for applications [AMJ24].

We extend this approach to the dual questions mentioned above. In addition to the applications contexts of [AB01] and Chapter 4, we prove that we can extend it to questions and concepts about objects that are nowadays considered purely discrete questions, namely tilings and subshifts. In particular, we provide a formal explanation of the phenomenon observed experimentally in simulations of tilesets such as the one of [JR21].

We consider the context of tilings and subshifts. The domino problem is known to be undecidable in general, even for sets of Wang tiles: unit squares tiles with colours on their edges, introduced in the early '60s to study the undecidability of the $\forall\exists\forall$ fragment of first-order logic [Wan61].

We prove the Domino problem is decidable for *robust* tilesets and that it holds for many other tilesets of the literature, including the Robinson tileset [Rob71] (Section 6.5). A tileset is robust if the membership of its patterns is decided by reasoning on strips of the plane. Applying our general theory, we prove that if the number of strips involved by a given pattern is polynomial, then the Domino problem is solvable in polynomial space. We show the undecidability of the Domino problem holds in Berger's construction because it involves a non-robust Turing machine, i.e. a non-robust tiling. We argue it is true for all tilesets in the literature unless they are produced from non-robust Turing machines.

- We start from the ideas of [AB01] and Chapter 4. While they were restricted to point-to-point computability, our theory is widely more general: it also covers the computation of invariance kernel, invariance envelope, or reachability sets (sets-computations, with all the associated difficulties).
- Doing so, we specialise and extend several statements known in control theory for differential inclusions [SP94, Aub91] or hybrid systems [Lyg04] to the theory of discrete time dynamical systems. Several statements seem original, even if inspired by this rather different context.
- We prove that robustness concepts from [AB01] have a topological interpretation (Definition 6.2.6, Proposition 6.2.7, Corollary 6.2.8). No such explanations have been observed. Another side effect is that many statements stated in the tiling subshift community get an interpretation regarding statements for dynamical systems over the continuum and conversely (e.g. minimality of a subshift and the invariance kernel), used in our arguments.
- We prove that several statements from Chapter 4 have a wider interpretation. We extend several statements to non-deterministic systems and over more general time-space (group actions), needed for our theory about subshifts.

Chapter 1

Mathematical background

But once an object is found inedible, that does not always mean it's uninteresting.

Peter Godfrey-Smith, Other Minds: The Octopus, the Sea, and the Deep Origins of Consciousness

In this chapter, we shall introduce the concepts we need in the other chapters, covering the theory of Ordinary Differential Equations, Computable Analysis, Computability and Complexity, Dynamical Systems and Tilings. The objective is to find and prove links between all those concepts. Analysis and complexity remain the backbones of all our results. We apply various notions of both domains to characterise classes and study the complexity properties of different types of dynamical systems.

A tabular referencing the notations we use can be found at the end of this chapter.

1.1 Some Analysis and Topology

We recall some usual concepts of topology and analysis. We introduce only the concepts we need throughout the chapter. Hence, we assume some familiarity with these concepts. See [Cla24] or [Rud91] and [Rud87] for a more comprehensive view.

1.1.1 Topology

Sets and spaces

Definition 1.1.1 (*Metric Space, e.g. in [Cla24]*)

A metric space is a pair (X, d) , where X is a set and $d : X \times X \rightarrow \mathbb{R}_+$ (for \mathbb{R}_+ the non-negative real numbers) a distance, that is to say, for $x, y, z \in X$:

- $d(x, x) = 0$;
- $d(x, y) \geq 0$ (positivity);
- $d(x, y) = d(y, x)$ (symmetry);
- $d(x, z) \leq d(x, y) + d(y, z)$ (triangle inequality).

Example 1.1.2

The pair $(\mathbb{R}, |\cdot|)$ is a metric space. The pair (Σ^ω, d_C) with Σ^ω the set of infinite words on alphabet Σ and d_C such that $d_C(w_1, w_2) = 2^{-\min\{i \mid (w_1)_i \neq (w_2)_i\}}$ (and is equal to 0 when $w_1 = w_2$) is also a metric space.

The metric spaces of Example 1.1.2 are the ones we will work on the most in this document.

Definition 1.1.3 (Compact set)

Let X be a metric space. A subset $S \subseteq X$ is compact if and only if (iff) every cover by a collection \mathcal{C} of open sets $\bigcup_{\mathcal{O} \in \mathcal{C}} \mathcal{O}$ of S has a finite subcover by a finite subcollection $\mathcal{F} \subseteq \mathcal{C}$: $S = \bigcup_{\mathcal{O} \in \mathcal{F}} \mathcal{O}$.

Definition 1.1.4 (Closure, e.g. in [Cla24])

Let X be a metric space. The closure of $S \subseteq X$, denoted by \bar{S} or $\text{cl}(S)$, is the set containing S and all its limit points.

Example 1.1.5

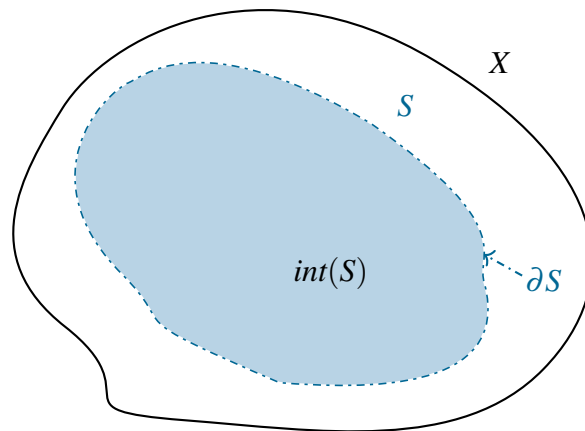
The closure of \mathbb{Q} is \mathbb{R} , as any real number is the limit of a rational sequence. Only real numbers can be reached with such sequences.

Definition 1.1.6 (Interior and Frontier, e.g. in [Cla24])

Let X be a metric space. The interior of $S \subseteq X$, denoted by $\text{int}(S)$, is the union of all the open subsets \mathcal{O} of X such that $\mathcal{O} \subseteq S$.
The frontier of S is $\partial(S) = \bar{S} - \text{int}(S)$.

Example 1.1.7

The interior of \mathbb{Q} is empty, and its frontier is \mathbb{R} . On $(\mathbb{R}, |\cdot|)$, $\text{int}([0, 1]) =]0, 1[$ and $\partial([0, 1]) = \{0, 1\}$



Definition 1.1.8 (Balls, e.g. in [Cla24])

Let (X, d) be a metric space. The (open) ball of radius ε centered on $x \in X$, denoted by $B_\varepsilon(x)$, is the set $\{y \in X \mid d(x, y) < \varepsilon\}$.

Definition 1.1.9 (Closed Rational Box)

A closed rational box X is a subset of \mathbb{R}^n of the form $X = \prod_{i=0}^{n-1} [a_i, b_i]$, with, for all $i \in \llbracket 0, n \rrbracket$, $a_i, b_i \in \mathbb{Q}$.

Note that, according to the definition, a rational box X is a subset of \mathbb{R}^n , not of rational numbers: it is a finite product of intervals. Consequently, such an X is compact for the topology of \mathbb{R}^n .

Functions

In the rest of the thesis, for A and B sets, we will often write B^A for the set of functions f such that $f : A \rightarrow B$. As we deal with complexity over the reals, measuring the resources we use during the computations is crucial. We define the following to do so:

Definition 1.1.10 (Modulus of convergence, e.g. in [Cla24])

Let $g : \mathbb{N} \rightarrow \mathbb{R}$ and $h : \mathbb{N}^2 \rightarrow \mathbb{R}$, with $g(n)$ and $h(\cdot, n)$ be functions converging to 0 when n goes to ∞ .

- A function $M : \mathbb{N} \rightarrow \mathbb{N}$ is a modulus of convergence of g iff for all $n \in \mathbb{N}$, for all $i > M(n)$, we have $\|g(i)\| \leq 2^{-n}$.
- A function $M : \mathbb{N} \rightarrow \mathbb{N}$ is a uniform modulus of convergence of a sequence h iff for all $n \in \mathbb{N}$, for all $i > M(n)$, we have, for all $m \in \mathbb{N}$, $\|h(m, i)\| \leq 2^{-n}$.

Intuitively, the modulus of convergence bounds the speed of convergence of a sequence.

Definition 1.1.11 (Continuity, e.g. in [Cla24])

Let (E, d_E) and (F, d_F) be two metric spaces. A function $f : E \rightarrow F$ is (uniformly) continuous iff for all $\varepsilon > 0$, there exists $\delta > 0$ such that, for all $x, y \in E$, $d_E(x, y) \leq \delta$ implies $d_F(f(x), f(y)) \leq \varepsilon$.

If E and F are subsets of \mathbb{R} , the definition becomes: $|x - y| \leq \delta \Rightarrow |f(x) - f(y)| \leq \varepsilon$. In this context, we will often write $x =_\delta y$ for $|x - y| \leq \delta$.

In Chapter 6, we will also have to consider set-valued functions, for which we consider a weaker versions of continuity:

Definition 1.1.12 (Set-valued map)

A set-valued map f is a function mapping any element of $\text{Dom}(f)$ to subsets of another set.

Definition 1.1.13 (Upper-semicontinuity, e.g. [Mus00])

Let $f : X \rightarrow \mathcal{P}(\mathbb{R} \cup \{-\infty, +\infty\})$ be a set-valued map with X a topological space. We say f is an upper-semicontinuous map in $x_0 \in X$ iff for all open set V with $f(x_0) \subseteq V$, there exists a neighbourhood U of x_0 such that, for all $x \in U$, $f(x) \subseteq V$.

Definition 1.1.14 (Lower-semicontinuity, e.g. [Mus00])

Let $f : X \rightarrow \mathcal{P}(\mathbb{R} \cup \{-\infty, +\infty\})$ be a set-valued map with X a topological space. We say f is an lower-semicontinuous map in $x_0 \in X$ iff for all open set V with $f(x_0) \cap V \neq \emptyset$, there exists a neighbourhood U of x_0 such that, for all $x \in U$, $f(x) \cap V \neq \emptyset$.

Definition 1.1.15 (Lipschitz functions, e.g. in [Cla24])

Let (E, d_E) and (F, d_F) be two metric spaces. Let $k \in \mathbb{R}_+$. $f : E \rightarrow F$ is k -Lipschitz iff for all $x, y \in E$, $d_F(f(x), f(y)) \leq k \cdot d_E(x, y)$.

For simplicity, we denote “ k -Lipschitz functions” for some k , by “Lipschitz functions”. When $F = \mathbb{R}$ and E is a subset of \mathbb{R} , the definition can be simply restated as: $|f(x) - f(y)| \leq k \cdot |x - y|$.

A function f is *locally Lipschitz* if every point $x \in \text{Dom}(f)$ has a neighbourhood U such that $f|_U$ is Lipschitz.

A function f is a contraction if it is k -Lipschitz with $k \in (0, 1)$.

We explicitly use those functions in Chapter 4 and Chapter 5.

1.1.2 A classical statement from numerical analysis

The field of numerical analysis studies ways to give numerical approximations to various analysis problems. See [Dem96] for more details. We give here a classical lemma that we will use later:

Lemma 1.1.16 (Discrete Grönwall’s lemma, e.g. [Dem96])

Consider sequences $(h_n)_{n \in \mathbb{N}}, (\theta_n)_{n \in \mathbb{N}} \in \mathbb{R}_+$, $\Lambda \in \mathbb{R}_+$, with \mathbb{R}_+ the set of non-negative reals, and $\varepsilon_n \in \mathbb{R}$ such that, for all $n \in \mathbb{N}$,

$$\theta_{n+1} \leq (1 + \Lambda h_n) \theta_n + |\varepsilon_n|.$$

Then, for $(t_n)_{n \in \mathbb{N}}$ such that $t_0 \in \mathbb{R}_+$ and $t_{n+1} = t_0 + \sum_{i=0}^{n-1} \Delta t_i$, $\Delta t_i = t_{i+1} - t_i$, we have:

$$\theta_n \leq e^{\Lambda(t_n - t_0)} \theta_0 + \sum_{0 \leq i \leq n-1} e^{\Lambda(t_n - t_{i+1})} |\varepsilon_i|.$$

Proof. By recurrence over n . For $n = 0$, the inequality $\theta_0 \leq \theta_0$ holds.

Suppose now the inequality at order n . Observe that $(1 + \Lambda h_n) \leq e^{\Lambda(t_{n+1} - t_n)}$.

By hypothesis, we have

$$\begin{aligned}\theta_{n+1} &\leq e^{\Lambda(t_{n+1}-t_n)}\theta_n + |\varepsilon_n| \\ &\leq e^{\Lambda(t_{n+1}-t_0)}\theta_0 + \sum_{0 \leq i \leq n-1} e^{\Lambda(t_{n+1}-t_{i+1})}|\varepsilon_i| + |\varepsilon_n|\end{aligned}$$

The inequality at order $n+1$ follows. \square

1.2 Concepts of Ordinary Differential Equations

In this section, we give some concepts and definitions for continuous and discrete ODEs. We assume some familiarity with differential equations: see [Cla24, Dem96].

1.2.1 Continuous Ordinary Differential Equations

In Chapter 5, we prove an algebraic characterisation of **FPSPACE** using *continuous* ODEs.

Definition 1.2.1 (Derivative, e.g. in [Cla24])

A function f such that $\text{Dom}(f) \subseteq \mathbb{R}$, is differentiable at a point x iff $x \in \text{Dom}(f)$ and there exists $L \in \mathbb{R}$ such that for all $\varepsilon > 0$, there exists some $\delta > 0$ such that for all $h < \delta$, $x+h \in \text{Dom}(f)$,

$$\left| L - \frac{f(x+h) - f(x)}{h} \right| < \varepsilon.$$

The derivative L of a function quantifies how much the function varies on an arbitrarily small interval. We denote by $\frac{\partial f}{\partial x}(x, \dots)$ the derivative of f with respect to x , and just f' when the derivation variable is not ambiguous. For $i \in \mathbb{N}$, we denote by $f^{(i)}$ the i 's derivative of f .

Definition 1.2.2 (Differentiability for a function)

A function f is differentiable on $X \subseteq \text{Dom}(f)$ iff it is differentiable on every point of X .

When it is not ambiguous, we say that “ f is differentiable”, without precising on which set, when it is differentiable on all its domain $\text{Dom}(f)$. We notice that a differentiable function f is Lipschitz iff the derivative of f is bounded. For $n \in \mathbb{N}$, we denote by \mathcal{C}^n the set of functions f differentiable n times and $f^{(n)}$ is continuous.

Theorem 1.2.3 (Fundamental theorem of analysis, e.g. in [Cla24])

Let f be a continuous function and F be a primitive of f on an open interval J . Then, for all $a, b \in J$ and $a < b$:

$$\int_a^b f(t)dt = F(b) - F(a)$$

Definition 1.2.4 (Ordinary Differential Equation, e.g. in [Cla24])

An ordinary differential equation (ODE) is a functional equation in $y \in \mathcal{C}^{n-1}(I, \mathbb{R})$, I an interval, for $n \in \mathbb{N} \cup \{+\infty\}$, and possibly depending on one parameter \mathbf{t} , such that

$$F(\mathbf{t}, y(\mathbf{t}), y'(\mathbf{t}), \dots, y^{(n)}(\mathbf{t})) = 0,$$

for $D \subseteq \bigcap_{i \leq n} \text{Dom}(y^{(i)}) \subseteq \mathbb{R}^n$ and $F : (I \times D^n) \subseteq (\mathbb{R} \times \mathbb{R}^n) \rightarrow \mathbb{R}$.

An ODE is *polynomial* when F is a polynomial.

Definition 1.2.5 (Initial Value Problem, e.g. [Pou15])

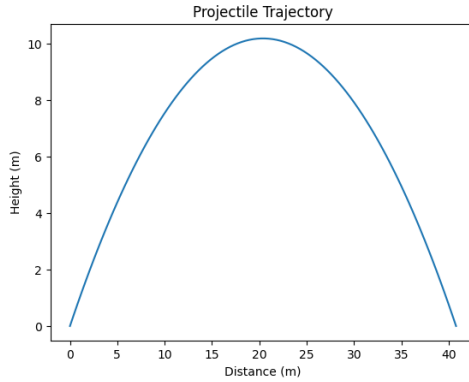
An initial value problem (IVP), also called Cauchy problem, is an ordinary differential equation together with an initial condition of the form $y(t_0) = y_0$ where $t_0 \in I$ and $y_0 \in \mathbb{R}$.

For example, every dynamic in physics, for example in mechanics, can be represented by an ODE as long as it is differentiable with a unique solution.

Example 1.2.6

Modelisation of the trajectory of an object thrown at initial velocity $v_0 = 20 \text{ m.s}^{-1}$ with an initial angle of $\theta = 45^\circ$. Re-using the notation of Definition 1.2.4,

$$F : t, \begin{pmatrix} x(t) \\ y(t) \end{pmatrix}, \begin{pmatrix} x'(t) \\ y'(t) \end{pmatrix} \rightarrow \begin{pmatrix} x'(t) - v_0 \cos(\theta) \\ y'(t) - v_0 \sin(\theta) + g \cdot t \end{pmatrix}.$$



$$\begin{cases} x(0) = v_0 \cdot \cos(\theta) \\ x(t) = v_0 \cdot \cos(\theta) + t \\ y(0) = v_0 \cdot \sin(\theta) \\ y(t) = v_0 \cdot \sin(\theta) \cdot t - 0.5 \cdot g \cdot t^2 \end{cases}$$

Cauchy problems, described by ODEs, have a good property on the solutions of IVPs:

Theorem 1.2.7 (Cauchy-Lipschitz, Picard–Lindelöf, e.g. in [Cla24])

Consider the IVP:

$$\begin{aligned} y'(t) &= f(t, y(t)) \\ y(0) &= t_0 \end{aligned}$$

with $f : I \times D \subset (\mathbb{R} \times \mathbb{R}) \rightarrow \mathbb{R}$, continuous in t and Lipschitz and continuous in $y(t)$. Then there exists an interval $J \subseteq I$ on which there exists a unique solution of this IVP.

For *Partial* Differential Equations, i.e. equations involving the derivative of several variables, the Cauchy-Lipschitz theorem no longer applies in the general case, as it does for ODEs. In this document, we only deal with ODEs.

Ultimately, we define the notion of *length* for a curve.

Definition 1.2.8 (Length of a curve, e.g. in [Pou15])

Let $n \in \mathbb{N}$, I be an interval and $y \in \mathcal{C}^1(I, \mathbb{R}^n)$. The length of y over I is defined by:

$$\text{length}_I(y) = \int_I \|y'(t)\| dt.$$

We use a discrete version (Definition 2.5.1) to have characterisations of **FPTIME** in Section 2.5 and in Chapter 3.

1.2.2 Discrete Ordinary Differential Equations

We redefine various notions of the previous subsection in the discrete settings. We mainly use those concepts in Chapter 3 and Chapter 5.

Basic definitions and properties

We first define the (right) *discrete* derivation:

Definition 1.2.9 (Discrete Derivative, e.g. in [Gel71])

The discrete derivative of $\mathbf{f}(x)$, where $\text{Dom}(\mathbf{f}) \subseteq \mathbb{N}$, is defined as $\Delta \mathbf{f}(x) = \mathbf{f}(x+1) - \mathbf{f}(x)$.

We also write $\mathbf{f}'(x)$ for $\Delta \mathbf{f}(x)$ when the context is not ambiguous with respect to their classical continuous counterparts. This type of derivative is also called *finite difference* in the literature. For $a, b \in \mathbb{N}$ we denote by $\llbracket a, b \rrbracket$ the set $\{a, a+1, \dots, b-1, b\}$.

NB 1.2.10

We can consider $\mathbb{N} \subset \mathbb{Z} \subset \mathbb{R}$ but, as functions may have different types of outputs, the composition is an issue. We consider that composition might not be defined in some cases: it is a partial operator. For example, given $f : \mathbb{N} \rightarrow \mathbb{R}$ and $g : \mathbb{R} \rightarrow \mathbb{R}$, the composition of g and f is defined as expected, but f cannot be composed with a function such as $h : \mathbb{N} \rightarrow \mathbb{N}$.

Several results for continuous derivatives generalise to the settings of discrete differences. It includes linearity of derivation $(a \cdot f(x) + b \cdot g(x))' = a \cdot f'(x) + b \cdot g'(x)$, formulas for products and division such as $(f(x) \cdot g(x))' = f'(x) \cdot g(x+1) + f(x) \cdot g'(x) = f(x+1)g'(x) + f'(x)g(x)$. However, there is no simple equivalent of the chain rule.

A fundamental concept is the following:

Definition 1.2.11 (Discrete Integral, e.g. in [Gel71])

Given $a, b \in \mathbb{N}$ such that $a \leq b$ and some function $\mathbf{f}: \llbracket a, b \rrbracket \rightarrow \mathbb{N}$, we write

$$\int_a^b \mathbf{f}(x) \delta x$$

as a synonym for $\int_a^b \mathbf{f}(x) \delta x = \sum_{x=a}^{x=b-1} \mathbf{f}(x)$. We fix the convention that it takes value 0 when $a = b$ and $\int_a^b \mathbf{f}(x) \delta x = -\int_b^a \mathbf{f}(x) \delta x$ when $a > b$.

The telescope formula yields the so-called Fundamental Theorem of Finite Calculus:

Theorem 1.2.12 (Fundamental Theorem of Finite Calculus, e.g. in [Gel71])

Let \mathbf{F} be some function on $\llbracket a, b \rrbracket$. Then,

$$\int_a^b \mathbf{F}'(x) \delta x = \mathbf{F}(b) - \mathbf{F}(a).$$

A classical concept in discrete calculus is the one of falling power defined as

$$x^{\underline{m}} = x \cdot (x-1) \cdot (x-2) \cdots (x-(m-1)).$$

This notion is motivated by the fact that it satisfies a derivative formula $(x^{\underline{m}})' = m \cdot x^{\underline{m-1}}$ similar to the classical one for powers in the continuous settings. Similarly, we introduce the concept of falling exponential.

Definition 1.2.13 (Falling exponential [BD19])

Given some function $\mathbf{U}(x)$, the expression \mathbf{U} to the falling exponential x , denoted by $\bar{2}^{\mathbf{U}(x)}$, stands for

$$\begin{aligned} \bar{2}^{\mathbf{U}(x)} &= (1 + \mathbf{U}'(x-1)) \cdots (1 + \mathbf{U}'(1)) \cdot (1 + \mathbf{U}'(0)) \\ &= \prod_{t=0}^{t=x-1} (1 + \mathbf{U}'(t)), \end{aligned}$$

with the convention that $\prod_0^0 = \prod_0^{-1} = \mathbf{id}$, where \mathbf{id} is the identity (sometimes denoted 1 hereafter).

This is motivated by the remarks that $2^x = \bar{2}^x$, and that the discrete derivative of a falling exponential is given, for all $x \in \mathbb{N}$, by

$$(\bar{2}^{\mathbf{U}(x)})' = \mathbf{U}'(x) \cdot \bar{2}^{\mathbf{U}(x)}$$

Lemma 1.2.14 (Derivation of an integral with parameters [BD19])

Consider

$$\mathbf{F}(x) = \int_{a(x)}^{b(x)} \mathbf{f}(x, t) \delta t.$$

Then

$$\begin{aligned} \mathbf{F}'(x) = \int_{a(x)}^{b(x)} \frac{\delta \mathbf{f}}{\delta x}(x, t) \delta t &+ \int_0^{-a'(x)} \mathbf{f}(x+1, a(x+1)+t) \delta t \\ &+ \int_0^{b'(x)} \mathbf{f}(x+1, b(x)+t) \delta t. \end{aligned}$$

In particular, when $a(x) = a$, $b(x) = b$ are constant functions,

$$\mathbf{F}'(x) = \int_a^b \frac{\delta \mathbf{f}}{\delta x}(x, t) \delta t,$$

and when $a(x) = a$, $b(x) = x$,

$$\mathbf{F}'(x) = \int_a^x \frac{\delta \mathbf{f}}{\delta x}(x, t) \delta t + \mathbf{f}(x+1, x).$$

Proof. By definition of the discrete derivative,

$$\begin{aligned} \mathbf{F}'(x) &= \mathbf{F}(x+1) - \mathbf{F}(x) \\ &= \sum_{t=a(x+1)}^{b(x+1)-1} \mathbf{f}(x+1, t) - \sum_{t=a(x)}^{b(x)-1} \mathbf{f}(x, t) \end{aligned}$$

Thus,

$$\begin{aligned} &= \sum_{t=a(x)}^{b(x)-1} (\mathbf{f}(x+1, t) - \mathbf{f}(x, t)) + \sum_{t=a(x+1)}^{t=a(x)-1} \mathbf{f}(x+1, t) + \sum_{t=b(x)}^{b(x+1)-1} \mathbf{f}(x+1, t) \\ &= \sum_{t=a(x)}^{b(x)-1} \frac{\delta \mathbf{f}}{\delta x}(x, t) + \sum_{t=a(x+1)}^{t=a(x)-1} \mathbf{f}(x+1, t) + \sum_{t=b(x)}^{b(x+1)-1} \mathbf{f}(x+1, t) \\ &= \sum_{t=a(x)}^{b(x)-1} \frac{\delta \mathbf{f}}{\delta x}(x, t) + \sum_{t=0}^{t=-a(x+1)+a(x)-1} \mathbf{f}(x+1, a(x+1)+t) \\ &+ \sum_{t=0}^{b(x+1)-b(x)-1} \mathbf{f}(x+1, b(x)+t). \end{aligned}$$



1.3 Computability, Complexity, Computable Analysis

1.3.1 Some notions of *classical* computability and complexity theory

We start by introducing some notions of classical computability and complexity theory, firstly because they are useful mainly in Chapter 4. Also, we use the same objects in computable analysis, so it is interesting to see how we can link both theories.

Computability

The usual model of computation we use to study discrete models is Turing machines. We assume the readers to be somewhat familiar with these concepts, a more comprehensive study can be found in [Sip97]. We give some basic definitions to fix the notations:

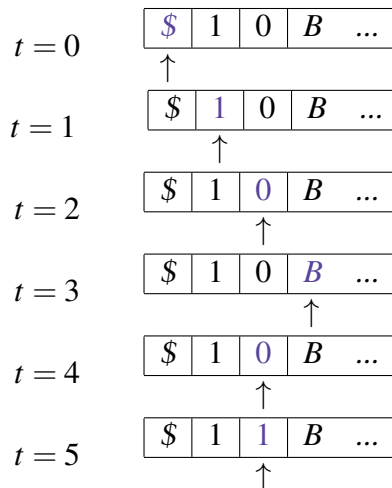
Definition 1.3.1 (Turing machine, e.g. in [Sip97])

A Turing machine is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, Q_{accept}, Q_{reject})$, where Q, Σ, Γ are all finite sets and:

1. Q is a set of states;
2. Σ is the input alphabet;
3. Γ is the tape and output alphabet, where $\Sigma \subseteq \Gamma$;
4. $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{\leftarrow, \rightarrow\}$ is the transition function;
5. $q_{init} \in Q$ is the starting state;
6. $Q_{accept} \subseteq Q$ is the set of accepting states;
7. $Q_{reject} \subseteq Q$ is the set of rejecting states, with $Q_{accept} \cap Q_{reject} = \emptyset$.

We give an example of the execution, or *run*, of a Turing machine computing the binary addition of some integer:

Example 1.3.2 (Binary Addition: doing +1)



$t = 6$

\$	1	1	B	...
----	---	---	---	-----

We represent what happens on the tape of a Turing machine performing a binary addition +1 on the integer 10 (written in big-endian binary) given as the input. The up-arrow represents the position of the head of the machine, \$ marks the beginning of the tape, $\Sigma = \{0, 1\}$, $\Gamma = \{\$, 0, 1, B\}$. The states of such machine are $Q = \{q_{init}, q_1, q_2, q_3, q_4^0\}$ and the transition function δ is described in the tabular below.

More formally, it can be defined by:

	\$	0	1	B
q_{init}	$q_{init}, \$, \rightarrow$	$q_{init}, 0, \rightarrow$	$q_{init}, 1, \rightarrow$	q_1, B, \leftarrow
q_1		$accept, 1$	$q_2, 0, \leftarrow$	
q_2	$q_3, \$, \rightarrow$	$accept, 1$	$q_2, 1, \leftarrow$	
q_3		$q_4^0, 1, \rightarrow$		
q_4^0		$q_4^0, 0, \rightarrow$		$accept, 0$

Where the cells left blank are rejecting states.

It is possible for a Turing machine M to take another machine M' as an input. In this case, we give some encoding (*the code*) of M' to M , denoted by $\langle M' \rangle$. More generally, we denote by $\langle M', w \rangle$ for both the code of M' and some input w .

We can also define multi-tapes Turing machines: each tape has its own head. One transition in such machines corresponds to a simultaneous movement of all heads. Thus the transition function δ for n tapes is $\delta : Q \times \Gamma^n \rightarrow Q \times \Gamma^n \times \{\leftarrow, \rightarrow\}^n$.

Regarding computational power, Turing machines can simulate more general Turing machines where the working alphabets are larger. Similarly, Turing machines whose transition function forbids the head to stay on a given cell (i.e. it must necessarily move right or move left at every step) can be simulated by the more general usual case where staying on a given cell is allowed: if the head stays on the same cell, this can be emulated by the head going back and forth, not altering the content of the tape on the way back. Furthermore, using standard techniques, we can assume, without loss of generality, that at any moment, the blank symbol B appears eventually on the right and eventually on the left, but never between non-blank symbols.

We presented *deterministic* Turing machines: one transition on a configuration gives at most one possible configuration. One variant is the *non-deterministic* Turing machine, where we allow the transition function to send a configuration to a set of configurations:

Definition 1.3.3 (Non-determinism, e.g. in [Sip97])

A Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, Q_{accept}, Q_{reject})$ is non-deterministic iff δ is a relation such that $\delta \subseteq (Q \setminus \{Q_{accept}, Q_{reject}\} \times \Gamma) \times (Q \times \Gamma \times \{\leftarrow, \rightarrow\})$.

In other words, non-deterministic machines are allowed to “make choices” during the computation. However, it does not affect the computational power of Turing machines, regarding computability:

NB 1.3.4 (e.g. in [Sip97])

Every language recognised by a non-deterministic machine can be recognised by a deterministic machine.

There exist other variants of Turing machines. We give here the definition of oracles, that we use in Chapter 5:

Definition 1.3.5 (Oracle Turing machine, e.g. in [Sip97])

An oracle for a language B is capable of reporting whether any string w is a member B .

An oracle Turing machine is a modified Turing machine with an additional capability of querying an oracle. We write M^B to describe an oracle Turing machine M having an oracle for the language B .

We are not interested in how the oracle actually “knows” the solution of the queries. Having an oracle for a machine, as the name suggests, can be considered as some “magical” computability ability.

NB 1.3.6

In this document, we will mainly use the variant of TMs described in Subsection 2.2.

Now that we have seen the main models of computation we use, let us focus on their computability properties. We will call *problem* sets of the form $\{I \in S \mid Q\}$ with $S \subseteq \Sigma^*$ some set (for example the set of codes of Turing machines) and $Q \subseteq S$ is a question to be solved algorithmically. We write problems this way:

Problem:

Input: $I \in S$

Output: Q

Definition 1.3.7 (Decision Problem, e.g. in [Sip97])

A decision problem is a subset $\mathfrak{P} \subseteq \Sigma^*$.

Intuitively, a problem is a decision problem if its possible solutions are yes or no: it can be interpreted as a yes/no question on some set of inputs.

For example, determining whether two integers are equal is a decision problem.

Definition 1.3.8 (Computable enumerability (c.e.), e.g. in [Sip97])

A decision problem \mathfrak{P} is computably enumerable (or semi-decidable) iff there exists a Turing machine accepting all the correct inputs of \mathfrak{P} .

If, on some input, the machine does not reach an accepting state, we cannot say more about its behaviour: it could halt in a rejecting state or never halt.

Example 1.3.9 ([Sip97])**Halting Problem:**

Input: A Turing Machine M and x

Output: Does M stops on x ?

The Halting problem is computably enumerable: if x is accepted by M , then the machine stops at some point. If not, then M could reach a rejecting state or just keep computing indefinitely.

We define a *class* as a set of problems related by some computability property. Informally, a class \mathcal{C} is a set of decision problems that are in the same equivalence class:

they are grouped according to their computability or complexity properties. For example the set of all the c.e. decision problems is a class.

Definition 1.3.10

For every class \mathcal{C} , $\text{co-}\mathcal{C}$ is the class of problems such that their respective complements are in \mathcal{C} .

Definition 1.3.11 (Co-computable enumerability (co-c.e.), e.g. in [Sip97])

A problem \mathfrak{P} is co-computably enumerable iff there exists a Turing machine rejecting all the incorrect inputs of \mathfrak{P} .

It means that, on some input, if the machine does not reach a rejecting state, we cannot say anything about its behaviour: it could halt in an accepting state or never halt.

Example 1.3.12

Non-Halting Problem:

Input: A Turing Machine M and x

Output: Does M loop forever on x ?

Corollary 1.3.13

A problem is co-computably enumerable if its complement is computably enumerable.

And there exist problems that are neither c.e. nor co-c.e.. For example, let ϕ be an oracle for the halting problem:

Example 1.3.14

Oracle Halting Problem:

Input: A Turing machine M^ϕ , an input x

Output: Does $M^\phi(x)$ halt?

NB 1.3.15

Coming back to oracle machines (Definition 1.3.5), note that the language recognised by the oracle is not necessarily decidable, as in this example.

NB 1.3.16

Typically, we study this kind of problem in the framework of the arithmetical hierarchy: see [Kle43] and [Mos47].

Definition 1.3.17 (Decidability)

A problem is decidable iff it is both computably enumerable and co-computably enumerable.

Definition 1.3.18 (Computable Functions, e.g. in [Sip97])

A function f is a computable if there exists a TM such that, on some input x , outputs $f(x)$ if $x \in \text{Dom}(f)$ and rejects otherwise.

They are somewhat more general than decision problems, as the output is not only *yes* or *no*.

Example 1.3.19 (A Decidable Problem)**Halting after n steps Problem:**

Input: A Turing Machine M , $n \in \mathbb{N}$ and x

Output: Does M stops on x in less than n steps?

As an example of non-decidability, let us prove the following:

Proposition 1.3.20 ([Tur37])

The Halting problem is not decidable.

Proof from [Str65]. Assume, by contradiction, that the Halting problem is decidable. Let M_H be a Turing machine taking as an input the encoding of some machine M' and x' , and accepts if M' halts on x' and rejects otherwise. Let us define the machine M taking M_H as an input and:

- If $M_H(< M, x >)$ accepts then M loops forever;
- If $M_H(< M, x >)$ rejects then M accepts.

Thus, if $M_H(< M, x >)$ accepts, M never halts: contradiction. If $M_H(< M, x >)$ rejects, M halts: contradiction.

Hence, the Halting problem is undecidable. \square

There exist many models of computations, for various purposes. Many of them are equivalent, as long as they deal with *effectively computable* functions:

Definition 1.3.21 (Effectively Computable [Chu36, Gan80])

A function in $\mathbb{N}^{\mathbb{N}}$ is effectively computable if there exists a procedure, e.g. an algorithm, computing it (in some pre-defined class) and terminating.

The Church-Turing thesis ([Chu36, Tur37, Kle36]) states that a function over the integers is effectively computable iff it can be computed by a Turing machine. It stipulates the equivalence, in terms of computation power, between Turing machines and other effective models of computations such as λ -calculus ([Chu36]), or more intuitive sufficiently powerful models. In the rest of the thesis, we will write “computable” instead of “effectively computable”.

Let us now recall the definitions of reduction and complete problems.

Definition 1.3.22 (Reduction, e.g. in [Sip97])

A decision problem A is reducible to a problem B , written $A \leq B$, if there exists a computable function $f : \Sigma^* \rightarrow \Sigma^*$, where for every w ,

$$w \in A \Leftrightarrow f(w) \in B.$$

The function f is called the reduction from A to B .

Corollary 1.3.23

Let A, B be two decision problems such that $A \leq B$. Then,

- If A is undecidable, then B is undecidable.
- If B is decidable, then A is decidable.

Example 1.3.24 ([Sip97])

We consider the following problem:

E_{LBA} :

Input: A Turing machine M

Output: M is a Linear Bounded Automaton with empty acceptance language ($L(M) = \emptyset$)?

where a Linear Bounded Automaton (LBA) is a Turing machine where the head is not allowed to move off the portion of the tape containing the input. Let us proof this problem is undecidable, by reducing from the **Halting Problem**. Assume, by contradiction, there exists a TM R deciding E_{LBA} . Let M be a Turing machine and x an input. We assume M terminates when given the input x . The reduction is as follows:

- We construct the LBA B from M and x . Its input is the sequence of configurations of M on x . This computation history is represented by a string, each configuration separated by a fresh symbol $\#$:

$$\begin{array}{ccccccc} \# & & \# & & \# \dots \# & & \# \\ & \underbrace{\hspace{1cm}} & & \underbrace{\hspace{1cm}} & & \underbrace{\hspace{1cm}} & \\ & C_1 & & C_2 & & C_l & \end{array}$$

B accepts if M accepts on some input by checking 3 conditions:

- C_1 is the initial condition of M on x ;
 - Each C_{i+1} is a legal successor of C_i ;
 - C_l is a terminating configuration of M .
- Run R on $\langle B \rangle$.
 - If R rejects, then accept; if R accepts, reject.

Thus, if R accepts $\langle B \rangle$, then $L(B) = \emptyset$, so M has no terminating computation history on x , so M does not halt x and we reject. If R rejects $\langle B \rangle$ then $L(B) \neq \emptyset$. B accepts only a finite computation history of M on x . Thus, M must halt on x and we halt on $\langle M, x \rangle$: contradiction.

Consequently, E_{LBA} is undecidable.

NB 1.3.25

For LBAs, and contrary to Turing machines, knowing whether a word is in a language is decidable. E_{LBA} is one of the problems where undecidability remains.

Theorem 1.3.26 (Completeness)

A problem A is complete for some class \mathcal{C} of decision problems, or \mathcal{C} -complete, iff $A \in \mathcal{C}$ and for every problem B in \mathcal{C} , $B \leq A$.

Complexity

Now that we have a criterion to say whether we can solve some problem, it makes sense to investigate how much resources we need to actually do so. We need a way to properly measure time and space. Using the framework of Turing machines, time corresponds to the number of transitions, or steps, needed to solve the problem, and space to the number of cells required. In this context, time and space are given with respect to the size of the input. Here, all inputs have finite size.

Definition 1.3.27 (Time classes, e.g. in [Sip97])

Let $f : \mathbb{N} \rightarrow \mathbb{R}_+$,

- $\text{TIME}(f)$ (respectively $\text{NTIME}(f)$) defines the class of languages decidable in time bounded by $O(f(n))$ by a deterministic (resp. non-deterministic) Turing machine;
- $\text{FTIME}(f)$ (resp. $\text{NFTIME}(f)$) is the class of functions computable in time bounded by $O(f(n))$ by a deterministic (resp. non-deterministic) Turing machine.

Definition 1.3.28 (Space classes, e.g. in [Sip97])

Let $f : \mathbb{N} \rightarrow \mathbb{R}_+$,

- $\text{SPACE}(f)$ (resp. $\text{NSPACE}(f)$) is the class of languages decidable in space bounded by $O(f(n))$ by a deterministic (resp. non-deterministic) Turing machine;
- $\text{FSPACE}(f)$ (resp. $\text{NFSPACE}(f)$) is the class of functions computable in space bounded by $O(f(n))$ by a deterministic (resp. non-deterministic) Turing machine.

In the literature, there are two possible definitions for **FSPACE**, according to whether functions with non-polynomial size values are allowed. When we talk about **FSPACE**,

we will always assume the outputs remain of polynomial size. Otherwise, the class is not closed by composition: the issue is about the usual convention of not counting the input and output as part of the total space used. Given f computable in polynomial space and g in logarithmic space, $f \circ g$ (and $g \circ f$) is computable in polynomial space. But, if exponential size output is allowed, this is not true: the problem comes from the fact that, if we assumed only f and g to be computable in polynomial space, g might give an output of exponential size.

Definition 1.3.29 (Some usual classes, e.g. in [Sip97])

- **L** is the class of languages decidable by a deterministic Turing machine in logarithmic space.
- **P** = **PTIME** = $\bigcup_{k \in \mathbb{N}} \text{TIME}(n^k)$;
- **NP** = **NPTIME** = $\bigcup_{k \in \mathbb{N}} \text{NTIME}(n^k)$;
- **PSPACE** = $\bigcup_{k \in \mathbb{N}} \text{SPACE}(n^k)$

The notions of stability (or closure) and reduction are still relevant for those classes. However, they are not stable (or close) under the previous definition of reduction: it is too general. Thus, when dealing with complexity classes, we restrain the reduction to be computable in logarithmic space or polynomial time.

Theorem 1.3.30 (Savitch theorem, e.g. in [Sip97])

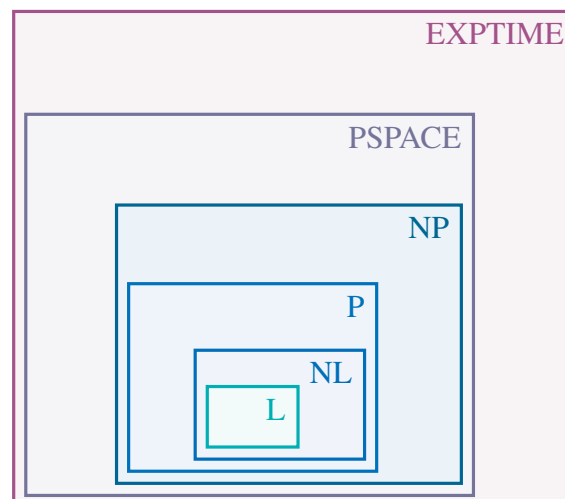
For any function $f : \mathbb{N} \rightarrow \mathbb{N}$ with $f(n) \geq \log n$, we have:

$$\text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n)).$$

Corollary 1.3.31

$$\text{PSPACE} = \text{NPSpace}$$

We have the following hierarchy of complexity classes:



The previous hierarchy can be divided more finely, by introducing classes such as co-NP or Π_n^P and Σ_n^P : see [Sip97] and [AB09] for more details.

Usual Complete Problems

Circuit Value:

Input: A boolean circuit C , an input x

Output: $C(x) = 1$?

Proposition 1.3.32 ([Sip97])

Circuit Value is P -complete.

SAT:

Input: A boolean formula ϕ

Output: Is ϕ satisfiable?

Proposition 1.3.33 ([Sip97], [Coo71], [Lev73] and [Kar72])

SAT is NP -complete.

QBF:

Input: A quantified boolean formula ϕ

Output: Is ϕ satisfiable?

Proposition 1.3.34 ([Sip97])

QBF is PSPACE -complete.

Let us focus on the following decision problem:

$\text{PATH}(G, u, v)$:

Input: A directed graph $G = (V, \rightarrow)$, some vertices $u, v \in V$

Output: Is there a path between u and v in G (denoted by $u \xrightarrow{*} v$)?

Lemma 1.3.35 (Reachability for graphs, e.g. in [Sip97])

$\text{PATH}(G, u, v) \in \text{NL}$.

The following is known:

Lemma 1.3.36 (Immerman–Szelepcsényi's theorem [Imm88, Sze88])

$\text{NL} = \text{coNL}$.

We also define the complement of $\text{PATH}(G, u, v)$:

NOPATH(G, u, v):

Input: A directed graph $G = (V, \rightarrow)$, some vertices $u, v \in V$

Output: Is there no path between u and v in G ?

Corollary 1.3.37

Then $\text{NOPATH}(G, u, v) \in \mathbf{NL}$.

Corollary 1.3.38

$\text{PATH}(G, u, v) \in \text{SPACE}(\log^2(n))$ and $\text{NOPATH}(G, u, v) \in \text{SPACE}(\log^2(n))$.

NB 1.3.39

*Notice that detecting whether there is no path between u and v is equivalent to determining whether all paths starting from u remain disjoint from v . More precisely, the path either loops or ends without reaching v . The above statement is established using a more subtle method than a simple depth-of-width search of the graph. One uses the trick of the proof of Savitch theorem, i.e. a recursive procedure (expressing reachability in less than 2^t steps, called **CANYIELD**($\mathbf{C}_1, \mathbf{C}_2, \mathbf{t}$) in [Sip97]) guaranteeing the claimed space complexity. The formal proof of the Savitch theorem is detailed in [Sip97].*

1.3.2 On computable analysis: Computability

For discrete-time models of computations over the reals, the most famous models are computable analysis, based on the Turing machine model (see[Tur37] and [Wei00]) and algebraic models such as the Blum Shub Smale (BSS) model of computation [BSS89, BCSS98]. Both models were tailored for different applications and it is well-known we cannot unify existing models with the equivalent of a Church-Turing thesis. For example, computable functions in a computable analysis model need to be continuous, while the BSS model intends to consider functions and problems over the polynomials that are not. It is also explained by the fact that some models have not been introduced with the idea of being associated with actual physical machines.

Among models of computation over the reals, we can highlight continuous-time models. This includes models of old, first-ever-built computers, such as the Differential Analysers [Ulm20]. A famous mathematical model of such machines is the General Purpose Analog Computer model of Claude Shannon [Sha41]. It covers many historical machines and today's analogue devices [Ulm13, Ver22] too. It also includes various recent approaches and models from deep learning such as Neural ODEs [CRBD18, Kid22] with many variants.

In the context of continuous-time, the situation is clearer than with discrete-time models, as there is a unifying way to describe these models provided by Ordinary Differential Equations (ODEs). Each model corresponds to a particular class of ODEs. For example, the GPAC corresponds to polynomial ODEs [GC03], and Neural ODEs are made by selecting the best solution among a parameterised class of ODEs: see, e.g. [Kid22].

When we say that a function $f : S_1 \times \dots \times S_d \rightarrow \mathbb{R}^{d'}$ is (respectively: polynomial-time) computable it will always be in the sense of computable analysis. We recall here the basic

concepts and definitions, mostly following the book [Ko91], whose subject is complexity theory in computable analysis. Alternative presentations include [BHW08, Wei00].

NB 1.3.40

The monograph [Ko91] formulates explicitly most of the statements only for functions over the reals. As we talk about functions in $\mathbb{R}^{\mathbb{N}}$ in Chapter 3, we need to mix complexity issues dealing with integer and real arguments. Hence, we extend some of the statements to these more general settings (in the spirit of [Ko91], [Kaw09], [KORZ14], [KMRZ15] and [KST18]).

We present here a formalisation equivalent to [Tak01] but with our own words. This will be used to fix the framework and the notations.

To talk about computable analysis, we consider a variant of Turing machines in this context:

Definition 1.3.41 (Type-2 Turing machine [Wei00])

A Type-2 Turing machine is a Turing machine with k input and working tapes together with a type specification (Y_1, \dots, Y_k, Y_0) with $Y_i \in \{\Sigma^, \Sigma^\omega\}$, giving the type (finite or infinite) for each input tape and output tape.*

This machine contains some read-only input tapes, containing the inputs, which can be either finite or infinite words, a read-write working tape and a write-only output tape. It looks like a classical Turing machine, the only difference is that we allow it to output an infinite word written forever on its write-only infinite output tape(s).

Definition 1.3.42 (Computable String Function $f : Y_0 \rightarrow Y_1 \times \dots \times Y_k$ [Wei00])

The initial tape configuration for input $(y_1, \dots, y_k) \in Y_1 \times \dots \times Y_k$ is as follows: for each input tape i the (finite or infinite) sequence $y_i \in Y_i$ is placed on the tape immediately to the right of the head, all other tape cells contain the symbol B. For all $y_0 \in Y_0, y_1 \in Y_1, \dots, y_k \in Y_k$ we define:

1. **Case $Y_0 = \Sigma^*$:** $f(y_1, \dots, y_k) := y_0 \in \Sigma^*$ iff the associated Type-2 Turing machine M halts on input (y_1, \dots, y_k) with y_0 written on the output tape.
2. **Case $Y_0 = \Sigma^\omega$:** $f(y_1, \dots, y_k) := y_0 \in \Sigma^\omega$ iff M computes forever on input (y_1, \dots, y_k) and writes y_0 on the output tape.

We can now define a proper notion of computable enumerability in that context:

Definition 1.3.43 (Computably enumerable sets [Wei00])

A closed set $A \in \mathbb{R}^n$ is computably enumerable if there exists a Turing machine enumerating all the open rational cubes intersecting A .

NB 1.3.44

We can also consider oracle Turing machines to represent functions in $\mathbb{R}^{\mathbb{R}}$, as we will see in Definition 1.3.48.

We approximate real numbers with dyadics. A dyadic number d is a rational number with a finite binary expansion. That is to say $d = m/2^n$ for some integers $m \in \mathbb{Z}, n \in \mathbb{N}, n \geq 0$. Let \mathbb{D} be the set of all dyadic rational numbers.

We denote by \mathbb{D}_n the set of all dyadic rationals d with a representation s of precision $\text{prec}(s) = n$: that is, $\mathbb{D}_n = \{m \cdot 2^{-n} \mid m \in \mathbb{Z}\}$.

NB 1.3.45

\mathbb{D} is dense in \mathbb{R} .

Thus, we can approximate any real number x with a sequence of dyadics converging to x . This leads us to the definition of computable real numbers:

Definition 1.3.46 ([Ko91])

For each real number x , a function $\phi : \mathbb{N} \rightarrow \mathbb{D}$ is said to *binary converge* to x if for all $n \in \mathbb{N}$, $\text{prec}(\phi(n)) = n$ and $|\phi(n) - x| \leq 2^{-n}$. Let CF_x denote the set of all functions binary converging to x .

CF is a set of Cauchy sequences.

Definition 1.3.47 (Computable real number [Ko91])

A real number x is *computable* iff CF_x contains a computable function: there exists a TM such that, on input $n \in \mathbb{N}$, computes $\phi(n)$, defined in Definition 1.3.46.

There exist many more non-computable than computable real numbers. One intuition is to notice that, according to the previous definition, there is an injection between computable reals and the set of Turing machines: the set of computable real numbers is countable, whereas \mathbb{R} is not. However, the set of computable reals is dense in \mathbb{R} , so any non-computable real can be approximated by a sequence of computable reals.

In Chapter 3 and Chapter 5, we are interested in having characterisations of functions computable in **FPTIME** and **FPSpace**. So, we have to formally define what is a computable function in the framework of computable analysis:

Definition 1.3.48 ([Ko91])

A real function $f : \mathbb{R} \rightarrow \mathbb{R}$ is *computable* if there is a function-oracle TM M such that for each $x \in \mathbb{R}$ and each $\phi \in CF_x$, the function ψ computed by M with oracle ϕ (i.e. $\psi(n) = M^\phi(n)$) is in $CF_{f(x)}$. We say the function f is *computable on interval $[a, b]$* if the above condition holds for all $x \in [a, b]$.

The intuition is that an oracle Turing machine M computes a function f over the reals, if we have, for any real x in the domain of f , for any $\phi \in CF_x$, M with oracle ϕ , the following: given the output precision 2^{-n} , given in the form of integer n , M eventually outputs $d \in \mathbb{D}$ with $\|d - f(x)\| \leq 2^{-n}$.

During this computation, M asks necessarily finitely many questions to its oracle ϕ . From the considered oracles, any such question can be seen as follows: for some queried precision 2^{-m} , M asks to oracle ϕ to provide some $\phi(m)$, such that $\|\phi(m) - x\| \leq 2^{-m}$.

We notice that, in computable analysis, we only deal with continuous functions:

Theorem 1.3.49 ([Wei00])

Every computable real function is continuous.

NB 1.3.50

From the model of computable analysis, given the names (or a representation) of the function \mathbf{f} and $\mathbf{x}, \mathbf{y} \in \mathbb{Q}$, it is impossible in general to tell effectively if $\mathbf{f}(\mathbf{x}) = \mathbf{y}$.

For closed sets, the notion of computability can be interpreted as the possibility of being plotted with an arbitrarily chosen precision: $\mathbf{z}/2^n$ corresponds to a pixel at precision 2^{-n} , 1 is black (the pixel is plotted black), 0 is white (the pixel is plotted white).

Theorem 1.3.51 ([BHW08])

For a closed set $A \subseteq \mathbb{R}^k$, A is computable iff it can be plotted: there exists a computable function $f : \mathbb{N} \times \mathbb{Z}^k \rightarrow \{0, 1\}$ and such that for all $n \in \mathbb{N}$ and $\mathbf{z} \in \mathbb{Z}^k$

$$f(n, \mathbf{z}) = \begin{cases} 1 & \text{if } B\left(\frac{\mathbf{z}}{2^n}, 2^{-n}\right) \cap A \neq \emptyset, \\ 0 & \text{if } B\left(\frac{\mathbf{z}}{2^n}, 2 \cdot 2^{-n}\right) \cap A = \emptyset, \\ 0 \text{ or } 1 & \text{otherwise.} \end{cases}$$

1.3.3 On computable analysis: Complexity

Assume that M is an oracle machine which computes f on domain $\text{Dom}(f)$.

For simplicity, we may assume in our discussion that $\text{Dom}(f)$ is some closed interval or \mathbb{R} . For any oracle $\phi \in CF_x$, with $x \in \text{Dom}(f)$, let $T_M(\phi, n)$ be the number of steps for M to halt on input n with oracle ϕ , and $T'_M(x, n) = \max \{T_M(\phi, n) \mid \phi \in CF_x\}$.

We define the time complexity of f as follows:

Definition 1.3.52 (From [Ko91])

Let $\text{Dom}(f)$ be a bounded closed interval $[a, b]$. Let $f : \text{Dom}(f) \rightarrow \mathbb{R}$ be a computable function. Then, we say that the time complexity of f on $\text{Dom}(f)$ is bounded by a function $t : \text{Dom}(f) \times \mathbb{N} \rightarrow \mathbb{N}$ if there exists an oracle TM M which computes f such that for all $x \in \text{Dom}(f)$ and all $n > 0$, $T'_M(x, n) \leq t(x, n)$.

In other words, as mentioned in [Ko91], the idea is to measure the time complexity of a real function based on the input and the output precision 2^{-n} . However, it is usually more convenient to simplify the complexity measure and make it depend only on the output precision. They define the uniform time complexity of f on G with a bound function $t' : \mathbb{N} \rightarrow \mathbb{N}$.

However, if we do so, it is important to realise that if we took $\text{Dom}(f) = \mathbb{R}$ in previous definition, for unbounded functions f , [Ko91] observes that “the uniform time complexity does not exist, because the number of moves required to write down the integral part of $f(x)$ grows as x approaches $+\infty$ or $-\infty$ ”.

Therefore, the approach of [Ko91] is to do as follows (the bounds -2^X and 2^X , $X \in \mathbb{N}$, are somewhat arbitrary, but are chosen here because the binary expansion of any $x \in (-2^n, 2^n)$ has n bits in the integral part):

Definition 1.3.53 (Adapted from [Ko91])

For functions $f(x)$ whose domain is \mathbb{R} , we say that the (non-uniform) time complexity of f is bounded by a function $t' : \mathbb{N}^2 \rightarrow \mathbb{N}$ if the time complexity of f on $[-2^X, 2^X]$ is bounded by a function $t : \text{Dom}(f) \times \mathbb{N} \rightarrow \mathbb{N}$ such that $t(x, n) \leq t'(X, n)$ for all $x \in [-2^X, 2^X]$.

We extend the approach to more general functions. When $\mathbf{x} = (x_1, \dots, x_p)$ and $\mathbf{X} = (X_1, \dots, X_p)$, we write $\mathbf{x} \in [-2^{\mathbf{X}}, 2^{\mathbf{X}}]$ as short for $x_1 \in [-2^{X_1}, 2^{X_1}], \dots, x_p \in [-2^{X_p}, 2^{X_p}]$.

For $n \in \mathbb{N}$, we denote by $\ell(n)$ the number of bits of the binary representation of n .

Definition 1.3.54 (Complexity for real functions: general case, from [Ko91])

Consider a computable function $f(x_1, \dots, x_p, n_1, \dots, n_q)$ whose domain is $\mathbb{R}^p \times \mathbb{N}^q$. We say that the (non-uniform) time complexity of f is bounded by a function $t' : \mathbb{N}^{p+q+1} \rightarrow \mathbb{N}$ if the time complexity of $f(\cdot, \dots, \cdot, n_1, \dots, n_q)$ on $[-2^{X_1}, 2^{X_1}] \times \dots \times [-2^{X_p}, 2^{X_p}]$ is bounded by a function:

$$t(\cdot, \dots, \cdot, \ell(n_1), \dots, \ell(n_q), \cdot) : \mathbb{N}^p \times \mathbb{N} \rightarrow \mathbb{N}$$

such that $t(\mathbf{x}, \ell(n_1), \dots, \ell(n_q), n) \leq t'(\mathbf{X}, \ell(n_1), \dots, \ell(n_q), n)$ whenever $\mathbf{x} \in [-2^{\mathbf{X}}, 2^{\mathbf{X}}]$. We say that f is polynomial time computable if t' can be chosen as a polynomial. We say that a vectorial function is polynomial time computable iff all its components are.

NB 1.3.55

There is some major subtlety. When considering $f : \mathbb{N} \rightarrow \mathbb{Q}$, as $\mathbb{Q} \subset \mathbb{R}$, stating f is computable might mean two things:

- in the classical sense, given integer y , i.e. one can compute p_y and q_y some integers such that $f(y) = p_y/q_y$;
- or in the sense of computable analysis, i.e. given some precision n , given arbitrary y and $n \in \mathbb{N}$ we can provide some rational q_n such that $|q_n - f(y)| \leq 2^{-n}$.

We always consider the latter.

The motivation behind those previous definitions is similar to the one in [Pou15] and [BGP16b]: we want this measure of complexity to extend the usual complexity for functions over the integers, where complexity of integers is measured with respects of their lengths, and over the reals, where complexity is measured with respect to their approximation.

In particular, in the specific case of a polynomial-time computable function $f : \mathbb{N}^d \rightarrow \mathbb{R}^{d'}$, it means there is some polynomial $t' : \mathbb{N}^{d+1} \rightarrow \mathbb{N}$ such that the time complexity of producing some dyadic approximating $f(\mathbf{m})$ at precision 2^{-n} is bounded by $t'(\ell(m_1), \dots, \ell(m_d), n)$, writing $\mathbf{m} = (m_1, \dots, m_d)$.

In other words, when considering that a function is polynomial time computable, the bounding polynomial is given by the length of all its integer arguments, as it is the usual

convention.

However, we sometimes also need to consider polynomial dependency directly in one of some specific integer argument, say n_i and not on its length $\ell(n_i)$. We say that *the function is polynomial time computable, with respect to the value of n_i* when this holds (keeping possible other integer arguments n_j , $j \neq i$, measured by their length).

A key observation is the stability of polynomial time under composition:

Lemma 1.3.56 ([Ko91])

The class of polynomial time computable functions is stable under composition.

Proof. It is proved in [Ko91]: in short, the idea of the proof for $\text{composition}(f, g)$, is that by the induction hypothesis, there exists M_f and M_g two Turing machines computing in polynomial time $f : R \rightarrow R$ and $g : R \rightarrow R$. In order to compute $\text{composition}(f, g)(x)$ with precision 2^{-n} , we just need to compute $g(x)$ with a precision $2^{-m(n)}$, where $m(n)$ is the polynomial modulus function of f . Then, we compute $f(g(x))$, which, by definition of M_f takes a polynomial time in n . Thus, since polynomial time with an oracle in polynomial time is polynomial time, $\text{composition}(f, g)$ is computable in polynomial time so the class of polynomial time computable functions is closed under composition. \square

There exist various characterisations of functions computable in polynomial time over the reals. For example, the following is proved in [Ko91] for functions from $[a, b] \rightarrow \mathbb{R}$. It could be extended to more general functions.

Theorem 1.3.57 (Alternative characterisation [Ko91])

A function $f : [a, b] \rightarrow \mathbb{R}$ is computable in polynomial time iff there exist polynomial functions m and q and a function $\psi : (\mathbb{D} \cap [a, b]) \times \mathbb{N} \rightarrow \mathbb{D}$ such that

1. *m is a polynomial modulus function for f on $[a, b]$: $|x - y| \leq 2^{-m(n)}$ implies $|f(x) - f(y)| \leq 2^{-n}$ for every $x, y \in [a, b]$ and every integer n ;*
2. *for any $d \in \mathbb{D} \cap [a, b]$ and all $n \in \mathbb{N}$, $|\psi(d, n) - f(d)| \leq 2^{-n}$;*
3. *$\psi(d, n)$ is computable in time $q(\text{prec}(d) + n)$.*

In particular, a well-known observation is the following (see e.g. [Ko91]):

Theorem 1.3.58

Consider \mathbf{f} as in Definition 1.3.54 computable in polynomial time. Then \mathbf{f} has a polynomial modulus function, that is to say there is a polynomial function $m_{\mathbf{f}} : \mathbb{N}^{p+q+1} \rightarrow \mathbb{N}$ such that for all $\mathbf{x}, \mathbf{y} \in [-2^{\mathbf{X}}, 2^{\mathbf{X}}]$ and all $n > 0$,

$$\|\mathbf{x} - \mathbf{y}\| \leq 2^{-m_{\mathbf{f}}(\mathbf{X}, \ell(n_1), \dots, \ell(n_q), n)}$$

implies

$$\|\mathbf{f}(\mathbf{x}, n_1, \dots, n_q) - \mathbf{f}(\mathbf{y}, n_1, \dots, n_q)\| \leq 2^{-n}.$$

1.3.4 Reductions in continuous settings

Having proper computability and complexity classes over the reals depends a lot on the representation of the reals we consider. We can even have complexity results holding for some representation, but not for another.

Definition 1.3.59 (Translation and equivalence [Wei00])

Let $S, S' \subseteq \mathbb{R}$ be sets. For arbitrary functions $\gamma : Y \rightarrow S$, $\gamma' : Y' \rightarrow S'$ with $Y, Y' \in \{\Sigma^*, \Sigma^\omega\}$:

- $f : Y \rightarrow Y'$ translates γ to γ' iff, for all $y \in \text{Dom}(\gamma)$, $\gamma(y) = \gamma'(f(y))$;
- γ is equivalent to γ' iff some computable function translates γ to γ' and some computable function translates γ' to γ .

All equivalent representations of the reals give the same computational properties (see [Wei00]). Here we only use equivalent representations of Definition 1.3.46, with quickly converging names: a name of $x \in \mathbb{R}^d$, with I_n an interval of radius $< 2^{-n}$. When s is a word, here we denote by $|s|$ its length.

Once again, we do not work directly on the real numbers themselves, but on some representation of them. In this context, we will use the set **Reg** of regular functions to represent them:

Definition 1.3.60 (Regular Functions [KC12])

A function $f : \Sigma^* \rightarrow \Sigma^*$ is regular iff, for all $u, v \in \text{Dom}(f)$ such that $|u| \leq |v|$, then $|f(u)| \leq |f(v)|$.

Definition 1.3.61 (Representation [Wei00] and [KC12])

A representation of a set S is a surjective function $\delta : \mathbf{Reg} \rightarrow S$.

A representation can be seen as a naming system. The problem with using infinite sequences in Σ^ω is that there is no robust measure of size for complexity, which is critical in complexity theory. Following [KC12], we will use **Reg** to name real numbers, sets and functions. The complexity is then defined regarding the length of the words representing the reals and the size of the oracle, also denoted by $|\cdot|$ such that $|\phi|(|u|) = |\phi(u)|$.

Notice that, in Subsection 1.3.1, we only had to deal with sets of languages or decidable problems as inputs. Here, we have functions as inputs, thus we are doing reductions for the complexity on pseudofunctors (operators sending functions to functions, preserving composition). In that context, we say the complexity classes are defined in *second-order*. A second-order polynomial ([KC12]) is defined inductively as follows: a positive integer and a variable are a second-order polynomial and if P and Q are second-order polynomials, then so are $P + Q$, $P \cdot Q$ and $P(L)$ where $L \in \mathbb{N}^{\mathbb{N}}$ is a polynomial (type-1 variable). In other words, a second-order polynomial P is in $(\mathbb{N}^{\mathbb{N}})^{\mathbb{N}^{\mathbb{N}}}$.

Definition 1.3.62 (Second-order polynomial time [KC12])

A (deterministic or non-deterministic) TM M runs in second-order polynomial time if there is a second-order polynomial P such that, given any oracle $\phi \in \mathbf{Reg}$ and any input $u \in \Sigma^*$, M halts within $P(|\phi|)(|u|)$ steps. Define second-order polynomial space analogously by counting the number of visited cells on all tapes.

We can then define the usual ones in this context:

Definition 1.3.63 (Type-2 Time-Complexity Classes)

- \mathbf{PTIME}^2 defines the class of languages decidable in second-order polynomial time by a Type-2 Turing machine;
- \mathbf{FPTIME}^2 defines the class of functions computable in second-order polynomial time by a Type-2 Turing machine.

Definition 1.3.64 (Type-2 Space-Complexity Classes)

- \mathbf{PSPACE}^2 defines the class of languages decidable in second-order polynomial space by a Type-2 Turing machine;
- \mathbf{FSPACE}^2 defines the class of functions computable in second-order polynomial space by a Type-2 Turing machine.

To obtain hardness results on computability and complexity classes, we must define the notion of reduction we have to use, with respect to Type-2 Turing machines (Definition 1.3.41).

Following the discussions of [KC12], the reductions work on the set of regular functions \mathbf{Reg} . We define the adapted notion of reduction (Definition 1.3.65) to have complete problems.

For decision problems, we assume that the set S of Definition 1.3.61 is $\{0, 1\}$. We denote by \mathbf{Pred} the set of $\{0, 1\}$ -valued regular functions, like in [KC12].

Thus, we give the definition of Type-2 reductions for decision of [KC12]:

Definition 1.3.65 (Type-2 reductions [KC12])

Let A and B be functions from \mathbf{Reg} to \mathbf{Pred} . We write $A \leq_m^2 B$ if there exists $s, t \in \mathbf{FPTIME}^2$ such that for any $\phi \in \text{Dom}(A)$, we have $s(\phi) \in \text{Dom}(B)$ and for $\theta = B(s(\phi))$ the function $\theta \circ t(\phi) = A(\phi)$.

More intuitively, for A and B two decision problems, we define

$$\text{Dom}(A) = \{\phi \text{ a function}, x \in \text{Dom}(\phi)\}$$

and we have $A(\phi)(x) = (B(s(\phi)) \circ t(\phi))(x)$.

$$\begin{array}{ccc}
\phi & \xrightarrow{A} & A(\phi) \\
\downarrow t & & \parallel \\
t(\phi) & \xrightarrow{B \circ s \circ \phi} & \theta \circ t(\phi)
\end{array}
\quad \text{or} \quad
\begin{array}{ccc}
\phi & \xrightarrow{A} & A(\phi) \\
\downarrow t & \nearrow B \circ s \circ \phi & \\
t(\phi) & &
\end{array}$$

As highlighted in [KC12], this construction might seem somewhat artificial. The point is, it is strong enough to make the following classical problems complete and it is invariant under replacing the representations to equivalent ones.

We can deduce the definition of $\mathbf{PSPACE}^2\text{-}\leq_m^2$ -completeness:

Definition 1.3.66 ($\mathbf{PSPACE}^2\text{-}\leq_m^2$ -completeness)

A problem P is $\mathbf{PSPACE}^2\text{-}\leq_m^2$ -complete iff:

- $P \in \mathbf{PSPACE}^2$;
- P is $\mathbf{PSPACE}^2\text{-}\leq_m^2$ -hard: for all problem $P' \in \mathbf{PSPACE}^2$, $P' \leq_m^2 P$.

With this notion of reduction several $\mathbf{PSPACE}^2\text{-}\leq_m^2$ -complete problems were defined:

We denote by “ $g^{[k]}$ ” the fact that g is iterated k times. If g is a function, then g is composed k times with itself. If it is a string or an integer, then $g^{[k]}$ is a string with g repeated k times. We denote by $|s|$ the length of the string s .

Basic \mathbf{PSPACE}^2 :

Input: $\langle M, \bar{\mu}, \phi \rangle$, with M a Type-2 Turing machine, $\bar{\mu} \in \mathbf{Reg}$ such that $\bar{\mu}(u) = 0^{\mu(|u|)}$ where $\mu : \mathbb{N} \rightarrow \mathbb{N}$ a non-decreasing polynomial bounding the space of M , $\phi \in \mathbf{Reg}$ and a string u .

Output: Does M with an oracle ϕ accept u ?

Power 2 :

Input: $f \in \mathbf{Reg}$ length-preserving ($|f(x)| = |x|$), a string u .

Output: $f^{2^{|u|}}(u) = 0^{|u|}$?

As defined in [KC12], if ϕ_p is a boolean formula involving predicate p ($\{0, 1\}$ -valued function), then

$$Q_1 a_1 \cdots Q_k a_k \phi_p(a_1, \dots, a_k),$$

where each $Q_i \in \{\forall, \exists\}$ is a quantified boolean formula involving predicate p . The truth value is determined relatively to the predicate to be substituted into p .

QBF 2 :

Input: $p \in \mathbf{Pred}$, a string u .

Output: Is u , a quantified boolean formula involving predicate p , made true by p ?

Proposition 1.3.67 ([KC12])

Basic PSPACE², Power² and QBF² are PSPACE²- \leq_m^2 -complete for the reduction of Definition 1.3.65.

We use this type of reduction in Chapter 5.

1.4 Dynamical Systems

Dynamical systems are often used to model natural phenomena and in many applied fields. The last decades have seen an impressive use of computers in studying and analysing them, with several visible theoretical breakthroughs. A famous, notable example is the discovery of strange attractors, in models such as Lorenz attractors through numerical simulations [Lor63]. The mathematical proof of their existence arrived only 40 years later in [Tuc02]. The proof was obtained by checking that some quantitative invariant holds through computer-certified computations.

Discrete-Time Dynamical Systems

Definition 1.4.1 (Discrete-time Dynamical System)

A discrete-time dynamical system is given by a set D , called domain and some function \mathbf{u} from D to D . A trajectory in the system is a sequence $\mathbf{f}(t)$ evolving according to \mathbf{u} : that is $\mathbf{f}(t+1) = \mathbf{u}(\mathbf{f}(t))$ for all t .

We recall we denote by $\mathbf{f}^{[t]}$, $t \in \mathbb{N}$, the t -th iteration of function \mathbf{f} (\mathbf{f} composed with itself t times).

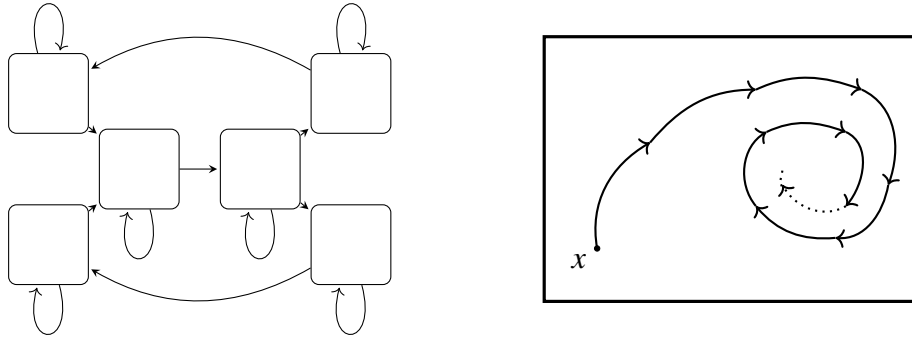


Figure 1.1: On the left, an example of dynamical system over a domain made of 6 states and \mathbf{f} is represented by the arrows between the states. On the right, an example of a trajectory of a dynamical system over \mathbb{R}^2 starting from x .

We also write $\mathbf{x} \rightarrow^* \mathbf{y}$ if there exists $n \in \mathbb{N}$ such that $\mathbf{y} = \mathbf{f}^{[n]}(\mathbf{x})$. When $n > 0$, then we write $\mathbf{x} \rightarrow^+ \mathbf{y}$.

A dynamical system can equivalently be described by its flow Φ : by definition, $\Phi(\mathbf{f}_0, t)$ gives the position of the dynamics at time t , for an initial position $\mathbf{f}(0) = \mathbf{f}_0$. It satisfies

the flow property

$$\Phi(\mathbf{f}_0, 0) = \mathbf{f}_0 \quad \Phi(\mathbf{f}_0, t + t') = \Phi(\Phi(\mathbf{f}_0, t), t') \quad (1.1)$$

for all t, t' . The transition dynamic \mathbf{u} can be recovered from the flow function since $\mathbf{u}(\cdot) = \Phi(\cdot, 1)$. Hence, describing a dynamical system by its dynamic \mathbf{u} or by a flow function is equivalent. All of this can be parametrised by some \mathbf{x} : \mathbf{u} is also some function of \mathbf{x} , and $\mathbf{f}(t + 1) = \mathbf{u}(\mathbf{f}(t), \mathbf{x})$ for all t , and the flow function is $\Phi(\mathbf{x}, \mathbf{f}_0, t) = \Phi_{\mathbf{x}}(\mathbf{f}_0, t)$.

Continuous-Time Dynamical Systems

We can also consider continuous-time dynamical systems:

Definition 1.4.2

A continuous-time dynamical system \mathcal{H} is given by a set $X \subseteq \mathbb{R}^d$, called the domain, and some function $\mathbf{f} \in X^X$. A trajectory starting from \mathbf{f}_0 is a solution of the associated Initial Value Problem (IVP) $\mathbf{u}' = \mathbf{f}(\mathbf{u}(t))$, with $\mathbf{u}(0) = \mathbf{f}_0$.

If we consider $\Phi(\mathbf{f}_0, t)$ as giving the value of the solution at time t , starting from \mathbf{f}_0 , it still satisfies the flow property (1.1). Assuming sufficient regularity on Φ (namely that it is continuous and differentiable), Φ satisfies some ODE, and hence giving a dynamical system is equivalent to giving its flow: $\mathbf{u}(\cdot)$ can be recovered by $\mathbf{u}' = \Phi'(\cdot, 0)$. Here, we can consider that all of this is parameterised by some \mathbf{f}_0 .

Hence, in a very general view, a dynamical system can also be given by some function Φ satisfying the flow property. The fact that it is continuous or discrete time, is related to the nature of its time (i.e. second) variable, that belongs to \mathbb{N} or \mathbb{Z} in the latter case, and to \mathbb{R} in the former. See [HSD03] for a monography on the theory of dynamical systems from a mathematical point of view.

In all previous discussions, we considered homogeneous dynamics in the sense that \mathbf{u} was only a function of $\mathbf{f}(t)$ and not also of t : but, for example, for discrete-time, to cover the case $\mathbf{f}(t + 1) = \mathbf{u}(t, \mathbf{f}(t))$ for all t , it is sufficient to consider $\bar{\mathbf{f}}(t) = (\mathbf{f}(t), t)$, and we come back to the previous settings, as we can write $\bar{\mathbf{f}}(t + 1) = \bar{\mathbf{u}}(\bar{\mathbf{f}}(t))$, with $\bar{\mathbf{u}}(\mathbf{f}, t) = (\mathbf{u}(\mathbf{f}, t), t + 1)$, that is a homogeneous dynamic.

Dynamical systems as group actions

We can also adopt another point of view and study dynamical systems with group theory.

Let \mathcal{G} denote either \mathbb{Z} or \mathbb{N} , or a finite products of \mathbb{Z} or \mathbb{N} , such as $\mathbb{Z} \times \mathbb{N}$.

The flow of a discrete-time dynamical system can be seen as a semi-group (\mathbb{N} -)action. If G is a group with identity element e , and X is a set, then a (left) group action α of G on X is a function $\Phi : G \times X \rightarrow X$ satisfying the following two axioms:

- $\Phi(e, x) = x$;
- $\Phi(g, \Phi(h, x)) = \Phi(gh, x)$ for all g and h in G and all x in X .

It is precisely the translation of the flow property we already described. We restrict here to \mathbb{N}^d or \mathbb{Z}^d -actions, and denote by \mathcal{G} the sets \mathbb{N} or \mathbb{Z} , depending on the context. We can then consider more general classes of dynamical systems, defined by (semi-)group actions.

More formally in our context, a \mathcal{Z}^d -**action on** X is a function $\Phi : \mathcal{Z}^d \times X \rightarrow X$ that satisfies the following two axioms $\Phi(0, x) = x$ and $\Phi(g, \Phi(h, x)) = \Phi(g + h, x)$ for all g and h in G and all x in X .

Let's restrict to the case where the subgroup G is finitely generated, and commutative, namely \mathcal{Z}^d . For example, for \mathbb{Z}^2 , there are two generators, that we can call *east* and *north*. We write \mathcal{G} for the finitely many generators. Here $\mathcal{G} = \{\text{east}, \text{north}\}$.

NB 1.4.3

Giving a subgroup, with finitely many generators, is the same as giving a function f^g for each generator $g \in \mathcal{G}$, respecting the group equations, considering equation $f^g(x) = \Phi(g, x)$, for a generator, and the flow equation.

Giving a \mathcal{Z}^d -action is the same as giving a function f^g for each element of the canonical basis of \mathcal{Z}^d (the generators of the (semi-)group \mathcal{Z}^d), respecting the relations among those generators. Thus, we sometimes present things with some function f^g (or forget about the generator g when this is clear).

Non-deterministic dynamical systems

We also need to talk about **non-deterministic discrete-time dynamical systems**. A non-deterministic *discrete-time dynamical system* \mathcal{H} is now given by a set X , called *domain* and some set-valued function $F : X \rightarrow \mathcal{P}(X)$, with $\mathcal{P}(X)$ the power set of X . An *infinite trajectory* (also called *run*, as we do for TMs) of \mathcal{H} is some sequence $(x_t)_{t \in \mathbb{N}}$ such that $x_{t+1} \in F(x_t)$ for all t . Similarly for finite trajectories.

Definition 1.4.4 (Reachability)

We say x^* (or a set X^*) is *reachable from* x if there is a trajectory with $x_0 = x$ and $x_t = x^*$ (respectively $x_t \in X^*$) for some t .

This can still be described by a set-valued flow.

Reachability Questions

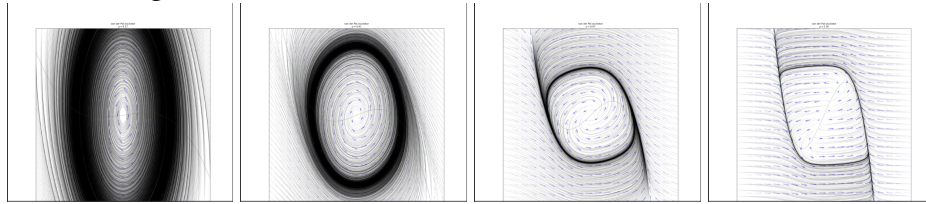
Dynamical systems exhibit a very rich class of possible behaviours. In particular, dynamical systems may be chaotic. We do not intend here to recall how chaoticity can be defined in mathematical terms (see [Dev89, HSD03]), but this includes at least high sensitivity to initial conditions. From our point of view, we just need to say this leads, in practice, to high unpredictability in the long run. As observed in [RS23], while countless papers exist in the literature about computations in dynamical systems, only a small fraction of them address the problem rigorously: i.e., how far is the sought actual quantity from the computed one? And can a computer perform such computation up to a very small pre-specified error? The reachability problem gives an important example where these questions are of interest: given a (finite) description of a dynamical system, a description of its initial state, and the description of some “unsafe” states, the question is to tell whether a trajectory can reach an unsafe state.

In the long run, dynamical systems may also exhibit attractors. Although there is a clear agreement about this intuitive concept, corresponding to the set of points to which most points evolve, it is a hard task to provide a mathematical definition covering all cases. It is a similar problem to the concept of chaos. We refer to [Mil85] for discussions

of many possible ways of defining attractors and their relations. In the general case, the attractors of dynamical systems, even if the system is very simple, can be very rich. We refer to [RS23] for a characterisation of the hardness of computing attractors from a computable analysis point of view. In this document, we somehow restrict to a very robust one (numerically stable ones) for which the problem of computing attractors is tractable.

Example 1.4.5

The van der Pol equation gives a very famous example of a dynamical system admitting an attractor: $y'' - \mu(1 - y^2)x' + y = 0$ where μ is a parameter. Introducing $z = y - y^3/3 - \dot{y}/\mu$, it can be also described by $y' = \mu(y - \frac{1}{3}y^3 - z)$, $z' = \frac{1}{\mu}y$. It is a classical mathematical exercise to prove that the system has a limit cycle: this is usually done as an application of Liénard's theorem, which is established by studying the flow of the dynamics, using qualitative arguments based on some formal mathematical statements: see e.g. [HSD03].



Simulations between discrete-time dynamical systems

A discrete-time dynamical system Φ' over X' *simulates* some dynamical system Φ over X if there is some relation $R \subset X \times X'$ such that for all $(\mathbf{x}, \mathbf{x}') \in R$, if $\mathbf{y} \in \Phi(g, \mathbf{x})$ for some generator g , then there is some $\mathbf{y}' \in \Phi'(g, \mathbf{x}')$ with $(\mathbf{y}, \mathbf{y}') \in R$.

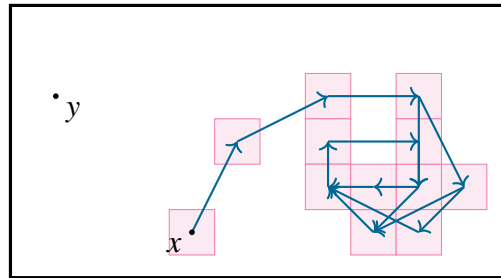


Figure 1.2: A discrete-time dynamical system simulating the dynamical system of Figure 1.1: its states are indicated by boxes and the possible transitions between states by an arrow.

1.5 Tiling theory

1.5.1 Tilings as dynamical systems

Tiling and (discrete-time) dynamical systems theories are strongly related. A mathematical formalisation for such is the field of *symbolic dynamic*. We give here some basic

definitions and theorems, taken from the lecture notes [JV20, BBEP10, Sab12, FBF⁺02]. We assume $d \in \mathbb{N}$ and A is a finite set unless stated otherwise.

Definition 1.5.1 (Configuration)

We consider the set of functions $x : \mathbb{Z}^d \rightarrow A$. An element $x \in A^{\mathbb{Z}^d}$ is called a configuration.

Let $\mathbb{U} \subset \mathbb{Z}^d$ be a finite set.

Definition 1.5.2 (Pattern)

A pattern is an element $p \in A^{\mathbb{U}}$, the support of p is $\text{supp}(p) = \mathbb{U}$. Given a configuration x , we write $x_{\mathbb{U}}$ for the pattern corresponding to the restriction of x to \mathbb{U} .

Definition 1.5.3

A pattern $p \in A^{\mathbb{U}}$ appears in x if there exists $\mathbf{i} \in \mathbb{Z}^d$ such that $x_{\mathbf{i}+\mathbb{U}} = p$, this is written $p \sqsubset x$.

Shifts act on configurations:

Definition 1.5.4 (Shift)

Let A be an alphabet. The shift $\sigma : \mathbb{Z}^d \times A^{\mathbb{Z}^d} \rightarrow A^{\mathbb{Z}^d}$ is the function such that, for all $\mathbf{i} \in \mathbb{Z}^d$ and for all $w \in A^{\mathbb{Z}^d}$, if we write $\sigma^{\mathbf{i}}(w)$ for $\sigma(\mathbf{i}, w)$, we have $\sigma^{\mathbf{i}}(w) = w'$, where w' is the configuration satisfying, for all $\mathbf{j} \in \mathbb{Z}^d$, $w'(\mathbf{j}) = w(\mathbf{j} - \mathbf{i})$.

In this context, we usually consider the following space:

Definition 1.5.5 (Subshift)

A subshift on the finite alphabet A is a subset X of $A^{\mathbb{Z}^d}$ which is closed (for the topology (A, d_C)) and invariant by shifting ($x \in X \Rightarrow \sigma(x) \in X$, σ a shift).

Usually, as in the previous definition and as we said, the alphabet A is assumed to be finite. We will call a *generalised subshift* a closed set of $A^{\mathbb{Z}^d}$ stable by shifting, with A a possibly infinite set.

Subshifts can also be defined by their forbidden patterns:

Theorem 1.5.6

Let $S \subseteq \{0, 1\}^{\mathbb{Z}^d}$. A set $\mathbf{T} \subset A^S$ is a subshift if and only if there exists a set of patterns \mathcal{F} such that $\mathbf{T} = \mathbf{T}(A, S, \mathcal{F})$, where $\mathbf{T}(A, S, \mathcal{F}) = \{x \in A^S \mid p \text{ does not appear in } x \text{ for all } p \in \mathcal{F}\}$.

When a subshift can be defined by a computably enumerable family \mathcal{F} of forbidden patterns, it is called an *effectively closed subshift*.

There exist many variants of subshifts, depending on their additional properties:

Definition 1.5.7 (SFT)

A shift of finite type (SFT) is a subshift $X \subset A^{\mathbb{Z}^d}$ that can be defined by a finite set \mathcal{F} of forbidden patterns.

SFT can also be defined by their authorised patterns, which correspond to Wang tile-sets. It is the object we study in Subsection 1.5.3.

With a set of configurations, we can associate a language:

Definition 1.5.8 (Language of patterns)

For $\mathbb{U} \subset \mathbb{Z}^d$, the language of support \mathbb{U} of the set $T \subset A^{\mathbb{Z}^d}$ is:

$$\mathcal{L}(T) = \{p \in \mathbb{U} \mid \exists x \in T, p \sqsubset x\}.$$

1.5.2 Minimality and Essential Minimality

We are interested in *minimal subshifts* in Chapter 6:

Definition 1.5.9 (Minimal subshift)

A subshift \mathbf{T} is minimal if it does not strictly contain a non-empty subshift.

We define the notion of recurrence for patterns in a configuration:

Definition 1.5.10 (Recurrence)

Let \mathbb{F}_n be the set $\llbracket -n, n \rrbracket^d$. A pattern p is recurrent in a configuration x iff there exists some $m \in \mathbb{N}^d$ such that for all $\mathbf{i} \in \mathbb{Z}^d$ we have $x_{\mathbf{i}+\mathbb{F}_m}$ contains p .

For a given subshift \mathbf{T} , we denote by $\mathcal{R}ecurrent(\mathbf{T})$ the set of recurrent patterns in the configurations of \mathbf{T} .

Proposition 1.5.11 (Properties of minimal subshifts)

For some configuration x in subshift \mathbf{T} , we define the orbit of x : $\mathcal{O}(x) = \{\sigma^i(x) \mid i \in \mathbb{Z}^d\}$. The following statements are equivalent:

1. \mathbf{T} is minimal;
2. for all $x \in \mathbf{T}$, $\mathcal{O}(x)$ is dense in \mathbf{T} ;
3. for all $x, y \in \mathbf{T}$, $\mathcal{L}(x) = \mathcal{L}(y)$.

A proof can be found for example in [Sab12], similarly for the following proposition:

Proposition 1.5.12

Every subshift contains a minimal subshift.

Furthermore,

Proposition 1.5.13

A subshift X is minimal iff each pattern $p \in \mathcal{L}(X)$ is recurrent in X .

Minimality for a subshift is a rather strong property. For example, the Robinson tileset (discussed later) is not minimal.

We consider in Chapter 6 a weaker concept of minimality:

Definition 1.5.14 (Essential minimality)

A subshift \mathbf{T} is essentially minimal if it contains a unique (non-empty) minimal subshift \mathbf{Y} that is to say, a unique non-empty closed shift-invariant set.

We write $\min(\mathbf{T})$ for that \mathbf{Y} . This notion of minimality is weaker than Definition 1.5.9. For example, the Robinson tileset is essentially minimal.

Proposition 1.5.15 ([Bru19])

Given a subshift \mathbf{T} and a point $y \in \mathbf{T}$, the following are equivalent:

1. \mathbf{T} is essentially minimal and $y \in \min(\mathbf{T})$;
2. for every $x \in \mathbf{T}$, $y \in \omega(x) = \bigcap_{n \in \mathbb{N}} \overline{\{\sigma^i(x) \mid i \notin \llbracket -n, n \rrbracket^d\}}$;
3. For every open set U with $y \in U$, $\mathbf{T} = \bigcup_{n \in \mathbb{Z}} \sigma^n(U)$.

1.5.3 Tiles and Tilings

As we mentionned, SFTs can also be described by their authorised patterns. Those patterns corresponds to Wang tiles.

Here, we focus on Wang tilesets, in dimension 2:

Definition 1.5.16 (Wang tilesets)

A Wang tileset T is a triple (H, V, τ) where:

- H is a finite set that corresponds to the horizontal colours;
- V is a finite set that corresponds to vertical colours;
- $\tau \subset H^2 \times V^2$ is the set of tiles.

A tile $t \in H^2 \times V^2$ will also be denoted by $t = (w, e, s, n)$ where w (resp. e, s, n) is the colour on the left (resp. right, bottom, top) edge. We will also write $t(w), t(e), t(s), t(n)$ for w, e, s, n respectively.

Example 1.5.17

We give an example of a Wang tileset, made with 4 tiles:

- $H = \{\text{Red}, \text{Green}, \text{Blue}\}$
- $V = \{\text{Red}, \text{Green}\}$

- And τ is the set:



Between tiles, we will talk about *east-west* and *north-south* compatibilities.

Definition 1.5.18 (Tiling)

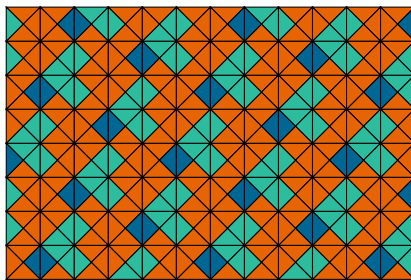
Let $P \subseteq \mathbb{Z}^2$. A tiling of P by a tileset τ assigns to each element of P a tile from τ such that colors of adjacent tiles agree on their common border. Formally, a tiling is a map $x : P \rightarrow \tau$ such that, for every $(i, j) \in \mathbb{Z}^2$ on the plane:

- If $(i, j) \in P$ and $(i + 1, j) \in P$ then $x(i, j)(e) = x(i + 1, j)(w)$ (east-west compatibility);
- If $(i, j) \in P$ and $(i, j + 1) \in P$ then $x(i, j)(n) = x(i, j + 1)(s)$ (north-south compatibility).

When $P = \mathbb{Z}^2$, it is a tiling of the plane. We say that τ tiles P if there exists a tiling of P by τ .

Example 1.5.19

Using the set of tiles of Example 1.5.17, we have the following tiling of a rectangle:



By a compactness argument, the following holds:

Proposition 1.5.20 (Compactness)

A tileset τ tiles the plane iff for all n , τ tiles a square of size $n \times n$.

A very fundamental problem in tiling theory is the following:

Domino Problem:

Input: A tileset τ

Output: Can τ tile \mathbb{Z}^2 ?

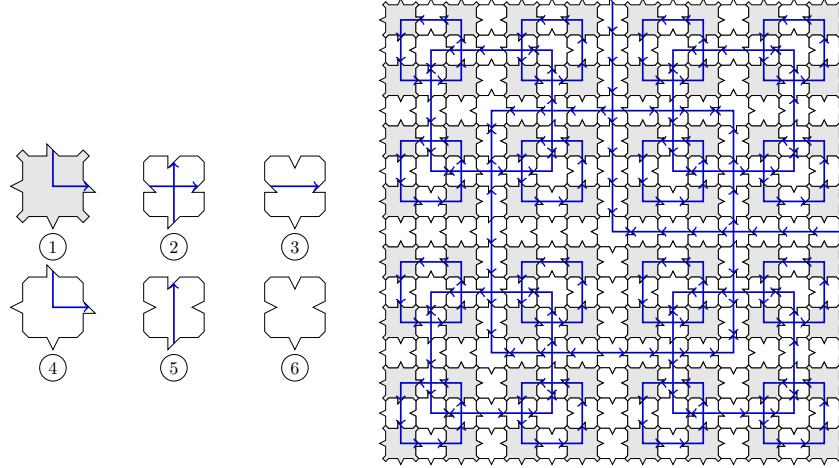
The following is true from Proposition 1.5.20:

Proposition 1.5.21 ([Ber66])

The **Domino Problem** is co-c.e..

It holds because there is a finite number of tiles in the tileset, so it is possible to decide whether the tiling does not tile a given square. However, the Domino problem is not c.e., hence undecidable, in the general case.

A tiling x is *periodic* of period $(a, b) \in \mathbb{Z}^2 \setminus \{0, 0\}$ iff, for all $(u, v) \in \mathbb{Z}^2$, $x(u, v) = x(u + a, v + b)$. In [Wan61], Wang gave the conjecture that a tiling of \mathbb{Z}^2 is necessarily periodic. However, this conjecture is false: there exist aperiodic tilings (valid and non-periodic), such as the Robinson tiling ([Rob71]), made of the following tiles (on left), that can be turned.



An example of a tiling of the plane is illustrated on the right. Intuitively, any Robinson tiling of the plane is aperiodic because an arbitrarily big blue square must be in a bigger blue square. Inductively, for every $n \in \mathbb{N} \setminus \{0, 1, 2\}$, we can construct a square of size $2^n - 1$, containing 4 squares of size $2^{n-1} - 1$. Proofs of the statement can be found in [Rob71].

We focused on 2-dimensional Wang tilesets. More generally, we can consider Wang tilings on d -dimensional spaces. We can also have tilesets with other shapes than squares. It is interesting to investigate what features of the tilings make the domino problems decidable. In Chapter 6, we study those features.

1.5.4 Transducers and meta-transducers

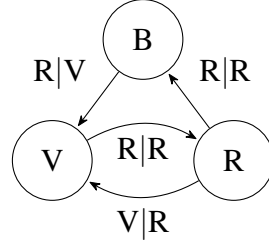
Another point of view on SFTs and Wang tilesets we consider in this thesis are transducers. They are associated with a *graph* point of view. We will use them in Chapter 6.

Definition 1.5.22 (Transducer)

Let Σ be a finite alphabet. A transducer is a triple (Q, Σ, δ) with

- Q a finite set of states, Σ a finite alphabet;
- $\delta \subseteq Q \times Q \times \Sigma \times \Sigma$ a set of transitions.

For example the transducer \mathcal{T}_1 with $Q = \{V, B, R\}$, $\Sigma = Q$ is pictured below. The fact that $(q, r, w, w') \in \delta$ is represented by an edge between q and r labelled by $w|w'$.



A Wang tileset expressed as triple (H, V, τ) exactly matches the definition of a transducer where tiles are the transitions, the horizontal colours H are states Q and the vertical colours V are letters of the alphabet Σ .

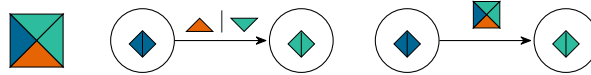


Figure 1.3: Two graphical representations of a Wang tile as a transition in the associated transducer.

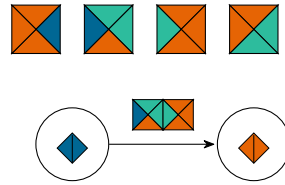


Figure 1.4: An example a of transducer derived from the tileset τ composed by the four tiles pictured on the top.

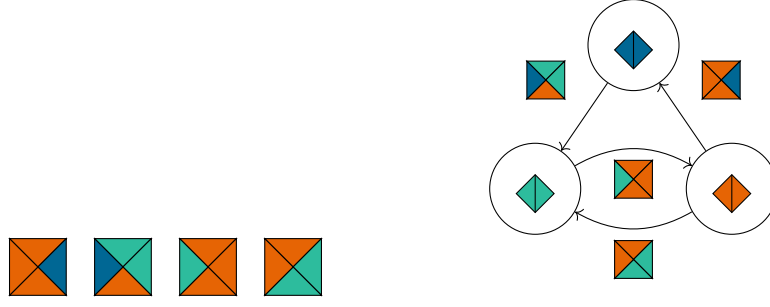
The choice of H as states and V as colours is arbitrary. We could also have considered the opposite. The transducer that we would obtain will be called the *vertical transducer*, as opposed to the (horizontal) transducer considered here.

The existence of a path of length two between two vertices of the transducer tests the horizontal compatibility between two tiles.

There is a bijection between the tiles of a Wang tileset and the edges of its associated transducer. From now on we use tileset or transducer interchangeably without any possible confusion.

Example 1.5.23

On the tiles of Example 1.5.19, starting from the right of the transducer, we put the tile according the next transition.



A natural operation on transducers, denoted by \cup , is the union: given $\mathcal{T}_1 = (Q_1, \Sigma_1, \delta_1)$ and $\mathcal{T}_2 = (Q_2, \Sigma_2, \delta_2)$, their union $\mathcal{T}_1 \cup \mathcal{T}_2$ is given by (Q, Σ, δ) where $Q = Q_1 \cup Q_2$, $\Sigma = \Sigma_1 \cup \Sigma_2$, and $(q, q', w, w') \in \delta$ iff $(q, q', w, w') \in \delta_1$ or $(q, q', w, w') \in \delta_2$.

Any transducer is thus the union of its edges. Combined with the remark immediately above, the transducer associated with a Wang tileset can be seen as the union of its tiles.

To simplify notations and the writing of proofs, it is often more convenient to use *meta-transducers* rather than transducers. Meta-transducers allow us to deal not only with single letters but also with words of arbitrary finite length as labels of transitions.

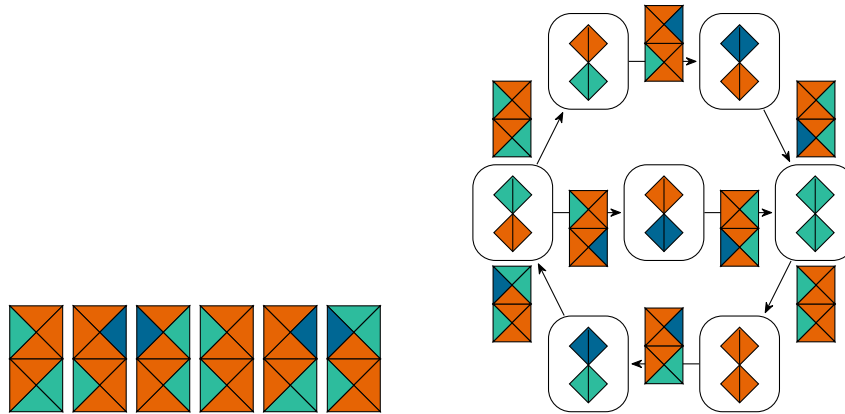
Definition 1.5.24 (Meta-transducer)

Let Σ be a finite alphabet. A meta-transducer is a triple (Q, Σ, δ) with

- Q a finite set of states, Σ a finite alphabet;
- $\delta \subseteq Q \times Q \times \Sigma^* \times \Sigma^*$ a set of transitions – or edges – with $|w|, |w'| \geq 1$ where $|w|$ denotes the length of w .

Example 1.5.25

Here is the meta-transducer of height 2 associated with the tileset of Example 1.5.17. Starting from the couple of tiles on the upper left, we just follow the transitions all around the meta-transducer:



1.6 Notations Tabulars

Name	Definition
$\mathbb{N}, \mathbb{Z}, \mathbb{D}, \mathbb{Q}, \mathbb{R}$	sets of non-negative integers, integers, dyadics, rationals, real numbers
$\mathbb{Q}_+, \mathbb{R}_+$	sets of non-negative rationals and reals
$\mathbb{Q}_-, \mathbb{R}_-$	sets of non-positive rationals and reals
\mathcal{Z}	denotes either \mathbb{Z} or \mathbb{N} , or a finite products of \mathbb{Z} or \mathbb{N} , such as $\mathbb{Z} \times \mathbb{N}$
$\bar{S}, \mathbf{cls}(S)$	closure of S
$\mathit{int}(S)$	interior of S
$\partial(S)$	frontier of S
$B_\varepsilon(x)$	ball of radius ε centered on x : $\{y d(x, y) < \varepsilon\}$
$[f_1, f_2, \dots, f_k; \mathit{op}_1, \mathit{op}_2, \dots, \mathit{op}_\ell]$	set containing f_1, \dots, f_k , closed under $\mathit{op}_1, \dots, \mathit{op}_\ell$
$(X, f), X \text{ a set}, f \in X^X$	dynamical system on X with dynamic function f
$[a, b]$	interval between a and $b, \subset \mathbb{R}$
$\llbracket a, b \rrbracket$	set $\{a, a+1, \dots, b\} \subset \mathbb{N}$
B^A	set of functions $\{f : A \rightarrow B\}$
$\mathit{Dom}(f)$	domain of the function f
$\mathcal{P}(E)$	power set of set E

Name	Definition
$\ell(n)$	$\in \mathbb{N}^{\mathbb{N}}$, the length, or size, function, mapping some integer to the length of its binary representation,
$ x , x \in \mathbb{R}$	absolute value of x
$ s , s$ a string	size of s
$ f , f \in \mathbb{R}^{\mathbb{R}}$	norm 1 of f
$\ f\ $	the sup-norm: $\sup\{f(x) x \in Dom(f)\}$
$\lfloor x \rfloor$	floor of x
$\lceil x \rceil$	ceiling of x
$\{x\}$	the fractional part of x
$\mathbf{a} =_{\varepsilon} \mathbf{b}$	$\ \mathbf{a} - \mathbf{b}\ \leq \varepsilon$
$g^{[t]}$ or $g^t, t \in \mathbb{N}$	the t -th iteration of function g (g composed with itself t times)
Name	Definition
$x \rightarrow^* y$	In a dynamical system, y is reachable from x in $n \in \mathbb{N}$ steps
$x \rightarrow^+ y$	In a dynamical system, y is reachable from x in $n \in (\mathbb{N} \setminus \{0\})$ steps
$x \rightarrow_{\varepsilon}^* y, x \rightarrow_{\varepsilon}^+ y$	the analogous of $x \rightarrow^* y$ and $x \rightarrow^+ y$ but for ε -perturbed dynamical systems

Chapter 2

State of the art

Là où les étoiles bougent, d'aucuns s'extasient. Moi, je cherche à comprendre. Et à découvrir se qui se cache derrière. J'ai toujours été curieuse et, après tout, qu'est-ce que la science, si ce n'est la forme de curiosité la plus absolue?

Élodie Serrano, Les Étoiles ne Fileront Plus

We first review results about the difficulty of solving ODEs in computer science in Section 2.1. In Section 2.2, we define the variant of the Turing machine we will consider in the rest of the document for our characterisations of complexity classes.

Furthermore, in Section 2.3, we will relate Turing machines and dynamical systems, and mention some complexity results regarding the properties of dynamical systems.

We will present the field of implicit complexity in Section 2.4. In Section 2.5, we will see previous results about discrete ODEs and give an implicit characterisation of polynomial time for functions over the integers. Finally, we study the relation between programming languages and Turing machines in Section 2.7.

2.1 Solving efficiently ODEs: what is known

We first review what is known about the complexity of ODEs solving. A more complete survey is [GZ18]. First, it is important to distinguish the case where we want to solve the ODE on a bounded (hence a compact) domain, from the case of the full domain \mathbb{R}^d : in the latter case, we might ask questions about the evolution of the system on the long run, which is harder.

Proposition 2.1.1 ([Ko83], extending [PER79, Abe71])

Over a compact domain, there exists some polynomial-time computable function $u : [-1, 1] \times [0, 1] \rightarrow \mathbb{R}$ such that $f' = u(f, t)$ has no computable solution, even over $[0, \delta]$, for any $\delta > 0$.

The involved ODE has no unique solution. It is known over compact and non-compact domains that if unicity holds, its solution is computable [CG08, CG09, Ruo96]. However, the complexity can be arbitrarily high [Ko91, Mil70].

If we want to get to tractability, some regularity hypotheses must be assumed. A classical hypothesis is to assume the ODE to be Lipschitz.

Over a compact domain, it has been observed in several references (see e.g. [Ko91]) that a careful analysis of Euler's method proves that:

Proposition 2.1.2 ([GZ18])

If $u : [0, 1]^n \times [0, 1] \rightarrow \mathbb{R}^n$, is a polynomial time computable Lipschitz function then any solution $f : [0, 1] \rightarrow [0, 1]^n$ of $f' = u(f, t)$ must be polynomial-space computable.

See the discussions around Theorem 3.2 in [GZ18] with several references. Kawamura has proved the following result:

Theorem 2.1.3 ([Kaw09])

*There exists a polynomial-time computable Lipschitz function $u : [-1, 1] \times [0, 1] \rightarrow \mathbb{R}$, such that the unique solution $f : [0, 1] \rightarrow \mathbb{R}$ takes values in $[-1, 1]$ and the computation of such function is a **PSPACE**-complete problem.*

Hence, the question of solving ODEs over a compact domain in polynomial time is the question **PTIME** = **PSPACE** [Kaw09], even for \mathcal{C}^∞ -functions [KORZ14].

However, all these results are over compact domains, and dealing with non-compact domains, i.e. in the long run, is harder. **PSPACE** membership is not true, without stronger hypotheses.

Example 2.1.4

If we consider $f_1(t) = e^t$, and $f_{i+1}(t) = e^{f_i(t)-1}$ then $f_d(t)$ is

$$e^{e \cdot e^{e^t} - 1} - 1,$$

while all these functions are solutions of a simple polynomial ODE over \mathbb{R}^d , namely $f'_1(t) = f_1(t)$ and $f'_d(t) = f_1(t) \cdot \dots \cdot f_d(t)$. Hence, a solution can grow faster than a tower of exponentials in the description of the ODE, and hence is necessarily intractable for time or space: see the discussion in [GZ18, Section 3.2].

The difficulty, even with a bounded domain, comes from the possibility of simulating any Turing machine by some finite-dimensional polynomial ODE [GZB06] over a non-compact domain. It leads to many undecidability results for analytic, and even very simple ODEs: see Chapter 5. For example, it follows that there is an analytic and computable function $u : \mathbb{R} \rightarrow \mathbb{R}$ such that the unique solution of the associated homogeneous ODE is defined on a non-computable maximal interval of existence [GZB06].

A possible way to analyse efficiency is to analyse the complexity of the solution, assuming a bound on the growth of the function (i.e. using parameterised complexity):

Proposition 2.1.5 ([BGP12])

A polynomial ODE is solvable in polynomial time assuming a bound on $\mathbf{Y}(T) = \max_{0 \leq t \leq T} \|\mathbf{f}(t)\|$.

We can extend this result to non-polynomial ODEs assuming polynomial-time computability of the higher derivatives of \mathbf{f} and an appropriate (polynomial) bound on the growth of those derivatives [BGP12].

The result for polynomial ODEs was later improved in [PG16]: the time T and parameter \mathbf{Y} can be replaced by a single parameter, namely the length of the curve for polynomial ODEs. This parameter is not required to be given as input to the algorithm. It is a key argument for one direction of the motto “time complexity = length”. We will focus on this in Chapter 3.

It is obtained using a non-classical method, from the point of view of classical numerical analysis: this is not a fixed-order numerical method, but somehow a method whose order is a function of the inputs. The authors of [Thi18, KST18] proposed independently a similar approach, considering parametrised complexity for analytic functions.

To get polynomial-time complexity over a non-compact domain, it is mandatory not to use most classical methods from numerical analysis. From the general theory of numerical methods, presented for example in [Dem96], every such numerical method come with a given fixed order k , and from the general theory of step-based methods, interval $[0, T]$ is divided into N steps, each of width $h_n \leq h$, and the error θ_n at step n can be bounded by $\theta_{n+1} \leq (1 + \Lambda h_n) \theta_n + |\varepsilon_n|$: there is a multiplicative error due to the Lipschitz constant Λ of u , and some numerical additive error ε_n due to the used precision.

Using Discrete Grönwall Lemma (Lemma 1.1.16), the final error between computed approximation \tilde{f}_n and exact solution f_n at step n satisfies

$$\max_{0 \leq n \leq N} |\tilde{f}_n - f_n| \leq e^{\Lambda T} |\tilde{f}_0 - f_0| + \frac{e^{\Lambda T} - 1}{\Lambda} \left| \max \frac{\varepsilon_n}{h} \right|.$$

If $N = T/h$, we have

$$\max_{0 \leq n \leq N} |\tilde{f}_n - f_n| \leq e^{\Lambda T} |\tilde{f}_0 - f_0| + \frac{e^{\Lambda T} - 1}{\Lambda} \left| \max \frac{\varepsilon_n}{h} \right|.$$

To go to zero, we need to choose the last factor to be of the form $\exp(-\mathcal{O}(T))$, so the number of steps cannot be polynomial.

The same problem happens when discussing space complexity: a non-classical method is required to guarantee polynomial space complexity in the long run (i.e. on the non-compact domain). As far as we know, no such method has yet been proposed, and this is the purpose of Subsection 5.2.1. Actually, for space complexity, in addition to all the problems mentioned, in all the above space or time analyses, the problem is that the complexity is (possibly implicitly) dependent on the Lipschitz constant or the length of the solution. In a system as simple as linear dynamics, the state at time T depends in Lipschitz way from the state at time 0, and the number of additional bits required to guarantee some precision 2^{-n} growth linearly with T . But the problem is that in a space polynomial in the input size, T has no reason to remain polynomial. Hence, the required precision is possibly exponential in the input size.

NB 2.1.6

The above comment can be interpreted informally as the fact that “most” (this could be “generic” in the sense of [RS23], i.e. (effective) descriptive theory) dynamical systems are intrinsically unstable, and an error method introduced at some step can make the approach unavoidably incorrect in the long run unless we have a means to “guess” what happens next. This method will be used in Chapter 5, relying on the fact that $\mathbf{NPSpace} = \mathbf{PSPACE}$.

2.2 A variant definition of Turing machines

In the rest of this thesis, we will consider Turing machines with bi-infinite tapes. We can also consider that a Turing machine is given by

$$M = (Q, \{0, 1, 3\}, q_{init}, \delta, F),$$

where the input alphabet and the output alphabet are the same (except for the blank symbol) and F is the set of final states. We assume that the input alphabet contains the symbols 0, 1, 3 and that $B = 0$ is the blank symbol and $q_{init} \in Q$. The reason for the choice of symbols 1 and 3 will be made clear later. This definition is equivalent to the classical definition (Definition 1.3.1).

2.3 Dynamical systems and associated complexity issues

2.3.1 Embedding TMs into dynamical systems

Many authors have embedded TMs in various classes of dynamical systems to get undecidability or several hardness results. For example, in [CMPS23], the authors proved that universal TMs can be simulated by a 3-dimensional steady fluid flow in a Euclidean space, linking computability and hydrodynamical systems. The paper [CR24] also tackles the question of efficiently embed TMs into dynamical systems, but focuses on using the BSS model for the dynamic.

We present in this section how this is usually done, to point out where (intuitively) (non-)robustness issues appear.

Theorem 2.3.1

Let M be a Turing machine. Then, we can construct a dynamical system $(\mathbb{N} \times \Sigma^\omega \times \Sigma^\omega, \mathbf{f})$, \mathbf{f} a PAM with rational coefficients, simulating one transition of M .

Proof. We consider a Turing machine of the form described in Section 2.2. Up to renaming, we assume $Q = \{0, 1, \dots, |Q| - 1\} \subset \mathbb{N}$. Let

$$\dots l_{-k} l_{-k+1} \dots l_{-1} l_0 r_0 r_1 \dots r_n \dots$$

denote the content of the tape of the Turing machine M .

We can write $l = l_0 l^\bullet$ and $r = r_0 r^\bullet$, respectively the left part of the tape with respect to the head and the right part of the tape, where l_0 and r_0 are the first letters of l and r , and l^\bullet and r^\bullet corresponding to the (possibly infinite) word $l_{-1} l_{-2} \dots$ and $r_1 r_2 \dots$ respectively. Notice that l is written in reverse order on the tape.

$$\begin{array}{|c|c|c|c|} \hline \dots & l^\bullet & l_0 & r_0 & r^\bullet & \dots \\ \hline \end{array}$$

$\underbrace{\hspace{10em}}_l \quad \underbrace{\hspace{10em}}_r$

In this representation, the head is in front of symbol r_0 , and $l_i, r_i \in \{0, 1, 3\}$ for all i . Furthermore, if the blank symbol $B = 0$ appears at some letter r_i then it appears at letter r_j

for all $j \geq i$; if the blank symbol $B = 0$ appears at some letter l_{-i} then it appears at letter l_{-j} for all $j \geq i$.

We denote by δ the transition function: given some control state $q \in Q$, and some symbol r_0 , $\delta(q, r_0) = (q', x, m)$ indicates the new control state $q' \in Q$, the symbol x to be written, and some movement $m \in \{\leftarrow, \rightarrow\}$, corresponding to shifting left or right. For practical reasons, we assume that when M reaches a final state, it stays in the same configuration.

A configuration C of such TM can be denoted by $C = (q, l, r)$, where

$$\begin{aligned} r &= r_0 r_1 \cdots r_n \cdots, \\ l &= l_0 l_{-1} \cdots l_{-k} \cdots \end{aligned}$$

are words over alphabet $\Sigma = \{0, 1, 3\}$ and where $q \in Q$ denotes the internal state of M . We work with infinite words, but as expected, this covers the case of finite words. Namely, given a finite word $w = w_1 w_2 \dots w_n$ on the tape, in control state q , with the head in front of symbol $1 \leq i \leq n$, corresponds to the configuration

$$C = (q, w_{i-1} w_{i-2} \dots w_1 0^\omega, w_i w_{i+1} \dots w_n 0^\omega),$$

that is to say, with eventually the blank symbol $B = 0$.

The idea is that such a configuration C can also be encoded by some element $\gamma_{\text{config}}(C) = (q, \bar{l}, \bar{r}) \in \mathbb{N} \times \mathbb{R}^2$, by considering

$$\begin{aligned} \bar{r} &= r_0 4^{-1} + r_1 4^{-2} + \cdots + r_n 4^{-(n+1)} + \cdots, \\ \bar{l} &= l_0 4^{-1} + l_{-1} 4^{-2} + \cdots + l_{-k} 4^{-(k+1)} + \cdots \end{aligned}$$

In other words, we encode the configuration of bi-infinite tape Turing machine M by real numbers using their radix 4 encoding, but using only digits $\{0, 1, 3\}$. As digit 0 is used only at the end of the words, this corresponds to some rational numbers for a finite word, while for an infinite word, this corresponds to some real numbers, possibly irrational. If we write: $\gamma_{\text{word}} : \Sigma^\omega \rightarrow \mathbb{R}$ for the function that maps word $w = w_0 w_1 w_2 \cdots$ to

$$\gamma_{\text{word}}(w) = w_0 4^{-1} + w_1 4^{-2} + \cdots + w_n 4^{-(n+1)} + \cdots,$$

we can also write

$$\gamma_{\text{config}}(C) = \gamma_{\text{config}}(q, l, r) = (q, \gamma_{\text{word}}(l), \gamma_{\text{word}}(r)).$$

Here, the transition function δ leads naturally to a function $\bar{\delta} : C \mapsto C'$ that maps a configuration $C = (q, l, r)$ to a configuration $C' = (q', l', r')$ after one step in M . Via the above encoding, this can be seen as a function $\mathbf{f} : \mathbb{N} \times \mathbb{R}^2 \mapsto \mathbb{N} \times \mathbb{R}^2$.

Notice that \mathbf{f} is defined in $Q \times [0, 1]^2$ and it is a piecewise affine map. Actually, if we denote the image of $\gamma_{\text{word}} : \Sigma^\omega \rightarrow \mathbb{R}$ by \mathcal{J} , it even takes its values in $Q \times \mathcal{J}^2$.

It remains to prove that \mathbf{f} is a PAM:

The function \mathbf{f} is basically of the form $\mathbf{f}(q, l, r) = \mathbf{f}(q, l_0 l^\bullet, r_0 r^\bullet) = (q', l', r')$, defined as a definition by case of type:

$$(q', l', r') = \begin{cases} (q', x l_0 l^\bullet, r^\bullet) & \text{whenever } \delta(q, r_0) = (q', x, \rightarrow) \\ (q', l^\bullet, l_0 x r^\bullet) & \text{whenever } \delta(q, r_0) = (q', x, \leftarrow) \end{cases}$$

This rewrites as a function $\bar{\mathbf{f}}$ which is similar, working over the representation of the configurations as reals, considering $r_0 = \lfloor 4\bar{r} \rfloor$

$$\begin{aligned}\bar{\mathbf{f}}(q, \bar{l}, \bar{r}) &= \bar{\mathbf{f}}(q, \overline{l_0 l^\bullet}, \overline{r_0 r^\bullet}) = (q', \bar{l}', \bar{r}') \\ &= \begin{cases} (q', \overline{x l_0 l^\bullet}, \overline{r^\bullet}) & \text{whenever } \delta(q, r_0) = (q', x, \rightarrow) \\ (q', \bar{l}^\bullet, \overline{l_0 x r^\bullet}) & \text{whenever } \delta(q, r_0) = (q', x, \leftarrow) \end{cases}\end{aligned}$$

where $r_0 = \lfloor 4\bar{r} \rfloor$ and

- in the first case “ \rightarrow ” : $\bar{l}' = 4^{-1}\bar{l} + 4^{-1}x$ and $\bar{r}' = \bar{r}^\bullet = \{4\bar{r}\}$
 - in the second case “ \leftarrow ” : $\bar{l}' = \bar{l}^\bullet = \{4\bar{l}\}$ and $\bar{r}' = 4^{-2}\bar{r}^\bullet + 4^{-2}x + 4^{-1}\lfloor 4\bar{l} \rfloor$
- (2.1)

We introduce the following functions:

$$\begin{aligned}\rightarrow: Q \times \{0, 1, 3\} &\mapsto \{0, 1\} \\ (q, a) &\mapsto 1 \quad \text{if } \delta(q, a) = (\cdot, \cdot, \rightarrow) \\ (q, a) &\mapsto 0 \quad \text{otherwise}\end{aligned}$$

corresponding to the head moving to the right, and

$$\begin{aligned}\leftarrow: Q \times \{0, 1, 3\} &\mapsto \{0, 1\} \\ (q, a) &\mapsto 1 \quad \text{if } \delta(q, a) = (\cdot, \cdot, \leftarrow) \\ (q, a) &\mapsto 0 \quad \text{otherwise}\end{aligned}$$

corresponding to the head moving to the left.

Hence, we can rewrite,

$$\bar{\mathbf{f}}(q, \bar{l}, \bar{r}) = (q', \bar{l}', \bar{r}')$$

as

$$\bar{l}' = \sum_{q, r_0} \left[\rightarrow(q, r_0) \left(\frac{\bar{l}}{4} + \frac{x}{4} \right) + \leftarrow(q, r_0) \{4\bar{l}\} \right]$$

and

$$\bar{r}' = \sum_{q, r_0} \left[\rightarrow(q, r_0) \{4\bar{r}\} + \leftarrow(q, r_0) \left(\frac{\{4\bar{r}\}}{4^2} + \frac{x}{4^2} + \frac{\lfloor 4\bar{l} \rfloor}{4} \right) \right].$$

And $\{\cdot\}$ and $\lfloor \cdot \rfloor$ are PAMs. \square

NB 2.3.2

We will use this construction for the proof of Lemma 3.1.10, with a function *Next* (to be defined) instead of \mathbf{f} .

The embedding of configurations of TM into reals, that we just used and will mainly use in Chapter 4 and Chapter 6, is $\Upsilon(C) = \gamma_{\text{config}}(C)$. But, without loss of generality, we could have taken $\Upsilon(q, w_1, w_2) = (q, \gamma(w_1), \gamma(w_2))$ with $\gamma: \Sigma^* \rightarrow \mathbb{R}$ taken as:

- the encoding $\gamma_{\mathbb{N}}$ mapping the word $w = a_1 \dots a_n$ to the integer whose binary expression is w ,

- or $\gamma_{[0,1]}$ mapping w to the real number of $[0, 1]$ whose binary expansion is w ,
- or more generally, $\gamma_{[0,1]}^k$ or $\gamma_{\mathbb{N}}^k$, using base k instead of base 2 for $k \geq 2$,
- or $\gamma_{[0,1]}^{\prime k}$ mapping w to $(\gamma_{[0,1]}^k(w), \ell(w))$.

In any case, we can then consider a function $\mathbf{f}: X \subseteq \mathbb{R}^d \rightarrow X$ such that for any configuration C , denoting by C' the next configuration, $\mathbf{f}(\Upsilon(C)) = \Upsilon(C')$: one step of the TM corresponds to one step in the dynamical system (X, \mathbf{f}) with respect to γ . Then, we have that the diagram of the execution commutes for any number of steps:

$$\begin{array}{ccccccc}
 C & \xrightarrow{\vdash} & C' & \xrightarrow{\vdash} & C'' & \xrightarrow{\vdash} & C''' \cdots \xrightarrow{\vdash} \\
 \Upsilon \downarrow & & \Upsilon \downarrow & & \Upsilon \downarrow & & \Upsilon \downarrow \\
 \Upsilon(C) & \xrightarrow{\mathbf{f}} & \Upsilon(C') & \xrightarrow{\mathbf{f}} & \Upsilon(C'') & \xrightarrow{\mathbf{f}} & \Upsilon(C''') \cdots \xrightarrow{\mathbf{f}}
 \end{array}$$

The questions related to the existence of trajectories in the dynamical system (X, \mathbf{f}) associated with the TM lead to questions about the existence of suitable trajectories of the Turing machine. Specifically, it provides c.e.-hardness of reachability for various classes of dynamical systems. Call such a situation a *step-by-step* emulation. However, encodings such as $\gamma_{[0,1]}$, whose image is compact, or on γ_{word} as above, map intrinsically distinct configurations to points arbitrarily close to each other (a sequence over a compact must have some accumulation point). Encodings like $\gamma_{\mathbb{N}}$ do not have a compact image but involve emulations with arbitrarily large integers, which is another issue. These observations led to the already mentioned (informal) conjecture that undecidability/hardness may hold only for non-robust systems (sensitive to arbitrarily small perturbations). This leads to discussing more formally robustness issues for general dynamical systems over \mathbb{R}^d for some $d \in \mathbb{N}$. We will come back to those questions in Chapter 4.

These statements and underlying constructions, which allow the simulation of Turing machines, led to solving several open problems: the existence of a universal ODE [BP17], the proof of the Turing-completeness of chemical reactions [FGBP17], or statements about the hardness of several dynamical systems problems [GZ18].

Thus, we need the transition function in a Turing machine to be Lipschitz:

Lemma 2.3.3

Let M be a Type-2 Turing machine. The transition function \mathbf{f} of M is Lipschitz.

2.3.2 Some complexity results on graphs

We discussed the hardness of solving IVP, or, equivalently, of computing $\Phi(y, t)$ for continuous-time dynamical systems. We will also need to discuss the case of discrete-time ODEs.

For pedagogical reasons, we first discuss the case of a simple setting, namely the case of a (deterministic) discrete-time systems over a finite space, i.e. directed graphs. Indeed, observe that a deterministic discrete-time dynamical system (X, \mathbf{f}) can also be seen as

a particular (deterministic) directed graph $G = (V, \rightarrow)$, where, in the general case, V is not necessarily finite: G corresponds to $V = X$ and \rightarrow to the graph of the function \mathbf{f} , i.e. $\mathbf{x}_t \rightarrow \mathbf{x}_{t+1}$ iff $\mathbf{x}_{t+1} = \mathbf{f}(\mathbf{x}_t)$. If the dynamical system is deterministic, the obtained graph is deterministic because any vertex has an outdegree 1.

Starting from some point \mathbf{x}_0 , there is at most one possible path, and consequently, for a given time T , we can talk about its position at time T , i.e. $\Phi(\mathbf{x}_0, T)$ is T th element of this path.

We come to the space complexity of computing path, i.e. of computing $\Phi(\mathbf{x}_0, T)$, when V is finite (or equivalently when $V = X$ can be abstracted by some finite set). As usual in complexity theory, the length of some integer is the length of its binary representation. Also, a function $f \in \mathbb{N}^{\mathbb{N}}$ is *space-constructible* when there is a Turing machine that, on input n written in unary, outputs the binary representation of $f(n)$ using at most $O(f(n))$ cells.

Proposition 2.3.4 (The case of finite graphs)

Let $s(n) \geq \log(n)$ be space-constructible. Assume the vertices of $G = (V, \rightarrow)$ can be encoded in binary using words of length $s(n)$. Assume the relation \rightarrow is decidable using a space polynomial in $s(n)$. Then,

- given the encoding of $\mathbf{u} \in V$ and of $\mathbf{v} \in V$, we can decide whether there is some path from \mathbf{u} to \mathbf{v} , in a space polynomial in $s(n)$.
- given the encoding of $\mathbf{u} \in V$, and integer T in binary, we can compute $\Phi(\mathbf{u}, T)$, in a space polynomial in $s(n)$ and the length of T .

The second item is even a characterisation of the complexity of the problem. Indeed, the converse is true: if, given the encoding of $\mathbf{u} \in V$ and integer T in binary, we can compute $\Phi(\mathbf{u}, T)$ in a space polynomial in $s(n)$ and the length of T , then as \rightarrow is given by $\Phi(\cdot, 1)$, then \rightarrow is decidable using a space polynomial in $s(n)$.

Proof. Given a directed graph $G = (V, \rightarrow)$ and some vertices $\mathbf{u}, \mathbf{v} \in V$, determining whether there is some path between \mathbf{u} and \mathbf{v} in G , denoted by $\mathbf{u} \xrightarrow{*} \mathbf{v}$ is in **NL** (Corollary 1.3.37). The algorithm described in NB 1.3.39, working over representations of vertices, when vertices are encoded using words of length $s(n)$ works in $\text{NSPACE}(s(n))$ (with the addition of the binary encoding of T for the second item, if it greater than $s(n)$). We then observe that $\text{NSPACE}(s(n)) = \text{SPACE}(s^2(n))$ from the Savitch theorem. \square

NB 2.3.5 (Attractor point of view)

We presented the above statement in terms of computing the flow $\Phi(\mathbf{x}, T)$. This could alternatively be interpreted in terms of attractors. Indeed, when the hypothesis of Proposition 2.3.4 holds, then the dynamics are captured by a graph. In the long run, in particular, if T is greater than the number of vertices, any trajectory loops (i.e. reaches an attractor). The above statement could then also be read as the fact that such an attractor is then polynomial space computable.

2.4 Implicit complexity

Having “simple” characterisations of computability and complexity classes is useful for various fundamental and applied science fields. We are interested here in “algebraic” characterisations of those classes: we want to define them as the smallest set

$$[f_1, \dots, f_k; o_1, \dots, o_l]$$

where the $f_i, i \in \llbracket 1, k \rrbracket$ are functions, closed under the operators $o_j, j \in \llbracket 1, l \rrbracket$. For example, the set of computable functions over the integers is well-known to be:

$$[0, 1, \pi_k^i; \text{composition}, \text{minimisation}, \text{primitive recursion}]$$

Algebraic characterisations aims to give similar algebras for classes of complexity theory: a reference survey is [Clo98, CK02]. The main benefit is to avoid the use of the framework of Turing machines, which is rather heavy and not necessarily well-known outside fundamental computer science.

When dealing with complexity classes, the problem of resources arises: somehow, we have to bound the time and the space. The first idea is to give an explicit bound on the functions we can construct in the algebra. Several characterisations for **P**TIME over the integers were proposed.

The first is due to Cobham in [Cob62], but relies on explicit bounds and a bounded recursion scheme for functions in $\mathbb{N}^{\mathbb{N}}$:

Definition 2.4.1 ([Cob62])

A function $f : \mathbb{N}^2 \rightarrow \mathbb{N}$ is defined by a bounded recursion on notations, aka BRN, from functions g, h_0, h_1 and k iff:

$$\begin{cases} f(0, \mathbf{y}) = g(\mathbf{y}) \\ f(\mathbf{s}_0(x), \mathbf{y}) = h_0(x, \mathbf{y}, f(x, \mathbf{y})) \text{ for } x \neq 0 \\ f(\mathbf{s}_1(x), \mathbf{y}) = h_1(x, \mathbf{y}, f(x, \mathbf{y})) \\ f(x, \mathbf{y}) \leq k(x, \mathbf{y}) \end{cases}$$

with $\mathbf{s}_0, \mathbf{s}_1 : \mathbb{N} \rightarrow \mathbb{N}$ such that $\mathbf{s}_0(x) = 2 \cdot x$ and $\mathbf{s}_1(x) = 2 \cdot x + 1$.

Leading to the following algebraic characterisation of **F**PTIME over the integers:

Theorem 2.4.2 ([Cob62])

Consider

$$F_P = [\mathbf{0}, \pi_i^k, \mathbf{s}_0, \mathbf{s}_1, \sharp; \text{composition}, \text{BRN}]$$

with \sharp the “smash funtion” defined by $\sharp(x, y) = 2^{\ell(x) \times \ell(y)}$. We have:

$$F_P = \mathbf{FPTIME} \cap \mathbb{N}^{\mathbb{N}}$$

The idea behind the proof of this statement is that if we have a function like in Definition 2.4.1, if $\ell(k(x, \mathbf{y}))$ is polynomial in $\ell(x) + \ell(y)$, so is $\ell(f(x, \mathbf{y}))$. Hence, the inner terms do not grow too fast. Also, the definition of the successor functions ($\ell(\mathbf{s}_1(x)) = \ell(\mathbf{s}_0(x)) = \ell(x) + 1$) guarantees that the number of induction steps is in $O(\ell(x))$.

It is possible to extend this characterisation to polynomial space, by considering a simple bounded recursion scheme (BR) instead of bounded recursion on notations:

Theorem 2.4.3 ([Tho72])

Consider

$$F_Q = [\mathbf{0}, \pi_i^k, s_0, s_1, \sharp; \text{composition}, \text{BR}]$$

We have:

$$F_Q = \mathbf{FPSPACE} \cap \mathbb{N}^{\mathbb{N}}$$

The problem with these characterisations is that they involve the exhibition of some artificial bound functions k .

Here, we want to avoid it and work directly on *how* we write the functions the algebras can compute: in other words, we want to handle an *implicit* bound or a notational one.

Similarly, Bellantoni and Cook in [BC92] proposed a characterisation of polynomial time based on distinguishing *normal* arguments from *safe* arguments (separated by a semicolon in the input of the functions). We say an argument is *safe* if it can be turned into an impredicative value, meaning a value on which we can do inductions. A normal argument can become safe but the contrary is not possible.

Firstly, we give the definitions of “safe composition” and of “safe recursion”:

Definition 2.4.4 (Safe Composition [BC92])

The function f is defined by safe composition (SCOMP) from $g, u_1, \dots, u_n, v_1, \dots, v_m$ iff

$$f(x; a) = g(u_1(x;), \dots, u_n(x;); v_1(x; a), \dots, v_m(x; a))$$

where the arguments on the left side of the semicolon are normal and the ones on the right side are safe.

Definition 2.4.5 (Safe Recursion [BC92])

The function f is defined by recursion on notation (SRN) from the functions g, h_0, h_1 iff:

$$\begin{cases} f(0, y; a) = g(y; a) \\ f(s_0(x), y; a) = h_0(x, y; a, f(x, y; a)) \\ f(s_1(x), y; a) = h_1(x, y; a, f(x, y; a)) \end{cases} \text{ for } x \neq 0$$

In other words, the predicative parameter is necessarily safe.

Leading to the following algebra:

Theorem 2.4.6 ([BC92])

We consider

$$\mathcal{B} = \left[0, \pi_i^k, s_0, s_1, pred, if; \text{SCOMP}, \text{SRN} \right],$$

with $pred$ the binary predecessor function. We have:

$$\mathbf{FPTIME} \cap \mathbb{N}^{\mathbb{N}} = \mathcal{B} \cap \mathbf{NORMAL} \cap \mathbb{N}^{\mathbb{N}}$$

where $\mathcal{B} \cap \mathbf{NORMAL}$ means the arguments of the functions in \mathcal{B} are all normal.

Recently, Bournez and Durand in [BD23] suggested an algebra using the so-called "linear-length" discrete ODEs (see Definition 2.5.10). Instead of having explicit bounds, the linearity of these discrete ODEs guarantees polynomial time complexity (see Lemma 2.5.12). The characterisations of polynomial space, which we extensively study in Chapter 5, involve another type of linear ODEs: see Definition 5.1.1 for the discrete ODEs.

There also have been some attempts using continuous ODEs [BCGSH07], or the so-called \mathbb{R} -recursive functions [BP21]. In Chapter 5, we give a characterisation of polynomial space in Section 5.2 with a scheme of continuous ODEs (Definition 5.2.3). The main ideas are first to allow a computation by dichotomy and to impose a polynomial precision computation at each step of the dichotomy (Definition 5.2.3).

2.5 Using Discrete ODEs in complexity

We recall in this section some results and concepts from [BD19, BD23]. We provide some of the proofs when they are not in these articles.

The most up-to-date survey is [BP21]. Dealing with discrete ODEs is really different, as most of the constructions heavily rely on some closure properties of continuous ODEs not true for discrete ODEs, in particular because there is no chain rule formula for discrete derivation.

2.5.1 Length Ordinary Differential Equations

For that discrete framework, we define the notion of length. It is adapted from Definition 1.2.8:

Definition 2.5.1 (Length of a (discrete) curve [Pou15])

Let $d \in \mathbb{N}$, $I = \llbracket a, b \rrbracket$ be an interval. The length of $y : I \rightarrow \mathbb{R}^d$ over I is defined by:

$$length_I(y) = \sum_{i=a}^{b-1} |y'(x)| = \sum_{i=a}^{b-1} |y(x+1) - y(x)|$$

Thus, we introduce the following definition for length ODE, with $\ell(\cdot) \in \mathbb{N}^{\mathbb{N}}$ giving the length of the binary representation of the input:

Definition 2.5.2 (Length ODE [BD19, BD23])

A function \mathbf{f} is length ODE definable (from u , g and h) if it is a solution of

$$\begin{aligned}\mathbf{f}(0, \mathbf{y}) &= \mathbf{g}(\mathbf{y}) \\ \frac{\delta \mathbf{f}(x, \mathbf{y})}{\delta \ell} &= \mathbf{u}(\mathbf{f}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y})\end{aligned}$$

where a synonym of the previous equation is $\mathbf{f}(x+1, \mathbf{y}) = \mathbf{f}(x, \mathbf{y}) + (\ell(x+1) - \ell(x)) \cdot \mathbf{u}(\mathbf{f}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y})$.

NB 2.5.3

This concept, introduced in [BD19, BD23], is motivated by the fact that a length ODE is similar to the classical formula for classical continuous ODEs:

$$\frac{\delta f(x, \mathbf{y})}{\delta x} = \frac{\delta \mathcal{L}(x, \mathbf{y})}{\delta x} \cdot \frac{\delta f(x, \mathbf{y})}{\delta \mathcal{L}(x, \mathbf{y})},$$

and hence this is similar to a change of variable. Consequently, a linear length-ODE is a linear ODE over variable t , once the change of variable $t = \ell(x)$ is done.

Example 2.5.4

Consider the following length ODE:

$$\begin{cases} f(0) = 2 \\ \frac{\delta f}{\delta \ell}(x) = f(x) \cdot f(x) - f(x) \end{cases}$$

Its unique solution is $f(x) = 2^{2^{\ell(x)}}$.

Indeed, we rewrite the ODE, this way:

$$f(x+1) - f(x) = (\ell(x+1) - \ell(x)) (f(x)f(x) - f(x))$$

We fix $f(x) = F(z)$, with $z = \ell(x)$. Then, $F(z+1) = 2^{2^z+2^z} = F(z)F(z)$. Thus, if $\ell(x+1) = \ell(x) + 1$, $F(z+1) = F(z) + (z+1-z)(F(z)F(z) - F(z))$. If $\ell(x+1) = \ell(x)$, we have $f(x+1) = f(x)$ and the equation still holds. So f is a solution.

Let us show it is unique. Let g be a solution of this length ODE, we show by induction it is equal to f :

- At $n = 0$, we necessarily have $g(0) = 2 = f(0)$;
- Assuming $f(n) = g(n)$, we deduce

$$\begin{aligned}g(n+1) &= g(n) + (\ell(n+1) - \ell(n))(g(n)g(n) - g(n)) && \text{by definition} \\ &= f(n) + (\ell(n+1) - \ell(n))(f(n)f(n) - f(n)) && \text{by induction hypothesis} \\ &= f(n+1) && \text{by definition}\end{aligned}$$

Thus, the solution is unique.

Intuitively, the length derivative corresponds to a change of variable: instead of doing a derivative with a fixed step of size one, we do a derivative where the size of the steps

changes as the successive powers of 2. Thus, there is a variation only when $\ell(x+1) - \ell(x) \neq 0$. We will often define some functions by defining their value in 2^0 , and then 2^{n+1} from their value in 2^n as it corresponds to some discrete ODE after this change of variable. Namely:

Lemma 2.5.5 ([BD19, BD23])

Let $f : \mathbb{N}^{p+1} \rightarrow \mathbb{Z}^d$, $\mathcal{L} : \mathbb{N}^{p+1} \rightarrow \mathbb{Z}$ be some functions. Assume that the equation of Definition 2.5.2 holds considering $\mathcal{L}(x, \mathbf{y}) = \ell(x)$. Then $\mathbf{f}(x, \mathbf{y})$ is also given by $\mathbf{f}(x, \mathbf{y}) = \mathbf{F}(\ell(x), \mathbf{y})$ where \mathbf{F} is the solution of:

$$\begin{aligned} \mathbf{F}(1, \mathbf{y}) &= \mathbf{f}(0, \mathbf{y}), \\ \frac{\delta \mathbf{F}(t, \mathbf{y})}{\delta t} &= \mathbf{h}(\mathbf{F}(t, \mathbf{y}), 2^t - 1, \mathbf{y}). \end{aligned}$$

We will also implicitly use in what follows that some basic functions such as $n \mapsto 2^n$ can be easily defined and that we can produce $2^{T(\ell(\omega))}$ for any polynomial T : see [BD19, BD23] where this is explicitly proved and detailed.

We notice that without more constraints on the length ODEs, we can compute an exponential function, as in Example 2.5.4. Since we use them to characterise *polynomial* complexity classes, we must prevent non-polynomial functions from being produced. For that, we must define *linear* length ODEs in Definition 2.5.10.

Linearity

We now review the work of Bournez and Durand in [BD19, BD23]. However, we slightly extend the concept of sg-polynomial expression from [BD19, BD23] to allow expressions with $\overline{\text{cond}}$ instead of sg ($\text{sg}(x) = 0$ when $x < 0$ and 1 otherwise). We define the discrete sigmoid $\overline{\text{cond}}(x) : \mathbb{R} \rightarrow \mathbb{R}$ as the continuous piecewise affine function valuing 1 for $x > \frac{3}{4}$ and 0 for $x < \frac{1}{4}$, and affine between $\frac{1}{4}$ and $\frac{3}{4}$.

Definition 2.5.6 ($\overline{\text{cond}}$ -polynomial expression)

A $\overline{\text{cond}}$ -polynomial expression $P(x_1, \dots, x_h)$ is an expression built-on $+, -, \times$ (also often write \cdot also for \times) and $\overline{\text{cond}}$ functions over a set of variables $V = \{x_1, \dots, x_h\}$ and integer constants. The degree $\deg(x, P)$ of a term $x \in V$ in P is defined inductively as follows:

- $\deg(x, x) = 1$ and for $x' \in V \cup \mathbb{Z}$ such that $x' \neq x$, $\deg(x, x') = 0$;
- $\deg(x, P + Q) = \max\{\deg(x, P), \deg(x, Q)\}$;
- $\deg(x, P \times Q) = \deg(x, P) + \deg(x, Q)$;
- $\deg(x, \overline{\text{cond}}(P)) = 0$.

A $\overline{\text{cond}}$ -polynomial expression P is essentially constant in x if $\deg(x, P) = 0$.

In other words, compared to the classical notion of degree in polynomial expression,

all subterms that are within the scope of a $\overline{\text{cond}}$ function contributes 0 to the degree. A vectorial function (resp. a matrix or a vector) is said to be a $\overline{\text{cond}}$ -polynomial expression if all its coordinates (resp. coefficients) are. It is said to be *essentially constant* if all its coefficients are.

Definition 2.5.7 (Essential Linearity [BD19, BD23])

A $\overline{\text{cond}}$ -polynomial expression $\mathbf{g}(\mathbf{f}(x, \mathbf{y}), x, \mathbf{y})$ is *essentially linear* in $\mathbf{f}(x, \mathbf{y})$ if it is of the form:

$$\mathbf{g}(\mathbf{f}(x, \mathbf{y}), x, \mathbf{y}) = \mathbf{A}[\mathbf{f}(x, \mathbf{y}), x, \mathbf{y}] \cdot \mathbf{f}(x, \mathbf{y}) + \mathbf{B}[\mathbf{f}(x, \mathbf{y}), x, \mathbf{y}]$$

where \mathbf{A} and \mathbf{B} are $\overline{\text{cond}}$ -polynomial expressions essentially constant in $\mathbf{f}(x, \mathbf{y})$.

For example,

- the expression $P(x, y, z) = x \cdot \overline{\text{cond}}((x^2 - z) \cdot y) + y^3$ is essentially linear in x , essentially constant in z and not linear in y .
- The expression $P(x, 2^{\ell(y)}, z) = \overline{\text{cond}}(x^2 - z) \cdot z^2 + 2^{\ell(y)}$ is essentially constant in x , essentially linear in $2^{\ell(y)}$ (not essentially constant) and not essentially linear in z .
- The expression $z + (1 - \overline{\text{cond}}(x)) \cdot (1 - \overline{\text{cond}}(-x)) \cdot (y - z)$ is essentially constant in x and linear in y and z .
- The following matrix is essentially linear in z and y and constant in x :

$$A(x, y, z) = \begin{pmatrix} \overline{\text{cond}}(x - y) & \overline{\text{cond}}(x) \cdot y \\ \overline{\text{cond}}(z^5 - x^3) & z \end{pmatrix}$$

In Chapter 3, we use a particular type of discrete ODE, namely linear length ODE.

Lemma 2.5.8 (Solution of linear ODE)

For matrices \mathbf{A} and vectors \mathbf{B} and \mathbf{G} , the solution of the equation $\mathbf{f}'(x, \mathbf{y}) = \mathbf{A}(\mathbf{f}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y}) \cdot \mathbf{f}(x, \mathbf{y}) + \mathbf{B}(\mathbf{f}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y})$ with initial conditions $\mathbf{f}(0, \mathbf{y}) = \mathbf{G}(\mathbf{y})$ is

$$\begin{aligned} \mathbf{f}(x, \mathbf{y}) &= \left(2^{\int_0^x \mathbf{A}(\mathbf{f}(t, \mathbf{y}), \mathbf{h}(t, \mathbf{y}), t, \mathbf{y}) \delta t} \right) \cdot \mathbf{G}(\mathbf{y}) \\ &\quad + \int_0^x \left(2^{\int_{u+1}^x \mathbf{A}(\mathbf{f}(t, \mathbf{y}), \mathbf{h}(t, \mathbf{y}), t, \mathbf{y}) \delta t} \right) \cdot \mathbf{B}(\mathbf{f}(u, \mathbf{y}), \mathbf{h}(u, \mathbf{y}), u, \mathbf{y}) \delta u. \end{aligned}$$

Proof. Denoting the right-hand side by $\mathbf{rhs}(x, \mathbf{y})$, we have

$$\begin{aligned}
\overline{\mathbf{rhs}}'(x, \mathbf{y}) &= \mathbf{A}(\mathbf{f}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y}) \cdot \left(\bar{2}^{\int_0^x \mathbf{A}(\mathbf{f}(t, \mathbf{y}), \mathbf{h}(t, \mathbf{y}), t, \mathbf{y}) \delta t} \right) \cdot \mathbf{G}(\mathbf{y}) \\
&\quad + \int_0^x \left(\bar{2}^{\int_{u+1}^x \mathbf{A}(\mathbf{f}(t, \mathbf{y}), \mathbf{h}(t, \mathbf{y}), t, \mathbf{y}) \delta t} \right)' \cdot \mathbf{B}(\mathbf{f}(u, \mathbf{y}), \mathbf{h}(u, \mathbf{y}), u, \mathbf{y}) \delta u \\
&\quad + \left(\bar{2}^{\int_{x+1}^{x+1} \mathbf{A}(\mathbf{f}(t, \mathbf{y}), \mathbf{h}(t, \mathbf{y}), t, \mathbf{y}) \delta t} \right) \cdot \mathbf{B}(\mathbf{f}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y}) \\
&= \mathbf{A}(\mathbf{f}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y}) \cdot \left(\bar{2}^{\int_0^x \mathbf{A}(\mathbf{f}(t, \mathbf{y}), \mathbf{h}(t, \mathbf{y}), t, \mathbf{y}) \delta t} \right) \cdot \mathbf{G}(\mathbf{y}) \\
&\quad + \mathbf{A}(\mathbf{f}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y}) \cdot \\
&\quad \int_0^x \left(\bar{2}^{\int_{u+1}^x \mathbf{A}(\mathbf{f}(t, \mathbf{y}), \mathbf{h}(t, \mathbf{y}), t, \mathbf{y}) \delta t} \right) \mathbf{B}(\mathbf{f}(u, \mathbf{y}), \mathbf{h}(u, \mathbf{y}), u, \mathbf{y}) \delta u \\
&\quad + \mathbf{B}(\mathbf{f}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y}) \\
&= \mathbf{A}(\mathbf{f}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y}) \cdot \mathbf{rhs}(x, \mathbf{y}) + \mathbf{B}(\mathbf{f}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y})
\end{aligned}$$

where we have used linearity of derivation and definition of falling exponential for the first term, and derivation of an integral (Lemma 1.2.14) providing the other terms to get the first equality, and then the definition of falling exponential. This proves the property by unicity of solutions of a discrete ODE, observing that $\overline{\mathbf{rhs}}(0, \mathbf{y}) = \mathbf{G}(\mathbf{y})$. \square

We write also 1 for the identity.

NB 2.5.9

Notice that this can be rewritten as $\mathbf{f}(x, \mathbf{y}) = \sum_{u=-1}^{x-1} \left(\prod_{t=u+1}^{x-1} (1 + \mathbf{A}(\mathbf{f}(t, \mathbf{y}), \mathbf{h}(t, \mathbf{y}), t, \mathbf{y})) \right) \cdot \mathbf{B}(\mathbf{f}(u, \mathbf{y}), \mathbf{h}(u, \mathbf{y}), u, \mathbf{y})$, with the (not so usual) conventions that for any function $\kappa(\cdot)$, $\prod_x^{x-1} \kappa(x) = 1$ and $\mathbf{B}(-1, \mathbf{y}) = \mathbf{G}(\mathbf{y})$. Such equivalent expressions both have a clear computational content. They can be interpreted as an algorithm unrolling the computation of $\mathbf{f}(x+1, \mathbf{y})$ from the computation of $\mathbf{f}(x, \mathbf{y}), \mathbf{f}(x-1, \mathbf{y}), \dots, \mathbf{f}(0, \mathbf{y})$.

Thus, we can properly define *linear length ODEs*:

Definition 2.5.10 (Linear length ODE, adapted from [BD19, BD23])

A function \mathbf{f} is linear \mathcal{L} -ODE definable (from \mathbf{u} , \mathbf{g} and \mathbf{h}) if it corresponds to the solution of:

$$\begin{aligned}
\mathbf{f}(0, \mathbf{y}) &= \mathbf{g}(\mathbf{y}) \\
\frac{\delta \mathbf{f}(x, \mathbf{y})}{\delta \ell} &= \mathbf{u}(\mathbf{f}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y})
\end{aligned}$$

where \mathbf{u} is essentially linear in $\mathbf{f}(x, \mathbf{y})$.

2.5.2 Implicit definition of polynomial time

Stability property for discrete sigmoid

We need to ensure that the computation remains in polynomial time, which requires to prove that the class of polynomial time computable functions is preserved by the linear

length ODE schema, so we start by it. We write $\|\cdot\|$ for the sup norm of ceiling integer part (of absolute value): given some matrix $\mathbf{A} = (A_{i,j})_{1 \leq i \leq n, 1 \leq j \leq m}$, $\|\mathbf{A}\| = \max_{i,j} \lceil |A_{i,j}| \rceil$. In particular, given a vector \mathbf{x} , it can be seen as a matrix with $m = 1$, and $\|\mathbf{x}\|$ is the sup norm of the ceiling integer part of its components.

We need to prove the following result:

Lemma 2.5.11 (Essential linearity stability for $\overline{\text{cond}}$)

Consider the ODE

$$\mathbf{F}'(x, \mathbf{y}) = \mathbf{A}(\mathbf{F}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y}) \cdot \mathbf{F}(x, \mathbf{y}) + \mathbf{B}(\mathbf{F}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y}). \quad (2.2)$$

Assume:

- The initial condition $\mathbf{G}(\mathbf{y}) \stackrel{\text{def}}{=} \mathbf{F}(0, \mathbf{y})$ is polynomial time computable and $\mathbf{h}(x, \mathbf{y})$ are polynomial time computable with respect to the value of x .
- $\mathbf{A}(\mathbf{F}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y})$ and $\mathbf{B}(\mathbf{F}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y})$ are $\overline{\text{cond}}$ -polynomial expressions essentially constant in $\mathbf{F}(x, \mathbf{y})$.

Then, there exists a polynomial p such that $\ell \left(\|\mathbf{F}(x, \mathbf{y})\| \right) \leq p \left(x, \ell \left(\|\mathbf{y}\| \right) \right)$ and $\mathbf{F}(x, \mathbf{y})$ is polynomial time computable with respect to the value of x .

We give the proof of the previous lemma later, as we give a slight generalisation of it in Lemma 2.5.14.

The previous statements lead to the following:

Lemma 2.5.12 (Intrinsic complexity of linear \mathcal{L} -ODE)

Let \mathbf{f} be a solution of the linear \mathcal{L} -ODE

$$\begin{aligned} \mathbf{f}(0, \mathbf{y}) &= \mathbf{g}(\mathbf{y}), \\ \frac{\delta \mathbf{f}(x, \mathbf{y})}{\delta \ell} &= \mathbf{u}(\mathbf{f}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y}) \end{aligned}$$

where \mathbf{u} is essentially linear in $\mathbf{f}(x, \mathbf{y})$. Assume that the functions $\mathbf{u}, \mathbf{g}, \mathbf{h}$ are computable in polynomial time. Then, \mathbf{f} is computable in polynomial time.

Proof. From Lemma 2.5.5, $\mathbf{f}(x, \mathbf{y})$ can also be given by $\mathbf{f}(x, \mathbf{y}) = \mathbf{F}(\ell(x), \mathbf{y})$ where \mathbf{F} is the solution of the initial value problem

$$\begin{aligned} \mathbf{F}(1, \mathbf{y}) &= \mathbf{g}(\mathbf{y}), \\ \frac{\delta \mathbf{F}(t, \mathbf{y})}{\delta t} &= \mathbf{u}(\mathbf{F}(t, \mathbf{y}), 2^t - 1, \mathbf{y}). \end{aligned}$$

Functions \mathbf{u} are $\overline{\text{cond}}$ -polynomial expressions that are essentially linear in $\mathbf{f}(x, \mathbf{y})$. So there exist matrices \mathbf{A}, \mathbf{B} that are essentially constants in $\mathbf{f}(t, \mathbf{y})$ such that

$$\frac{\delta \mathbf{f}(x, \mathbf{y})}{\delta \ell} = \mathbf{A}(\mathbf{f}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y}) \cdot \mathbf{f}(x, \mathbf{y}) + \mathbf{B}(\mathbf{f}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y}).$$

In other words,

$$\mathbf{F}'(t, \mathbf{y}) = \overline{\mathbf{A}}(\mathbf{F}(t, \mathbf{y}), t, \mathbf{y}) \cdot \mathbf{F}(t, \mathbf{y}) + \overline{\mathbf{B}}(\mathbf{F}(t, \mathbf{y}), t, \mathbf{y}),$$

setting

$$\begin{aligned} \overline{\mathbf{A}}(\mathbf{F}(t, \mathbf{y}), t, \mathbf{y}) &= \mathbf{A}(\overline{\mathbf{F}}(t, \mathbf{y}), \mathbf{h}(2^t - 1, \mathbf{y}), 2^t - 1, \mathbf{y}) \\ \overline{\mathbf{B}}(\mathbf{F}(t, \mathbf{y}), t, \mathbf{y}) &= \mathbf{B}(\overline{\mathbf{F}}(t, \mathbf{y}), \mathbf{h}(2^t - 1, \mathbf{y}), 2^t - 1, \mathbf{y}) \end{aligned}$$

The corresponding matrix $\overline{\mathbf{A}}$ and vector $\overline{\mathbf{B}}$ are essentially constant in $\mathbf{F}(t, \mathbf{y})$. Also, functions \mathbf{g}, \mathbf{h} are computable in polynomial time, more precisely polynomial in $\ell(x)$, hence in t , and $\ell(y)$. Given t , obtaining $2^t - 1$ is immediate. This guarantees that all hypotheses of Lemma 2.5.12 are true. We can then conclude remarking, again, that $t = \ell(x)$. \square

This proves Lemma 2.5.12.

Stability property for \tanh -polynomial expressions

NB 2.5.13

The previous statements were initially stated in [BD19, BD23], using the sign function $\text{sg}(\cdot)$ ($\text{sg}(x) = 0$ if $x \leq 0$ and 1 otherwise) instead of $\overline{\text{cond}}(\cdot)$. We adapted their result to the framework we will use.

As we will see in Chapter 3, we will use a hyperbolic tangent instead of the function $\overline{\text{cond}}$ -polynomial (and hence also instead of $\text{sg}(\cdot)$ of [BD19, BD23]). The hyperbolic tangent has the advantage of being analytic. All the previous results and proofs remain true in this setting.

We will need to consider \tanh -polynomial expression. This is not exactly as in Definition 2.5.6: it uses analogous definitions and properties. The concepts are mostly the same, by replacing $\overline{\text{cond}}(\cdot)$ by \tanh or $\text{sg}(\cdot)$.

In particular, the definitions of *essentially constant* and *essentially linear* are the same, replacing $\overline{\text{cond}}$ by \tanh or even $\text{sg}(\cdot)$: a vectorial function (resp. a matrix or a vector) is said to be a \tanh -polynomial expression if all its coordinates (resp. coefficients) are, and *essentially constant* if all its coefficients are.

As for the proof of Lemma 2.5.12, we first need to prove the following lemma. The lemma and the proof are essentially the same as for Lemma 2.5.11.

Lemma 2.5.14 (Essential linear stability for tanh)

Consider the ODE

$$\mathbf{F}'(x, \mathbf{y}) = \mathbf{A}(\mathbf{F}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y}) \cdot \mathbf{F}(x, \mathbf{y}) + \mathbf{B}(\mathbf{F}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y}). \quad (2.3)$$

Assume:

- The initial condition $\mathbf{G}(\mathbf{y}) = \mathbf{F}(0, \mathbf{y})$ is polynomial time computable, and $\mathbf{h}(x, \mathbf{y})$ are polynomial time computable with respect to the value of x .
- $\mathbf{A}(\mathbf{F}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y})$ and $\mathbf{B}(\mathbf{F}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y})$ are \tanh -polynomial expressions essentially constant in $\mathbf{F}(x, \mathbf{y})$.

Then, there exists a polynomial p such that $\ell \left(\left\| \mathbf{F}(x, \mathbf{y}) \right\| \right) \leq p \left(x, \ell \left(\left\| \mathbf{y} \right\| \right) \right)$ and $\mathbf{F}(x, \mathbf{y})$ is polynomial time computable with respect to the value of x .

Proof. The solution of ordinary differential equation (2.3) can be put in some explicit form (this follows from [BD19, BD23]).

$$\mathbf{F}(x, \mathbf{y}) = \sum_{u=-1}^{x-1} \left(\prod_{t=u+1}^{x-1} (1 + \mathbf{A}(\mathbf{F}(t, \mathbf{y}), \mathbf{h}(t, \mathbf{y}), t, \mathbf{y})) \right) \cdot \mathbf{B}(\mathbf{F}(u, \mathbf{y}), \mathbf{h}(u, \mathbf{y}), u, \mathbf{y}). \quad (2.4)$$

with the conventions that $\prod_{x=1}^{x-1} \kappa(x) = 1$ and $\mathbf{B}(\cdot, -1, \mathbf{y}) = \mathbf{G}(\mathbf{y})$.

This formula allows us to evaluate $\mathbf{F}(x, \mathbf{y})$, using a dynamic programming approach, from the quantities $\mathbf{y}, u, \mathbf{h}(u, \mathbf{y})$, for $1 \leq u < x$, in several arithmetic steps that is polynomial in x . Indeed: for any $-1 \leq u \leq x$, $\mathbf{A}(\mathbf{F}(u, \mathbf{y}), \mathbf{h}(u, \mathbf{y}), u, \mathbf{y})$ and $\mathbf{B}(\mathbf{F}(u, \mathbf{y}), \mathbf{h}(u, \mathbf{y}), u, \mathbf{y})$ are matrices whose coefficients are \tanh -polynomial. Their coefficients involve finitely many arithmetic operations or \tanh operations from their inputs. Once this is done, computing $\mathbf{F}(x, \mathbf{y})$ requires polynomially in x many arithmetic operations: once the values for \mathbf{A} and \mathbf{B} are known, we have to sum up $x + 1$ terms, each of them involving at most $x - 1$ multiplications.

We need to take care not only of the arithmetic complexity, which is polynomial in x , but also of the bit complexity. We start by discussing the bit complexity of the integer parts. Each of the quantities $\mathbf{y}, u, \mathbf{h}(u, \mathbf{y})$, for $1 \leq u < x$, are computable in polynomial time, so their integer parts have a bit complexity remaining polynomial in x and $\ell \left(\left\| \mathbf{y} \right\| \right)$.

The bit complexity of a sum, product, etc is polynomial in the size of its arguments, it is sufficient to show that the growth rate of the function $\mathbf{F}(x, \mathbf{y})$ can be polynomially dominated. For this, recall that, for any $-1 \leq u \leq x$, coefficients of $\mathbf{A}(\mathbf{F}(u, \mathbf{y}), \mathbf{h}(u, \mathbf{y}), u, \mathbf{y})$ and $\mathbf{B}(\mathbf{F}(u, \mathbf{y}), \mathbf{h}(u, \mathbf{y}), u, \mathbf{y})$ are essentially constant in $\mathbf{F}(u, \mathbf{y})$. Hence, the size of the integer part of these coefficients does not depend on $\ell(\mathbf{F}(u, \mathbf{y}))$. Since, in addition, \mathbf{h} is computable in polynomial time in x and $\ell(\mathbf{y})$, there exists a polynomial p_M such that:

$$\max \left(\ell \left(\left\| \mathbf{A}(\mathbf{F}(u, \mathbf{y}), \mathbf{h}(u, \mathbf{y}), u, \mathbf{y}) \right\| \right), \ell \left(\left\| \mathbf{B}(\mathbf{F}(u, \mathbf{y}), \mathbf{h}(u, \mathbf{y}), u, \mathbf{y}) \right\| \right) \right) \leq p_M \left(u, \ell \left(\left\| \mathbf{y} \right\| \right) \right) \quad (2.5)$$

It then holds that,

$$\ell \left(\left\| \mathbf{F}(x+1, \mathbf{y}) \right\| \right) \leq p_M(x, \ell \left(\left\| \mathbf{y} \right\| \right)) + \ell \left(\left\| \mathbf{F}(x, \mathbf{y}) \right\| \right) + 2$$

It follows from an easy induction that we must have

$$\ell \left(\left\| \mathbf{F}(x, \mathbf{y}) \right\| \right) \leq \ell \left(\left\| G(\mathbf{y}) \right\| \right) + x \cdot (p_M(x, \ell \left(\left\| \mathbf{y} \right\| \right)) + 2),$$

which gives the desired bound on the length of the integer part of the values for function \mathbf{F} , after observing that, since \mathbf{G} is polynomial time computable, necessarily $\ell \left(\left\| G(\mathbf{y}) \right\| \right)$ remains polynomial in $\ell \left(\left\| \mathbf{y} \right\| \right)$.

We now take care of the bit complexity of involved quantities to prove that $\mathbf{F}(x, \mathbf{y})$ is indeed polynomial time computable with respect to the value of x . Given \mathbf{y} , we can determine some integer Y such that $\mathbf{y} \in [2^{-Y}, 2^Y]$. We just need to prove that given n , we can provide some dyadic \mathbf{z}_x approximating $\mathbf{F}(x, \mathbf{y})$ at precision n , i.e. with $\|\mathbf{F}(x, \mathbf{y}) - \mathbf{z}_x\| \leq 2^{-n}$, in a time polynomial in x and Y .

Basically, this follows from the possibility of evaluating $\mathbf{F}(x, \mathbf{y})$ using formula (2.4). This latter formula is made of a sum of $x+1$ terms, that we can call $\mathbf{T}_0, \mathbf{T}_1, \dots, \mathbf{T}_x$, each of them \mathbf{T}_i corresponding to a product of k matrices (or vectors) $\mathbf{T}_i = \mathbf{C}_1 \mathbf{C}_2 \dots \mathbf{C}_{k(i)}$, with $k(i) \leq x+1$, where each \mathbf{C}_j is either some $\mathbf{B}(\mathbf{F}(u, \mathbf{y}), \mathbf{h}(u, \mathbf{y}), u, \mathbf{y})$ for some u or some $(1 + \mathbf{A}(\mathbf{F}(t, \mathbf{y}), \mathbf{h}(t, \mathbf{y}), t, \mathbf{y}))$ for some t .

To solve our problem, it is sufficient to be able to approximate the value of each \mathbf{T}_i by some dyadic \mathbf{d}_i with precision 2^{-n-m} , considering m with $x+1 \leq 2^m$. Indeed, taking $\mathbf{z}_x = \sum_{i=0}^x \mathbf{d}_i$ guarantees an error on the approximation of $\mathbf{F}(x, \mathbf{y})$ less than $(x+1)2^{-n-m} \leq 2^{-n}$.

So we focus on the problem of estimating $\mathbf{T}_i = \mathbf{C}_1 \mathbf{C}_2 \dots \mathbf{C}_{k(i)}$ with precision 2^{-n-m} . If we write $\tilde{\mathbf{C}}_j$ for some approximation of \mathbf{C}_j , we can write

$$\begin{aligned} \left\| \mathbf{C}_1 \mathbf{C}_2 \dots \mathbf{C}_{k(i)} - \tilde{\mathbf{C}}_1 \tilde{\mathbf{C}}_2 \dots \tilde{\mathbf{C}}_{k(i)} \right\| &\leq \left\| \mathbf{C}_1 \mathbf{C}_2 \dots \mathbf{C}_{k(i)-1} \left(\mathbf{C}_{k(i)} - \tilde{\mathbf{C}}_{k(i)} \right) \right\| \\ &+ \left\| \mathbf{C}_1 \mathbf{C}_2 \dots \mathbf{C}_{k(i)-2} \left(\mathbf{C}_{k(i)-1} - \tilde{\mathbf{C}}_{k(i)-1} \right) \tilde{\mathbf{C}}_{k(i)} \right\| \\ &+ \left\| \mathbf{C}_1 \mathbf{C}_2 \dots \mathbf{C}_{k(i)-3} \left(\mathbf{C}_{k(i)-2} - \tilde{\mathbf{C}}_{k(i)-2} \right) \tilde{\mathbf{C}}_{k(i)-1} \tilde{\mathbf{C}}_{k(i)} \right\| \\ &\vdots \\ &+ \left\| \left(\mathbf{C}_1 - \tilde{\mathbf{C}}_1 \right) \tilde{\mathbf{C}}_1 \tilde{\mathbf{C}}_2 \dots \tilde{\mathbf{C}}_{k(i)} \right\|. \end{aligned} \quad (2.6)$$

We just need then to compute some approximation $\tilde{\mathbf{C}}_{k(i)}$ of $\mathbf{C}_{k(i)}$, guaranteeing the first term in (2.6) to be less than 2^{-n-m-m} , and then choose some approximation $\tilde{\mathbf{C}}_{k(i)-1}$ of

$\mathbf{C}_{k(i)-1}$, guaranteeing the second term in (2.6) to be less than 2^{-n-m-m} , and so on. Doing so, we solve our problem, as $\tilde{\mathbf{C}}_1 \tilde{\mathbf{C}}_2 \dots \tilde{\mathbf{C}}_{k(i)}$ provides an estimation of \mathbf{T}_i , with an error less than $(x+1)2^{-n-m-m} \leq 2^{-n-m}$ by (2.6).

For the first term of (2.6), the point is that we know from a reasoning similar to previous computations (namely bounds such as (2.5)) that we have

$$\left\| \mathbf{C}_1 \mathbf{C}_2 \dots \mathbf{C}_{k(i)-1} \right\| \leq 2^{p_1(x,Y)}$$

for some polynomial p_1 . Consequently, it is sufficient to take

$$\left\| \mathbf{C}_{k(i)} - \tilde{\mathbf{C}}_{k(i)} \right\| \leq 2^{-n-2m-p_1(x,Y)}$$

to guarantee an error less than 2^{-n-m-m} .

A similar analysis applies for the second, and other terms, that involve finitely many terms. The whole approach takes a time that remains polynomial in x and Y . \square

Thus, Lemma 2.5.12 remains true in this framework.

NB 2.5.15

In the context of neural network models, there have been several characterisations of complexity classes over the discrete (see the monograph [Sie99] about the approach discussed above, but not only), as far as we know, the relation between formal neural networks with classes of computable analysis has never been established before.

If we do not restrict ourselves to neural network-related models, as in all these previous contexts, as far as we know, only a few papers have been devoted to characterisations of complexity and even computability classes in the sense of computable analysis. There have been some attempts using continuous ODEs [BCGSH07], that we already mentioned, or the so-called \mathbb{R} -recursive functions [BP21, BGH11]. For discrete schemata, we only know [Bra96] and [NRTY21], focusing on computability and not complexity.

We also mention the references [FGH14] discussing the complexity of operators in computable analysis. Notice that most of the classical complexity classes of the Blum Shub Smale model [BCSS98] have been characterised using the implicit complexity approach [BCdNM06, BCdNM05].

A characterisation of PTIME over the integers with Linear Length ODEs

A main result of [BD19, BD23] is the following ($\mathbb{L}\mathbb{D}\mathbb{L}$ stands for “Linear Derivation on Length”):

Theorem 2.5.16 ([BD19])

For discrete functions, we have $\mathbb{L}\mathbb{D}\mathbb{L} = \mathbf{FPTIME}$ where

$$\mathbb{L}\mathbb{D}\mathbb{L} = [\mathbf{0}, \mathbf{1}, \pi_i^k, \ell(x), +, -, \times, \text{sg}(x) ; \text{composition, linear length ODE}].$$

That is to say, $\mathbb{L}\mathbb{D}\mathbb{L}$ (and hence \mathbf{FPTIME} for discrete functions) is the smallest subset of functions, that contains the constant functions $\mathbf{0}$ and $\mathbf{1}$, the projections π_i^k , the

length function $\ell(x)$ (that maps an integer to the length of its binary representation), the addition function $x+y$, the subtraction function $x-y$, the multiplication function $x \times y$ (that we also often denote $x \cdot y$), the sign function $\text{sg}(x)$ and closed under composition (when defined) and linear length-ODE scheme: the linear length-ODE scheme basically (a formal definition is provided in Definition 2.5.10) corresponds to defining functions from linear ODEs with respect to derivation with respect to the length of the argument. We write $\mathbf{A}[\mathbf{f}(x, \mathbf{y}), x, \mathbf{y}]$ to mean that its coefficients (may) depend on $\mathbf{f}(x, \mathbf{y}), x, \mathbf{y}$, and similarly for \mathbf{B} . Essentially constant means that $\mathbf{f}(x, \mathbf{y})$ is actually either not appearing, or under the scope of some $\text{sg}(\cdot)$ function.

Theorem 2.5.17 ([BD19])

$$\mathbf{LDL} \cap \mathbb{N}^{\mathbb{N}} = \mathbf{FPTIME} \cap \mathbb{N}^{\mathbb{N}}.$$

This provides a characterisation of **FPTIME** for discrete functions that do not require to specify an explicit bound in the recursion, in contrast to Cobham’s work [Cob62], nor to assign a specific role or type to variables, in contrast to safe recursion or ramification [BC92, Lei95]. The characterisation happens to be very simple using only natural notions from the world of Ordinary Differential Equations.

Our purpose in Chapter 3 and in Chapter 5 is to extend this to more general classes of functions. In particular, we will characterise polynomial time functions from the reals to the reals. We consider here computability and complexity over the reals in the most classical sense, that is to say, computable analysis (see e.g. [Wei00]). Indeed, considering that $\mathbb{N} \subset \mathbb{R}$, most of the basic functions and operations in the above characterisation (this includes for example, $+$, $-$, \dots) have a clear meaning over the reals.

Link with Neural Networks

The point of view considered in the previous subsection is very close to the one of [Sie99]. Knowing whether Turing machines can be simulated by recurrent neural networks of finite size using the tanh activation function (and not the “exact” saturated linear function) is a long-standing open problem. Despite some attempts, such as the one described in [Sie99], up to our knowledge, there remain some of the statements not yet formally proved, or at least not generally accepted, in the existing proofs. Our statements in Chapter 3 deal with the tanh activation function, in some sense, but we avoid this open question by restricting it to finite space or time computations. Our proofs state this is possible if the space or the time of the machine is bounded, up to some controlled error.

2.6 A previous characterisation of **FPSPACE**

Very recently, a characterisation of **FPSPACE** has also been obtained: see [Goz22], for full details.

Theorem 2.6.1 ([Goz22])

A language $L \subseteq \Gamma^*$ belongs to **FPSPACE** iff there are ODEs $\mathbf{y}' = \mathbf{p}(\mathbf{y}, \mathbf{z})$ and $\mathbf{z}' = \mathbf{q}(\mathbf{y}, \mathbf{z})$, where \mathbf{p}, \mathbf{q} are vectors of polynomials, there is a polynomial u , (vector-valued) functions $\mathbf{r}, \mathbf{s} \in \text{GPVAL}$, a $\gamma_{\mathbb{N}}^k$ -bound ϕ , and $\varepsilon > 0$, $\tau \geq \alpha > 0$, $\alpha, \tau \in \mathbb{Q}$, such that, for all $w \in \Sigma^*$, one has that the solution (\mathbf{y}, \mathbf{z}) of the ODEs with the initial condition $\mathbf{y}(0) = \mathbf{r}(x)$ and $\mathbf{z}(0) = \mathbf{s}(\mathbf{y}(0)) = \mathbf{s} \circ \mathbf{r}(0)$, with $d(\gamma_{\mathbb{N}}^k(w), \mathbf{x}) < 1/4$, satisfies:

1. If $\bar{t}_1 > 0$ is such that $|\mathbf{y}_1(\bar{t}_1)| \geq 1$, then $|\mathbf{y}_1(t)| \geq 1$ for all $t \geq \bar{t}_1$ and $|\mathbf{y}_1(t)| \geq 3/2$ for all $t \geq \bar{t}_1 + 1$;
2. If $w \in L$ (respectively $w \notin L$) then there is some $\bar{t}_1 > 0$ such that $\mathbf{y}_1(\bar{t}_1) \geq 1$ (respectively ≤ -1);
3. $\|(\mathbf{y}(t), \mathbf{z}(t))\| \leq \phi \circ u(|w|)$, for all $t \geq 0$;
4. Suppose that $(\tilde{\mathbf{y}}, \tilde{\mathbf{z}})$ satisfies the ODE with $\mathbf{y}(0) = \mathbf{r}(x)$, $\mathbf{z}(0) = \mathbf{s}(\mathbf{y}(0)) = \mathbf{s} \circ \mathbf{r}(x)$, except possibly at time instants t_i , for $i = 1, 2, \dots$, satisfying:
 - (a) $t_i - t_{i-1} \geq \tau$, where $t_0 = 0$;
 - (b) $\|\tilde{\mathbf{y}}(t_i) - \lim_{t \rightarrow t_i^-} \tilde{\mathbf{y}}(t)\| \leq \varepsilon$ and $\tilde{\mathbf{z}}(t_i) = \mathbf{s}(\tilde{\mathbf{y}}(t_i))$;
 - (c) $\lim_{t \rightarrow t_i^-} \tilde{\mathbf{z}}_1(t) > 1$.

Then conditions 1,2,3,5 hold for $(\tilde{\mathbf{y}}, \tilde{\mathbf{z}})$;

5. For any $b > a \geq 0$ such that $|b - a| \geq \tau$, there is an interval $I = [c, d] \subseteq [a, b]$, with $|d - c| \geq \alpha$, such that $\mathbf{z}_1(t) \geq 3/2$ for all $t \in I$.

Condition $d(\gamma_{\mathbb{N}}^k(w), x) < \frac{1}{4}$, 4) and 5) impose that there exists some abstraction graph. Conditions 3) make it keep a polynomial log-size, and conditions 1) and 2) impose to the system to be eventually decidable. This guarantees **PSPACE** and allows a robust emulation of a TM. However, the space used by the ODEs cannot be “read” easily (as by a concept such as length in the previous theorem). We will propose a simpler characterisation in Chapter 5.

2.7 Programs and Turing machines

Turing machines are closely related to programs and proof theory. We can use Hoare’s logic to formalise this relation. We review some basic results about Hoare logic. To formally express the properties of programs, we fix a logic language *Assn* ([Win93]), the syntax rules are the following:

$$A ::= \mathbf{true} \mid \mathbf{false} \mid a_0 = a_1 \mid a_0 \leq a_1 \mid A_0 \wedge A_1 \mid A_0 \vee A_1 \mid \neg A \mid A_0 \Rightarrow A_1 \mid \forall i. A \mid \exists i. A$$

where a_0, a_1 are arithmetic expressions $a ::= n \mid X \mid i \mid a_0 + a_1 \mid a_0 - a_1 \mid a_0 \times a_1$, with $n \in \mathbb{N}$, i an integer variable and X a location.

We can use standard first-order logic for this purpose (with the signature of arithmetic). Any other choice of sufficiently expressive language, able to talk about words, configurations, and sequences, could be appropriate.

We review some statements of [Win93].

Theorem 2.7.1 (Gödel incompleteness theorem [Göd31])

Any sufficiently expressive consistent formal system F can be incomplete: there are statements of the language of F which can neither be proved nor disproved in F .

Consequently, from Gödel's incompleteness theorem, we get:

Theorem 2.7.2 (Uncompleteness see e.g. [Win93])

There is no effective proof system for $Assn$ such that the theorems coincide with the valid assertions of $Assn$.

Hence, there is no hope of getting a proof system with total correctness. Asking whether a program terminates or not is equivalent to asking about its *partial correctness*.

Proposition 2.7.3 (Uncompleteness see e.g. [Win93])

There is no effective proof system for partial correctness assertions such that its theorems are precisely valid partial correctness assertions.

However, relative *partial correctness* is possible. We can devise *relatively complete* proof systems, i.e. for any partial correctness assertion $\{A\}c\{B\}$, if $\{A\}c\{B\}$ is satisfiable then it is provable. A famous proof method for this is provided by Hoare's logic. In this logic, a partial correctness assertion has the form : $\{A\}c\{B\}$, where A and B are in $Assn$, and c is a program. Such an assertion states that starting from any initial configuration C satisfying A if the system evolves according to the program – or Turing machine – c , then it will either diverge or reach a configuration C' satisfying B . Formulas A and B are allowed to depend on some parameters, i.e. some interpretation I of some variables.

Theorem 2.7.4 (Relative completeness [Win93])

Hoare's logic is relatively complete.

NB 2.7.5

Observe that the Hoare triplet $\{A\}c\{false\}$ holds iff the program c diverges from any initial configuration C satisfying A .

Hoare's Logic

Hoare's logic includes the following:

1. Rule for while loops:

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

2. Rule of consequence:

$$\frac{\models^I (A \Rightarrow A') \quad \{A'\}c\{B'\} \quad \models^I (B' \Rightarrow B)}{\{A\}c\{B\}}$$

Hoare's logic provides a relatively complete method for proving non-termination. A Turing machine can be seen as a program corresponding to a while loop so we can focus on the case of a while loop: it will be the purpose of Section 4.6.

The rule for while loops, in the case of triplets of the form $\{A\}c\{false\}$, when c corresponds to a Turing machine gives a way to discuss termination of a TM. We will come back to this in Section 4.6.

Namely, given a triplet $\{A\}c\{B\}$, the idea will be to reason about the weakest precondition: this is a predicate $Q = wp(c, B)$ such that for any precondition A , $\{A\}c\{B\}$ if and only if $A \Rightarrow Q$. This is the least restrictive requirement needed to guarantee that B holds after A .

NB 2.7.6

The uniqueness of the weakest precondition follows easily from the definition: if both Q and Q' are weakest preconditions, then by the definition $\{Q'\}c\{B\}$ so $Q' \Rightarrow Q$ and $\{Q'\}c\{B\}$ so $Q \Rightarrow Q'$ and hence $Q = Q'$.

We will use those concepts in Section 4.6.

Chapter 3

Algebraic Characterisations of FPTIME with Discrete ODEs

W życiu niczego nie należy się bać, należy to tylko zrozumieć.

Maria Skłodowska-Curie

NB 3.0.1

*This chapter is based on articles published in MCU 2022 ([BB22], it received **Best Student Paper Award**) and MFCS 2023 ([BB23], it received **Best Paper Award**), co-authored with Olivier Bournez. We wrote journal versions for both articles, respectively accepted for publication in the *International Journal of Foundations of Computer Science* ([BB24a]) and in the *Journal of Logic and Analysis* ([BB25]).*

The class of functions from the integers to the integers computable in polynomial time has been characterised using discrete ordinary differential equations (ODE) in [BD23]. Their results are recalled in Chapter 2. In doing so, Olivier Bournez and Arnaud Durand pointed out the fundamental role of linear discrete and classical ODEs tools such as changes of variables to capture computability and complexity measures or for programming.

In this chapter,

- we extend the previous approach to characterise functions from the integers to the reals (we mean *real sequences*) computable in polynomial time in the sense of computable analysis. In particular, we provide a characterisation of such functions in terms of the smallest class of functions containing some basic functions, closed by composition, linear length ODEs, and an effective limit schema;
- then, we prove that functions over the reals computable in polynomial time can also be characterised using discrete ordinary differential equations (ODE);
- furthermore, we prove we do not need any artificial sign or test function for time complexity.

At a technical level, this is obtained by proving that Turing machines can be simulated with analytic discrete ordinary differential equations. These results give underlying well-understood notions of time complexity for programming with ODEs. They can be related

to [BGP16b, BGP17] and Chapter 2: Olivier Bournez, Daniel Graça and Amaury Pouly provide a characterisation of **PTIME** with *continuous* ODEs, establishing that time complexity corresponds to the length of the involved curve, i.e. the motto **time complexity = length**.

To do so, we naturally study an algebra of functions more general than discrete functions, that is to say over more general space than \mathbb{N} and \mathbb{Z} . This introduces some subtleties, and difficulties, that we discuss in this document, with our various concepts, definitions and statements.

We hence consider here the functions of type $f : S_1 \times \cdots \times S_d \rightarrow S_0$, where each S_i is either \mathbb{N} , \mathbb{Z} or \mathbb{Q} or \mathbb{R} . Or possibly vectorial functions whose components are of this type. We denote \mathcal{F} for the class of such functions. Clearly, we can consider $\mathbb{N} \subset \mathbb{Z} \subset \mathbb{Q} \subset \mathbb{R}$, but as functions may have different types of outputs, composition is an issue. We simply admit that composition may not be defined in some cases. In other words, we consider that composition is a partial operator: for example, given $f : \mathbb{N} \rightarrow \mathbb{R}$ and $g : \mathbb{R} \rightarrow \mathbb{R}$, the composition of g and f is defined as expected, but f cannot be composed with a function such as $h : \mathbb{N} \rightarrow \mathbb{N}$.

Considering discrete ODE as a model of computation is due to [BD19, BD23]. From a non-ODE-centric point of view, we are characterizing some complexity classes using particular discrete schemata. Recursion schemes constitute a major approach to computability theory and, to some extent, complexity theory. The foundational characterisation of **FPTIME** due to Cobham [Cob62], and then others based on safe recursion [BC92] or ramification ([LM93, Lei94]), or for other classes [LM95], gave birth to the very vivid field of *implicit complexity* at the interplay of logic and theory of programming: see [Clo98, CK02] for monographs.

Our ways of simulating Turing machines are rather similar to the constructions used in other contexts such as Neural Networks [SS95, Sie99]. But, as far as we know, only a few papers have been devoted to characterisations of complexity and even computability, classes in the sense of computable analysis. For discrete schemata, we only know [Bra96] and [NRTY21], focusing on computability and not complexity.

In Section 3.1, we present and prove an algebraic characterisation of real sequences computable in polynomial time. In Section 3.2, we generalise the characterisation to functions over the real: the proof techniques are very different, as, for example, in the second characterisation, we no longer do exact computations, but the algebras are rather similar.

3.1 Algebraic characterisation of the real sequences computable in polynomial time

In this section, we consider the class

$$\mathbb{L}\mathbb{D}\mathbb{L}^\bullet = \left[\mathbf{0}, \mathbf{1}, \pi_i^k, \ell(x), +, -, \times, \overline{\text{cond}}(x), \frac{x}{2}; \text{composition, linear length ODE} \right]$$

Here

- $\frac{x}{2} : \mathbb{R} \rightarrow \mathbb{R}$ is the function that divides by 2, and all other basic functions are defined exactly as for $\mathbb{L}\mathbb{D}\mathbb{L}$, but considered here as functions from the reals to reals.

- $\overline{\text{cond}}(\cdot)$ is the function defined in Section 1.2.2 for Definition 2.5.6: its restrictions to the integer is the function $\text{sg}(x)$ considered in \mathbb{LDL} .

We are going to prove, in Subsection 3.1.2, the following theorem:

Theorem 3.1.1 (Encoding a Turing machine in \mathbb{LDL}^\bullet)

A function $\mathbf{f} : \mathbb{N}^d \rightarrow \mathbb{R}^{d'}$, that is to say, a real sequence, is computable in polynomial time if and only if there exists $\tilde{\mathbf{f}} : \mathbb{N}^{d+1} \rightarrow \mathbb{R}^{d'} \in \mathbb{LDL}^\bullet$ such that for all $\mathbf{m} \in \mathbb{N}^d$, $n \in \mathbb{N}$,

$$\|\tilde{\mathbf{f}}(\mathbf{m}, 2^n) - \mathbf{f}(\mathbf{m})\| \leq 2^{-n}.$$

From the fact that we have the reverse direction in the previous theorem, it is natural to consider the operation that maps $\tilde{\mathbf{f}}$ to \mathbf{f} . For that purpose, we introduce the operation $ELim$ ($ELim$ stands for Effective Limit):

Definition 3.1.2 (Operation $ELim$)

Given $\tilde{\mathbf{f}} : \mathbb{N}^{d+1} \rightarrow \mathbb{R}^{d'} \in \mathbb{LDL}^\bullet$ such that for all $\mathbf{m} \in \mathbb{N}^d$, $n \in \mathbb{N}$, $\|\tilde{\mathbf{f}}(\mathbf{m}, 2^n) - \mathbf{f}(\mathbf{m})\| \leq 2^{-n}$, then $ELim(\tilde{\mathbf{f}})$ is the (clearly uniquely defined) corresponding function $\mathbf{f} : \mathbb{N}^d \rightarrow \mathbb{R}^{d'}$.

Proposition 3.1.3

The class of functions from the integers computable in polynomial time is preserved by the operation $ELim$.

Take such a function $\tilde{\mathbf{f}}$. By definition, given \mathbf{m} , we can compute $\tilde{\mathbf{f}}(\mathbf{m}, 2^n)$ with precision 2^{-n} in time polynomial in n . This must be by definition of $ELim$ schema some approximation of $\mathbf{f}(\mathbf{m})$, and hence \mathbf{f} is computable in polynomial time.

NB 3.1.4

We will reuse generalisations of this statement for functions over the reals in Sections 3.2, 5.1 and 5.2.

Then, we obtain our second main result of this chapter, providing a characterisation of polynomial time computable functions for functions from the integers to the reals. We prove it in Subsection 3.1.3.

Theorem 3.1.5 (Polynomial time in $\overline{\mathbb{LDL}^\bullet}$)

A function $\mathbf{f} : \mathbb{N}^d \rightarrow \mathbb{R}^{d'}$ is computable in polynomial time if and only if all its components belong to $\overline{\mathbb{LDL}^\bullet}$, where

$$\overline{\mathbb{LDL}^\bullet} = [\mathbf{0}, \mathbf{1}, \pi_i^k, \ell(x), +, -, \times, \overline{\text{cond}}(x), \frac{x}{2};$$

composition, linear length ODE, $ELim$]

In particular, we obtain a characterisation of the functions computable in polynomial time from the integers to the reals:

Theorem 3.1.6 (Characterisation of PTIME for real sequences)

$$\overline{\text{LDL}}^\bullet \cap \mathbb{R}^\mathbb{N} = \text{FPTIME} \cap \mathbb{R}^\mathbb{N}$$

The proof of $\overline{\text{LDL}}^\bullet \cap \mathbb{R}^\mathbb{N} \subseteq \text{FPTIME} \cap \mathbb{R}^\mathbb{N}$ is the purpose of Proposition 3.1.9: every function we can construct in $\overline{\text{LDL}}^\bullet$ is computable in polynomial time. The proof of $\overline{\text{LDL}}^\bullet \cap \mathbb{R}^\mathbb{N} \supseteq \text{FPTIME} \cap \mathbb{R}^\mathbb{N}$ relies on encoding the execution of Turing machines into linear length ODEs.

Our ways of simulating Turing machines have some similarities with constructions used in other contexts such as neural networks [SS95, Sie99]. In particular, we use a Cantor-like encoding set \mathcal{J} with inspiration from these references. These references use some particular sigmoid function σ (called sometimes the *saturated linear function*) that values 0 when $x \leq 0$, x for $0 \leq x \leq 1$, 1 for $x \geq 1$. The latter corresponds to $\overline{\text{cond}}\left(\frac{1}{4} + \frac{1}{2}x\right)$ and hence their constructions can be reformulated using the $\overline{\text{cond}}$ function.

The models considered in [SS95, Sie99] rely on inductions while our constructions are not neural networks. Also, they are restricted to live on the compact domain $[0, 1]$, which forbids getting functions from $\mathbb{R} \rightarrow \mathbb{R}$. Our settings allow more general functions. Our proofs also require functions taking some integer arguments, that would be impossible to consider in their hypotheses (unless at the price of an artificial recoding).

NB 3.1.7

In some sense, our constructions can be seen as operators mapping to a family of neural networks in the spirit of these models, instead of considering fixed recurrent neural networks, but also dealing with tanh, and not requiring the saturated linear function.

It will be seen later in the chapter (Subsection 3.1.3) that, in this context, the multiplication can be avoided.

Theorem 3.1.8 (No need of multiplication)

Previous theorems remain true if multiplication is not among the basic functions.

Let us now go to the proofs.

3.1.1 Towards functions from integers to the reals: simulation with Turing machines

Functions in LDL^\bullet are in FPTIME

The aim of this subsection is to prove that all functions of LDL^\bullet are computable (in the sense of computable analysis) in polynomial time (Proposition 3.1.9), giving one inclusion of Theorem 3.1.6.

Proposition 3.1.9

All functions of LDL^\bullet are computable (in the sense of computable analysis) in polynomial time.

Proof. We prove this proposition by induction. It is true for basis functions, from basic arguments from computable analysis. In particular, as $\overline{\text{cond}}$ is a continuous piecewise affine function with rational coefficients, it is computable in polynomial time from standard arguments.

The class of polynomial time computable functions is stable under composition: it is proved in Lemma 1.3.56.

It remains to prove that the class of polynomial time computable functions is closed under the linear length ODE schema: this is Lemma 2.5.12. \square

Functions in FPTIME are in \mathbb{LDL}^\bullet

This subsection is devoted to proving a kind of reverse implication of Proposition 3.1.9: for any polynomial-time computable function $\mathbf{f} : \mathbb{N}^d \rightarrow \mathbb{R}^{d'}$, we can construct some function $\tilde{\mathbf{f}} \in \mathbb{LDL}^\bullet$ that simulates the computation of \mathbf{f} . It requires to be able to simulate the computation of a Turing machine using functions from \mathbb{LDL}^\bullet .

Consequently, to simplify the coming proofs, we consider the variant of TMs defined in Section 2.2.

A key point is to observe that

Lemma 3.1.10

We can construct some function $\overline{\text{Next}}$ in \mathbb{LDL}^\bullet that simulates one step of M , i.e. that computes the Next function sending (the encoding of) a configuration C of Turing machine M to the next one. This function is essentially linear.

Proof. We restart from arguments of the proof of Theorem 2.3.1.

NB 3.1.11

However, the problem with the expressions of the proof of Theorem 2.3.1 is that we cannot expect the integer part and the fractional part function to be in \mathbb{LDL}^\bullet (as functions of this class are computable, and hence continuous, unlike the fractional part). But, a key point is that from our trick of using only symbols 1 and 3, we are sure that in an expression like $\lfloor \bar{r} \rfloor$, either it values 0 (this is the specific case where there remain only blanks in r), or that $4\bar{r}$ lives in the interval $[1, 1+1)$ or in interval $[3, 3+1)$.

That means that we could replace in the proof of Theorem 2.3.1 $\{4\bar{r}\}$ by $\sigma(4\bar{r})$ where σ is some (piecewise affine) function obtained by composing in a suitable way the basic functions of \mathbb{LDL}^\bullet .

Namely, define $\text{If}(b, T, E)$ as a synonym for $\overline{\text{cond}}(b) \times T + (1 - \overline{\text{cond}}(b)) \times E$. Then, considering $i(x) = \text{If}(x - 2, 3, \text{If}(x, 1, 0))$, $\sigma(x) = x - i(x)$, then $i(4\bar{r})$ would be the same as $\lfloor 4\bar{r} \rfloor$, and $\sigma(4\bar{r})$ would be the same as $\{4\bar{r}\}$ in our context in above expressions. In other words, we could replace the items (2.1) above by:

where $r_0 = i(4\bar{r})$

- in the first case “ \rightarrow ” : $\bar{l}' = 4^{-1}\bar{l} + 4^{-1}x$ and $\bar{r}' = \bar{r}^\bullet = \sigma(4\bar{r})$
- in the second case “ \leftarrow ” : $\bar{l}' = \bar{l}^\bullet = \sigma(4\bar{l})$ and $\bar{r}' = 4^{-2}\bar{r}^\bullet + 4^{-2}x + 4^{-1}i(4\bar{l})$

And get something that would still work exactly, but using only functions from \mathbb{LDL}^\bullet . Notice that this imbrication of If rewrites to an essentially constant expression.

We can then write:

$$q' = \text{If}(q - (|Q| - 1), \text{next}q^{|Q|-1}, \text{If}(q - (|Q| - 2), \text{next}q^{|Q|-2}, \dots, \text{If}(q - 1, \text{next}q^1, \text{next}q^0)))$$

where $\text{next}q^q = \text{If}(v - 3, \text{next}q_3^q, \text{If}(v - 1, \text{next}q_1^q, \text{next}q_0^q))$ and where $\text{next}q_v^q = q'$ if $\delta(q, v) = (q', x, m)$ for $m \in \{\leftarrow, \rightarrow\}$, for $v \in \{0, 1, 3\}$. Similarly, we can write

$$r' = \text{If}\left(q - (|Q| - 1), \text{next}r^{|Q|-1}, \text{If}\left(q - (|Q| - 2), \text{next}r^{|Q|-2}, \dots, \text{If}\left(q - 1, \text{next}r^1, \text{next}r^0\right)\right)\right)$$

where $\text{next}r^q = \text{If}(v - 3, \text{next}r_3^q, \text{If}(v - 1, \text{next}r_1^q, \text{next}r_0^q))$

and where $\text{next}r_v^q$ corresponds to the associated expression in the item above according to the value of $\delta(q, v)$. We can clearly write a similar expression for l' . This imbrication of If rewrites to some essentially linear expressions. \square

Once we have one step, we can simulate some arbitrary computation of a Turing machine, using some linear length ODE:

Proposition 3.1.12

Consider some Turing machine M computing some function $f : \Sigma^* \rightarrow \Sigma^*$ in some time $T(\ell(\omega))$ on input ω . We can construct some function $\tilde{\mathbf{f}} : \mathbb{N} \times \mathbb{R} \rightarrow \mathbb{R}$ in \mathbb{LDL}^\bullet that does the same, with respect to the previous encoding: $\tilde{\mathbf{f}}(2^{T(\ell(\omega))}, \gamma_{\text{word}}(\omega))$ provides $f(\omega)$.

Proof. The idea is to define the function \overline{Exec} that maps some time 2^t and some initial configuration C to the configuration at time t . It can be obtained using some linear length ODE using Lemma 3.1.10.

$$\overline{Exec}(0, C) = C \quad \text{and} \quad \frac{\delta \overline{Exec}}{\delta \ell}(t, C) = \overline{Next}(\overline{Exec}(t, C))$$

We can then get the value of the computation as $\overline{Exec}(2^{T(\ell(\omega))}, C_{\text{init}})$ on input ω , considering $C_{\text{init}} = (q_0, 0, \gamma_{\text{word}}(\omega))$. By applying some projection, we get the following function $\tilde{\mathbf{f}}(x, y) = \pi_3^3(\overline{Exec}(x, q_0, 0, y))$ that satisfies the property. \square

3.1.2 Proof of Theorem 3.1.1: $f \in \mathbf{FPTIME}$ is equivalent to the existence of some $\tilde{f} \in \mathbb{LDL}^\bullet$

The purpose of this section is to prove Theorem 3.1.1: a function $\mathbf{f} : \mathbb{N}^d \rightarrow \mathbb{R}^{d'}$ is computable in polynomial time if and only if there exists $\tilde{\mathbf{f}} : \mathbb{N}^{d+1} \rightarrow \mathbb{R}^{d'} \in \mathbb{LDL}^\bullet$ such that for all $\mathbf{m} \in \mathbb{N}^d$, $n \in \mathbb{N}$, $\|\tilde{\mathbf{f}}(\mathbf{m}, 2^n) - \mathbf{f}(\mathbf{m})\| \leq 2^{-n}$.

More precisely, we prove that the function \tilde{f} we construct in \mathbb{LDL}^\bullet converges exponentially fast to the function f , computable in polynomial time. And we prove that for every function $f \in \mathbf{FPTIME}$, there exists $\tilde{f} \in \mathbb{LDL}^\bullet$ converging exponentially fast to f .

Reverse implication: the existence of some $\tilde{f} \in \text{LDL}^\bullet$ implies $f \in \text{FPTIME}$

Proof. The reverse implication of Theorem 3.1.1 mostly follows from Proposition 3.1.9 and arguments from computable analysis. We prove this by induction.

Indeed, the only non-trivial inductive case is preservation by effective limit: assume there exists $\tilde{\mathbf{f}}: \mathbb{N}^{d+1} \rightarrow \mathbb{R}^{d'} \in \text{LDL}^\bullet$ such that for all $\mathbf{m} \in \mathbb{N}^d, n \in \mathbb{N}$,

$$\|\tilde{\mathbf{f}}(\mathbf{m}, 2^n) - \mathbf{f}(\mathbf{m})\| \leq 2^{-n}.$$

From Proposition 3.1.9, we know that $\tilde{\mathbf{f}}$ is computable in polynomial time (in the binary length of its arguments). Then $\mathbf{f}(\mathbf{m})$ is computable: indeed, given some integers \mathbf{m} and n , we can approximate $\mathbf{f}(\mathbf{m})$ at precision 2^{-n} as follows: approximate $\tilde{\mathbf{f}}(\mathbf{m}, 2^{n+1})$ at precision $2^{-(n+1)}$ by some rational q , and output q . We will then have

$$\begin{aligned} \|q - \mathbf{f}(\mathbf{m})\| &\leq \|q - \tilde{\mathbf{f}}(\mathbf{m}, 2^{n+1})\| + \|\tilde{\mathbf{f}}(\mathbf{m}, 2^{n+1}) - \mathbf{f}(\mathbf{m})\| \\ &\leq 2^{-(n+1)} + 2^{-(n+1)} \\ &\leq 2^{-n}. \end{aligned}$$

All of this is done in polynomial time in n and the size of \mathbf{m} and, hence, we get that \mathbf{f} is polynomial time computable from definitions. \square

Direct implication: $f \in \text{FPTIME}$ implies the existence of some $\tilde{f} \in \text{LDL}^\bullet$

For the direct implication of Theorem 3.1.1, the difficulty is that we know from the previous section (Section 3.1.1) how to simulate Turing machines working over \mathcal{I} (the image of $\gamma_{\text{word}}: \Sigma^\omega \rightarrow \mathbb{R}$, and γ_{word} defined in the proof of Theorem 2.3.1), while we want functions that work directly over the integers and the reals. A key point is to be able to convert from integers/reals to representations using only symbols 1 and 3, that is to say, to map integers to \mathcal{I} , and \mathcal{I} to reals.

Lemma 3.1.13 (From \mathcal{I} to \mathbb{R})

We can construct some function $\text{Encode}: \mathbb{N} \times [0, 1] \rightarrow \mathbb{R}$ in LDL^\bullet mapping $\gamma_{\text{word}}(\bar{d})$ with $\bar{d} \in \{1, 3\}^$ to some real d . It is surjective over the dyadic, in the sense that for any dyadic $d \in \mathbb{D}$, there is some (easily computable) such \bar{d} with $\text{Encode}(2^{\ell(\bar{d})}, \bar{d}) = d$.*

Proof. Consider the following transformation: every digit in the binary expansion of d is encoded by a pair of symbols in the radix 4 encoding of $\bar{d} \in [0, 1]$: digit 0 (respectively: 1) is encoded by 11 (resp. 13) if before the “decimal” point in d , and digit 0 (respectively: 1) is encoded by 31 (resp. 33) if after. For example, for $d = 101.1$ in base 2, $\bar{d} = 0.13111333$ in base 4. To compute d , given \bar{d} , the intuition is to consider a two-tapes Turing machine $(Q, \Sigma, q_{\text{init}}, \delta, F)$: the first tape contains the input (\bar{d}) and is read-only, the second one is write-only and empty at the beginning. We just use a different encoding on the second tape than the previous one: for the first tape, we restrict to digits 0, 1, 3, while for the second, we use binary encoding.

Writing the natural Turing machine doing the transformation, this would do the following (in terms of real numbers), if we forget the encoding of the internal state.

With more detail, from the fact that we have only 1 and 3 in \bar{r} , the reasoning is valid as soon as $i(16\bar{r})$ is correct for $16\bar{r} \in \{\overline{11}, \overline{13}, \overline{31}, \overline{33}\}$. So $i(x) = \text{If}(x - 15, 15, \text{If}(x - 13, 13, \text{If}(x - 7, 5)))$ works. Then take $\sigma(x) = x - i(x)$.

The transformation from \bar{d} to d can be done by iterating a function $F : [0, 1]^2 \rightarrow [0, 1]^2$ satisfying:

$$F(\bar{r}_1, \bar{l}_2) = \begin{cases} (\sigma(16\bar{r}_1), 2\bar{l}_2 + 0) & \text{whenever } i(16\bar{r}_1) = 5 \\ (\sigma(16\bar{r}_1), 2\bar{l}_2 + 1) & \text{whenever } i(16\bar{r}_1) = 7 \\ (\sigma(16\bar{r}_1), (\bar{l}_2 + 0)/2) & \text{whenever } i(16\bar{r}_1) = 13 \\ (\sigma(16\bar{r}_1), (\bar{l}_2 + 1)/2) & \text{whenever } i(16\bar{r}_1) = 15 \end{cases}$$

Here we write \overline{ab} for the integer whose base 4 radix expansion is ab . A natural candidate for F is an expression such as

$$\begin{aligned} & \text{If}(i(16\bar{r}_1) - 15, (\sigma(16\bar{r}_1), (\bar{l}_2 + 1)/2), \\ & \quad \text{If}(i(16\bar{r}_1) - 13, (\sigma(16\bar{r}_1), (\bar{l}_2 + 0)/2), \\ & \quad \quad \text{If}(i(16\bar{r}_1) - 7, (\sigma(16\bar{r}_1, 2\bar{l}_2 + 1), (\sigma(16\bar{r}_1, 2\bar{l}_2 + 0)))) \end{aligned}$$

with σ and i constructed as a suitable approximation of the fractional and integer part and $\text{If}(\cdot, \cdot, \cdot)$ defined in the proof of Lemma 3.1.10.

Then, we just need to apply $\ell(\bar{d})$ th times F on $(\bar{d}, 0)$, and then project on the second component to get a function *Encode* doing the encoding. That is $\text{Encode}(x, y) = \pi_3^3(G(x, y))$ with

$$G(0, y) = (\bar{d}, 0) \quad \text{and} \quad \frac{\delta G}{\delta \ell}(t, \bar{d}, \bar{l}) = F(G(t, \bar{d}, \bar{l})).$$

This is how we can get the function F . Then the previous reasoning applies to the iterations of function F that would provide some encoding function. \square

Lemma 3.1.14 (From \mathbb{N} to \mathcal{I})

We can construct some function $\text{Decode} : \mathbb{N}^d \rightarrow \mathbb{R}$ in LDL^\bullet that maps $n \in \mathbb{N}$ to some (easily computable) encoding of n in \mathcal{I} .

Proof. We discuss only the case $d = 1$. The generalisation to $d > 1$ is similar.

Let div_2 (respectively: mod_2) denote integer (resp. remainder of) division by 2: as these functions are from $\mathbb{N} \rightarrow \mathbb{N}$, from Theorem 3.1.6, they belongs to LDL . Their expression in LDL , replacing sg by cond , provides some extensions $\overline{\text{div}_2}$ and $\overline{\text{mod}_2}$ in LDL^\bullet .

We do something similar as in Lemma 3.1.13 but now with the function

$$F(\bar{r}_1, \bar{l}_2) = \begin{cases} (\overline{\text{div}_2}(\bar{r}_1), (\bar{l}_2 + 0)/2) & \text{whenever } \overline{\text{mod}_2}(\bar{r}_1) = 0 \\ (\overline{\text{div}_2}(\bar{r}_1), (\bar{l}_2 + 1)/2) & \text{whenever } \overline{\text{mod}_2}(\bar{r}_1) = 1. \end{cases}$$

\square

We can now prove the direct direction of Theorem 3.1.1:

Proof of Theorem 3.1.1. Assume $\mathbf{f} : \mathbb{N}^d \rightarrow \mathbb{R}^{d'}$ is computable in polynomial time. It means that each of its components are, so we can consider without loss of generality that $d' = 1$. We assume also that $d = 1$ (otherwise consider either multi-tape Turing machines, or some suitable alternative encoding in *Encode*). That means that we know that there is a TM polynomial time computable functions $d : \mathbb{N}^{d+1} \rightarrow \{1, 3\}^*$ so that on \mathbf{m}, n it provides the encoding of some dyadic $\phi(\mathbf{m}, n)$ with $\|\phi(\mathbf{m}, n) - \mathbf{f}(\mathbf{m})\| \leq 2^{-n}$ for all \mathbf{m} .

From Proposition 3.1.12, we can construct \tilde{d} with $\tilde{d}(2^{p(\max(\mathbf{m}, n))}, \text{Decode}(n, \mathbf{m})) = d(\mathbf{m}, n)$ for some polynomial p corresponding to the time required to compute d .

Both functions $\ell(\mathbf{x}) = \ell(x_1) + \dots + \ell(x_p)$ and $B(\mathbf{x}) = 2^{\ell(\mathbf{x}) \cdot \ell(\mathbf{x})}$ are in \mathbb{LDL} (see [BD19, BD23]). It is easily seen that $\ell(\mathbf{x})^c \leq B^{(c)}(\ell(\mathbf{x}))$ where $B^{(c)}$ is the c -fold composition of function B .

Then

$$\tilde{\mathbf{f}}(\mathbf{m}, n) = \text{Encode}(\tilde{d}(B^{(c)}(\max(\mathbf{m}, n)), \text{Decode}(n, \mathbf{m})))$$

provides a solution such that

$$\|\tilde{\mathbf{f}}(\mathbf{m}, 2^n) - \mathbf{f}(\mathbf{m})\| \leq 2^{-n}$$

□

3.1.3 Proving Theorems 3.1.5 and 3.1.6: $\overline{\mathbb{LDL}}^\bullet$ characterises **FPTIME** for real sequences

It remains to prove that $\overline{\mathbb{LDL}}^\bullet$ characterises **FPTIME** for real sequences.

We recall the statement of Theorem 3.1.6: $\overline{\mathbb{LDL}}^\bullet \cap \mathbb{R}^\mathbb{N} = \mathbf{FPTIME} \cap \mathbb{R}^\mathbb{N}$. It follows from the case where $d = 1$ and $d' = 1$ from Theorem 3.1.5.

Hence, there only remains to prove Theorem 3.1.5, stating that a function $\mathbf{f} : \mathbb{N}^d \rightarrow \mathbb{R}^{d'}$ is computable in polynomial time if and only if all its components belong to $\overline{\mathbb{LDL}}^\bullet$.

The direct direction is immediate from Theorem 3.1.1.

For the reverse direction, is direct from Proposition 3.1.3.

We now improve this statement by getting rid of the multiplication in the algebra.

Getting rid of the multiplication

We notice that, in $\overline{\mathbb{LDL}}^\bullet$, we could have removed the multiplication.

Theorem 3.1.15

We have that $\overline{\mathbb{LDL}}^\bullet \cap \mathbb{R}^\mathbb{N}$ is equal to:

$$[\mathbf{0}, \mathbf{1}, \pi_i^k, \ell(x), +, -, \overline{\text{cond}}(x), \frac{x}{2}; \text{composition, linear length ODE, ELim}] \cap \mathbb{R}^\mathbb{N}.$$

In order to prove this theorem, we must introduce the following function:

Lemma 3.1.16

Consider $T(d, l) = \overline{\text{cond}}(d - 3/4 + l/2)$. For $l \in [0, 1]$, we have $T(0, l) = 0$, and $T(1, l) = l$.

Proof. We check that it is true for $d = 0$ and then for $d = 1$, from the definition $\overline{\text{cond}}$. \square

Then, we also need to modify the way we do the conditions (conditional loops), by rewriting them to avoid multiplications and introducing a function send:

Lemma 3.1.17

An expression such as, for If defined in the proof of Lemma 3.1.10:

$$\text{If}(q - \alpha_n, T_n, \text{If}(q - \alpha_{n-1}, T_{n-1}, \dots, \text{If}(q - \alpha_1, T_1, E_1))),$$

for integers $\alpha_1 < \alpha_2 < \dots < \alpha_n$ is some piecewise continuous function that values E_1 for $x \leq \alpha_1 + \frac{1}{4}$, T_i on $[\alpha_i + \frac{3}{4}, \alpha_{i+1} + \frac{1}{4}]$ for $i \in \{1, 2, \dots, n-1\}$, T_n for $x \geq \alpha_n + 3/4$. It is equivalent to some function obtained without any multiplication.

Proof. Observe that any product of the form $\overline{\text{cond}}(q - \alpha_i) \overline{\text{cond}}(q - \alpha_j)$ rewrites to:

$$\overline{\text{cond}}(q - \alpha_j)$$

if $\alpha_i < \alpha_j$ from the definition of $\overline{\text{cond}}$.

It follows from induction that above expression rewrites to $E_1 + \overline{\text{cond}}(q - \alpha_1)(T_1 - E_1) + \overline{\text{cond}}(q - \alpha_2)(T_2 - T_1) + \dots + \overline{\text{cond}}(q - \alpha_n)(T_n - T_{n-1})$, from which the result follows easily. \square

Actually, let's propose the following notation:

Lemma 3.1.18

Assume you are given some integers $\alpha_1, \alpha_2, \dots, \alpha_n$, and values V_1, V_2, \dots, V_n . Then there is some function obtained without any multiplication, that we write $\text{send}(\alpha_i \mapsto V_i)_{i \in \{1, \dots, n\}}$, mapping any $x \in [\alpha_i - 1/4, \alpha_i + 1/4]$ to V_i , for all $i \in \{1, \dots, n\}$.

Proof. Sort the α_i so that $\alpha_1 < \alpha_2 < \dots < \alpha_n$. Then consider $V_1 + \overline{\text{cond}}(x - \alpha_1 + 1)(V_2 - V_1) + \overline{\text{cond}}(x - \alpha_2 + 1)(V_3 - V_2) + \dots + \overline{\text{cond}}(x - \alpha_{n-1} + 1)(V_n - V_{n-1})$. \square

More generally:

Lemma 3.1.19

Let N be some integer. Assume we are given some integers $\alpha_1, \alpha_2, \dots, \alpha_n$, and some values $V_{i,j}$ for $1 \leq i \leq n$, and $0 \leq j < N$. Then there is some function,

$$\text{send}((\alpha_i, j) \mapsto V_{i,j})_{i \in \{1, \dots, n\}, j \in \{0, \dots, N-1\}},$$

mapping any $x \in [\alpha_i - 1/4, \alpha_i + 1/4]$ and $y \in [j - 1/4, j + 1/4]$ to $V_{i,j}$, for all $i \in \{1, \dots, n\}$, $j \in \{0, \dots, N-1\}$, and obtained without any multiplication.

Proof. If we define the function by

$$\begin{aligned} \text{send}((\alpha_i, j) \mapsto V_{i,j})_{i \in \{1, \dots, n\}, j \in \{1, \dots, N\}}(x, y) \\ = \text{send}(N\alpha_i + j \mapsto V_{i,j})_{i \in \{1, \dots, n\}, j \in \{1, \dots, N\}}(Nx + y) \end{aligned}$$

this works when $x = \alpha_i$ for some i .

Considering instead:

$$\text{send}(N\alpha_i + j \mapsto V_{i,j})_{i \in \{1, \dots, n\}, j \in \{1, \dots, N\}}(N \text{send}(\alpha_i \mapsto \alpha_i)_{i \in \{1, \dots, n\}}(x) + y)$$

works for any $x \in [\alpha_i - 1/4, \alpha_i + 1/4]$. \square

We can now prove Theorem 3.1.15:

Proof. We only needed multiplication for the If instructions. The above lemmas give way to realise similar operations without any multiplication.

For example, we used the formula

$$q' = \text{If}(q - (|Q| - 1), \text{next}q^{|Q|-1}, \text{If}(q - (|Q| - 2), \text{next}q^{|Q|-2}, \dots, \text{If}(q - 1, \text{next}q^1, \text{next}q^0)))$$

where

$$\text{next}q^q = \text{If}(v - 3, \text{next}q_3^q, \text{If}(v - 1, \text{next}q_1^q, \text{next}q_0^q))$$

in the proof of Lemma 3.1.10.

Using this new formalism, we could write instead:

$$q' = \text{send}((q, r) \mapsto \text{next}q_r^q)_{q \in Q, r \in \{0, 1, 3\}}(q, \sigma(4\bar{r})),$$

where $\text{next}q_{r_0}^q = q'$ if $\delta(q, r_0) = (q', x, m)$ for $m \in \{\leftarrow, \rightarrow\}$.

This would do exactly the same but without any multiplication. \square

Generalisations

The results we prove remain true for a more general notion of effective limit (Definition 3.1.2):

Definition 3.1.20 (Operation $E_2\text{Lim}$)

Given $\tilde{\mathbf{f}} : \mathbb{N}^{d+1} \rightarrow \mathbb{R} \in \text{LIDL}^\bullet$, $g : \mathbb{N}^{d+1} \rightarrow \mathbb{R}$ such that for all $\mathbf{m} \in \mathbb{N}^d$, $n \in \mathbb{N}$, $\|\tilde{\mathbf{f}}(\mathbf{m}, 2^n) - \mathbf{f}(\mathbf{m})\| \leq g(\mathbf{m}, n)$ under the condition that $0 \leq g(\mathbf{m}, n)$ is decreasing to 0, with $\|g(\mathbf{m}, p(n))\| \leq 2^{-n}$ for some polynomial $p(n)$ then $E_2\text{Lim}(\tilde{\mathbf{f}}, g)$ is the (clearly uniquely defined) corresponding function $\mathbf{f} : \mathbb{N}^d \rightarrow \mathbb{R}^e$.

Theorem 3.1.21

We could replace $E\text{Lim}$ by $E_2\text{Lim}$ in the statements of Theorems 3.1.5 and 3.1.6.

It is equivalent to proving the following, and observe from the proof that we can replace in the above statement “ $g(\mathbf{m}, n)$ going to 0” by “decreasing to 0”, and last condition by $\|g(\mathbf{m}, p(n))\| \leq 2^{-n}$.

Theorem 3.1.22

$\mathbf{F} : \mathbb{N}^d \rightarrow \mathbb{R}^{d'}$ is computable in polynomial time iff there exists $\mathbf{f} : \mathbb{N}^{d+1} \rightarrow \mathbb{Q}^{d'}$, with $\mathbf{f}(\mathbf{m}, n)$ computable in polynomial time with respect to the value of n , and $g : \mathbb{N}^{d+1} \rightarrow \mathbb{Q}$ such that

- $\|\mathbf{f}(\mathbf{m}, n) - \mathbf{F}(\mathbf{m})\| \leq g(\mathbf{m}, n)$;
- $0 \leq g(\mathbf{m}, n)$ and $g(\mathbf{m}, n)$ converging to 0 when n goes to $+\infty$;
- with a uniform polynomial modulus of convergence $p(n)$.

Proof. (\Rightarrow): if we assume that \mathbf{F} is computable in polynomial time, we set $g(\mathbf{m}, n) = 2^{-n}$, and we take the identity as uniform modulus of convergence (see Definition 1.1.10).

(\Leftarrow): given \mathbf{m} and n , approximate $\mathbf{f}(\mathbf{m}, p(n+1)) \in \mathbb{Q}^{d'}$ at precision $2^{-(n+1)}$ by some dyadic rational q and output q . It can be done in polynomial time with respect to the value of n .

We will then have

$$\begin{aligned} \|q - \mathbf{F}(\mathbf{m})\| &\leq \|q - \mathbf{f}(\mathbf{m}, p(n+1))\| + \|\mathbf{f}(\mathbf{m}, p(n+1)) - \mathbf{F}(\mathbf{m})\| \\ &\leq 2^{-(n+1)} + g(\mathbf{m}, p(n+1)) \\ &\leq 2^{-(n+1)} + 2^{-(n+1)} \leq 2^{-n} \end{aligned}$$

□

From the proofs, we also get a normal form theorem. In particular:

Theorem 3.1.23 (Normal form theorem)

Any function $f : \mathbb{N}^d \rightarrow \mathbb{R}^{d'}$ can be obtained from the class $\overline{\text{LDL}}^\bullet$ using only one schema $ELim$ (or E_2Lim).

3.2 Algebraic characterisation of FPTIME for functions over the reals

Now that we have an algebraic characterisation of **FPTIME** for functions from \mathbb{N} to \mathbb{R} , we are going to extend these results to functions in $\mathbb{R}^{\mathbb{R}}$.

But we cannot use the same kind of proof as before as the previous encoding function would not be correct in that framework. It comes from the fact that it is not possible to *continuously* send \mathbb{R} in base 4 in a set of numbers using only 1s and 3s, intuitively a set which is not dense in \mathbb{R} . Also, it was obtained by assuming that some **non-analytic exact function** is among the basic available functions to simulate a Turing machine, namely $\overline{\text{cond}}$. We want to avoid this and use only analytic functions.

First, we extend all previous results to real functions. Consider

$$\text{LDL}^\circ = \left[\mathbf{0}, \mathbf{1}, \pi_i^k, \ell(x), +, -, \tanh, \frac{x}{2}, \frac{x}{3}; \text{composition, linear length ODE} \right],$$

where $\frac{x}{2} : \mathbb{R} \rightarrow \mathbb{R}$ is the function dividing by 2 (similarly for $\frac{x}{3}$) and all other basic functions defined exactly as for \mathbb{LDL} , but are considered here as functions from the reals to reals.

The core of our proof relies on two principles:

- As we need non-continuous functions, such as the integer parts, we provide *continuous* approximations of them, using our discrete ODEs schemes.
- We encode the execution of polynomial time or polynomial space Turing machines into ODEs using, in particular, the previous approximations.

Since \mathbb{LDL}° is about functions over the reals, and Theorem 2.5.16 is about functions over the integers, we want a way to compare these classes. Given a function $\mathbf{f} : \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$ sending every integer $\mathbf{n} \in \mathbb{N}^d$ to the vicinity of some integer of $\mathbb{N}^{d'}$, say at a distance less than $1/4$, we write $\text{DP}(f)$ for its discrete part: this is the function from $\mathbb{N}^d \rightarrow \mathbb{N}^{d'}$ mapping $\mathbf{n} \in \mathbb{N}^d$ to the integer rounding of $\mathbf{f}(\mathbf{n})$. Given a class \mathcal{C} of such functions, we write $\text{DP}(\mathcal{C})$ for the class of the discrete parts of the functions of \mathcal{C} . It is inspired by the work done in the BSS model [BCSS98] comparing complexity classes to their discrete part. We have:

Theorem 3.2.1

$$\text{DP}(\mathbb{LDL}^\circ) = \mathbf{FPTIME} \cap \mathbb{N}^{\mathbb{N}}.$$

We improve Theorem 3.1.6: Write $\overline{\mathbb{LDL}^\circ}$ for the class obtained by adding some effective limit operation similar to the one considered in Definition 3.1.2.

Definition 3.2.2 (Operation *ELim* for \mathbb{LDL}°)

Given $\tilde{\mathbf{f}} : \mathbb{R}^d \times \mathbb{N}^{d''} \times \mathbb{N} \rightarrow \mathbb{R}^{d'} \in \mathbb{LDL}^\circ$ such that for all $\mathbf{x} \in \mathbb{R}^d, X \in \mathbb{N}, \mathbf{x} \in [-2^X, 2^X], \mathbf{m} \in \mathbb{N}^{d''}, n \in \mathbb{N}$,

$$\|\tilde{\mathbf{f}}(\mathbf{x}, \mathbf{m}, 2^X, 2^n) - \mathbf{f}(\mathbf{x}, \mathbf{m})\| \leq 2^{-n},$$

then $\text{ELim}(\tilde{\mathbf{f}})$ is the (uniquely defined) corresponding function $\mathbf{f} : \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$.

We get a characterisation of functions over the reals (and not only sequences as in the previous section) computable in polynomial time.

Theorem 3.2.3 (Generic functions over the reals)

$\overline{\mathbb{LDL}^\circ} \cap \mathbb{R}^{\mathbb{R}} = \mathbf{FPTIME} \cap \mathbb{R}^{\mathbb{R}}$. More generally:

$$\overline{\mathbb{LDL}^\circ} \cap \mathbb{R}^{\mathbb{N}^d \times \mathbb{R}^{d'}} = \mathbf{FPTIME} \cap \mathbb{R}^{\mathbb{N}^d \times \mathbb{R}^{d'}}.$$

3.2.1 Preliminary results about continuous functions

Basic results about various common functions

A key part of our proofs is the construction of very specific functions in \mathbb{LDL}° . These functions will be used to simulate the execution of the Turing machines in the following

section. For many of them, we have functions parameterised by some m , providing a way to approximate some “ideal” function: the error between the function and the ideal function remains at less than 2^{-m} . This sometimes holds on a subdomain controlled by some parameter n . We start with some basic facts about the hyperbolic tangent.

Lemma 3.2.4

$$|1 + \tanh x| \leq 2 \exp(2|x|) \text{ for } x \in (-\infty, 0].$$

Proof. For $x \leq 0$, $|1 + \tanh x| = 1 + \tanh x$, and $|x| = -x$. We have $f(x) = 2 \exp(2x) - \tanh(x) - 1 = 2 \exp(2x) - \frac{1 - \exp(-2x)}{1 + \exp(-2x)} - 1 = \frac{2 \exp(4x)}{1 + \exp(2x)} \geq 0$. \square

Lemma 3.2.5

$$|1 - \tanh x| \leq 2 \exp(-2|x|) \text{ for } x \in [0, +\infty).$$

Proof. This follows from Lemma 3.2.4, using the fact that \tanh is odd. \square

A first observation is that we can uniformly approximate the (very famous in deep learning context) $\text{ReLU}(x) = \max(0, x)$ function using an essentially constant function:

Lemma 3.2.6

Consider (see Figure 3.1) $Y(x, 2^{m+2}) = \frac{1 + \tanh(2^{m+2}x)}{2}$. For all integer m , for all $x \in \mathbb{R}$,

$$|\text{ReLU}(x) - xY(x, 2^{m+2})| \leq 2^{-m}.$$

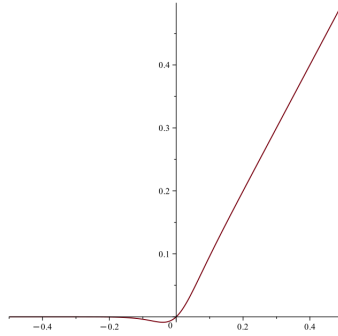


Figure 3.1: Graphical representation of $xY(x, 2^{2+2})$ obtained with Maple.

NB 3.2.7

Here, we are basically programming with \tanh and sigmoids. We express the sigmoids in terms of the ReLU function, through Lemma 3.2.6. Function \tanh could be replaced by \arctan : the key is to be able to approximate the ReLU function with \tanh (Lemma 3.2.6) and this can be proved to hold also for \arctan , using error bounds on \arctan established in [Gra07].

Proof of Lemma 3.2.6. Let $m \in \mathbb{N}$. Consider $Y(x, K) = \frac{1+\tanh(Kx)}{2}$, with $K > 0$.

For $0 \leq x$, $\text{ReLU}(x) = x$, and $|\text{ReLU}(x) - xY(x, K)| = \frac{x}{2}|1 - \tanh(Kx)| \leq x \exp(-2Kx)$ from Lemma 3.2.5.

For $x \leq 0$, $\text{ReLU}(x) = 0$, and $|\text{ReLU}(x) - xY(x, K)| = \frac{|x|}{2}|1 + \tanh(Kx)| \leq |x| \exp(-2K|x|)$ from Lemma 3.2.4, which is the same value as above for $0 \leq x$.

Function $g(x) = x \exp(-2Kx)$ has its derivative $g'(x) = \exp(-2Kx)(1 - 2Kx)$. We deduce the maximum of this function $g(x)$ over \mathbb{R} is in $\frac{1}{2K}$, and that the maximum value of $g(x)$ is $\frac{1}{e^{2K}}$.

Consequently, if we take $K = 2^{m+2}$, then $g(x) \leq 2^{-m}$ for all x , and we conclude. \square

We deduce we can uniformly approximate the continuous sigmoid functions (when $\frac{1}{b-a}$ is in \mathbb{LDL}°) defined as: $\mathfrak{s}(a, b, x) = 0$ whenever $w \leq a$, $\frac{x-a}{b-a}$ whenever $a \leq x \leq b$, and 1 whenever $b \leq x$. We denote by the gothic style the exact function and by \mathcal{C} – the continuous approximation.

Lemma 3.2.8 (Uniform approximation of any piecewise continuous sigmoid)

Assume $a, b, \frac{1}{b-a}$ is in \mathbb{LDL}° . Then there is some function (illustrated by Figure 3.2) $\mathcal{C}\text{-}\mathfrak{s}(m, a, b, x) \in \mathbb{LDL}^\circ$ such that for all integer m ,

$$|\mathcal{C}\text{-}\mathfrak{s}(m, a, b, x) - \mathfrak{s}(a, b, x)| \leq 2^{-m}.$$

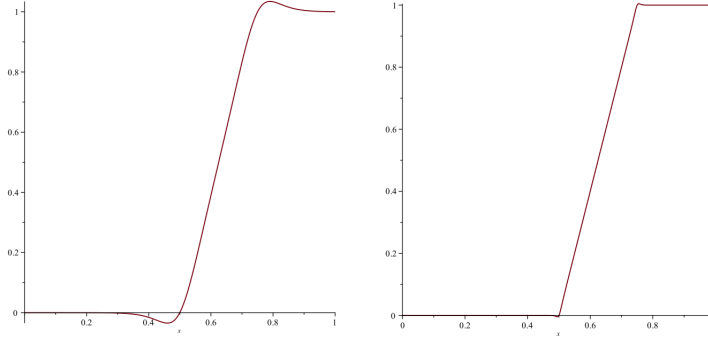


Figure 3.2: Graphical representations of $\mathcal{C}\text{-}\mathfrak{s}(2, \frac{1}{2}, \frac{3}{4}, x)$ and $\mathcal{C}\text{-}\mathfrak{s}(2^5, \frac{1}{2}, \frac{3}{4}, x)$ obtained with Maple.

Proof. We can write $\mathfrak{s}(a, b, x) = \frac{\text{ReLU}(x-a) - \text{ReLU}(x-b)}{b-a}$. Consider

$$\mathcal{C}\text{-}\mathfrak{s}(z, a, b, x) = \frac{(x-a)Y(x-a, z2^{1+c}) - (x-b)Y(x-b, z2^{1+c})}{b-a}$$

Thus, $|\mathcal{C}\text{-}\mathfrak{s}(m+1+c, a, b, x) - \mathfrak{s}(a, b, x)| \leq \frac{2 \cdot 2^{-m-1-c}}{b-a}$, using the triangle inequality. Take c such that $\frac{1}{b-a} \leq 2^c$. \square

Now that we have Lemma 3.2.8, we can continuously approximate other classical discrete (piecewise affine) functions.

Now, given some function $\mathbf{f}: \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$, some error and some time t , our expressions provide explicit expressions in \mathbb{LDL}° of an approximation of what is computed by a Turing machine at time t uniformly over any compact domain.

Constructing a zoo: continuous approximations of discrete functions

We prove the existence of a continuous approximation of a threshold, written in \mathbb{LDL}° . We will need it to express continuous approximations of discrete functions, such as the fractional part (Corollary 3.2.10), the floor function (Corollary 3.2.11), a function to express an adaptive barycenter (Corollary 3.2.12), inspired by some constructions of [BCGSH07], the modulo 2 (Corollary 3.2.1), and the Euclidian division by 2 (Corollary 3.2.14). In the same spirit, the function \tanh is a uniform controlled approximation of the piecewise affine function $\overline{\text{cond}}$. We write $\{x\}$ for the fractional part of the real x , i.e. $\{x\} = x - \lfloor x \rfloor$.

We start with the following function ξ and we will construct the other discrete functions based on this continuous approximation:

Theorem 3.2.9

There exists some function, illustrated by Figure 3.3, $\xi: \mathbb{N}^2 \rightarrow \mathbb{R}$ in \mathbb{LDL}° such that for all $n, m \in \mathbb{N}$ and $x \in [-2^n, 2^n]$, whenever $x \in \left[\lfloor x \rfloor + \frac{1}{8}, \lfloor x \rfloor + \frac{7}{8} \right]$,

$$\left| \xi(2^m, 2^n, x) - \left\{ x - \frac{1}{8} \right\} \right| \leq 2^{-m}.$$

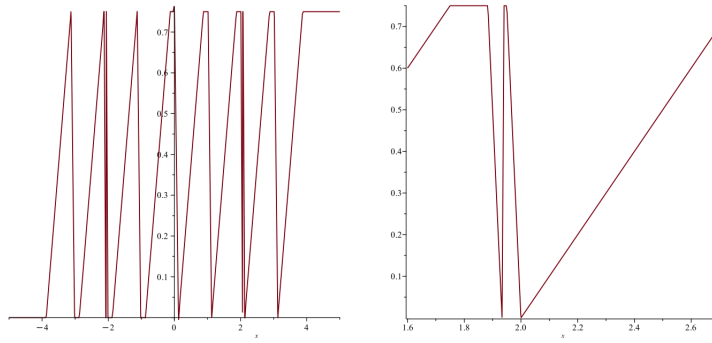


Figure 3.3: Graphical representations of $\xi(2, 4, x)$ obtained with Maple: some details on the right.

Proof. If we take ξ' that satisfies the constraint only when $x \geq 0$, and that values 0 for $x \leq 0$, then $\frac{3}{4} - \xi'(\cdot, \cdot, -x)$ would satisfy the constraint when $x \leq 0$, but values $3/4$ for $x \geq 0$. So,

$$\xi(2^m, N, x) = \xi'(2^{m+2}, N, x) - \xi'(2^{m+2}, N, -x) + \frac{3}{4} - \frac{3}{4} \mathcal{C}_{-\mathfrak{s}}\left(2^{m+2}, 0, \frac{1}{8}, x\right)$$

would work for all x . So it remains to construct ξ' such that for all $n \in \mathbb{N}$, $x \in [0, 2^n]$ and $m \in \mathbb{N}$, whenever $x \in \left[\lfloor x \rfloor + \frac{1}{8}, \lfloor x \rfloor + \frac{7}{8}\right]$,

$$\left| \xi'(2^m, 2^n, x) - \left\{x - \frac{1}{8}\right\} \right| \leq 2^{-m} \text{ and } |\xi'(2^m, N, x) - 0| \leq 2^{-m} \text{ for } x \leq 0.$$

Let $s(x) = \frac{3}{4} \mathfrak{s} \left(\lfloor x \rfloor + \frac{1}{8}, \lfloor x \rfloor + \frac{7}{8}, x \right)$. Over $\left[\lfloor x \rfloor + \frac{1}{8}, \lfloor x \rfloor + \frac{7}{8}\right]$, we have $s(x) = \left\{x - \frac{1}{8}\right\}$. Actually, we will even construct ξ' with the stronger properties that whenever $x \in \left[\lfloor x \rfloor + \frac{1}{8} - 2^{-m}, \lfloor x \rfloor + \frac{7}{8} + 2^{-m}\right]$,

$$\left| \xi'(2^m, 2^n, x) - s\left(x - \frac{1}{8}\right) \right| \leq 2^{-m}$$

It suffices to define ξ' by induction by

$$\begin{cases} \xi'(2^m, 2^0, x) &= \frac{3}{4} \mathcal{C}\text{-}\mathfrak{s} \left(2^m, \frac{1}{8}, \frac{7}{8}, x \right) \\ \xi'(2^m, 2^{n+1}, x) &= \xi'(2^{m+1}, 2^n, F(2^{m+1}, 2^n, x)) \end{cases}$$

where

$$F(2^{m+1}, K, x) = x - K \cdot \mathcal{C}\text{-}\mathfrak{s} \left(2^{m+1}, K + \frac{1}{32}, K + \frac{3}{32}, x \right).$$

Let $I_{\lfloor x \rfloor}$ be $\left[\lfloor x \rfloor + \frac{1}{8}, \lfloor x \rfloor + \frac{7}{8}\right]$, $x \in I_{\lfloor x \rfloor}$, and let us first study the value of $F(2^{m+1}, 2^n, x)$:

- If $x \leq 2^n$, by definition of $\mathcal{C}\text{-}\mathfrak{s}$, $|F(2^{m+1}, 2^n, x) - x| \leq 2^{-(m+1)}$, with $x \in I_{\lfloor x \rfloor}$.
- The case $2^n < x < 2^n + \frac{1}{8}$ cannot happen as we assume $x \in I_{\lfloor x \rfloor}$.
- If $2^n + \frac{1}{8} \leq x$ then

$$|F(2^{m+1}, 2^n, x) - (x - 2^n)| \leq 2^{-(m+1)}$$

with $x - 2^n \in I_{\lfloor x \rfloor - 2^n}$.

Now, the property is true by induction. Indeed, it is true for $n = 0$ by definition of $\xi'(2^m, 2^0, x)$. We now assume it is true for some $n \in \mathbb{N}$. We have $\xi'(2^m, 2^{n+1}, x) = \xi'(2^{m+1}, 2^n, F(2^{m+1}, 2^n, x))$. Thus, by induction hypothesis,

$$\left| \xi'(2^{m+1}, 2^n, F(2^{m+1}, 2^n, x)) - s\left(F(2^{m+1}, 2^n, x) - 1/8\right) \right| \leq 2^{-(m+1)}.$$

Now:

- If $x \leq 2^n$, by definition of $\mathcal{C}\text{-}\mathfrak{s}$,

$$\left| F(2^{m+1}, 2^n, x) - x \right| \leq 2^{-(m+1)},$$

and as s is 1-Lipschitz,

$$\left| s\left(F(2^{m+1}, 2^n, x) - \frac{1}{8}\right) - s\left(x - \frac{1}{8}\right) \right| \leq \left| F(2^{m+1}, 2^n, x) - x \right| \leq 2^{-(m+1)}.$$

Consequently, $\left| \xi'(2^m, 2^{n+1}, x) - s\left(x - \frac{1}{8}\right) \right| \leq 2^{-m}$ and the property holds for $n+1$.

- The case $2^n < x < 2^n + \frac{1}{8}$ cannot happen with our constraint $x \in I_{[x]}$.

- If $2^n + \frac{1}{8} \leq x$ then

$$\left| F(2^{m+1}, 2^n, x) - (x - 2^n) \right| \leq 2^{-(m+1)}$$

and as s is 1-Lipschitz,

$$\left| s\left(F(2^{m+1}, 2^n, x) - \frac{1}{8}\right) - s\left(x - 2^n - \frac{1}{8}\right) \right| \leq \left| F(2^{m+1}, 2^n, x) - x + 2^n \right| \leq 2^{-(m+1)}.$$

Consequently, $\left| \xi'(2^m, 2^{n+1}, x) - s\left(x - 2^n - \frac{1}{8}\right) \right| \leq 2^{-m}$ and the property holds for $n+1$.

There remains to prove that the function ξ' is in \mathbb{LDL}° . Unfortunately, this is not clear from the recursive definition, but this can be written in another way, from which this follows. Indeed, we have from an easy induction that:

$$\xi'(2^m, 2^n, x) = F\left(2^{m+n-1}, 2^0, F\left(2^{m+n-2}, 2^1, F\left(2^{m+n-3}, 2^2, \dots, F(2^m, 2^{n-1}, x)\right)\right)\right),$$

if we define

$$F(2^m, 2^0, x) = \xi'(2^m, 2^0, x) = \frac{3}{4}\mathcal{C}\text{-}\mathfrak{s}\left(2^m, \frac{1}{8}, \frac{7}{8}, x\right).$$

Then, we can obtain $\xi'(2^m, 2^n, x) = H(2^m, 2^{n-1}, 2^n, x)$ with

$$\begin{aligned} H(2^m, 2^0, 2^n, x) &= F(2^m, 2^{n-1}, x) \\ H(2^m, 2^{t+1}, 2^n, x) &= F(2^{m+t}, 2^{n-1-t}, H(2^m, 2^t, 2^n, x)) \\ &= H(2^m, 2^t, 2^n, x) - 2^{n-1-t} \cdot \mathcal{C}\text{-}\mathfrak{s}\left(2^{m+t}, 2^{n-1-t}, 2^{n-1-t} \right. \\ &\quad \left. + \frac{1}{8}, H(2^m, 2^t, 2^n, x)\right) \end{aligned}$$

Such recurrence can be then seen as a linear length ordinary differential equation, in the length of its first argument. It follows that ξ' is in \mathbb{LDL}° . \square

From the construction of ξ , we can then obtain continuous approximations of the fractional part, the floor function, an adaptative barycenter, the modulo 2 and the Euclidian division by 2:

Corollary 3.2.10 (Uniformly controlled approximation of the fractional part)

There exists (see Figure 3.4) $\xi_1, \xi_2 : \mathbb{N}^2 \times \mathbb{R} \mapsto \mathbb{R} \in \text{LDL}^\circ$ such that, for all $n, m \in \mathbb{N}$, $[x] \in [-2^n + 1, 2^n]$,

- whenever $x \in \left[[x] - \frac{1}{2}, [x] + \frac{1}{4} \right]$,

$$|\xi_1(2^m, 2^n, x) - \{x\}| \leq 2^{-m},$$

- and whenever $x \in \left[[x], [x] + \frac{3}{4} \right]$,

$$|\xi_2(2^m, 2^n, x) - \{x\}| \leq 2^{-m}.$$

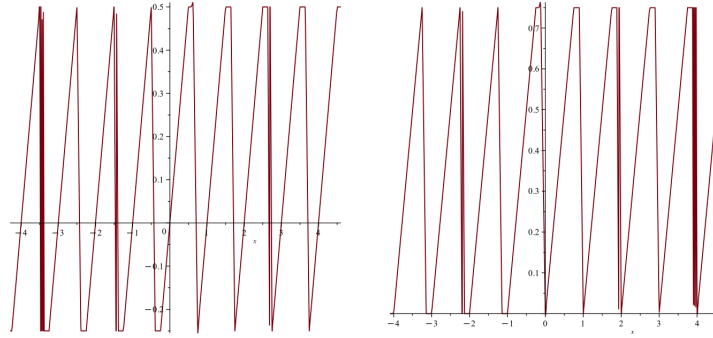


Figure 3.4: Graphical representation of $\xi_1(2, 4, x)$ and $\xi_2(2, 4, x)$ obtained with Maple.

Proof. Consider

$$\xi_1(2^m, N, x) = \xi\left(2^m, N, x - \frac{3}{8}\right) - \frac{1}{2} \quad \text{and} \quad \xi_2(2^m, N, x) = \xi\left(2^m, N, x - \frac{7}{8}\right).$$

□

Corollary 3.2.11 (Uniformly controlled approximation of the floor function)

There exists (see Figure 3.5) $\sigma_1, \sigma_2 : \mathbb{N}^2 \times \mathbb{R} \mapsto \mathbb{R} \in \text{LDL}^\circ$ such that, for all $n, m \in \mathbb{N}$, $[x] \in [-2^n + 1, 2^n]$,

- whenever $x \in \left[[x] - \frac{1}{2}, [x] + \frac{1}{4} \right]$,

$$|\sigma_1(2^m, 2^n, x) - [x]| \leq 2^{-m},$$

- and whenever $x \in I_2 = \left[[x], [x] + \frac{3}{4} \right]$,

$$|\sigma_2(2^m, 2^n, x) - [x]| \leq 2^{-m}.$$

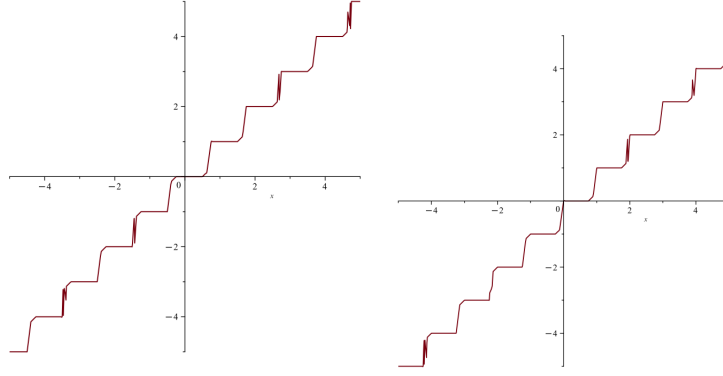


Figure 3.5: Graphical representation of $\sigma_1(2, 4, x)$ and $\sigma_2(2, 4, x)$ obtained with Maple.

Proof. Consider $\sigma_i(2^n, x) = x - \xi_i(2^n, x)$ with the function defined in Corollary 3.2.10.

□

Corollary 3.2.12 (Uniformly controlled approximation of the indicator function)

There exist (see Figure 3.6) $\lambda : \mathbb{N}^2 \times \mathbb{R} \mapsto [0, 1] \in \mathbb{LDL}^\circ$ such that for all $m, n \in \mathbb{N}$, $\lfloor x \rfloor \in [-2^n + 1, 2^n]$,

- whenever $x \in \left[\lfloor x \rfloor + \frac{1}{4}, \lfloor x \rfloor + \frac{1}{2} \right]$,

$$|\lambda(2^m, 2^n, x) - 0| \leq 2^{-m},$$

- and whenever $x \in \left[\lfloor x \rfloor + \frac{3}{4}, \lfloor x \rfloor + 1 \right]$,

$$|\lambda(2^m, 2^n, x) - 1| \leq 2^{-m}.$$

Proof. Consider $\lambda(2^m, 2^n, x) = F\left(\xi\left(2^{m+1}, 2^n, x - 9/8\right)\right)$ where

$$F(x) = \mathcal{C}\text{-}\mathfrak{s}\left(2^{m+1}, 1/4, 1/2, x\right).$$

□

Corollary 3.2.13 (Uniformly controlled approximation of the modulo 2)

There exists (see Figure 3.7) $\text{mod}_2 : \mathbb{N}^2 \times \mathbb{R} \mapsto [0, 1] \in \mathbb{LDL}^\circ$ such that for all $m, n \in \mathbb{N}$, $\lfloor x \rfloor \in [-2^n + 1, 2^n]$, whenever $x \in \left[\lfloor x \rfloor - \frac{1}{4}, \lfloor x \rfloor + \frac{1}{4} \right]$,

$$|\text{mod}_2(2^m, 2^n, x) - \lfloor x \rfloor \bmod 2| \leq 2^{-m}.$$

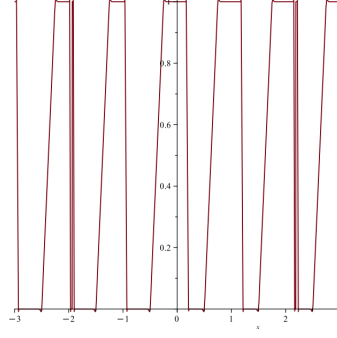


Figure 3.6: Graphical representation of $\lambda(2, 4, x)$ obtained with Maple.

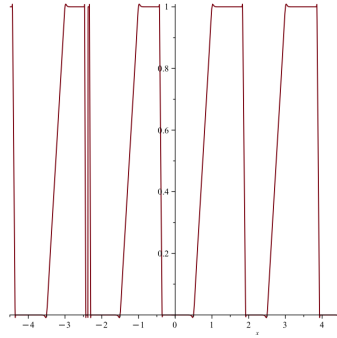


Figure 3.7: Graphical representation of $\text{mod}_2(2, 4, x)$ obtained with Maple.

Proof. We can take

$$\text{mod}_2(2^m, N, x) = 1 - \lambda\left(2^m, N/2, \frac{1}{2}x + \frac{7}{8}\right)$$

where λ is the function given by Corollary 3.2.12. \square

Corollary 3.2.14 (Uniformly controlled approximation of \div_2)

There exists (see Figure 3.8) $\div_2 : \mathbb{N}^2 \times \mathbb{R} \mapsto [0, 1] \in \mathbb{LDL}^\circ$ such that for all $m, n \in \mathbb{N}$, $\lfloor x \rfloor \in [-2^n + 1, 2^n]$, whenever $x \in \left[\lfloor x \rfloor - \frac{1}{4}, \lfloor x \rfloor + \frac{1}{4}\right]$,

$$|\div_2(2^m, 2^n, x) - \lfloor x \rfloor // 2| \leq 2^{-m},$$

where $//$ is the integer division.

Proof. We can take

$$\div_2(2^m, N, x) = \frac{1}{2}(\sigma_1(2^m, N, x) - \text{mod}_2(2^m, N, x))$$

where mod_2 is the function given by Corollary 3.2.13, and σ_2 is the function given by Corollary 3.2.11. \square

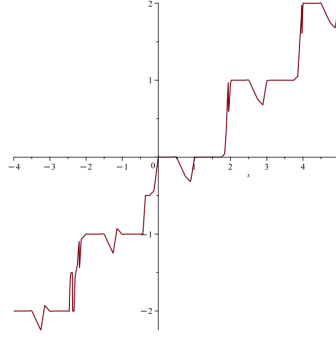


Figure 3.8: Graphical representation of $\div_2(2, 4, x)$ obtained with Maple.

Some properties of sigmoids

In this section, we encode a conditional function in $\mathbb{L}\mathbb{D}\mathbb{L}^\circ$. We observe that, for the function

$$\bar{\text{if}}(d, \ell) = 4\mathfrak{s}\left(1, 2, \frac{1}{2} + d + \ell/4\right) - 2,$$

with $\ell \in [0, 1]$, we have

$$\begin{aligned}\bar{\text{if}}(0, \ell) &= 0 \\ \bar{\text{if}}(1, \ell) &= \ell\end{aligned}$$

Applying Lemma 3.2.8 on this sigmoid, we get:

Lemma 3.2.15 (Generalisation of Lemma 3.1.16)

For $\ell \in [0, 1]$, if $d \in \left[-\frac{1}{4}, \frac{1}{4}\right]$, then $\bar{\text{if}}(d, \ell) = 0$, and if $d \in \left[\frac{3}{4}, \frac{5}{4}\right]$, then $\bar{\text{if}}(d, \ell) = 4(d - 1) + \ell$.

Consider $\text{if}(d, \ell) = \bar{\text{if}}\left(\mathfrak{s}\left(\frac{1}{4}, \frac{3}{4}, x\right), \ell\right)$.

- if we take $|d' - 0| \leq \frac{1}{4}$, then $\text{if}(d', \ell) = 0$,
- and if we take $|d' - 1| \leq \frac{1}{4}$, then $\text{if}(d', \ell) = \ell$.

Proof. Just check that for $d \leq \frac{1}{4}$, we have $\frac{1}{2} + d + \frac{\ell}{4} \leq 1$, and hence $\mathfrak{s}\left(1, 2, \frac{1}{2} + d + \frac{\ell}{4}\right) = 0$, so $\bar{\text{if}}(d, \ell) = 0$. For $\frac{3}{4} \leq d \leq \frac{5}{4}$, we have $\frac{5}{4} \leq \frac{1}{2} + d + \frac{\ell}{4} \leq 2$, and hence $\mathfrak{s}\left(1, 2, \frac{1}{2} + d + \frac{\ell}{4}\right) = d + \frac{\ell}{4} - \frac{1}{2}$, and hence $\bar{\text{if}}(d, \ell) = \ell$. The other observation follows. \square

Then we go to versions using tanh:

Lemma 3.2.16

Consider $\overline{\mathcal{C}\text{-if}}(2^m, d, \ell) = 4\mathcal{C}\text{-}\mathfrak{s}\left(2^{m+2}, 1, 2, \frac{1}{2} + d + \frac{\ell}{4}\right) - 2$.

For $\ell \in [0, 1]$, we have $\left|\overline{\mathcal{C}\text{-if}}(0, \ell) - 0\right| \leq 2^{-m}$ and $\left|\overline{\mathcal{C}\text{-if}}(1, \ell) - \ell\right| \leq 2^{-m}$.

If $d \in \left[-\frac{1}{4}, \frac{1}{4}\right]$, then $\left|\overline{\mathcal{C}\text{-if}}(2^m, d, \ell) - 0\right| \leq 2^{-m}$ and if $d \in \left[\frac{3}{4}, \frac{5}{4}\right]$, then $\left|\overline{\mathcal{C}\text{-if}}(2^m, d, \ell) - 4(d - 1) + \ell\right| \leq 2^{-m}$.

The proof of the previous lemma is direct from the Lemma 3.2.15 and Lemma 3.2.8. Then Lemma 3.2.15 follows.

We can now generalise Lemma 3.1.18, using only analytic functions:

Lemma 3.2.17 (Generalisation of Lemma 3.1.18)

Let $\alpha_1, \alpha_2, \dots, \alpha_n$ be some pairwise distinct integers, and V_1, V_2, \dots, V_n some constants. There is some function in \mathbb{LDL}° , that we write $\mathcal{C}\text{-send}(2^m, \alpha_i \mapsto V_i)_{i \in \{1, \dots, n\}}$, mapping any $x \in [\alpha_i - 1/4, \alpha_i + 1/4]$ to a real at a distance at most 2^{-m} of V_i , for all $i \in \{1, \dots, n\}$.

We see $\overline{\text{cond}}(x)$ as $\mathfrak{s}(1/4, 3/4, x)$ and can consequently consider $\overline{\mathcal{C}\text{-cond}}(2^m, x)$ as $\mathcal{C}\text{-}\mathfrak{s}(2^m, 1/4, 3/4, x)$. We prove the following lemma about some ideal sigmoids, approximated by analytic functions:

Lemma 3.2.18 (Generalisation of Lemma 3.1.19)

Assume you are given some pairwise distinct integers $\alpha_1, \alpha_2, \dots, \alpha_n$, and some constants V_1, V_2, \dots, V_n . Then there is some function in \mathbb{LDL}° , written

$$\text{send}(\alpha_i \mapsto V_i)_{i \in \{1, \dots, n\}},$$

mapping any $x \in [\alpha_i - 1/4, \alpha_i + 1/4]$ to V_i , for all $i \in \{1, \dots, n\}$.

Proof. Sort the α_i 's so that $\alpha_1 < \alpha_2 < \dots, \alpha_n$. Then consider $T_1 + \overline{\text{cond}}(x - \alpha_1)(T_2 - T_1) + \overline{\text{cond}}(x - \alpha_2)(T_3 - T_2) + \dots + \overline{\text{cond}}(x - \alpha_{n-1})(T_n - T_{n-1})$. \square

We can now go to versions with tanh.

Proof of Lemma 3.2.17. Sort the α_i 's so that $\alpha_1 < \alpha_2 < \dots, \alpha_n$. Then consider $T_1 + \overline{\mathcal{C}\text{-cond}}(2^{m+c}, x - \alpha_1)(T_2 - T_1) + \overline{\mathcal{C}\text{-cond}}(2^{m+c}, x - \alpha_2)(T_3 - T_2) + \dots + \overline{\mathcal{C}\text{-cond}}(2^{m+c}, x - \alpha_{n-1})(T_n - T_{n-1})$, for some constant c so that $n \max_j (T_j - T_{j+1}) \leq 2^c$. \square

More generally:

Lemma 3.2.19 (Continuous send)

Let $N \in \mathbb{N}$, $\alpha_1, \alpha_2, \dots, \alpha_n \in \mathbb{N}^n$, pairwise distinct, and, for $1 \leq i \leq n$ and $1 \leq j \leq N$, let $V_{i,j}$ be some constants. Then there exists some function

$$\mathcal{C}\text{-send}(2^m, (\alpha_i, j) \mapsto V_{i,j})_{i \in \{1, \dots, n\}, j \in \{1, \dots, N\}}$$

in \mathbb{LDL}° , such as, for all $i \in \{1, \dots, n\}$, $j \in \{1, \dots, N\}$, it maps any $x \in [\alpha_i - 1/4, \alpha_i + 1/4]$ and any $y \in [j - 1/4, j + 1/4]$ to a real number at a distance at most 2^{-m} of $V_{i,j}$.

Before proving Lemma 3.2.19, we prove the following lemma regarding some ideal sigmoids:

Lemma 3.2.20

Let N be some integer. Assume some pairwise distinct integers $\alpha_1, \alpha_2, \dots, \alpha_n$, and some constants $V_{i,j}$ for $1 \leq i \leq n$, and $0 \leq j < N$ are given. Then there is some function in \mathbb{LDL}° , that we write $\text{send}((\alpha_i, j) \mapsto V_{i,j})_{i \in \{1, \dots, n\}, j \in \{0, \dots, N-1\}}$, mapping any $x \in [\alpha_i - 1/4, \alpha_i + 1/4]$ and $y \in [j - 1/4, j + 1/4]$ to $V_{i,j}$, for all $i \in \{1, \dots, n\}$, $j \in \{0, \dots, N-1\}$.

Proof. If we define the function

$$\text{send}((\alpha_i, j) \mapsto V_{i,j})_{i \in \{1, \dots, n\}, j \in \{1, \dots, N\}}(x, y)$$

by $\text{send}(N\alpha_i + j \mapsto V_{i,j})_{i \in \{1, \dots, n\}, j \in \{1, \dots, N\}}(Nx + y)$ this works when $x = \alpha_i$ for some i .

Considering instead $\text{send}(N\alpha_i + j \mapsto V_{i,j})_{i \in \{1, \dots, n\}, j \in \{1, \dots, N\}}(N \text{send}(\alpha_i \mapsto \alpha_i)_{i \in \{1, \dots, n\}}(x) + y)$ works for any $x \in [\alpha_i - \frac{1}{4}, \alpha_i + \frac{1}{4}]$. \square

We then go to versions with \tanh :

Proof of Lemma 3.2.19. If we define the function

$$\mathcal{C}\text{-send}(2^m, (\alpha_i, j) \mapsto V_{i,j})_{i \in \{1, \dots, n\}, j \in \{1, \dots, N\}}(x, y)$$

by $\mathcal{C}\text{-send}(2^m, N\alpha_i + j \mapsto V_{i,j})_{i \in \{1, \dots, n\}, j \in \{1, \dots, N\}}(Nx + y)$ this works when $x = \alpha_i$ for some i .

Considering instead

$$\mathcal{C}\text{-send}(2^m, N\alpha_i + j \mapsto V_{i,j})_{i \in \{1, \dots, n\}, j \in \{1, \dots, N\}}(N \mathcal{C}\text{-send}(2^{m+c}, \alpha_i \mapsto \alpha_i)_{i \in \{1, \dots, n\}}(x) + y)$$

that works for any $x \in [\alpha_i - \frac{1}{4}, \alpha_i + \frac{1}{4}]$, for some constant c selected as above. \square

We now have continuous approximations of various discrete functions. We use them in the next section to have a continuous approximation of the execution of a Turing machine, thus proving our characterisation of **FPTIME** for functions over the reals.

3.2.2 Simulating Turing machines with functions of \mathbb{LDL}°

This section is devoted to the simulation of a Turing machine using some analytic functions and in particular functions from \mathbb{LDL}° . We use some ideas from Section 3.1 but with several improvements, as we need to deal with errors and avoid multiplications, as they are still not necessary in this framework.

First, the key point is to observe that we can write a function in \mathbb{LDL}° simulating a one-step reduction of the TM M . The idea is similar to what we did for \mathbb{LDL}^\bullet , except here we must deal with controlled errors.

Lemma 3.2.21

Given a TM M , there exists some function $\overline{\text{Next}}$ in \mathbb{LDL}° simulating one step of M : for any configuration C of M , writing C' the configuration after one transition in M , for all $m \in \mathbb{N}$, $\|\overline{\text{Next}}(2^m, \overline{C}) - \overline{C'}\| \leq 2^{-m}$.

Proof. We reuse the same construction as in the proof of Lemma 3.1.10 for LDL^\bullet , with the real numbers written this way on the tape:

$$\begin{array}{c} \overline{\dots \quad l^\bullet \quad l_0 \quad r_0 \quad r^\bullet \quad \dots} \\ \underbrace{\hspace{1.5cm}}_l \quad \underbrace{\hspace{1.5cm}}_r \end{array}$$

We also define $\text{next}q_a^q = q'$ if $\delta(q, a) = (q', \cdot, \cdot)$, i.e. values (q', x, m) for some x and $m \in \{\leftarrow, \rightarrow\}$.

As before, we can rewrite $\overline{\text{Next}}(q, \bar{l}, \bar{r}) = (q', \bar{l}', \bar{r}')$ as

$$\bar{l}' = \sum_{q, r_0} \left[\rightarrow (q, r_0) \left(\frac{\bar{l}}{4} + \frac{x}{4} \right) + \leftarrow (q, r_0) \{4\bar{l}\} \right]$$

and

$$\bar{r}' = \sum_{q, r_0} \left[\rightarrow (q, r_0) \{4\bar{r}\} + \leftarrow (q, r_0) \left(\frac{\{4r\}}{4^2} + \frac{x}{4^2} + \frac{\lfloor 4\bar{l} \rfloor}{4} \right) \right],$$

and, using notation of Lemma 3.2.19, $q' = \text{send}((q, r) \mapsto \text{next}q_r^q)_{q \in Q, r \in \{0, 1, 3\}}(q, \lfloor 4\bar{r} \rfloor)$.

Then, following the intuition of Remark 3.1.11, we can replace $\lfloor 4\bar{r} \rfloor$ by $\sigma(4\bar{r})$ if we take σ as some continuous function that would be affine and values respectively 0, 1 and 3 on $\{0\} \cup [1, 2] \cup [3, 4]$ (that is to say matches $\lfloor 4\bar{r} \rfloor$ on this domain). A possible candidate is $\sigma(x) = \mathfrak{s}\left(\frac{1}{4}, \frac{3}{4}, x\right) + \mathfrak{s}\left(\frac{9}{4}, \frac{11}{4}, x\right)$. Then considering $\xi(x) = x - \sigma(x)$, then $\xi(4\bar{r})$ would be the same as $\{4\bar{r}\}$: that is, considering $r_0 = \sigma(4\bar{r})$, replacing in the above expression every $\{4\cdot\}$ by $\xi(\cdot)$, and every $\lfloor \cdot \rfloor$ by $\sigma(\cdot)$, and get something that would still work the same, but using only continuous functions.

But here, we need to go to some analytic functions and not only continuous functions, and it is well-known that an analytic function that equals some affine function on some interval (e.g. on $[1, 2]$) must be affine, and hence cannot be 3 on $[3, 4]$. But the point is that we can try to tolerate errors and replace $\mathfrak{s}(\cdot, \cdot)$ by $\mathcal{C}\text{-}\mathfrak{s}(2^{m+c}, \cdot, \cdot)$ in the expressions above for σ and ξ , taking c such that $\left(3 + \frac{1}{4^2}\right) 3|Q| \leq 2^c$. This would just introduce some error at most $\left(3 + \frac{1}{4^2}\right) 3|Q|2^{-c}2^{-m} \leq 2^{-m}$.

We can also replace every $\rightarrow (q, r)$ in above expressions for \bar{l}' and \bar{r}' by

$$\mathcal{C}\text{-send}(k, (q, r) \mapsto \rightarrow (q, r)) (q, \sigma(4\bar{r})),$$

for a suitable error bound k , and symmetrically for $\leftarrow (q, r)$. However, if we do so, we still might have some multiplications in the above expressions.

The key is to use Lemma 3.2.15: we can also write the above expressions as

$$\begin{aligned} \bar{l}' = \sum_{q, r} & \left[\mathcal{C}\text{-if}\left(2^{m+c}, \mathcal{C}\text{-send}(2^2, (q, r) \mapsto \rightarrow (q, r))(q, \sigma(4\bar{r})), \frac{\bar{l}}{4} + \frac{x}{4}\right) \right. \\ & \left. + \mathcal{C}\text{-if}\left(2^{m+c}, \mathcal{C}\text{-send}(2^2, (q, r) \mapsto \leftarrow (q, r))(q, \sigma(4\bar{r})), \xi(4\bar{l})\right) \right] \end{aligned}$$

$$\bar{r}' = \sum_{q,r} \left[\mathcal{C}\text{-if} \left(2^{m+c}, \mathcal{C}\text{-send}(2^2, (q,r) \mapsto \rightarrow (q,r))(q, \sigma(4\bar{r})), \xi(4\bar{r}) \right) \right. \\ \left. + \mathcal{C}\text{-if} \left(2^{m+c}, \mathcal{C}\text{-send}(2^2, (q,r) \mapsto \leftarrow (q,r))(q, \sigma(4\bar{r})), \frac{\xi(4r)}{4^2} + \frac{x}{4^2} + \frac{\sigma(4\bar{l})}{4} \right) \right]$$

and still have the same bound on the error. \square

Once again, as for \mathbb{LDL}^\bullet , we can simulate one step and we would like to simulate some arbitrary computation of a Turing machine, by considering the iterations of function *Next*.

The problem with the above construction is that even if we start from the exact encoding \bar{C} of a configuration, it introduces some error (even if at most 2^{-m}). If we want to apply the function *Next* again, we will start not exactly from the encoding of a configuration.

Looking at the choice of the function σ , a small error can be tolerated (roughly if the process does not involve points at a distance less than $1/4$ of \mathcal{J}), but this error is amplified (roughly multiplied by 4 on some component), before introducing some new errors (even if at most 2^{-m}). The point is that if we repeat the process, very soon it will be amplified, up to a level where we have no true idea or control about what becomes the value of the above function.

However, if we know some bound on the space used by the Turing machine, we can correct it to get at most some fixed additive error: a Turing machine using a space S uses at most S cells to the right and the left of the initial position of its head.

Consequently, a configuration $C = (q, l, r)$ of such a machine involves words l and r of length at most S . Their encoding \bar{l} , and \bar{r} are expected to remain in \mathcal{J}_{S+1} . Consider $\text{round}_{S+1}(\bar{l}) = \lfloor 4^{S+1} \bar{l} \rfloor / 4^{S+1}$. For a point \bar{l} of \mathcal{J}_{S+1} , $4^{S+1} \bar{l}$ is an integer, and $\bar{l} = \text{round}_{S+1}(\bar{l})$. But now, for a point \tilde{l} at a distance less than $4^{-(S+2)}$ from a point $\bar{l} \in \mathcal{J}_{S+1}$, $\text{round}_{S+1}(\tilde{l}) = \bar{l}$. In other words, round_{S+1} “deletes” errors of order $4^{-(S+2)}$.

Consequently, we can replace every \bar{l} in the above expressions by

$$\sigma_1 \left(2^{2S+4}, 2^{2S+3}, 4^{S+1} \bar{l} \right) / 4^{S+1},$$

as this is close to $\text{round}_{S+1}(\bar{l})$, and the same for \bar{r} , where σ_1 is the function from Corollary 3.2.11. We could also replace m by $m + 2S + 4$ to guarantee that $2^{-m} \leq 4^{-(S+2)}$.

We get the following major improvement of the previous lemma:

Lemma 3.2.22 (Generalisation of Lemma 3.1.10)

*For any TM M , there exists a function $\overline{\text{Next}} \in \mathbb{LDL}^\circ$ simulating one step of M , i.e. computing the *Next* function sending any configuration \bar{C} of M to the next configuration \bar{C}' , such as $\|\text{Next}(2^m, 2^S, \bar{C}) - \bar{C}'\| \leq 2^{-m}$.*

Furthermore, it is robust to errors on its input, up to space S : considering $\|\tilde{C} - \bar{C}\| \leq 4^{-(S+2)}$, $\|\text{Next}(2^m, 2^S, \tilde{C}) - \bar{C}'\| \leq 2^{-m}$ remains true.

We can deduce the following proposition, adding a time complexity constraint:

Proposition 3.2.23 (Generalisation of Proposition 3.1.12)

Consider some Turing machine M computing some function $f : \Sigma^* \rightarrow \Sigma^*$ in some time $T(\ell(\omega))$ on input ω . Then, there exists some function $\tilde{f} : \mathbb{N}^2 \times \mathbb{R} \rightarrow \mathbb{R}$ in \mathbb{LDL}° such that $\|\tilde{f}(2^m, 2^{T(\ell(\omega))}, \gamma_{word}(\omega)) - \gamma_{word}(f(\omega))\| \leq 2^{-m}$.

Proof. The idea is to define the function \overline{Exec} that maps some time 2^t and some initial configuration C to the configuration at time t . It can be obtained using previous lemmas by

$$\begin{cases} \overline{Exec}(2^m, 0, 2^T, C) &= C \\ \overline{Exec}(2^m, 2^{t+1}, 2^T, C) &= \overline{Next}\left(2^m, 2^t, \overline{Exec}(2^m, 2^t, 2^T, C)\right). \end{cases}$$

We can then get the value of the computation as $\overline{Exec}(2^m, 2^{T(\ell(\omega))}, 2^{T(\ell(\omega))}, C_{init})$ on input ω , considering $C_{init} = (q_0, 0, \gamma_{word}(\omega))$. By applying some projection, we get $\tilde{f}(2^m, 2^T, y) = \pi_3\left(\overline{Exec}(2^m, 2^T, 2^T, (q_0, 0, y))\right)$ satisfying the property. \square

3.2.3 Converting integers and dyadics to words, and conversely

It is necessary to encode and decode the real numbers to do the computations, as in Section 3.1.1. As before, we need to prove that the encoding and decoding functions are in \mathbb{LDL}° .

Lemma 3.2.24 (From \mathbb{N} to \mathcal{I} , generalisation of Lemma 3.1.14)

There exists some function $Decode : \mathbb{N}^2 \rightarrow \mathbb{R}$ in \mathbb{LDL}° , mapping m and n to some point $\eta_{m,n}$ such that $\|\gamma_{word}(\bar{n}) - \eta_{m,n}\| \leq 2^{-m}$.

Proof. Recall the functions provided by Corollaries 3.2.13 and 3.2.14. The idea is to iterate $\ell(n)$ times the function

$$F(\bar{r}_1, \bar{l}_2) = \begin{cases} \left(\div_2(\bar{r}_1), \frac{\bar{l}_2+5}{4} \right) & \text{whenever } \text{mod}_2(\bar{r}_1) = 0 \\ \left(\div_2(\bar{r}_1), \frac{\bar{l}_2+7}{4} \right) & \text{whenever } \text{mod}_2(\bar{r}_1) = 1. \end{cases}$$

over $(n, 0)$, and then projects on the second argument.

This can be done in \mathbb{LDL}° by considering

$$\begin{aligned} F(2^m, 2^M, \bar{r}_1, \bar{l}_2) &= \mathcal{C}\text{-send}(2^{m+1}, 0 \mapsto (\div_2(2^{m+1}, 2^M, \bar{r}_1), (\bar{l}_2+5)/4), \\ &\quad 1 \mapsto (\div_2(2^{m+1}, 2^M, \bar{r}_1), (\bar{l}_2+7)/4))(\bar{r}_1) \end{aligned}$$

and then, we define

$$Decode(2^m, n) = \pi_2^2\left(G\left(2^{m+\ell(n)}, 2^{\ell(n)+1}, 2^{\ell(n)}, n, 0\right)\right),$$

with

$$\begin{cases} G(2^m, 2^M, 2^l, 2^0, n, 0) &= (n, 0) \\ G(2^m, 2^M, 2^{l+1}, r, l) &= F\left(2^m, 2^M, G(2^m, 2^l, r, l)\right). \end{cases}$$

The global error will be at most $2^{-m-\ell(n)} \times \ell(n) \leq 2^{-m}$. \square

This technique can be extended to consider decoding of tuples: there is a function $Decode : \mathbb{N}^{d+1} \rightarrow \mathbb{R}$ in \mathbb{LDL}° that maps m and \mathbf{n} to some point at distance less than 2^{-m} from $\gamma_{word}(\bar{\mathbf{n}})$, with $\bar{\mathbf{n}}$ defined componentwise.

Conversely, given \bar{d} , we need a way to construct d . As we will need to avoid multi-
 plications, we state that we can even do something stronger: given \bar{d} , and (some bounded) λ we can construct λd .

We give a generalisation of Lemma 3.1.13 in our new framework:

Lemma 3.2.25 (From \mathcal{I} to \mathbb{R} , multiplying in parallel)

We can construct some function $EncodeMul : \mathbb{N}^2 \times [0, 1] \times \mathbb{R} \rightarrow \mathbb{R}$ in \mathbb{LDL}° that maps $m, 2^S, \gamma_{word}(\bar{d})$ and (bounded) λ to some real at distance at most 2^{-m} from λd , whenever \bar{d} is of length less than S .

Proof. The idea is to do as in the proof of previous lemma, but considering

$$F(\bar{r}_1, \bar{l}_2, \lambda) = \begin{cases} \left(\sigma(16\bar{r}_1), 2\bar{l}_2 + 0, \lambda \right) & \text{whenever } i(16\bar{r}_1) = 5 \\ \left(\sigma(16\bar{r}_1), 2\bar{l}_2 + \lambda, \lambda \right) & \text{whenever } i(16\bar{r}_1) = 7 \\ \left(\sigma(16\bar{r}_1), (\bar{l}_2 + 0)/2, \lambda \right) & \text{whenever } i(16\bar{r}_1) = 13 \\ \left(\sigma(16\bar{r}_1), (\bar{l}_2 + \lambda)/2, \lambda \right) & \text{whenever } i(16\bar{r}_1) = 15 \end{cases}$$

iterated S times over suitable approximation of the rounding $\text{round}_{S+1}(\gamma_{word}(\bar{d}), 0, \lambda)$, with σ and ξ constructed as an approximation of the integer and fractional part, as before. \square

We have the continuous approximations in \mathbb{LDL}° of the functions we need to prove our characterisation of **FPTIME**.

3.2.4 Proof of Theorem 3.2.3: $\overline{\mathbb{LDL}^\circ}$ characterises **FPTIME** for real functions

We prove that: $\overline{\mathbb{LDL}^\circ} \cap \mathbb{R}^\mathbb{R} = \mathbf{FPTIME} \cap \mathbb{R}^\mathbb{R}$. We reason, as previously, by double inclusion.

$\mathbb{LDL}^\circ \subseteq \mathbf{FPTIME}$

Theorem 3.2.1 follows from the next Proposition for one inclusion and the previous simulation of Turing machines for the other.

Proposition 3.2.26

All functions of $\mathbb{L}\mathbb{D}\mathbb{L}^\circ$ are computable (in the sense of computable analysis) in polynomial time, with

$$\mathbb{L}\mathbb{D}\mathbb{L}^\circ = \left[\mathbf{0}, \mathbf{1}, \pi_i^k, \ell(x), +, -, \tanh, \frac{x}{2}, \frac{x}{3}; \text{composition, linear length ODE} \right].$$

Proof of Proposition 3.2.26. It is proved by induction. It is true for basis functions, from basic arguments from computable analysis. In particular, \tanh is computable in polynomial time from standard arguments. It is stable by composition (Lemma 1.3.56) and closed under the linear length ODE schema: it is Lemma 2.5.12. \square

We now go to various applications of the proposition and our toolbox. First, we state a characterisation of **FPTIME** for general functions, covering both the case of a function $\mathbf{f}: \mathbb{N}^d \rightarrow \mathbb{R}^{d'}$ and $\mathbf{f}: \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$ as a special case: only the first type (sequences) was covered by Section 3.1.

Theorem 3.2.27

A function $\mathbf{f}: \mathbb{R}^d \times \mathbb{N}^{d''} \rightarrow \mathbb{R}^{d'}$ is computable in polynomial time iff there exists $\tilde{\mathbf{f}}: \mathbb{R}^d \times \mathbb{N}^{d''+2} \rightarrow \mathbb{R}^{d'} \in \mathbb{L}\mathbb{D}\mathbb{L}^\circ$ such that for all $\mathbf{x} \in \mathbb{R}^d$, $X \in \mathbb{N}$, $\mathbf{x} \in [-2^X, 2^X]$, $\mathbf{m} \in \mathbb{N}^{d''}$, $n \in \mathbb{N}$, $\|\tilde{\mathbf{f}}(\mathbf{x}, \mathbf{m}, 2^X, 2^n) - \mathbf{f}(\mathbf{x}, \mathbf{m})\| \leq 2^{-n}$.

The reverse implication of Theorem 3.2.27 follows from Proposition 3.2.26, (1.) and arguments from computable analysis, as for $\mathbb{L}\mathbb{D}\mathbb{L}^\bullet$.

Proof of Theorem 3.2.27. Assume there exists $\tilde{\mathbf{f}}: \mathbb{R}^d \times \mathbb{N}^{d''+2} \rightarrow \mathbb{R}^{d'} \in \mathbb{L}\mathbb{D}\mathbb{L}^\circ$ such that for all $\mathbf{x} \in \mathbb{R}^d$, $X \in \mathbb{N}$, $\mathbf{x} \in [-2^X, 2^X]$, $\mathbf{m} \in \mathbb{N}^{d''}$, $n \in \mathbb{N}$, $\|\tilde{\mathbf{f}}(\mathbf{x}, \mathbf{m}, 2^X, 2^n) - \mathbf{f}(\mathbf{x}, \mathbf{m})\| \leq 2^{-n}$.

From Proposition 3.2.26, (1.), we know that $\tilde{\mathbf{f}}$ is computable in polynomial time (in the binary length of its arguments). Then $\mathbf{f}(\mathbf{x}, \mathbf{m})$ is computable: indeed, given \mathbf{x} , \mathbf{m} and n , we can approximate $\mathbf{f}(\mathbf{x}, \mathbf{m})$ at precision 2^{-n} on $[-2^X, 2^X]$ as follows: approximate $\tilde{\mathbf{f}}(\mathbf{x}, \mathbf{m}, 2^X, 2^{n+1})$ at precision $2^{-(n+1)}$ by some rational q , and output q . We will then have

$$\begin{aligned} \|q - \mathbf{f}(\mathbf{x}, \mathbf{m})\| &\leq \|q - \tilde{\mathbf{f}}(\mathbf{x}, \mathbf{m}, 2^X, 2^{n+1})\| + \|\tilde{\mathbf{f}}(\mathbf{x}, \mathbf{m}, 2^X, 2^{n+1}) - \mathbf{f}(\mathbf{x}, \mathbf{m})\| \\ &\leq 2^{-(n+1)} + 2^{-(n+1)} \\ &\leq 2^{-n}. \end{aligned}$$

All of this is done in polynomial time in n and the size of \mathbf{m} , and hence we get that \mathbf{f} is polynomial time computable from definitions.

For the direct implication, for real sequences, we are almost done: reasoning componentwise, we only need to consider $f: \mathbb{N}^{d''} \rightarrow \mathbb{R}$ (i.e. $d' = 1$). As the function is polynomial time computable, this means that there is a polynomial time computable function $g: \mathbb{N}^{d''+1} \rightarrow \{1, 3\}^*$ so that on $\mathbf{m}, 2^n$, it provides the encoding $\overline{\phi(\mathbf{m}, n)}$ of some dyadic $\phi(\mathbf{m}, n)$ with $\|\phi(\mathbf{m}, n) - f(\mathbf{m})\| \leq 2^{-n}$ for all \mathbf{m} .

This is basically what is done in Subsection 3.1.2, except that we do it here with analytic functions. However, as already observed in Section 3.1, this cannot be done for the case $d \geq 1$, e.g. for $f: \mathbb{R} \rightarrow \mathbb{R}$. The problem is we used the fact that we can decode:

Decode maps an integer n to its encoding \bar{n} (but is not guaranteed to do something valid on non-integers). There cannot exist such functions that would be valid over all reals, as such functions must be continuous, and there is no way to map continuously real numbers to finite words. It is where the approach of Section 3.1 for $\mathbb{L}\mathbb{D}\mathbb{L}^\bullet$ is stuck.

The problem is then to decode, compute and encode the result to produce this dyadic number, using our previous toolbox.

More precisely, from Proposition 3.2.23, we get \tilde{g} with

$$\left| \tilde{g}\left(2^e, 2^{p(\max(\mathbf{m}, n))}, \text{Decode}(2^e, \mathbf{m}, n)\right) - \gamma_{\text{word}}(g(\mathbf{m}, n)) \right| \leq 2^{-e}$$

for some polynomial p corresponding to the time required to compute g and $e = \max(p(\max(\mathbf{m}, n)), n)$. We need to transform the value to the correct dyadic: we mean

$$\tilde{\mathbf{f}}(\mathbf{m}, n) = \text{EncodeMul}\left(2^e, 2^t, \tilde{g}\left(2^e, 2^t, \text{Decode}(2^e, \mathbf{m}, n)\right), 1\right),$$

where $t = p(\max(\mathbf{m}, n))$, $e = \max(p(\max(\mathbf{m}, n)), n)$ provides a solution such that $\|\tilde{\mathbf{f}}(\mathbf{m}, 2^n) - \mathbf{f}(\mathbf{m})\| \leq 2^{-n}$. \square

To solve this, we use an adaptive barycentric technique. For simplicity and pedagogy, we discuss only the case of a polynomial time computable function $f : \mathbb{R} \times \mathbb{N} \rightarrow \mathbb{R}$. From standard arguments from computable analysis (see e.g. [Corollary 2.21][Ko91]), the following holds and the point is to be able to realize all this with functions from $\mathbb{L}\mathbb{D}\mathbb{L}^\circ$.

Lemma 3.2.28

Assume $f : \mathbb{R} \times \mathbb{N} \rightarrow \mathbb{R}$ is computable in polynomial time. There exists some polynomial $m : \mathbb{N}^2 \rightarrow \mathbb{N}$ and some $\tilde{f} : \mathbb{N}^4 \rightarrow \mathbb{Z}$ computable in polynomial time such that for all $x \in \mathbb{R}$, $|2^{-n} \tilde{f}(\lfloor 2^{m(n, M)} x \rfloor, u, 2^M, 2^n) - f(x, u)| \leq 2^{-n}$ whenever $\frac{x}{2^{m(n, M)}} \in [-2^M, 2^M]$.

Considering an approximation σ_i (with either $i = 1$ or $i = 2$) of the floor function given by Lemma 3.2.11. Then, given n, M , when $2^{m(n, M)} x$ falls in some suitable interval I_i for σ_i (see the statement of Lemma 3.2.11), we are sure $\sigma_i(2^e, 2^{m(n, M)+X+1}, 2^{m(n, M)} x)$ is at some distance upon control from $\lfloor 2^{m(n, M)} x \rfloor$. Consequently,

$$2^{-n} \tilde{f}\left(\sigma_i\left(2^{m(n, M)+X+1}, 2^{m(n, M)} x\right), u, 2^M, 2^n\right)$$

provides some 2^{-n} -approximation of $f(x, u)$, up to some error upon control. When this holds, we then use an argument similar to what we describe for sequences: using functions from $\mathbb{L}\mathbb{D}\mathbb{L}^\circ$, we can decode, compute, and encode the result to provide this dyadic. It is provided by an expression $\text{Formula}_i(x, u, M, n)$ of the form $\text{EncodeMul}(2^e, 2^t, \tilde{f}(2^e, 2^t, \text{Decode}(2^e, \sigma_i(2^e, 2^{m(n, M)+X+1}, 2^{m(n, M)} x))), 2^{-n})$.

The problem is that it might also be the case that $2^{m(n, M)} x$ falls in the complement of the intervals $(I_i)_i$. In that case, we have no clear idea of what could be the value of $\sigma_i(2^e, 2^{m(n, M)+X+1}, 2^{m(n, M)} x)$, and consequently of what might be the value of the above expression $\text{Formula}_i(x, u, M, n)$. But the point is that when it happens for an x for σ_1 , we could have used σ_2 , and this would work, as one can check that the intervals of type I_1 cover the complements of the intervals of type I_2 and conversely. They also overlap,

but when x is both in some I_1 and I_2 , $Formula_1(x, u, M, n)$ and $Formula_2(x, u, M, n)$ may differ, but they are both 2^{-n} approximations of $f(x)$.

The key is to compute some suitable "adaptive" barycenter, using function λ , provided by Corollary 3.2.12. Writing \approx for the fact that two values are closed up to some controlled bounded error, observe from the statements of Corollary 3.2.12 and 3.2.11

- that whenever $\lambda(\cdot, 2^n, x) \approx 0$, we know that $\sigma_2(\cdot, 2^n, x) \approx \lfloor x \rfloor$;
- that whenever $\lambda(\cdot, 2^n, x) \approx 1$ we know that $\sigma_1(\cdot, 2^n, x) \approx \lfloor x \rfloor$;
- that whenever $\lambda(\cdot, 2^n, x) \in (0, 1)$, we know that $\sigma_1(\cdot, 2^n, x) \approx \lfloor x \rfloor + 1$ and $\sigma_2(\cdot, 2^n, x) \approx \lfloor x \rfloor$.

That means that if we consider

$$\lambda(\cdot, 2^n, x)Formula_1(x, u, M, n) + (1 - \lambda(\cdot, 2^n, x))Formula_2(x, u, M, n)$$

we are sure to be close (up to some bounded error) to some 2^{-n} approximation of $f(x)$. There remains that this requires some multiplication with λ . But from the form of $Formula_i(x, u, M, n)$, this could be also be written as follows, ending the proof of Theorem 3.2.27.

$$\begin{aligned} & EncodeMul \left(2^e, 2^t, \tilde{f} \left(2^e, 2^t, Decode \left(2^e, \sigma_1(2^e, 2^M, 2^{m(n, M)} x) \right) \right), \lambda \left(2^e, 2^M, 2^{m(n, M)} x \right) 2^{-n} \right) + \\ & EncodeMul \left(2^e, 2^t, \tilde{f} \left(2^e, 2^t, Decode \left(2^e, \sigma_2(2^e, 2^M, 2^{m(n, M)} x) \right) \right), \left(1 - \lambda \left(2^e, 2^M, 2^{m(n, M)} x \right) \right) 2^{-n} \right) \end{aligned} \quad (3.1)$$

NB 3.2.29

The formula (3.1) can be seen as a function that generates uniformly a family of circuits/formal \mathcal{C} -s approximating a given function at some given precision over some given domain. The functions we obtain are the composition of essentially linear functions, which can be considered as layers of formal neural networks using a concept not assuming the last layer of the network to be made of neurons and that result may be outputted by some linear combination of the neurons in the final layer.

Which enables us to prove Theorem 3.2.1:

Proof of Theorem 3.2.1. We know that a function $\mathbf{f} : \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$ from $\mathbb{L}\mathbb{D}\mathbb{L}^\circ$ is polynomial time computable by Proposition 3.2.26, (1.). That means we can approximate it with arbitrary precision, in particular, precision $\frac{1}{4}$ in polynomial time. Given such an approximation \mathbf{q} , if we know it is some integer, it is easy to determine which integer it is: return (componentwise) the closest integer to \mathbf{q} .

Conversely, if we have a function $\mathbf{f} : \mathbb{N}^d \rightarrow \mathbb{N}^{d'}$ that is polynomial time computable, our previous simulations of Turing machines provide a function in $\mathbb{L}\mathbb{D}\mathbb{L}^\circ$ that computes it at any required precision, in particular $1/4$. \square

From the fact that we have the reverse direction in Theorem 3.2.27, it is natural to consider the operation that maps $\tilde{\mathbf{f}}$ to \mathbf{f} .

Theorem 3.2.30

A continuous function \mathbf{f} is computable in polynomial time if and only if all its components belong to $\overline{\text{LDL}}^\circ$, where

$$\overline{\text{LDL}}^\circ = \left[\mathbf{0}, \mathbf{1}, \pi_i^k, \ell(x), +, -, \tanh x, \frac{x}{2}, \frac{x}{3}; \text{composition, linear length ODE, } E\text{Lim} \right]$$

with $E\text{Lim}$ defined in Definition 3.2.2.

For the reverse direction, by induction, the only thing to prove is that the class of functions from reals to integers computable in polynomial time is preserved by the operation $E\text{Lim}$. The proof is the same as Proposition 3.1.3. This also gives directly Theorem 3.1.6 as a corollary.

From the proofs, we also get a normal form theorem, namely formula (3.1). In particular,

Theorem 3.2.31

Any function $f : \mathbb{N}^d \times \mathbb{R}^{d''} \rightarrow \mathbb{R}^{d'}$ can be obtained from the class $\overline{\text{LDL}}^\circ$ using only one schema $E\text{Lim}$.

3.3 Chapter Conclusion

In this chapter, we proved that real sequences and functions over the reals, computable in polynomial time, can be algebraically characterised using discrete ordinary differential equations (ODE). We have provided algebras $\overline{\text{LDL}}^\bullet$ for $\mathbf{FPTIME} \cap \mathbb{R}^\mathbb{N}$ (Theorem 3.1.6) and $\overline{\text{LDL}}^\circ$ for $\mathbf{FPTIME} \cap \mathbb{R}^\mathbb{R}$ (Theorem 3.2.3).

Actually, our characterisations also cover $\mathbf{FPTIME} \cap \mathbb{R}^{\mathbb{N}^d \times \mathbb{R}^{d'}}$ for integers d and d' . A major improvement of our characterisation of functions over the reals with $\overline{\text{LDL}}^\circ$, in comparison to the one for real sequences, is that it uses only analytic functions (i.e. no need for $\text{sg}(\cdot)$ function). It required continuous approximations of several discontinuous or non-analytic functions (floor function, the Euclidean division, etc.). Furthermore, it also required a barycentric method, inspired by some constructions of [BCGSH07].

In a more abstract view, the core of the proofs was proving that we can simulate Turing machines with analytic discrete ordinary differential equations. We believe this result opens the way to many applications, as it opens the possibility of programming with (discrete time) ordinary differential equations, with an underlying well-understood time and space complexity. We will use this in Chapter 5 to algebraically characterise $\mathbf{FPSPACE}$ with discrete and continuous ODEs.

Chapter 4

Robustness and applications

Det är underligt med vägar och floder, funderade Sniff, man ser dem gå förbi och får en hemsk lust att vara nån annanstans. Att följa med och se var de slutar...

Tove Jansson, Kometen Kommer

NB 4.0.1

This chapter is based on an article published in CSL 2024 ([BB24c]), co-authored with Olivier Bournez, except Section 4.6, which is co-authored with Nathalie Aubrun and Olivier Bournez. A journal version of the CSL article has been submitted.

Reasoning about dynamical systems evolving over the reals is well-known to lead to undecidability. In particular, there are no reachability decision procedures for first-order theories over the reals extended with even very basic functions, for logical theories reasoning about real-valued functions, or decision procedures for state reachability. It mostly comes from the fact that reachability for dynamical systems over the reals is undecidable, as Turing machines can be embedded into dynamical systems.

However, various results in the literature have shown that decision procedures exist when restricting to robust systems, with a suitably-chosen notion of robustness. In particular, in the field of verification, if the state reachability is not sensitive to infinitesimal perturbations, then decision procedures for state reachability exist. In the context of logical theories over the reals, decision procedures exist if we focus on properties not sensitive to arbitrarily small perturbations. For example, by considering properties that are either true or δ -far from being true for some $\delta > 0$.

In this chapter, we first propose a unified theory explaining in a uniform framework these statements, established in different contexts. We will extend the results of this chapter in Chapter 6.

More fundamentally and additionally, while all the previous statements were only about computability issues, we consider complexity theory aspects. We prove that robustness to some precision is inherently related to the complexity of the decision procedure. When a system is robust, we can quantify the level of perturbation we allow: assuming robustness to a polynomial perturbation on precision leads to a characterisation of **PSPACE**. Furthermore, assuming robustness to polynomial perturbation on time or length leads to similar statements for **PTIME**.

In other words, precision on computations is related to space complexity, while length or time of trajectories, is intrinsically related to time complexity.

Generally speaking, considering dynamical systems with respect to infinitesimal perturbations of dynamics is an old idea, sometimes with concepts reinvented later with other names, such as *noisy* dynamical systems (e.g. [BRS15]). In the field of verification, the idea of infinitely perturbed dynamics has been considered to provide alternative semantics of some models: see e.g. [Pur00] for timed automata. The approaches considered in [AB01] and [GAC12] belong to the line of investigation considering general dynamical systems and aiming at studying the frontier between decidability and undecidability. Somehow, our results state that the uncomputability discussed in [RS23] is intrinsically due to the non-numerical stability of the considered dynamical systems there.

To our knowledge, such a unifying framework has never been established. For the computability aspects, regarding some of the existing works: compared to [AB01], we allow more general discrete-time and continuous-time dynamical systems, such as those with unbounded domains. Some generalisations have also been obtained in [BGH10], but focusing on dynamical systems as language recognisers and mainly focusing on generalisations of [AB01, Theorem 4]. The logic considered in [GAC12] allows us to comment on finite-time reachability properties, but not reachable sets. As far as we know, complexity aspects have never been discussed this way. In [BRS15], the authors study the space complexity of computing the invariant measure of a dynamical system submitted to a Gaussian noise, but assume that the dynamic must not be an additional source of computational complexity. We assume here a somewhat more general type of perturbation and adopt a point of view of graph theory.

In Section 4.1, we review some existing results, due to [AB01] about the definitions and computability properties of perturbed TMs. Using the constructions of Section 2.3.1, we study the (polynomial) space complexity of the reachability relation in discrete-time rational dynamical systems in Section 4.2. We extend those properties to discrete-time and continuous-time real dynamical systems (Section 4.3 and Section 4.4). We study the time complexity of the same relation in Section 4.5. In Section 4.6, we study the robustness of TMs, using Haore’s logic.

4.1 Perturbed TMs

Our theory relies on consideration from [AB01] and some well-known observations from complexity theory. We start by recalling some facts and a few basic concepts.

Article [AB01] introduces the concept of space-perturbed TM: given $n > 0$, the idea is that the n -perturbed version of the machine M is unable to remain correct at a distance more than n from the head of the machine. Formally, the n -perturbed version M_n of M is defined exactly as M except before any transition, all the symbols at a distance n or more from the head can be altered at every step. Hence M_n is nondeterministic.

A word w is accepted by M_n iff there exists a run of this machine stopping in an accepting state. Let $L(M)$ be the language recognised by a TM M . Let $L_n(M)$ be the n -perturbed language of M . From definitions, if a word is accepted by M , then it is also recognised by all the M_n ’s: perturbed machines have more behaviours. Moreover, $L_{n+1}(M) \subseteq L_n(M)$.

Let $L_\omega(M) = \bigcap_n L_n(M)$: this is the set of words accepted by M when subject to arbitrarily “small” perturbations. $L_\omega(M)$ is called the ω -perturbed language of M .

We have $L(M) \subseteq L_\omega(M) \subseteq \dots \subseteq L_2(M) \subseteq L_1(M)$.

Here is a key observation: a TM's ω -perturbed language is co-computably enumerable:

Theorem 4.1.1 (Perturbed reachability is co-c.e. [AB01])

$$L_\omega(M) \in \Pi_1^0.$$

Proof. We consider the definition of TM similar to Subsection 2.3.1. Given a bi-infinite configuration C of M of the form

$$(q, \dots a_{-n-1} a_{-n} \dots a_{-1}, a_0 a_1 \dots a_n a_{n+1} \dots),$$

we define $\varphi_n(C) = (q, a_{-n} \dots a_{-1}, a_0 a_1 \dots a_n) \in Q \times \Sigma^n \times \Sigma^{n+1}$ made of a state and words of length n and $n+1$. We denote by $C[w]$, where $w = a_1 a_2 \dots a_n$ the configuration $(q, \dots BBB, a_1 a_2 \dots a_n BBB \dots)$, and $C_0[w]$ when $q = q_{init}$.

For every $n \in \mathbb{N}$, we associate to the n -perturbed version M_n of TM M some graph $G_n = (V_n, \rightarrow_n)$. The vertices, denoted $(\mathcal{V}_i)_i$, of G_n correspond to the $|Q| \times |\Sigma + 1|^{2n+1}$ possible values of $\varphi_n(C)$ for a configuration C of M . There is an edge between \mathcal{V}_i and \mathcal{V}_j in G_n iff there exist configurations C and C' such that $\varphi_n(C) = \mathcal{V}_i$ and $\varphi_n(C') = \mathcal{V}_j$ and M_n can go from configuration C to configuration C' in one step. In other words, G_n simulates the execution of M_n .

Determining whether $\mathcal{V}_i \rightarrow \mathcal{V}_j$ holds is easy (and in particular polynomial space computable) by considering that, when the head is moved to the left (resp. to the right) of \mathcal{V}_i a symbol in $\Sigma \cup \{B\}$ is non-deterministically chosen and appended to the left (resp. right) of the configuration and the right-most (resp. left-most) one is lost (it belongs now to the perturbed area of the configuration and hence it can be replaced by any other symbol).

Let $F_n = \varphi_n(Q_{accept})$ correspond to the accepting control states. By construction, the n -perturbed version M_n of M has an accepting run starting from a configuration C , iff F_n is reachable from $\varphi_n(C)$, that is to say $\text{PATH}(G_n, \varphi_n(C), F_n)$. By Corollary 2.3.4, this is decidable in a space polynomial in n .

Let Basis_n be the finite set of sequences $(s_n)_{n \in \mathbb{N}} \in \Sigma^n$, such that F_n is reachable, for all $n \in \mathbb{N}$, from $C_0[s_n]$. Let Short_n be the finite set of sequences $s_k \in \Sigma^k$ with $k < n$, such that F_n is reachable from $C_0[s_k]$. Then $L_n(M) = (\text{Short}_n \cup \text{Basis}_n) \cap \Sigma^*$. Consequently, $L_n(M)$ is decidable in space polynomial in n and hence its complement also is. Thus, $L_\omega(M)$ is co-c.e., as it is a (uniform in n) intersection of decidable sets. \square

Since a set c.e. and co-c.e. is decidable, following [AB01], we define robustness as:

Definition 4.1.2 (Robust Language)

A language L is robust if there exists a deterministic Turing machine M such that $L = L_\omega(M) = L(M)$.

Definition 4.1.3 (Robust Machine)

A TM M is robust if $L(M) = L_\omega(M)$.

Note that there could be non-robust TM that also have L as accepting language. With this definition, robust languages are necessarily decidable (i.e. the “robustness conjecture” holds). Actually, robustness is close to decidability for machines that always halt.

From definitions, if M always halts, then $L(M)$ is decidable and $L_\omega(M) = L(M)$.

Corollary 4.1.4 (Robust \approx decidable [AB01])

If $L_\omega(M) = L(M)$ then $L(M)$ is decidable.

Thus, a language L is decidable iff it is robust. A direct corollary is that a language L is undecidable iff it is not robust. Conversely, we think it is instructive to reason on a non-robust TM. Since we always have $L(M) \subseteq L_\omega(M)$, we have that $L_\omega(M) \neq L(M)$ iff there exists some word $w \in L_\omega(M)$, but $w \notin L(M)$.

Lemma 4.1.5

$L_\omega(M) \not\subseteq L(M)$ iff there is a word w nor accepted nor rejected by M , but accepted by any n -perturbed version M_n .

Proof. It is sufficient to prove that some word $w \in L_\omega(M)$, with $w \notin L(M)$ satisfies this. Since $w \notin L(M)$, w is not accepted by M . As $w \in L_\omega(M)$, w is accepted by any n -perturbed version M_n . Necessarily, such a w is also not rejected by M : it was, this must happen in finite time for M , using finitely many cells of the TM, so, by definition, there exists an n sufficiently big such as the behaviour of M_n is similar to the one of M on w on these cells, and hence M_n also necessarily reaches the rejecting state q_{reject} on input w . Once in state q_{reject} , from definitions, M_n stops: as M , its internal state and its heads do not move anymore, and there is no way that w becomes accepted. So, we get that necessarily, we cannot have w accepted by M_n for n sufficiently big, in contradiction with $w \in L_\omega(M)$. \square

In general, ω -perturbed languages are not c.e. [AB01]:

Proposition 4.1.6 ([AB01])

For any TMM, we can effectively construct another TMM' such that $L_\omega(M') = L(M)^c$.

A simple point, but a key observation for coming discussions, is the following: one can talk about complexity and not only computability. Indeed, when a language is robust, it makes sense to measure what level of perturbation s can be tolerated. This is the purpose of Definition 4.1.8.

NB 4.1.7

Assume L is robust, so there exists a TM M such that $L = L(M) = L_\omega(M)$. This means that for any word w , there must exist some n (depending possibly on w) such that $w \in L(M)$ and $w \in L_n(M)$ have the same truth value. This n can be read as the associated tolerated level of perturbation. It quantifies the tolerated level of robustness. Now, we can always consider that this function that associates n to w depends only on its length (as there are finitely many words of a given length, and we can always replace n with a bigger n).

Formally, assuming a language $L = L(M)$, for some TM M , is robust means that $L = L(M) = L_\omega(M)$. Let us consider some length ℓ , and reason about words of length ℓ (i.e. about words of Σ^ℓ where Σ is the alphabet of the Turing machine). We must have $L \cap \Sigma^\ell = L(M) \cap \Sigma^\ell = L_\omega(M) \cap \Sigma^\ell$. Now, by definition of $L_\omega(M)$, $L_\omega(M) \cap \Sigma^\ell$ is necessarily $L_n(M) \cap \Sigma^\ell$ for some n . Consequently, we must have $L(M) \cap \Sigma^\ell = L_n(M) \cap \Sigma^\ell$ for some $n = s(\ell)$ for a robust language.

In other words, for a robust language, we have necessarily

$$L = L(M) = L_{\{s\}}(M)$$

for some function s , for the coming definition. This function $n = s(\ell)$ quantifies the tolerated level of robustness. If one prefers, a robust language is necessary s -robust for some s , according to Definition 4.1.10.

Definition 4.1.8 (Level of robustness n given by s)

Given a function $s : \mathbb{N} \rightarrow \mathbb{N}$, we write $L_{\{s\}}(M)$ for the set of words accepted by M with space perturbation s :

$$L_{\{s\}}(M) = \{w \mid w \in L_{s(\ell(w))}(M)\}.$$

Lemma 4.1.9

For any function $s : \mathbb{N} \rightarrow \mathbb{N}$ and any deterministic Turing machine M , $L_\omega(M) \subseteq L_s(M)$.

Definition 4.1.10 (s -robust language)

For any function $s : \mathbb{N} \rightarrow \mathbb{N}$, a robust language is s -robust if there exists a TM M such that $L = L(M) = L_{\{s\}}(M)$.

It is natural to consider the case where the function s is a polynomial. It turns out that this corresponds to (and is even a characterisation of) **PSPACE**:

Theorem 4.1.11 (Polynomial robustness \Leftrightarrow PSPACE)

$L \in \mathbf{PSPACE}$ iff for some M and some polynomial p , $L = L(M) = L_{\{p\}}(M)$.

Proof. (\Rightarrow) Let a language $L \in \mathbf{PSPACE}$. Thus, there exists a polynomial p and a deterministic TM M working in space p and recognising L : we have $L = L(M)$. We claim that $L(M) = L_{\{p\}}(M)$. It is clear from our definition that $L(M) \subseteq L_\omega(M) \subseteq L_{\{p\}}(M)$. For the other direction, we consider $w \in L_{\{p\}}(M)$. We want to show that $w \in L(M)$. We assume w is of length n . Since $w \in L_{\{p\}}(M)$, $w \in L_{p(n)}(M)$, so $M_{p(n)}$ accepts w . But, on input w , M uses at most $p(n)$ cells, so the only run of M on w does not visit any symbol that could be altered non-deterministically during any run of $M_{p(n)}$ on w . The same applies to any run of $M_{p(n)}$ on w . Hence, the run of M on input w must also be accepting, thus $w \in L(M) = L$.

(\Leftarrow) For any TM M and any polynomial p , $L_{p(n)}(M) \in \mathbf{NPSPACE}$ is direct from the definitions. By Savitch's theorem, $\mathbf{NPSPACE} = \mathbf{PSPACE}$, so $L_{p(n)}(M) \in \mathbf{PSPACE}$. \square

Non-robust machines

We think it is very instructive to realise that it is not that easy to imagine, in a constructive way, a machine (or c.e language) that is non-robust. First of all, from definitions and previous discussions, any machine that always halts is robust. Furthermore, a machine that would loop over a finite set of configurations is also, from definition, robust (take n

sufficiently big, etc). The same is true if we relax and say that it enters a non-possibly ending loop over finitely many of its internal states. So, non-robustness has to do with the existence of inputs for which the machine does not reach an accepting state, without even looping, in many strong senses of “looping”. So, a simple example of non-robust TM?

Example 4.1.12 (An example of non-robust Turing machine)

Consider a TM M that zig-zags writing zeros if it reads a zero, and reach an accepting state if it reads a 1. The language $L(M)$ is the set of words containing a 1. For all $n \in \mathbb{N}$, $L_n(M)$ contains 0^n . Thus M is not robust. It is an example of a decidable language with a non-robust machine.

By some argument of countability or by the fact that a computably enumerable and co-computably enumerable set is recursive and that the halting problem of Turing machines is computable enumerable and non-recursive, we can deduce that there are some non-robust Turing machines.

The following example, inspired by [Koz97], is based on two well-known facts from computability and logic.

1. From Kleene’s recursion theorem ([Kle38]), we can consider that a Turing machine can obtain its own code, using the terminology and its presentation in [Sip97].
2. From logic that given the code m of some Turing machine M , we can explicitly write some first-order arithmetic formula $\gamma_{m,w}$ that is true if M accepts input w : this formula states, using the β -function from Gödel, that there is some finite sequence of successive configurations of M , starting from the initial configuration corresponding to w , ending with some accepting configuration. This is the formula considered in Remark 4.6.7. We write ε for the empty word.

Example 4.1.13 (Another example of non-robust Turing machine)

Let M be the Turing machine that, on any input w and using the recursion theorem, M obtains its code m . Then it constructs the arithmetic formula $\psi = \gamma_{m,\varepsilon}$. Then, it enumerates all the provable formulas until it finds the formula ψ in the iteration. If this loop eventually terminates, then it accepts (otherwise, it runs forever, of course). It is an example of a undecidable language with a non-robust machine.

We claim that M is not terminating. The formula $\psi = \gamma_{m,\varepsilon}$ is not provable: indeed, ψ is true iff m does accept the empty word and if M finds a proof of ψ , then M accepts the empty word and hence formula ψ is wrong. Unless the arithmetic is not coherent there is no way to prove some false formula, hence this case can not happen. Hence, necessarily, the loop will run forever. Now, if M does not find a proof of ψ , then M does not accept the empty word by construction. In other words, ψ is some valid formula that is not provable and M is not terminating, but there is no proof of it in arithmetic.

NB 4.1.14

The one used in the proof of Theorem 6.3.1, later in this document, is another example of a non-robust machine.

We have just used in the reasoning the fact that some things could be true, but with no proof of it (Gödel’s incompleteness theorem). Thinking about it, from previous considerations, somehow this is mandatory (if one wants to remain “constructive” and give explicitly a language or machine, and not conclude by arguments like diagonalisation or

counting arguments). Indeed, coming arguments show that non-robustness has to do with statements that could hold but are not provable, with proofs of a specific form, namely finite abstractions:

Definition 4.1.15 (Finite abstraction)

A finite abstraction of a set X is given by some (labelled) graph $G = (V, \rightarrow_g^g)$: each of its finitely many vertices corresponds to some finite union of basic open sets U_i . The edges between vertices are labelled by generators. It satisfies that for all $\mathbf{x} \in U_i$, if $\mathbf{f}^g(\mathbf{x}) = \mathbf{y}$ then there is an edge labelled by g from U_i to some U_j with $\mathbf{y} \in U_j$.

Such a finite abstraction corresponds to the subset of X obtained by taking the union of the open sets U_i corresponding to its vertices. We will come back to those statements in Section 4.6.

4.2 Discrete-Time Dynamical Systems

We now study the robustness of general discrete-time systems. We aim to focus on discrete-time systems of type $\mathbf{f}: \mathbb{R}^d \rightarrow \mathbb{R}^d$, but we start with a simpler framework: namely rational systems.

4.2.1 The case of rational systems

For clarity, as this general case requires talking about computability issues on the reals (we do so later in Section 4.2.2) we first focus on the case of systems of type $\mathbf{f}: \mathbb{R}^d \rightarrow \mathbb{R}^d$ such that $\mathbf{f}(\mathbb{Q}^d) \subseteq \mathbb{Q}^d$. In other words, we first focus on the case of *rational systems*, i.e. $\mathbf{f}: \mathbb{Q}^d \rightarrow \mathbb{Q}^d$ (possibly obtained as the restriction to the rationals of a function over the reals).

A rational discrete-time dynamical system will be called \mathbb{Q} -computable when the function (from the rationals to the rationals) is. A rational discrete-time dynamical system will be called Lipschitz when the function is: there exists some constant K such that $d(\mathbf{f}(\mathbf{x}), \mathbf{f}(\mathbf{y})) \leq Kd(\mathbf{x}, \mathbf{y})$, for all \mathbf{x}, \mathbf{y} .

With each rational discrete-time dynamical system \mathcal{H} is associated its reachability relation $R^{\mathcal{H}}(\cdot, \cdot)$ on $\mathbb{Q}^d \times \mathbb{Q}^d$. Namely, for two rational points \mathbf{x} and \mathbf{y} , $R^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ holds iff there exists a trajectory of \mathcal{H} from \mathbf{x} to \mathbf{y} .

The reachability relation of a \mathbb{Q} -computable system is computably enumerable: to enumerate, a Turing machine can just simulate the dynamics. Here is an example of a \mathbb{Q} -system:

Example 4.2.1 (Example of a \mathbb{Q} -system)

Take some recurrent neural network, with d neurons, with the ReLU activation function, defined as $\text{ReLU}(x) = \max(0, x)$. Its dynamics can be written as $\mathbf{x}_{t+1} = \text{ReLU}(\mathbf{A}\mathbf{x} + \mathbf{B})$ where \mathbf{A} is some $d \times d$ matrix with rational entries and \mathbf{B} is some vector of dimension d with rational entries, where the ReLU function is applied componentwise.

Another example is given by PAM systems:

Definition 4.2.2 (PAM System)

A Piecewise affine map system (PAM) is a discrete-time dynamical system \mathcal{H} where $\mathbf{f} : X \subset \mathbb{R}^d \mapsto X$ is a (possibly partial) function represented by: $\mathbf{f}(\mathbf{x}) = A_i \mathbf{x} + \mathbf{b}_i, \mathbf{x} \in P_i, \quad i = 1 \dots N$, where A_i are rational $d \times d$ -matrices, $\mathbf{b}_i \in \mathbb{Q}^d$ and P_i are convex rational polyhedral sets in X .

All constants in the PAM definitions are assumed to be rational so it remains a \mathbb{Q} -computable system (or a rational system). Notice that no form of continuity is assumed: we allow discontinuities between two affine pieces. The computational power of PAMs has been established in Subsection 2.3.1 using the technique of step-by-step emulation (with $\gamma_{[0,1]}$ and \mathbf{f} piecewise affine).

In [AB01], the authors consider only the special case of Piecewise Affine Maps (PAM), as representative of discrete-time systems, which are particular \mathbb{Q} -computable Lipschitz systems. They proved the following:

Theorem 4.2.3 (Computational power of PAMs [Moo91, KCG94, AB01])

Any c.e. language is reducible to the reachability relation of a PAM.

Theorem 4.2.4 (Computational power of PAMs [KCG94])

Any c.e. language is reducible to the reachability relation of a continuous PAM.

Let us discuss whether undecidability still holds for “robust systems”.

We apply the paradigm of small perturbations: consider a deterministic discrete-time dynamical system \mathcal{H} over a space X with a function \mathbf{f} . For any $\varepsilon > 0$ we consider the ε -perturbed system \mathcal{H}_ε . Its trajectories are defined as sequences \mathbf{x}_t satisfying $d(\mathbf{x}_{t+1}, \mathbf{f}(\mathbf{x}_t)) < \varepsilon$ for all t . This non-deterministic system is considered as \mathcal{H} submitted to a noise of magnitude ε . For convenience, we write $\mathbf{y} \in \mathbf{f}_\varepsilon(\mathbf{x})$ as a synonym for $d(\mathbf{f}(\mathbf{x}), \mathbf{y}) < \varepsilon$. We denote reachability in the system \mathcal{H}_ε by $R_\varepsilon^{\mathcal{H}}(\cdot, \cdot)$: $R_\varepsilon^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ holds iff from \mathbf{x} , it is possible to reach \mathbf{y} by finitely many applications of \mathbf{f}_ε . In particular, in zero steps, only \mathbf{x} is reachable from \mathbf{x} , so $R_\varepsilon^{\mathcal{H}}(\mathbf{x}, \mathbf{x})$ always holds.

We define the set-valued function \mathbf{f}_ε by $\mathbf{f}_\varepsilon(S) = \bigcup_{\mathbf{x} \in S} B(\mathbf{f}(\mathbf{x}), \varepsilon)$, for $S \subseteq \mathbb{R}^d$. We denote the t -th iteration of a function g by $g^{[t]}$. Then, $R_\varepsilon^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ is true iff there exists $t \in \mathbb{N}$ such that $\mathbf{y} \in \mathbf{f}_\varepsilon^{[t]}(\{\mathbf{x}\})$. We denote by $\mathbf{x} \rightarrow_\varepsilon^* \mathbf{y}$ and $\mathbf{x} \rightarrow_\varepsilon^+ \mathbf{y}$ the analogous of $\mathbf{x} \rightarrow^* \mathbf{y}$ and $\mathbf{x} \rightarrow^+ \mathbf{y}$ but for ε -perturbed systems.

All trajectories of a non-perturbed system \mathcal{H} are also trajectories of the ε -perturbed system \mathcal{H}_ε . If $\varepsilon_1 < \varepsilon_2$ then any trajectory of the ε_1 -perturbed system is also a trajectory of the ε_2 -perturbed system. Define $R_\omega^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ iff $\forall \varepsilon > 0 \ R_\varepsilon^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$: this relation encodes reachability with arbitrarily small perturbing noise. From definitions:

Lemma 4.2.5 ([AB01])

For any $0 < \varepsilon_2 < \varepsilon_1$ and any \mathbf{x} and \mathbf{y} the following implications hold: $R^{\mathcal{H}}(\mathbf{x}, \mathbf{y}) \Rightarrow R_\omega^{\mathcal{H}}(\mathbf{x}, \mathbf{y}) \Rightarrow R_{\varepsilon_2}^{\mathcal{H}}(\mathbf{x}, \mathbf{y}) \Rightarrow R_{\varepsilon_1}^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$.

Theorem 4.2.6 (Perturbed reachability is co-c.e.)

Consider a locally Lipschitz \mathbb{Q} -computable system whose domain $X \subset \mathbb{R}^d$ is a closed rational box and dynamic \mathbf{f} is continuous. Then the relation $R_\omega^\mathcal{H}(\mathbf{x}, \mathbf{y}) \subseteq \mathbb{Q}^d \times \mathbb{Q}^d$ is in the class Π_1 .

NB 4.2.7

A generalisation of this theorem will be given in Chapter 6 (Theorem 6.2.12).

Proof of Theorem 4.2.6. This corrects [AB01, Theorem 5], using an alternative proof. They use the fact that the function is Lipschitz, but apply it to discontinuous PAMs. Lipschitz functions being continuous, the argument requires to be adapted to cover the hypotheses. As \mathbf{f} is locally Lipschitz and X is compact, we know that \mathbf{f} is Lipschitz: there exists some $L > 0$ so that $d(\mathbf{f}(\mathbf{x}), \mathbf{f}(\mathbf{y})) \leq L \cdot d(\mathbf{x}, \mathbf{y})$. For every $\delta = 2^{-m}$, $m \in \mathbb{N}$, we associate some graph $G_m = (V_\delta, \rightarrow_\delta)$: its vertices, denoted by $(\mathcal{V}_i)_i$, correspond to some finite discretisation and covering of compact X by rational open balls $\mathcal{V}_i = B(\mathbf{x}_i, \delta_i)$ of radius $\delta_i < \delta$. There is an edge from \mathcal{V}_i to \mathcal{V}_j in this graph, that is to say $\mathcal{V}_i \rightarrow_\delta \mathcal{V}_j$, iff $B(\mathbf{f}(\mathbf{x}_i), (L+1)\delta) \cap \mathcal{V}_j \neq \emptyset$. With our hypothesis on the domain, such a graph can be effectively obtained from m , considering a suitable discretisation of the rational box X .

Before proving the rest of Theorem 4.2.6, we state and prove the two following lemmas:

Lemma 4.2.8

Assume $R_\varepsilon^\mathcal{H}(\mathbf{x}, \mathbf{y})$ for $\varepsilon = 2^{-n}$. Then, $\mathbf{x} = \mathbf{y}$, or, in the case $\mathbf{x} \neq \mathbf{y}$, for all i, j , with $\mathbf{x} \in \mathcal{V}_i$ and $\mathbf{y} \in \mathcal{V}_j$, $\mathcal{V}_i \not\rightarrow_\varepsilon \mathcal{V}_j$.

It holds because the graph for $\delta = \varepsilon$ is made to always have more trajectories/behaviours than $R_\varepsilon^\mathcal{H}$. More formally:

Proof. If $\mathbf{x} = \mathbf{y}$, then the claim is verified. Else, if $\mathbf{y} \in \mathbf{f}_\varepsilon(\mathbf{x})$, then $d(\mathbf{f}(\mathbf{x}_i), \mathbf{y}) \leq d(\mathbf{f}(\mathbf{x}_i), \mathbf{f}(\mathbf{x})) + d(\mathbf{f}(\mathbf{x}), \mathbf{y}) < Ld(\mathbf{x}_i, \mathbf{x}) + 2\varepsilon \leq L\varepsilon + 2\varepsilon = (L+2)\varepsilon$ and hence there is an edge from $\mathcal{V}_i \rightarrow_\varepsilon \mathcal{V}_j$ to any \mathcal{V}_j containing \mathbf{y} by definition of the graph. \square

Lemma 4.2.9

If $\mathbf{x} = \mathbf{y}$ then $R_\varepsilon^\mathcal{H}(\mathbf{x}, \mathbf{y})$. Furthermore, for any $\varepsilon = 2^{-n}$ there is some $\delta = 2^{-m}$ so that, if there exist some i, j with $\mathcal{V}_i \rightarrow_\delta^+ \mathcal{V}_j$ and $\mathbf{x} \in \mathcal{V}_i$ and $\mathbf{y} \in \mathcal{V}_j$, then $R_\varepsilon^\mathcal{H}(\mathbf{x}, \mathbf{y})$.

Lemma 4.2.9 states that $\neg R_\varepsilon^\mathcal{H}(\mathbf{x}, \mathbf{y})$ implies $\neg(\mathcal{V}_i \rightarrow_\delta^+ \mathcal{V}_j)$ for all i and j , with $\mathbf{x} \in \mathcal{V}_i$ and $\mathbf{y} \in \mathcal{V}_j$, for the corresponding δ .

Proof. If $\mathbf{x} = \mathbf{y}$, the first claim is clear.

Now, consider $\delta = 2^{-m}$ with $\delta < \varepsilon/(2L+1)$: It is sufficient to prove that when $\mathcal{V}_i \rightarrow_\delta^+ \mathcal{V}_j$ with $\mathbf{x} \in \mathcal{V}_i$, $\mathbf{y} \in \mathcal{V}_j$, we have $R_\varepsilon^\mathcal{H}(\mathbf{x}, \mathbf{y})$: indeed, then by induction, if we have

$$\mathcal{V}_{i_0} \rightarrow_\delta \mathcal{V}_{i_1} \dots \rightarrow_\delta \mathcal{V}_{i_l=j},$$

we can then reason on points $\mathbf{y}_1 \in \mathcal{V}_{i_1}, \dots, \mathbf{y}_{t-1} \in \mathcal{V}_{i_{t-1}}$, and obtain inductively that $R_{\varepsilon}^{\mathcal{H}}(\mathbf{x}, \mathbf{y}_1), R_{\varepsilon}^{\mathcal{H}}(\mathbf{y}_1, \mathbf{y}_2), \dots, R_{\varepsilon}^{\mathcal{H}}(\mathbf{y}_{t-1}, \mathbf{y})$, and hence that $R_{\varepsilon}^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$.

So, assume $\mathcal{V}_i \xrightarrow{1}_{\delta} \mathcal{V}_j$ with $\mathbf{x} \in \mathcal{V}_i, \mathbf{y} \in \mathcal{V}_j$. As $\mathcal{V}_i \xrightarrow{1}_{\delta} \mathcal{V}_j$ there is some $\bar{\mathbf{y}} \in \mathcal{V}_j$ with $d(\mathbf{f}(\mathbf{x}_i), \bar{\mathbf{y}}) < (L+1)\delta$. Then $d(\mathbf{f}(\mathbf{x}), \bar{\mathbf{y}}) \leq d(\mathbf{f}(\mathbf{x}), \mathbf{f}(\mathbf{x}_i)) + d(\mathbf{f}(\mathbf{x}_i), \bar{\mathbf{y}}) < L\delta + (L+1)\delta = (2L+1)\delta < \varepsilon$ and hence $R_{\varepsilon}^{\mathcal{H}}(\mathbf{x}, \bar{\mathbf{y}})$: we get the promised statement. \square

From Lemma 4.2.8 and Lemma 4.2.9, When $\mathbf{x} = \mathbf{y}$, $R_{\omega}^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ holds. Now, from the definitions, and the construction of our covering, when $\mathbf{x} \neq \mathbf{y}$, there exists $\delta_0 = 2^{-m_0}$, such that for all i, j with $\mathbf{x} \in \mathcal{V}_i$ and $\mathbf{y} \in \mathcal{V}_j$, we have $i \neq j$, hence $\neg(\mathcal{V}_i \xrightarrow{0}_{\delta_0} \mathcal{V}_j)$. Consequently, $\neg(\mathcal{V}_i \xrightarrow{*}_{\delta} \mathcal{V}_j)$ is the same as $\neg(\mathcal{V}_i \xrightarrow{0}_{\delta} \mathcal{V}_j)$ for $m \geq m_0$.

Now, when, $\mathbf{x} \neq \mathbf{y}$, then from the two above items, $\neg R_{\omega}^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ holds iff $\mathbf{x} \neq \mathbf{y}$ and there exists $\delta = 2^{-m}$, $m \geq m_0$, such that for all i, j with $\mathbf{x} \in \mathcal{V}_i$ and $\mathbf{y} \in \mathcal{V}_j$, one has $\neg(\mathcal{V}_i \xrightarrow{*}_{\delta} \mathcal{V}_j)$. This holds iff for some integer $m \geq m_0$, $\text{NOPATH}(G_m, \mathcal{V}_i, \mathcal{V}_j)$ for all $\mathcal{V}_i, \mathcal{V}_j$ with $\mathbf{x} \in \mathcal{V}_i, \mathbf{y} \in \mathcal{V}_j$. The latter property is computably enumerable, as it is a union of decidable sets (uniform in m), as $\text{NOPATH}(G_m, \mathcal{V}_i, \mathcal{V}_j)$ is a decidable property over finite graph G_m . \square

NB 4.2.10

This would work even only assuming the domain to be a computable compact and/or the map to be computable (non-necessarily Lipchitz) in the model of computable analysis: the proof only requires that given $\delta = 2^{-m}$, there is an effective way to determine an effective cover, using finitely many rational balls of radius strictly smaller than δ .

Definition 4.2.11 (Robust reachability relation)

We say that the reachability relation is robust when $R^{\mathcal{H}} = R_{\omega}^{\mathcal{H}}$.

We get the “robustness conjecture”:

Corollary 4.2.12 (Robust \Rightarrow decidable, [AB01])

Assume the hypotheses of Theorem 4.2.6. If the relation $R^{\mathcal{H}}$ is robust then it is decidable.

A continuous PAM simulating the execution of a Turing machine is an example of such relation.

Proof. $R^{\mathcal{H}}$ is c.e. and we know from Theorem 4.2.6 that $R_{\omega}^{\mathcal{H}}$ is co-c.e.. If they are equal, then they are decidable, as a c.e. and co-c.e. set is decidable. \square

A similar statement holds even if X is not compact: the existence of some family of graphs $\mathcal{G} = (G_m)$ with $G_m = (V_m, \rightarrow_m)$ is sufficient to get a similar reasoning with the following properties:

1. $R_{\varepsilon}^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ with $\mathbf{x} \in \mathcal{V}_i, \varepsilon = 2^{-n}$, implies $\mathbf{x} = \mathbf{y}$ or $\mathcal{V}_i \xrightarrow{+}_n \mathcal{V}_j$ for all \mathcal{V}_j containing \mathbf{y} .
2. For any $\varepsilon = 2^{-n}$, there is some m such that if we have $\mathcal{V}_i \xrightarrow{+}_m \mathcal{V}_j$ then $R_{\varepsilon}^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ whenever $\mathbf{x} \in \mathcal{V}_i, \mathbf{y} \in \mathcal{V}_j$.
3. For all m , G_m is a finite computable graph: determining whether $\mathcal{V}_i \rightarrow_m \mathcal{V}_j$ in G_m can be effectively determined given integers m, i and j .

When these three properties hold, \mathcal{G} is a *computable abstraction* of the discrete-time dynamical system.

Robustness versus decidability and δ -decidability

We now discuss how far the above statement is to a characterisation of decidability.

Before stating this in Corollary 4.2.19, we relate robustness to the concept of δ -decidability defined in [GAC12] and also to the existence of some witness of non-reachability.

We mix the notation δ and ε when talking about precision. They are indeed the same. Our problem is that the framework considered in [GAC12] uses the terminology δ -decidability, whereas [AB01] is in a context of analysis over the reals and uses ε to quantify error bounds. We decided to keep both δ and ε . Otherwise, this would conflict with their usual meaning in the two contexts.

Given \mathbf{x} , $R^{\mathcal{H}}(\mathbf{x})$ denotes the set of the points \mathbf{y} reachable from \mathbf{x} in the dynamical system \mathcal{H} : $R^{\mathcal{H}}(\mathbf{x}) = \{\mathbf{y} | R^{\mathcal{H}}(\mathbf{x}, \mathbf{y})\}$. This is also the smallest set such that $\mathbf{x} \in R^{\mathcal{H}}(\mathbf{x})$ and $\mathbf{f}(R^{\mathcal{H}}(\mathbf{x})) \subseteq R^{\mathcal{H}}(\mathbf{x})$.

Definition 4.2.13

$R^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ is said to be ε -far from being true if there is $\mathcal{R}^* \subseteq X$ so that

1. $\mathbf{x} \in \mathcal{R}^*$,
2. $\mathbf{f}_{\varepsilon}(\mathcal{R}^*) \subseteq \mathcal{R}^*$,
3. $\mathbf{y} \notin \mathcal{R}^*$.

When this holds, we have $\neg R^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$. Indeed, for all $\varepsilon > 0$, $R_{\varepsilon}^{\mathcal{H}}(\mathbf{x}) = \{\mathbf{y} | R_{\varepsilon}^{\mathcal{H}}(\mathbf{x}, \mathbf{y})\}$ is the smallest set satisfying $\mathbf{x} \in R_{\varepsilon}^{\mathcal{H}}(\mathbf{x})$ and $\mathbf{f}_{\varepsilon}(R_{\varepsilon}^{\mathcal{H}}(\mathbf{x})) \subseteq R_{\varepsilon}^{\mathcal{H}}(\mathbf{x})$. Thus, as \mathcal{R}^* also satisfies these properties by the first two conditions, $R_{\varepsilon}^{\mathcal{H}}(\mathbf{x}) \subseteq \mathcal{R}^*$ and hence $\mathbf{y} \notin R^{\mathcal{H}}(\mathbf{x})$ as $R^{\mathcal{H}}(\mathbf{x}) \subseteq R_{\varepsilon}^{\mathcal{H}}(\mathbf{x}) \subseteq \mathcal{R}^*$ and $\mathbf{y} \notin \mathcal{R}^*$ from the third condition.

In other words, \mathcal{R}^* is a *witness* of the non-reachability of \mathbf{y} from \mathbf{x} . We will say that it is *at level ε* . This provides a relation to δ -decidability considered in [GAC12]:

Proposition 4.2.14 (Robust \Leftrightarrow Reachability relation true or ε -far from being true)

We have $R_{\omega}^{\mathcal{H}} = R^{\mathcal{H}}$ if and only if for all $\mathbf{x}, \mathbf{y} \in \mathbb{Q}^d$, either

1. $R^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ is true
2. or $R^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ is false and there exists $\varepsilon > 0$ such that it is ε -far from being true.

Proof. (\Rightarrow): For all $\varepsilon > 0$, $R_{\varepsilon}^{\mathcal{H}}(\mathbf{x})$ satisfies $\mathbf{x} \in R_{\varepsilon}^{\mathcal{H}}(\mathbf{x})$ and $\mathbf{f}_{\varepsilon}(R_{\varepsilon}^{\mathcal{H}}(\mathbf{x})) \subseteq R_{\varepsilon}^{\mathcal{H}}(\mathbf{x})$ (this is even the smallest set such that this holds). Let $\mathbf{y} \in \mathbb{Q}^d$, let us assume that $R^{\mathcal{H}}(\mathbf{x}, \mathbf{y}) = R_{\omega}^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ is not true. Then, there exists ε such that $R_{\varepsilon}^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ is false, i.e. $\mathbf{y} \notin R_{\varepsilon}^{\mathcal{H}}(\mathbf{x})$. Consider $\mathcal{R}^* = R_{\varepsilon}^{\mathcal{H}}(\mathbf{x})$. Then, $\mathbf{x} \in \mathcal{R}^*$ and from the first paragraph $\mathbf{f}_{\varepsilon}(\mathcal{R}^*) \subseteq \mathcal{R}^*$ and $\mathbf{y} \notin \mathcal{R}^*$.

(\Leftarrow): When $R^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ is true, for all $\varepsilon > 0$, $R_{\varepsilon}^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ is true, so $R_{\omega}^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ is. When $R^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ is false, by hypothesis, $R^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ is ε -far from being true for some $\varepsilon > 0$: there

exists a set \mathcal{R}^* satisfying $\mathbf{x} \in \mathcal{R}^*$ and $\mathbf{f}_\varepsilon(\mathcal{R}^*) \subseteq \mathcal{R}^*$. As $R_\varepsilon^{\mathcal{H}}(\mathbf{x})$ is the smallest such set, $R_\varepsilon^{\mathcal{H}}(\mathbf{x}) \subseteq \mathcal{R}^*$. As $\mathbf{y} \notin \mathcal{R}^*$, $\mathbf{y} \notin R_\varepsilon^{\mathcal{H}}(\mathbf{x})$. Hence $R_\omega^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ is false. \square

We say that a subset R^* of X is ε -rejecting (with respect to \mathbf{y}) if it satisfies 2. and 3. of Definition 4.2.13: that is to say, $\mathbf{f}_\varepsilon(R^*) \subseteq R^*$, and $\mathbf{y} \notin R^*$. A trajectory reaching such a R^* will never leave it.

Definition 4.2.15

A system is eventually decisional if for all \mathbf{x}, \mathbf{y} , there is some R^* ε -rejecting (with respect to \mathbf{y}) so that either there is a trajectory starting from \mathbf{x} reaches \mathbf{y} or, when not, it reaches R^* .

We come back to the converse of Corollary 4.2.12: from Proposition 4.2.14, a robust dynamical system (i.e. $R_\omega^{\mathcal{H}} = R^{\mathcal{H}}$) is eventually decisional, by considering $R^* = \mathcal{R}^*$ for the \mathcal{R}^* given by item 2) there. Conversely:

Lemma 4.2.16

Take \mathbf{x} and \mathbf{y} with $R_\omega^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ but not $R^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$. For \mathbf{f} Lipschitz, the trajectory starting from $\mathbf{x} \in \mathbb{Q}^d$ can not reach any ε -rejecting subset.

Proof. By contradiction, assume the trajectory starting from \mathbf{x} reaches an ε -rejecting R^* . By considering one more step, we can assume that it reaches the interior of R^* for the first time at t , since, if it reaches the frontier at \mathbf{x}^* , $B(\mathbf{f}(\mathbf{x}^*), \varepsilon) \subseteq R^*$ and $\mathbf{f}(\mathbf{x}^*)$ is in the interior of that ball. So, $\mathbf{f}^{[t]}(\mathbf{x})$ is in the interior of R^* . As \mathbf{f} is Lipschitz, so are the s -th iterations of \mathbf{f} , for $0 \leq s \leq t$. Hence, by choosing ε' with $0 < \varepsilon' < \varepsilon$ sufficiently small, we obtain, on the one hand, $\mathbf{y} \notin \mathbf{f}_{\varepsilon'}^{[s]}(\{\mathbf{x}\})$, for $0 \leq s < t$, and, on the other hand, $\mathbf{f}_{\varepsilon'}^{[t]}(\{\mathbf{x}\}) \subseteq R^*$. The second assertion, together with the property $\mathbf{f}_\varepsilon(R^*) \subseteq R^*$, implies $\mathbf{f}_{\varepsilon'}^{[s]}(\{\mathbf{x}\}) \subseteq R^*$ for all $s \geq t$. As $\mathbf{y} \notin R^*$, we obtain $\mathbf{y} \notin \mathbf{f}_{\varepsilon'}^{[s]}(\{\mathbf{x}\})$ for all $s \geq t$. Thus, we have $\mathbf{y} \notin \mathbf{f}_{\varepsilon'}^{[s]}(\{\mathbf{x}\})$ for all $s \in \mathbb{N}$.

This shows $\neg R_{\varepsilon'}^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$, hence, $\neg R_\omega^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$, a contradiction. \square

Corollary 4.2.17

Consider a Lipschitz rational dynamical system over a rational box. It is robust iff it is eventually decisional.

We can even compute the witnesses under the hypotheses of Theorem 4.2.6. A dynamical system is *effectively eventually decisional* when there is an algorithm such that, given \mathbf{x} and \mathbf{y} , it outputs an R^* in the form of the finite union of rational balls. We can reinforce Corollary 4.2.12:

Proposition 4.2.18

Assume the hypotheses of Theorem 4.2.6. If $R_\omega^{\mathcal{H}} = R^{\mathcal{H}}$ then $R^{\mathcal{H}}$ is computable and the system is effectively eventually decisional.

Proof. The proof of Theorem 4.2.6 shows that when $R_\omega^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ is false, then $R_\varepsilon^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ is false for some $\varepsilon = 2^{-n}$ and there is a $\delta = 2^{-m}$ and some graph G_m with vertices \mathcal{V}_i and

\mathcal{V}_j , $\mathbf{x} \in \mathcal{V}_i$, $\mathbf{y} \in \mathcal{V}_j$ and $\neg(\mathcal{V}_i \xrightarrow{*}_\delta \mathcal{V}_j)$. Denote by R^{G_m} the union of the vertices \mathcal{V}_k such that $\mathcal{V}_i \xrightarrow{*}_\delta \mathcal{V}_k$, $\mathbf{x} \in \mathcal{V}_i$ in G_m . Consider $\mathcal{R}^* = R^{G_m}$: this is a witness at level $\delta = 2^{-m}$ from the properties of the construction. Then m can be found by testing increasing m until a proper graph is found. The corresponding $\mathcal{R}^* = R^{G_m}$ of the first graph found will be a witness at level $\delta = 2^{-m}$. \square

The reachability relation of an effectively eventually decisional system is necessarily decidable (given \mathbf{x} and \mathbf{y} , compute the path until it reaches \mathbf{y} (then accept), or R^* (then reject)):

Corollary 4.2.19

Under the hypotheses of Theorem 4.2.6, the following properties are equivalent:

- $R^{\mathcal{H}}$ is robust;
- $R^{\mathcal{H}}$ is decidable and the dynamical system is effectively eventually decisional;
- The dynamical system is effectively eventually decisional.

Until now, we mainly studied computability properties. We now focus on complexity in this framework.

Complexity issues

Assume the dynamical system is robust. Hence, for all $\mathbf{x}, \mathbf{y} \in \mathbb{Q}^d$, there exists ε (depending on \mathbf{x}, \mathbf{y}) such that $R^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ and $R_\varepsilon^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ have the same truth value (unchanged by smaller ε). It is then natural to quantify the level of required robustness according to \mathbf{x} and \mathbf{y} , i.e. on the value ε . As we may always assume $\varepsilon = 2^{-n}$ for some $n \in \mathbb{N}$, we write $R_n^{\mathcal{H}}$ for $R_{\varepsilon=2^{-n}}^{\mathcal{H}}$ and we introduce:

Definition 4.2.20 (Level of robustness ε given by s)

Given a function $s : \mathbb{N} \rightarrow \mathbb{N}$, we write $R_{\{s\}}^P$ for the relation defined as: for any rational points \mathbf{x} and \mathbf{y} the relation holds iff $R_{s(\ell(\mathbf{x})+\ell(\mathbf{y}))}^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$.

A robust dynamical system is necessarily s -robust for some function s , according to the next definition: this follows from the same arguments as the ones we used for the related concepts for Turing machines. This function s quantifies the tolerated level of robustness.

Definition 4.2.21 (s -robust system)

We say that a dynamical system is s -robust, when $R^{\mathcal{H}} = R_{\{s\}}^P$.

Lemma 4.2.22

Consider a locally Lipschitz \mathbb{Q} -computable system, with $\mathbf{f} : \mathbb{Q}^d \rightarrow \mathbb{Q}^d$ computable in polynomial time, whose domain X is a closed rational box. For $\delta = 2^{-m}$, consider the associated graph G_m considered in the proof of Theorem 4.2.6. Then $\text{NOPATH}(G_m, \mathcal{V}_i, \mathcal{V}_j)$ is decidable using a space polynomial in m .

Proof. This graph has less than $O(2^{d*m})$ vertices. It has a successor relation \rightarrow_δ computable in polynomial space in m . Hence, the analysis of Corollary 2.3.4 applies and we can determine if $\text{NOPATH}(G_m, \mathcal{V}_i, \mathcal{V}_j)$ using a space polynomial in m . \square

We can then naturally consider the case where s is a polynomial: considering robustness to polynomial perturbations corresponds to **PSPACE**:

Theorem 4.2.23

Consider a locally Lipschitz \mathbb{Q} -computable system, with $\mathbf{f} : \mathbb{Q}^d \rightarrow \mathbb{Q}^d$ computable in polynomial time, whose domain X is a closed rational box. Given some polynomial p , $R_{\{p\}}^P \in \mathbf{PSPACE}$.

Proof. From Theorem 4.2.6, for all n there exists some m (depending on n), such that $R_n^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ and $R^{G_m}(\mathbf{x}, \mathbf{y})$ have the same truth value, where R^{G_m} denotes reachability in the graph G_m . With the hypotheses, given \mathbf{x} and \mathbf{y} , we can determine whether $R_{\{p\}}^P(\mathbf{x}, \mathbf{y})$, by determining the truth value of $R_n^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$, taking $n = p(\ell(\mathbf{x}) + \ell(\mathbf{y}))$, p a polynomial. From the proof of Theorem 4.2.6, the corresponding m is linearly related to n . The analysis of Corollary 2.3.4 shows that the truth value of $R^{G_m}(\mathbf{x}, \mathbf{y})$ can be determined in space polynomial in m . \square

Theorem 4.2.24 (Polynomially robust to precision \Rightarrow PSPACE)

With the same hypotheses, if $R^{\mathcal{H}} = R_{\{p\}}^P$ for some polynomial p , then $R^{\mathcal{H}} \in \mathbf{PSPACE}$.

Proof. $R_{\{p\}}^P \in \mathbf{PSPACE}$ by Theorem 4.2.23. As $R^{\mathcal{H}} = R_{\{p\}}^P$, $R^{\mathcal{H}} \in \mathbf{PSPACE}$. \square

This is even a characterisation of **PSPACE**:

Theorem 4.2.25 (Polynomially robust to precision \Leftrightarrow PSPACE)

For any language L in **PSPACE** there exists a PAM \mathcal{H} , defined over a closed rational box, such that L is reducible to $R^{\mathcal{H}}$ and such that $R^{\mathcal{H}} = R_{\{p\}}^P$, for some polynomial p .

Proof. Let $L \in \mathbf{PSPACE}$. There is a TM M with $L(M) = L$ that works in polynomial space $q(\cdot)$. Its step-by-step emulation considered in Theorem 4.2.3, using $\gamma_{[0,1]}$ is done using a precision $O(2^{-q(n)})$ on words of length n . The obtained system satisfies $R_{\{q+O(1)\}}^P = R^{\mathcal{H}}$ from the properties of the emulation. \square

Assuming the hypotheses of Theorem 4.2.24, when $R^{\mathcal{H}} = R_{\{p\}}^P$ for some polynomial p , we also see that we can determine a witness of $\neg R^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ in polynomial space (using a suitable representation of it).

4.2.2 The case of computable systems

We now consider the case of general discrete-time dynamical systems. Then \mathbf{f} may take some non-rational values and we need the notion of computability of functions over the reals: this requires the model of computable analysis: see e.g. [Wei00] or [BHW08] for full presentations.

Computable systems

Given some rational ball $B(\mathbf{y}, \delta)$, we have to forbid “frontier reachability”: $B(\mathbf{y}, \delta)$ would not be reachable, but its frontier $\bar{B}(\mathbf{y}, \delta) - B(\mathbf{y}, \delta)$ would. It comes from the fact that testing equality is not computable (NB 1.3.50). A natural question arises: given some rational ball with the promise that either $B(\mathbf{y}, \delta)$ is reachable (that case implies that $\bar{B}(\mathbf{y}, \delta)$ is) from some \mathbf{x} , or that $\bar{B}(\mathbf{y}, \delta)$ is not, decide which possibility holds. We call this the *ball (decision) problem*. From definitions from computable analysis, when $R^{\mathcal{H}}(\mathbf{x})$ is a closed set, $R^{\mathcal{H}}(\mathbf{x})$ is a computable closed set iff its associated ball problem is algorithmically solvable.

For computable systems, the ball decision problem is c.e: we mean, there is a Turing machine whose halting set intersected with the rational balls satisfying the promise is the set of positive instances. Indeed, just simulate the system’s evolution, starting from \mathbf{x} until step T , with increasing precision and T , until one finds the guarantee that \mathbf{x}_T at time T is in $B(\mathbf{y}, \delta')$ for some $\delta' < \delta$. If the ball is reachable, it will terminate by computing a sufficient approximation of the corresponding \mathbf{x}_T . It cannot terminate without guaranteeing reachability. It is not co-c.e. in general.

NB 4.2.26

Our framework for discussing the computability of sets seems similar to the concept of a maximally partially decidable set, formalised in [Neu21, Neu23]. Similar ideas have also been implicitly used in many other articles considering various real problems, using computable analysis. A similar formalisation is also considered in [BCD23].

To a discrete-time system, we can also associate its reachability relation $R^{\mathcal{H}}(\cdot, \cdot, \cdot)$ over $\mathbb{Q}^d \times \mathbb{Q}^d \times \mathbb{N}$. For two points $\mathbf{x}, \mathbf{y} \in \mathbb{Q}^d$, $\eta = 2^{-p}$, encoded by $p \in \mathbb{N}$, $R^{\mathcal{H}}(\mathbf{x}, \mathbf{y}, p)$ iff there exists a trajectory of \mathcal{H} from \mathbf{x} to $\bar{B}(\mathbf{y}, \eta)$. We define $R_{\varepsilon}^{\mathcal{H}}$ similarly and $R_{\omega}^{\mathcal{H}} = \bigcap_{\varepsilon} R_{\varepsilon}^{\mathcal{H}}$. This relation encodes reachability with arbitrarily small perturbing noise to some closed ball.

Lemma 4.2.27

*For any $0 < \varepsilon_2 < \varepsilon_1$ and any \mathbf{x} and \mathbf{y} , η , the following implications hold:
 $R^{\mathcal{H}}(\mathbf{x}, \mathbf{y}, p) \Rightarrow R_{\omega}^{\mathcal{H}}(\mathbf{x}, \mathbf{y}, p) \Rightarrow R_{\varepsilon_2}^{\mathcal{H}}(\mathbf{x}, \mathbf{y}, p) \Rightarrow R_{\varepsilon_1}^{\mathcal{H}}(\mathbf{x}, \mathbf{y}, p)$.*

Given \mathbf{x} and $0 < \varepsilon_2 < \varepsilon_1$, $R^{\mathcal{H}}(\mathbf{x}, \mathbf{y}, p) \subseteq R_{\varepsilon_2}^{\mathcal{H}}(\mathbf{x}, \mathbf{y}, p) \subseteq \text{cls} \left(R_{\varepsilon_2}^{\mathcal{H}}(\mathbf{x}, \mathbf{y}, p) \right) \subseteq R_{\varepsilon_1}^{\mathcal{H}}(\mathbf{x}, \mathbf{y}, p) \subseteq \text{cls} \left(R_{\varepsilon_1}^{\mathcal{H}}(\mathbf{x}, \mathbf{y}, p) \right)$. Hence, $R_{\omega}^{\mathcal{H}}(\mathbf{x}, \mathbf{y}, p) = \bigcap_{\varepsilon > 0} R_{\varepsilon}^{\mathcal{H}}(\mathbf{x}, \mathbf{y}, p) = \bigcap_{\varepsilon > 0} \text{cls} \left(R_{\varepsilon}^{\mathcal{H}}(\mathbf{x}, \mathbf{y}, p) \right)$ is a closed set.

Theorem 4.2.28 (Perturbed reachability is co-r.e.)

Consider a locally Lipschitz computable system whose domain X is a computable compact. $R_\omega^{\mathcal{H}}(\mathbf{x}, \mathbf{y}, p) \subseteq \mathbb{Q}^d \times \mathbb{Q}^d \times \mathbb{N}$ is in Π_1 .

This can be considered as an extension of [BGH10, Theorem 1], established in a much simpler framework. We first prove the following lemma:

Lemma 4.2.29 (Compactness Argument)

Given a ball $B(\mathbf{y}, \eta)$, we have that $\bar{B}(\mathbf{y}, \eta) \cap R_\omega^{\mathcal{H}}(\mathbf{x}) = \emptyset$ iff $\bar{B}(\mathbf{y}, \eta) \cap \text{cls}(R_\varepsilon^{\mathcal{H}}(\mathbf{x})) = \emptyset$ for some $\varepsilon > 0$.

Proof. \Leftarrow (easy direction): If $\bar{B}(\mathbf{y}, \eta) \cap \text{cls}(R_\varepsilon^{\mathcal{H}}(\mathbf{x})) = \emptyset$ for some $\varepsilon > 0$, we cannot have $\bar{B}(\mathbf{y}, \eta) \cap R_\omega^{\mathcal{H}}(\mathbf{x}) \neq \emptyset$, as it would contain a point that would necessarily be in $\bar{B}(\mathbf{y}, \eta) \cap R_\varepsilon^{\mathcal{H}}(\mathbf{x})$.

\Rightarrow (compactness argument): Assume $\bar{B}(\mathbf{y}, \eta) \cap R_\omega^{\mathcal{H}}(\mathbf{x}) = \emptyset$. Since:

$$R_\omega^{\mathcal{H}}(\mathbf{x}) = \bigcap_{\varepsilon > 0} \text{cls}(R_\varepsilon^{\mathcal{H}}(\mathbf{x})),$$

it means that $\bigcup_{n \in \mathbb{N}} (\text{cls}(R_{2^{-n}}^{\mathcal{H}}(\mathbf{x})))^c$ is some covering of $\bar{B}(\mathbf{y}, \eta)$. As $\bar{B}(\mathbf{y}, \eta)$ is closed and bounded, it is compact. So, from the covering, we can extract some finite covering. Consequently, as n increases, the set $R_{2^{-n}}^{\mathcal{H}}(\mathbf{x})$ decreases, so does $\text{cls}(R_{2^{-n}}^{\mathcal{H}}(\mathbf{x}))$ and its complements $(\text{cls}(R_{2^{-n}}^{\mathcal{H}}(\mathbf{x})))^c$ increases. Hence, the set $(\text{cls}(R_{2^{-n_0}}^{\mathcal{H}}(\mathbf{x})))^c$, for some n_0 , is a covering of $\bar{B}(\mathbf{y}, \eta)$. In other words, $\bar{B}(\mathbf{y}, \eta) \cap \text{cls}(R_\varepsilon^{\mathcal{H}}(\mathbf{x})) = \emptyset$ for $\varepsilon = 2^{-n_0}$. This proves the direction from left to right. \square

Proof of Theorem 4.2.28. The idea is similar to Theorem 4.2.6, by constructing a graph satisfying Lemma 4.2.8 and Lemma 4.2.9. As \mathbf{f} is locally Lipschitz and X is compact, we know that \mathbf{f} is Lipschitz: there exists some $L > 0$ so that $d(\mathbf{f}(\mathbf{x}), \mathbf{f}(\mathbf{y})) \leq L \cdot d(\mathbf{x}, \mathbf{y})$.

For every $\delta = 2^{-m}$, $m \in \mathbb{N}$, we associate some graph $G_m = (V_\delta, \rightarrow_\delta)$: the vertices, denoted $(\mathcal{V}_i)_i$, of this graph correspond to some finite covering of compact X by rational open balls $\mathcal{V}_i = B(\mathbf{x}_i, \delta_i)$ of radius $\delta_i < \delta$.

There is an edge from \mathcal{V}_i to \mathcal{V}_j in this graph, that is to say $\mathcal{V}_i \rightarrow_\delta \mathcal{V}_j$, iff $B(\mathbf{f}_i, (L+2)\delta) \cap \mathcal{V}_j \neq \emptyset$, given some rational \mathbf{f}_i given by some (computed) δ -approximation of $\mathbf{f}(\mathbf{x}_i)$, i.e. \mathbf{f}_i such that $\mathbf{f}(\mathbf{x}_i) \in B(\mathbf{f}_i, \delta)$.

This is done to guarantee to cover $B(\mathbf{f}(\mathbf{x}_i), (L+1)\delta)$.

As we assumed compact X to be computable, such a graph can be effectively obtained from m , by computing suitable approximation \mathbf{f}_i of the $\mathbf{f}(\mathbf{x}_i)$'s at precision δ .

We write, as expected, $R_\omega^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ if there is a trajectory from \mathbf{x} to \mathbf{y} , allowing \mathbf{y} to be some real point (and similarly for $R_\varepsilon^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$).

From Lemma 4.2.8 and Lemma 4.2.9, $R_\omega^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ holds iff for all $\delta = 2^{-m}$, we have $\mathcal{V}_i \xrightarrow{*}_\delta \mathcal{V}_j$, for all $\mathcal{V}_i, \mathcal{V}_j$ with $\mathbf{x} \in \mathcal{V}_i, \mathbf{y} \in \mathcal{V}_j$. If one prefers, $\neg R_\omega^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ holds iff for some $\delta = 2^{-m}$, $\neg(\mathcal{V}_i \xrightarrow{*}_\delta \mathcal{V}_j)$ for some $\mathcal{V}_i, \mathcal{V}_j$ with $\mathbf{x} \in \mathcal{V}_i, \mathbf{y} \in \mathcal{V}_j$.

By Lemma 4.2.29, we basically can use arguments really similar to those of the proof of Theorem 4.2.6, where the role played by \mathbf{y} is now played by $\bar{B}(\mathbf{y}, \eta)$. With more details:

$R_\omega^\mathcal{H}(\mathbf{x}, \mathbf{y}, p)$ holds iff for all $\delta = 2^{-m}$, we have $\mathcal{V}_i \xrightarrow{*}_\delta \mathcal{V}_j$, for all $\mathcal{V}_i, \mathcal{V}_j$ with $\mathbf{x} \in \mathcal{V}_i$, $\mathcal{V}_j \cap \bar{B}(\mathbf{y}, 2^{-p}) \neq \emptyset$.

The direction from left to right is clear from Lemma 4.2.8. Conversely, assume that for all $\delta = 2^{-m}$, we have $\mathcal{V}_i \xrightarrow{*}_\delta \mathcal{V}_j$, for all $\mathcal{V}_i, \mathcal{V}_j$ with $\mathbf{x} \in \mathcal{V}_i$, $\mathcal{V}_j \cap \bar{B}(\mathbf{y}, 2^{-p}) \neq \emptyset$. Assume by contradiction that $\neg R_\omega^\mathcal{H}(\mathbf{x}, \mathbf{y}, \eta)$. From Claim*, we know that $\bar{B}(\mathbf{y}, \eta) \cap \text{cls}\left(R_\varepsilon^\mathcal{H}(\mathbf{x})\right) = \emptyset$ for some $\varepsilon > 0$. In particular $\bar{B}(\mathbf{y}, \eta) \cap R_\varepsilon^\mathcal{H}(\mathbf{x}) = \emptyset$. Then for the corresponding $\delta = 2^{-m}$ from Lemma 4.2.9, we cannot have $\mathcal{V}_i \xrightarrow{*}_\delta \mathcal{V}_j$, for any $\mathcal{V}_i, \mathcal{V}_j$ with $\mathbf{x} \in \mathcal{V}_i$, $\mathcal{V}_j \cap \bar{B}(\mathbf{y}, 2^{-p}) \neq \emptyset$. This proves the direction from right to left.

If one prefers, when $\mathbf{x} \notin \bar{B}(\mathbf{y}, 2^{-p})$, $\neg R_\omega^\mathcal{H}(\mathbf{x}, \mathbf{y}, \eta = 2^{-p})$ holds iff for some integer m , the following property P_m holds: $\text{NOPATH}(G_m, \mathcal{V}_i, \mathcal{V}_j)$, for all i, j , with $\mathbf{x} \in \mathcal{V}_i$ and $\mathcal{V}_j \cap \bar{B}(\mathbf{y}, 2^{-p}) \neq \emptyset$. The latter property is computably enumerable as it corresponds to a union of decidable sets (uniform in m), as the property P_m is a decidable property over finite graph G_m . \square

Corollary 4.2.30 (*Robust \Rightarrow decidable*)

Given some instance, $B(\mathbf{y}, \delta)$ of the ball problem, run in parallel the computably enumerable algorithm for it (and when its termination is detected, accepts) and the co-computably enumerable algorithm for $\left(R^\mathcal{H}(\mathbf{x})\right)^c = \left(R_\omega^\mathcal{H}(\mathbf{x})\right)^c$ (and when its termination is detected, rejects).

For example, it applies to the system described in Theorem 4.2.4, when the underlying TM is robust.

We now study the complexity of computable systems.

Complexity issues

Definition 4.2.31 (*Level of robustness ε given by s*)

Given some function $s : \mathbb{N} \rightarrow \mathbb{N}$, we write $R_{\{s\}}^\mathcal{H}$ as: for two rational points \mathbf{x} and \mathbf{y} and p , the relation holds iff $R_{s(\ell(\mathbf{x})+\ell(\mathbf{y})+p)}^\mathcal{H}(\mathbf{x}, \mathbf{y}, p)$.

As before, a robust dynamical system is necessary s -robust for some function s , according to the next definition. The function s quantifies the tolerated level of robustness.

Definition 4.2.32 (*s -robust system*)

We say that a dynamical system is s -robust, when $R^\mathcal{H} = R_{\{s\}}^\mathcal{H}$.

Theorem 4.2.33

Take a locally Lipschitz system, with \mathbf{f} polynomial time computable, whose domain X is a closed rational box. Then $R_{\{p\}}^\mathcal{H} \subseteq \mathbb{Q}^d \times \mathbb{Q}^d \times \mathbb{N} \in \mathbf{PSPACE}$, when p is a polynomial.

Proof. The proof of Theorem 4.2.28 (as in Theorem 4.2.6) shows that when the relation $R_\omega^\mathcal{H}(\mathbf{x}, \mathbf{y}, q)$ is false, then $R_\varepsilon^\mathcal{H}(\mathbf{x}, \mathbf{y}, q)$ is false for some $\varepsilon = 2^{-n}$. With the hypotheses, given \mathbf{x} , \mathbf{y} and q , we take n polynomial in $\ell(\mathbf{x}) + \ell(\mathbf{y}) + q$. The corresponding m is polynomially related to n (linear in n). An analysis similar to Theorem 4.2.23, shows the truth value of $R^{G_m}(\mathbf{x}, \mathbf{y}, p)$ can be determined in space polynomial in m . \square

NB 4.2.34

We prove the **PSPACE**-hardness of such problem in Section 5.3.

Then, once again:

Theorem 4.2.35 (Polynomially robust to precision \Rightarrow PSPACE)

Assuming Theorem 4.2.33's hypotheses, and that for all rational \mathbf{x} , $R^\mathcal{H}(\mathbf{x})$ is closed and $R^\mathcal{H}(\mathbf{x}) = R_{\{p\}}^P(\mathbf{x})$ for a polynomial p . Then the ball decision problem is in **PSPACE**.

Proof. We have $R_{\{p\}}^P \in \mathbf{PSPACE}$ by Theorem 4.2.33 and since $R^\mathcal{H} = R_{\{p\}}^P$, then $R^\mathcal{H} \in \mathbf{PSPACE}$. \square

Now that we have strong complexity properties for *rational* discrete-time dynamical systems, we want to extend them first to real systems, and then to continuous-time dynamical systems.

4.3 Relating robustness to drawability

We can go further and prove geometric properties: in the previous sections, we associated with every discrete-time dynamical system a reachability relation over the rationals. But we could also see it as a relation over the reals and use the framework of computable analysis, regarding subsets of $\mathbb{R}^d \times \mathbb{R}^d$. From the statements of [Wei00], the following holds:

Theorem 4.3.1

Consider a computable discrete-time system \mathcal{H} whose domain is a computable compact. For all computable \mathbf{x} , $\text{cls}(R^\mathcal{H}(\mathbf{x})) \subseteq \mathbb{R}^d$ is a c.e. closed subset.

Proof. Write $R^{\mathcal{H},T}(\mathbf{x}, \mathbf{y})$ iff there exists a trajectory of \mathcal{H} from \mathbf{x} to \mathbf{y} in at most T steps. We can write $R^{\mathcal{H},0}(\mathbf{x}, \mathbf{y})$ as $\{(\mathbf{x}, \mathbf{y}) \mid \mathbf{x} = \mathbf{y}\}$, which is a computable closed subset ([Wei00]). We reason by induction on the size of the trajectory.

- The property holds for $R^{\mathcal{H},0}(\mathbf{x}, \cdot) = \{\mathbf{x}\}$, as \mathbf{x} is computable.
- We can write $R^{\mathcal{H},T+1}(\mathbf{x}, \cdot) = \mathbf{F}(R^{\mathcal{H},T}(\mathbf{x}, \cdot))$ where \mathbf{F} is the operator that maps closed set K to $\mathbf{F}(K) := K \cup \mathbf{f}(K)$.

The function \mathbf{f} is computable so we know it is continuous. Thus, by induction on T , $R^{\mathcal{H},T+1}(\mathbf{x}, \mathbf{y})$ is a compact: K is a closed subset living in a compact and, by

induction, it is compact and the image of a compact by some continuous function is compact.

Hence, we know that $\mathbf{f}(K)$ is computable ([Wei00, Theorem 6.2.4]) and so is $\mathbf{F}(K)$ ([Wei00, Theorem 5.1.13]). By induction on T , $R^{\mathcal{H},T}(\mathbf{x}, \mathbf{y})$ is a closed computable subset, so it is c.e. closed (by [BHW08, Proposition 5.16]).

Furthermore, as it can be checked in all the above theorems from [Wei00] (see also [Wei00, Theorem 6.2.1] for the required iteration), our reasoning is even effective: we can produce effectively in T a name of $\text{cls}\left(R^{\mathcal{H},T}(\mathbf{x}, \cdot)\right)$ (even effectively from a name of \mathbf{f}). Consequently, this means by doing things in parallel (i.e. dovetailing) we can effectively enumerate the rational balls intersecting $\text{cls}\left(\bigcup_T R^{\mathcal{H},T}(\cdot, \cdot)\right)$, by considering increasing T and the balls in these enumerations. \square

A closed set is called *co-c.e. closed* if we can effectively enumerate the rational closed balls in its complement. Using proofs similar to Theorems 4.2.28 and 4.2.6:

Theorem 4.3.2

Consider a computable locally Lipschitz discrete-time system whose domain X is a computable compact. For all computable \mathbf{x} , $R_{\omega}^{\mathcal{H}}(\mathbf{x}) = \text{cls}\left(R_{\omega}^{\mathcal{H}}(\mathbf{x})\right) \subseteq \mathbb{R}^d$ is a co-c.e. closed subset.

Proof. Let \mathbf{x} and \mathbf{y} be such that $R_{\omega}^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ is false. There must exist some $\varepsilon = 2^{-n}$ and some $\eta = 2^{-p}$ so that $\bar{B}(\mathbf{y}, \eta) \cap R_{\varepsilon}^{\mathcal{H}}(\mathbf{x}) = \emptyset$, as otherwise, considering $n = p$, \mathbf{y} would be the limit of some accumulation points in $R_{\omega}^{\mathcal{H}}(\mathbf{x})$, and since $R_{\omega}^{\mathcal{H}}(\mathbf{x})$ is closed, we would have $\mathbf{y} \in R_{\omega}^{\mathcal{H}}(\mathbf{x})$.

Now, given $\varepsilon = 2^{-n}$ and $\eta = 2^{-p}$, the proof of Theorem 4.2.28 provides a strategy to find a m , so that graph G_m provides a proof that $\bar{B}(\mathbf{y}, \eta) \cap R_{\varepsilon}^{\mathcal{H}}(\mathbf{x}) = \emptyset$: If we denote by R^{G_m} the union of the vertices \mathcal{V}_k such that $\mathcal{V}_i \xrightarrow{\dagger}_{\delta} \mathcal{V}_k$, $\mathbf{x} \in \mathcal{V}_i$ in such a G_m , it acts as a witness of non-reachability from \mathbf{x} of $\bar{B}(\mathbf{y}, \eta)$, in the spirit of witness of non-reachability considered in previous sections.

Then a strategy to produce all the rational balls whose closure is not intersecting $R_{\omega}^{\mathcal{H}}(\mathbf{x}) = \text{cls}\left(R_{\omega}^{\mathcal{H}}(\mathbf{x})\right)$, is for increasing n , and p , generate in parallel all such balls in the corresponding witness R^{G_m} when one is found. This will enumerate all such balls, from previous arguments, and arguments from the proof of Theorem 4.2.28. \square

Corollary 4.3.3 (*Robust \Rightarrow computable*)

Assume the Theorem 4.3.2's hypotheses. For all computable \mathbf{x} , if $R^{\mathcal{H}}(\mathbf{x})$ is robust then $R^{\mathcal{H}}(\mathbf{x}) = \text{cls}\left(R^{\mathcal{H}}(\mathbf{x})\right) \subseteq \mathbb{R}^d$ is computable.

Proof. It follows from the fact that a closed set is computable iff it is c.e. closed and co-c.e. closed ([BHW08, Proposition 5.16]). \square

Notice that the above statements are effective in giving a name of \mathbf{x} .

Following Theorem 1.3.51, we have that being robust means being drawable.

Corollary 4.3.4 (Robust \Rightarrow drawable))

Assume Theorem 4.3.2's hypotheses. For all computable \mathbf{x} , if $R^{\mathcal{H}}(\mathbf{x})$ is robust then $\text{cls}\left(R^{\mathcal{H}}(\mathbf{x})\right) \subseteq \mathbb{R}^d$ can be plotted.

Proof. This follows from Corollary 4.3.3 and Theorem 1.3.51. \square

This is even effective in the name of \mathbf{x} and \mathbf{f} . The converse holds with additional topological properties.

Theorem 4.3.5

Assume $R^{\mathcal{H}}(\mathbf{x})$ is closed and can be plotted effectively in the name of \mathbf{x} and \mathbf{f} . Then the system is robust, i.e. $R_{\omega}^{\mathcal{H}}(\mathbf{x}) = R^{\mathcal{H}}(\mathbf{x})$.

We prove the statement: if $\text{cls}\left(R^{\mathcal{H}}(\mathbf{x})\right)$ can be plotted effectively in a name of \mathbf{x} and \mathbf{f} , then $R_{\omega}^{\mathcal{H}}(\mathbf{x}, \mathbf{y}) = R^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ except maybe for $(\mathbf{x}, \mathbf{y}) \in \text{cls}\left(R^{\mathcal{H}}(\mathbf{x})\right) - R^{\mathcal{H}}(\mathbf{x})$.

Proof. By Theorem 1.3.51, $\text{cls}\left(R^{\mathcal{H}}(\mathbf{x})\right)$ is computable which is equivalent to the computability of the distance function $d\left(\cdot, \text{cls}\left(R^{\mathcal{H}}(\mathbf{x})\right)\right)$ [Wei00, Corollary 5.1.8]. It means that given a rational ball, a name for \mathbf{x} and \mathbf{y} , with $\neg R^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$, the following procedure ends when $(\mathbf{x}, \mathbf{y}) \notin \text{cls}\left(R^{\mathcal{H}}(\mathbf{x})\right) - R^{\mathcal{H}}(\mathbf{x})$: compute a name of $d\left((\mathbf{x}, \mathbf{y}), \text{cls}\left(R^{\mathcal{H}}(\mathbf{x})\right)\right)$ until a strictly positive proof is found: we mean, until the machine computing a name of the distance outputs some rational interval $I_n = (a_n, b_n)$ containing $d\left((\mathbf{x}, \mathbf{y}), \text{cls}\left(R^{\mathcal{H}}(\mathbf{x})\right)\right)$ is such that $a_n > 0$.

Observe that $d\left((\mathbf{x}, \mathbf{y}), \text{cls}\left(R^{\mathcal{H}}(\mathbf{x})\right)\right) = 0$ would mean $(\mathbf{x}, \mathbf{y}) \in \text{cls}\left(R^{\mathcal{H}}(\mathbf{x})\right)$, but not in $R^{\mathcal{H}}$.

When the machine does so (i.e. answers such an I_n) it answers by reading $m \in \mathbb{N}$ cells of the names of \mathbf{x} , \mathbf{y} and \mathbf{f} . It returns the same if the names are altered after m symbols. Thus, there exists a precision ε (related to m , usually 2^{-m} for exponentially fast convergence) so $\neg R^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ remains true for an ε -neighborhood of \mathbf{x} and \mathbf{y} and unchanged by a small variation of \mathbf{f} . Hence, for all \mathbf{x} , \mathbf{y} , when $\neg R^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$, there exists some ε such that $\neg R_{\varepsilon}^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ ($\neg R_{\omega}^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$). When $R^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ holds, $R_{\omega}^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ holds. \square

4.4 Continuous-time systems

The previous ideas can be extended to continuous-time or hybrid systems.

The maximal interval of existence of solutions can be non-computable, even for computable Ordinary Differential Equations (ODEs) [GZB06]. To simplify, we assume the ODEs have solutions defined over all \mathbb{R} .

NB 4.4.1

A non-total solution must necessarily leave any compact, see e.g. [Har64], so X is

compact is not a restriction.

A trajectory of \mathcal{H} starting at $\mathbf{x}_0 \in X$ is a solution of the differential equation with initial condition $\mathbf{x}(0) = \mathbf{x}_0$, defined as a continuous right-derivable function $\xi : \mathbb{R}^+ \rightarrow X$ such that $\xi(0) = \mathbf{f}(\mathbf{x}_0)$ and for every t , $\mathbf{f}(\xi(t))$ is equal to the right-derivative of $\xi(t)$.

To each continuous-time dynamical system \mathcal{H} we associate its reachability relation $R^{\mathcal{H}}$ as for the discrete-time case.

For any $\varepsilon > 0$, the ε -perturbed system \mathcal{H}_ε is described by the differential inclusion $d(\dot{\mathbf{x}}, \mathbf{f}(\mathbf{x})) < \varepsilon$. This non-deterministic system can be seen as \mathcal{H} submitted to a noise of magnitude ε . We denote reachability in the system \mathcal{H}_ε by $R_\varepsilon^{\mathcal{H}}$. The limit reachability relation $R_\omega^{\mathcal{H}}$ is introduced as before: $R_\omega^{\mathcal{H}}$ iff $\forall \varepsilon > 0, R_\varepsilon^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$.

Definition 4.4.2 (Effective Local Lipschitz [GZB06])

Let $E = \bigcup_{n=0}^\infty B(a_n, r_n) \subseteq \mathbb{R}^{m+1}$ be a c.e. open set, $a_n \in \mathbb{Q}^m$ and $r_n \in \mathbb{Q}$ yield computable sequences satisfying $\overline{B(a_n, r_n)} \subseteq E$. $f : E \rightarrow \mathbb{R}^m$ is called effectively locally Lipschitz in the second argument if there exists a computable sequence $\{K_n\}$ of positive integers such that $|f(t, x) - f(t, y)| \leq K_n |y - x|$ whenever $(t, x), (t, y) \in \overline{B(a_n, r_n)}$.

Theorem 4.4.3 (Computability of the solutions [GZB06])

Let $E \subseteq \mathbb{R}^{m+1}$ be a c.e. open set and $f : E \rightarrow \mathbb{R}^m$ be effectively locally Lipschitz in the second argument. Let (α, β) be the maximal interval of existence of the solution $x(t)$ of the initial-value problem $\dot{x} = f(t, x)$, let $x(t_0) = x_0$, and let (t_0, x_0) be a computable point in E . Then:

1. The operator $(f, x_0) \mapsto (\alpha, \beta)$ is semicomputable (i.e. α can be computed from above and β can be computed from below) and
2. The operator $(f, x_0) \mapsto x(\cdot)$ is computable.

Theorem 4.4.4 (Perturbed reachability is co-r.e.)

Consider a continuous-time dynamical system, with \mathbf{f} locally Lipschitz, computable, whose domain is a computable compact, then, for all computable \mathbf{x} , $R_\omega^{\mathcal{H}}(\mathbf{x}) = \text{cls}(R_\omega^{\mathcal{H}}(\mathbf{x})) \subseteq \mathbb{R}^d$ is a co-c.e. closed subset.

Its proof can be considered as the main technical result established in [PBV95]. An alternative proof is similar to Theorems 4.2.28 and 4.2.6: adapt the construction of the involved graph G_m to cover the flow of the trajectory. With our hypotheses, the solutions are defined over all \mathbb{R} . It is proved in [GZB06] that Lipschitz (and even effectively locally Lipschitz) homogeneous computable ODEs have computable solutions over their maximal domain.

Corollary 4.4.5 (Robust \Rightarrow decidable)

Assume the hypotheses of Theorem 4.4.4. If $R^{\mathcal{H}}$ is robust then for all computable \mathbf{x} , then for all computable \mathbf{x} , $R^{\mathcal{H}}(\mathbf{x}) = \text{cls}(R^{\mathcal{H}}(\mathbf{x})) \subseteq \mathbb{R}^d$ is computable.

4.5 Other type of perturbations

Inspired by analogue computations [BGP17], when time has been related to the length of trajectories, we can also consider time or length perturbations.

Time-perturbation We can start by considering time-perturbed TM. The idea is that given $n > 0$, the n -perturbed version of M is unable to remain correct after a time n . Given $n > 0$, the n -perturbed version of M is defined exactly likewise, except after a time greater than n , its internal state q can change in a non-deterministic manner, if $q \neq q_{\text{reject}}$. The associated language is $L^n(M)$. From definitions: $L(M) \subseteq L^\omega(M) \subseteq \dots \subseteq L^2(M) \subseteq L^1(M)$. We call a language L *length-robust* if there exists a Turing machine M such that $L = L(M) = L^\omega(M)$.

Theorem 4.5.1 (*Length robust \Rightarrow decidable*)

$L^\omega(M)$ is in the class Π_1 . Consequently, whenever $L^\omega(M) = L(M)$, $L(M)$ is decidable.

Proof. For a word w , $w \notin L^\omega(M)$, iff there exists $n \in \mathbb{N}$ such that $w \notin L^n(M)$. As $L^n(M)$ is decidable uniformly in n , the complement of $L^\omega(M)$ is computably enumerable, as it is the uniform in n union of decidable sets. We get that $L^\omega(M) \in \Pi_1^0$ (co-computably enumerable). \square

Theorem 4.5.2

When M always stops, $L^\omega(M) = L(M)$.

Proof. We directly have $L(M) \subseteq L^\omega(M)$. Let $w \in L^\omega(M) = \bigcap_{n \in \mathbb{N}} L^n(M)$, so $\forall n \in \mathbb{N}$, $w \in L^n(M)$. By contradiction, we assume that $L(M) \subsetneq L^\omega(M)$. So there exists $w \in L^\omega(M)$ such that $w \notin L(M)$. Since M always terminates, it rejects w after using a time $q(\ell(w))$. But, then $w \notin L^n(M)$ for any $n \geq q(\ell(w)) + 2$ and hence $w \notin L^\omega(M)$, a contradiction. \square

Definition 4.5.3 (*Level of robustness n given by t*)

Given $t : \mathbb{N} \rightarrow \mathbb{N}$, we write $L^{\{t\}}(M)$ for the set of words accepted by M with time perturbation t : $L^{\{t\}}(M) = \{w \mid w \in L^{t(\ell(w))}(M)\}$.

A length-robust dynamical system is necessarily t -robust for some function t , according to the next definition:

Definition 4.5.4 (*t -robust to time language*)

We say that a language L is t -robust to time if there exists a TM M such that $L = L(M) = L^{\{t\}}(M)$.

Obviously, if a language is t -robust to time for some function $t : \mathbb{N} \rightarrow \mathbb{N}$, then it is length-robust.

Theorem 4.5.5 (Polynomially robust to time \Leftrightarrow PTIME)

A language L is in **PTIME** iff for some M and some polynomial p , $L = L(M) = L^{\{p\}}(M)$.

Furthermore, any **PTIME** language is reducible to PAM's reachability: $R^{\mathcal{H}} = R^{P,p}$ for some polynomial p .

Proof. This can be established for space perturbation. In an independent view, the intuition of the proof is that the polynomial in n can be seen as a time-out. M works in polynomial time $p(n)$, so in at most $p(n)$ steps, so the machine for $L^n(M)$ can reject if it has not accepted or rejected in $p(n)$ steps.

(\Rightarrow) If M always terminates and works in polynomial time, then there exists a polynomial q that bounds the execution time of M , so we have a polynomial $p := q(n) + 1$ such that, for $n \in \mathbb{N}$, $L^{\{p\}}(M) \subseteq L(M)$. We have the other inclusion by definition.

(\Leftarrow) We have $L^{\{p\}}(M) \in \mathbf{PTIME}$ and since $L^{\{p\}}(M) = L$, then $L \in \mathbf{PTIME}$. \square

Length-perturbation As we said, inspired by analogue computations [BGP17], we can also consider length perturbations: Fix a distance $\delta(\cdot, \cdot)$ over the domain X . A finite trajectory of a discrete-time dynamical system \mathcal{H} is a finite sequence $(\mathbf{x}_t)_{t \in 0 \dots T}$ such that $\mathbf{x}_{t+1} = \mathbf{f}(\mathbf{x}_t)$ for all $0 \leq t < T$. Its associated *length* is defined as $\mathcal{L} = \sum_{i=0}^{T-1} \delta(\mathbf{x}_i, \mathbf{x}_{i+1})$.

We consider a *length-perturbed* discrete-time dynamical system: given $L > 0$, the L -perturbed version of the system is unable to remain correct after a length L . We define $R^{\mathcal{H},L}(\mathbf{x}, \mathbf{y})$ as there exists a finite trajectory of \mathcal{H} from \mathbf{x} to \mathbf{y} of length $\mathcal{L} \leq L$.

When considering TMs as dynamical systems, $\delta(\cdot, \cdot)$ is a distance over configurations of TMs. Word w is said to be accepted in length d if the trajectory starting from $C_0[w]$ to the accepting configuration has length $\leq d$.

Definition 4.5.6

Distance $\delta(C, C')$ is called *time-metric* iff for all configurations C and C' of a Turing machines, if $C \vdash C'$, we have $\delta(C, C') \leq p(\ell(C))$, and $\delta(C, C') \geq \frac{1}{p(\ell(C))}$ for some polynomial p .

Write $\mathcal{L}(M, t)$ for the set of words accepted by M in length less than t .

Definition 4.5.7 (Tolerating some level of robustness L given by f)

Given $f : \mathbb{N} \rightarrow \mathbb{N}$, we write $L^{(f)}(M)$ for

$$L^{(f)}(M) = \{w \mid w \in \mathcal{L}(M, f(\ell(w)))\}.$$

Theorem 4.5.8 (Length robust for some time-metric distance \Leftrightarrow PTIME)

Assume $\delta(\cdot, \cdot)$ is time metric. Then, a language L is in **PTIME** iff for some TM M and some polynomial $p(n)$, $L = L(M) = L^{(f)}(M)$.

Proof. Let w be the input of size n . The execution of a TM is a sequence $(C_i) = (q_i, l_i, r_i)$.

(\Rightarrow) If L is in **PTIME**, so there is a TM M that computes L in polynomial time $p(n)$. Since the distance between two successive configurations C_i, C_{i+1} is bounded by a polynomial q in the length of C_i , and since $\ell(C_i) \leq n + i \leq n + p(n)$, for $i \leq p(n)$, we have that the total length \mathcal{L} is

$$\mathcal{L} \leq \sum_{i=0}^{p(n)-1} d(C_i, C_{i+1}) \leq \sum_{i=0}^{p(n)-1} q(n + p(n)),$$

which is a polynomial in n . Thus L is computable in polynomial length.

(\Leftarrow) If L is computable in polynomial length $p(n)$. Let T be fixed. Then we have, for all $i \in \{1 \dots T\}$: $d(C_i, C_{i+1}) \geq \frac{1}{\text{poly}(\ell(C_i))}$, thus

$$\sum_{i=0}^{T-1} d(C_i, C_{i+1}) \geq \frac{T}{\text{poly}(\ell(C_{\min}))},$$

where C_{\min} is chosen to minimise the previous lower bound.

Take $T = p(n) \times \text{poly}(\ell(C_{\min}))$, we have that a TM simulating the trajectory will accept or reject after a polynomial number of steps, thus $L \in \mathbf{PTIME}$. \square

One way to obtain a distance $\delta(C, C')$ is to take the Euclidean distance between $\Upsilon(C)$ and $\Upsilon(C')$ for $\gamma = \gamma_{[0,1]}$, where $\gamma_{[0,1]}$ and Υ are the functions considered in Section 2.3.1.

Proposition 4.5.9

The obtained distance is time-metric.

Proof. We consider $C_i = (q_i, l_i, r_i)$ and $C_{i+1} = (q_{i+1}, l_{i+1}, r_{i+1})$, with $C_i \vdash C_{i+1}$. We write $\bar{l}_i = \gamma(l_i)$ and $\bar{r}_i = \gamma(r_i)$. Then, from the definition of Υ and $\gamma_{[0,1]}$:

- $|\bar{r}_{i+1} - \bar{r}_i| \leq 1$.
- $|\bar{l}_{i+1} - \bar{l}_i| \leq 1$.
- And the gaps between \bar{r}_{i+1} and \bar{r}_i and \bar{l}_{i+1} and \bar{l}_i remain polynomial in the size of a configuration.

This provides property 1. By the encoding of the real numbers over the tapes of the TMs, the gap between two consecutive configurations is at least $\frac{1}{2}$ (we assume that the TM is not allowed not to do anything: that would correspond to a looping situation). This provides property 2. \square

Given $f : \mathbb{N} \rightarrow \mathbb{N}$, we write $R^{M,(f)}$ for the set of words accepted by M with length perturbation f : $R^{M,(f)} = \{w \mid w \in R^M, f(\ell(w))\}$.

Theorem 4.5.10 (Polynomially length robust \Leftrightarrow PTIME)

Assume distance d is time-metric and $R^M = R^{M,(p)}$ for some polynomial p . Then $R^M \in \mathbf{PTIME}$.

Proof. Since d is time-metric, a polynomial time and polynomial length are essentially the same, so the proof is very analogous to the one of Theorem 4.5.8. \square

4.6 Properties of provably robust TM

We adopt a point of view that may help us understand the type of results we established for dynamical systems in the previous sections. We will see in Chapter 6 that being robust means that some facts, here reachability properties, hold provably. For example, Proposition 4.2.14 states that reachability is true or (ϵ —) false. Somehow, we have a proof that (non-)reachability holds. We illustrate this idea in this section with TMs and what robustness means in that context.

In this section we fix some concept of proof: say, provability in first-order logic starting from Peano's axioms of arithmetic.

On some input w , a Turing machine either halts (in that case the sequence of successive configurations starting on $C_0[w]$ provides a proof of that fact) or it does not. From Gödel's incompleteness theorem (Theorem 2.7.1), there is a difference between provable valid statements and valid statements. We thus distinguish the case where there exists a proof of non-termination from the case where there is no proof of it. This leads to the introduction of the following concept:

Definition 4.6.1 (*Provably-Robust Turing machine*)

A Turing machine M is provably robust if, for all inputs, either the machine M halts or there is proof that it does not halt.

In other words, a non-provably robust Turing machine is a Turing for which there is an input w on which it does not halt, but for which there is no proof of that fact.

NB 4.6.2

We dealt with another notion of robustness of TMs in Section 4.1. It is not clear if those definitions of robustness and provable-robustness are the same or are equivalent.

Theorem 4.6.3 (*A statement for Turing machines*)

The halting problem is decidable for provably robust Turing machines.

Proof. Given some Turing machine M and some input w , run the two following procedures in parallel

1. simulate M on w : if M accepts w , then halt and accept; if M rejects w , then halt and reject.
2. search for a proof π of the fact that M does not halt on w . If such a proof is found, then halt and reject.

By the robustness hypothesis, this algorithm always halts and is correct. \square

Definition 4.6.1 has similarities with the concept of robustness for Turing machines, and more general dynamical systems, considered in [AB01]. The reasoning for establishing decidability can be interpreted as a variation on the previous proof, by playing on the concept of proof π involved: the problem is proved c.e. and co-c.e., hence decidable.

NB 4.6.4

There is no “free lunch theorem”: we can not decide if a Turing machine is provably

robust.

Uncompleteness and relative completeness

The statement of Theorem 4.6.3 might be considered as frustrating, as it does not provide an understanding of what proof of non-termination looks like. We show in this section that this corresponds to finding an invariant, preserved by the machine's transitions. We state that there is a parallel between what is done in this section for Turing machines and our discussion about more general dynamical systems.

Provably halting

Using the same ideas as in the proof of Theorem 2.7.4, we can derive the following theorem. Given a configuration C , we write $C \models^I \phi$ for the fact that $\phi \in \text{Assn}$ holds in C . As in Hoare's logic, this is a semantic notion (I stands for interpretation).

Theorem 4.6.5 (Relative Completeness)

A Turing machine M provably does not halt (resp. does not halt on input w) iff there is some formula $A^* \in \text{Assn}$ (Section 2.7) such that:

1. *Initialisation:* $C_0[w] \models^I A^*$: any (resp. the) initial configuration satisfies A^*

2. A^* is provably an invariant:

$$\frac{C \models^I A^* \quad C \vdash C'}{C' \models^I A^*} \text{invariance of } A^*$$

3. A^* implies non-termination: $C \models^I A^* \Rightarrow \text{nonhalting}(C)$.

Proof. If there is such an $A^* \in \text{Assn}$, then the Turing machine M is not terminating. Conversely, the point is that there always exists such an A^* .

The key is that Assn (first-order logic or arithmetic), or (any sufficiently expressive logic) is *expressive*: for every program c and assertion $B \in \text{Assn}$ there is an assertion $A_0 \in \text{Assn}$ corresponding to $\text{wp}(c, B)$.

Theorem 4.6.6 ([Win93])

Assn is expressive.

As a consequence, $\{A\}c\{B\}$ holds iff only $A \Rightarrow A_0$ iff it can be established using the rule of consequence in Hoare's logic. In the specific case of a triplet $\{A\}c\{\text{false}\}$ corresponding to the non-termination of Turing machine C , we get Theorem 4.6.5. \square

NB 4.6.7

It is also possible to do an independent direct proof of the theorem. Consider **NotHalt** to be the set of configurations C from which the execution of the Turing machine diverges. If we write \mathcal{C} for the set of configurations, the trick is that for a configuration

C and an interpretation I , we have $C \in \mathbf{NotHalt}$ iff

$$\begin{aligned} & \forall k \forall C_0, \dots, C_k \in \mathcal{C} [C = C_0 \& \\ & \forall i (0 \leq i < k). (C_i \models^I \mathbf{nonhalting}(C_i) \& \\ & C_i \vdash C_{i+1}) \Rightarrow (C_k \models^I \mathbf{nonhalting}(C_k))] \end{aligned} \quad (4.1)$$

This can be written as an equivalent formula of Assn, using β -predicate of Gödel (see e.g. [Win93, Lemma 7.4]). The obtained formula satisfies all the conditions 1), 2) and 3) of Theorem 4.6.5.

In short, relatively complete means the difficulty is “just” being able to prove assertions such as $A \Rightarrow w[c, B]$ (i.e. establishing $\models A \Rightarrow \mathbf{NotHalt}$) whenever $\models A \Rightarrow \mathbf{NotHalt}$, but not on the method of Theorem 4.6.5.

And Theorem 4.6.5, means that a Turing machine is non-provably terminating iff there is an invariant that is preserved by transitions of the Turing machine and that holds provably.

4.7 Chapter Conclusion

We have proposed a theory explaining in an unified framework of various statements relating robustness, defined as having a reachability relation non-sensitive to infinitesimal perturbations, to decidability. We extended the approach of [AB01] in various directions. In particular, we studied complexity properties of the problems, and not only computability.

Most of the statements in the spirit of the *robustness conjecture* have been established using arguments from computability over the rationals or the reals, playing with variations on the statement that a semi-computable and co-semi-computable set is decidable.

We also related the approach of [AB01] to the concept of δ -decidability of [GAC12], as well as the drawability of the associated dynamics, linked with computable analysis.

Notice that the proposed approach could also cover the so-called hybrid systems without deep difficulties. Various models have been considered in the literature for such systems, but one common point is that they all correspond to continuous-time dynamical systems, where the dynamics might be discontinuous, so not computable. In a very general view, a *hybrid system* \mathcal{H} is given by a set $X \subseteq \mathbb{R}^d$, a semi-group T and a flow function $\phi : X \times T \rightarrow X$ satisfying $\phi(\mathbf{x}, 0) = \mathbf{x}$ and $\phi(\phi(\mathbf{x}, t), t') = \phi(\mathbf{x}, t + t')$.

Regarding analogue models of computation, the theory developed here can be used to prove formally that space complexity corresponds to precision for continuous-time models of computation. This is what we do in the next chapter. It has been done over some compact domains, providing a more natural measure than the conditions considered in [BGGP23]. A variation on our concept of robustness can also be used to provide a characterisation of **PSPACE** for discrete-time ordinary differential equations on non-bounded domains: this is also presented in Chapter 5.

We will extend the results of this chapter in Chapter 6. More precisely, we will adopt a topological point of view, consider more general dynamical systems, in particular, defined by group actions, and generalise the result for reachability to invariance properties. This will allow us to apply the notion of robustness to subshifts and tilings.

Chapter 5

Algebraic Characterisations of FPSPACE

L'univers engendre la complexité. La complexité engendre l'efficacité. Mais l'efficacité n'engendre pas nécessairement le sens. Elle peut aussi conduire au non-sens.

Hubert Reeves

NB 5.0.1

This chapter is based on articles published in MFCS 2023 ([BB23], it received the Best Paper Award of the conference) and ICALP 2024 [BB24b], except for Section 5.3. A journal version of the MFCS paper has been accepted for publication in the Journal of Logic and Analysis ([BB25]). We submitted a journal version of the ICALP paper. Articles are both co-authored with Olivier Bournez.

We propose and prove in this chapter algebraic characterisations of **FPSPACE**, using *discrete* and *continuous* ODEs, still in the framework of computable analysis. Notice that some characterisations of complexity classes corresponding to **PSPACE** were obtained [BCdNM03, BCdNM05] in the BSS model. There exist previous characterisations of **FPSPACE** without explicit bounds, but for very different models of computation, e.g. for term rewrite systems in [MP08] and for λ -calculus in [GMR08]. As far as we know, this is the first time a characterisation of **FPSPACE** with discrete ODEs is proved.

If we forget the context of discrete ODEs, **FPSPACE** has been characterised in [Tho72] but using a bounded recursion scheme, i.e. requiring some explicit bound in the spirit of Cobham's statement [Cob62]. We avoid this issue by considering numerically stable schemes, which are very natural in the context of ODEs.

It is not the first time **FPSPACE** is characterised using continuous ODEs. However, the existing characterisation [Goz22, BGGP23] has complicated conditions on ODEs (see Section 2.6), while we have a simpler statement linking complexity to precision in a direct manner. Notice that the latter approach dealt with polynomial ODEs, while we do not restrict to polynomial ODEs. We obtain our statements by revisiting the approach of the latter papers but working over a compact domain and dealing with error correction more finely.

We argue here that space complexity is polynomially related and conversely to the numerical stability of ODEs and their associated precision. We prove that a problem can be

solved in polynomial space if and only if it can be simulated by some numerically stable ODE, using a polynomial precision. We prove it holds both for classical complexity over the discrete (functions over the integers) and for space complexity for real functions in the model of computable analysis. We give and prove here rather simple characterisations in the model of computable analysis and without explicit bounds.

We start by giving a characterisation of **FPSPACE** with a scheme of discrete ODEs. The proof is very similar to the characterisation of **FPTIME** by $\overline{\text{LDL}}^\circ$ in Chapter 3. The main difference is the scheme of discrete ODEs we consider.

Then, we give a characterisation of **FPSPACE** using continuous ODEs, hence with a more natural concept of continuous time associated with ODEs. We both characterise the case of polynomial space functions over the integers and the reals. The proof relies on two inclusions, involving the characterisation with discrete ODEs: The first uses some original polynomial space method for solving ODEs. For the other, we prove that Turing machines, with a proper representation of real numbers, can be simulated by continuous ODEs and not just discrete ODEs.

A significant consequence of our results is that the associated space complexity is provably related to the numerical stability of involved schemas and the associated required precision. We obtain that a problem can be solved in polynomial space if and only if it can be simulated by some numerically stable ODE, using a polynomial precision.

In Chapter 3, we had the motto **time complexity = length**. In this chapter, we establish that space complexity corresponds to the precision we need in the computation. Therefore, we get a motto of the form **space complexity = precision** here.

We organise this chapter as follows: in Section 5.1 we provide a characterisation of polynomial space for functions over the reals, using discrete ODEs. We use ideas and proof from Chapter 3. We generalise our framework, by characterising **FPSPACE** with *continuous* ODEs in Section 5.2.

5.1 Algebraic characterisation of PSPACE for functions over the reals with discrete ODEs

We prove that by replacing the linear length ODE with a robust linear ODE scheme (Definition 5.1.1) in $\overline{\text{LDL}}^\circ$, we get a class $\overline{\text{RLD}}^\circ$ (this stands for Robust Linear Derivation) with similar statements but for **FPSPACE**.

Definition 5.1.1 (Robust linear ODE)

A bounded function \mathbf{f} is robustly linear ODE definable from $\mathbf{g}, \mathbf{h}, \mathbf{u}$, with \mathbf{u} essentially linear in $\mathbf{f}(x, \mathbf{y})$ if:

1. it corresponds to the solution of

$$\mathbf{f}(0, \mathbf{y}) = \mathbf{g}(\mathbf{y}) \quad \text{and} \quad \frac{\delta \mathbf{f}(x, \mathbf{y})}{\delta x} = \mathbf{u}(\mathbf{f}(x, \mathbf{y}), \mathbf{h}(x, \mathbf{y}), x, \mathbf{y}), \quad (5.1)$$

2. where the schema (5.1) is polynomially numerically stable (see below).

The main issue for space is the need to prove the schema given by Definition 5.1.1

guarantees \mathbf{f} is in **FPSPACE** when \mathbf{u} , \mathbf{g} , and \mathbf{h} are. Assuming condition (1) of Definition 5.1.1 would not be sufficient: the problem is that $\mathbf{f}(x, \mathbf{y})$ may polynomially grow too fast or have a modulus function that would grow too fast. The point is, in Definition 5.1.1, we assumed \mathbf{f} to be both bounded and satisfying (2), i.e. polynomial numerical robustness. (2) means formally there exists some polynomial p such that, for all integer n , for $\varepsilon(n) = p(n + \ell(\mathbf{y}))$, with $a =_\varepsilon b$ standing for $|a - b| \leq \varepsilon$, considering any solution of

$$\begin{cases} \tilde{\mathbf{y}} &=_{2^{-\varepsilon(n)}} \mathbf{y} \\ \tilde{\mathbf{h}}(x, \tilde{\mathbf{y}}) &=_{2^{-\varepsilon(n)}} \mathbf{h}(x, \tilde{\mathbf{y}}) \\ \tilde{\mathbf{f}}(0, \tilde{\mathbf{y}}) &=_{2^{-\varepsilon(n)}} \mathbf{g}(\mathbf{y}) \\ \frac{\delta \tilde{\mathbf{f}}(x, \tilde{\mathbf{y}})}{\delta x} &=_{2^{-\varepsilon(n)}} \mathbf{u}(\tilde{\mathbf{f}}(x, \tilde{\mathbf{y}}), \tilde{\mathbf{h}}(x, \tilde{\mathbf{y}}), x, \tilde{\mathbf{y}}) \end{cases}$$

then, we have:

$$\tilde{\mathbf{f}}(x, \tilde{\mathbf{y}}) =_{2^{-\varepsilon(n)}} \mathbf{f}(x, \mathbf{y}).$$

These hypotheses are sufficient to work with the precision given by this robustness condition. These conditions guarantee the validity of computing with such approximated values.

At a technical level, all our results are still obtained by proving Turing machines can be suitably simulated with analytic discrete ODEs. We believe our constructions could be applied to many other situations, where programming with ODEs is needed.

NB 5.1.2

For linear length ODEs, we did not have to explicit the numerical stability as a hypothesis, as it comes directly from the fact that we consider solutions at most at some logarithmic with respect to their arguments. But this is required here to guarantee the computability of the solution (and even polynomial space computability).

NB 5.1.3

Notice that, over the continuum, even computable ODEs may have no computable solution [PER79]. Over the discrete, not all dynamics can be simulated and numerical stability becomes an issue. See Section 2.1 for more details.

In other words:

Proposition 5.1.4

All functions of \mathbb{RLD}° are computable (in the sense of computable analysis) in polynomial space.

$$\mathbb{RLD}^\circ = \left[\mathbf{0}, \mathbf{1}, \pi_i^k, \ell(x), +, -, \tanh, \frac{x}{2}, \frac{x}{3}; \text{composition, linear robust discrete ODE} \right]$$

5.1.1 Characterisation of FPSPACE and generalisations

We now prove that \mathbb{RLD}° provides a characterisation of **FPSPACE**.

First, the discrete part of \mathbb{RLD}° characterises **FPSPACE** for functions in $(\mathbb{N}^d)^{\mathbb{N}^{d'}}$:

Theorem 5.1.5

$$\text{DP}(\text{RLD}^\circ) = \mathbf{FPSPACE} \cap \mathbb{N}^{\mathbb{N}}$$

Theorem 5.1.6 (Generic functions over the reals)

$$\overline{\text{RLD}^\circ} \cap \mathbb{R}^{\mathbb{R}} = \mathbf{FPSPACE} \cap \mathbb{R}^{\mathbb{R}}.$$

More generally:

$$\overline{\text{RLD}^\circ} \cap \mathbb{R}^{\mathbb{N}^d \times \mathbb{R}^{d'}} = \mathbf{FPSPACE} \cap \mathbb{R}^{\mathbb{N}^d \times \mathbb{R}^{d'}}.$$

We obtain these statements for polynomial space computability (Theorems 5.1.5 and 5.2.2) replacing LDL° by RLD° , using similar reasoning about space instead of time in Proposition 3.2.26, and Proposition 5.1.7 instead of Proposition 3.2.23.

Namely, we prove:

Proposition 5.1.7

Consider some Turing machine M computing some function $f : \Sigma^* \rightarrow \Sigma^*$ in space $S(\ell(w))$, for some polynomial S , on input w . There exists some function $\tilde{f} : \mathbb{N}^2 \times \mathbb{R} \rightarrow \mathbb{R}$ in RLD° such that $\|\tilde{f}(2^m, 2^{S(\ell(w))}, \gamma_{\text{word}}(w)) - \gamma_{\text{word}}(f(w))\| \leq 2^{-m}$.

Proof. The idea is the same as Proposition 3.2.23, but not working with powers of 2 and with linear ODE: define the function $\overline{\text{Exec}}$ that maps some time t and some initial configuration C to the configuration at time t . This can be obtained using the fact that:

$$\begin{cases} \overline{\text{Exec}}(2^m, 0, 2^S, C) &= C \\ \overline{\text{Exec}}(2^m, t+1, 2^S, C) &= \overline{\text{Next}}\left(2^m, 2^S, \overline{\text{Exec}}(2^m, t, 2^S, C)\right). \end{cases}$$

In order to claim this is a robust linear ODE, we need to state that $\overline{\text{Exec}}(2^m, t, 2^S, C)$ is polynomially numerically stable: but this holds, since to estimate this value at 2^{-n} it is sufficient to work at precision $4^{-\max(m, n, S+2)}$ (independently of t , from the rounding).

We can then get the value of the computation as $\overline{\text{Exec}}(2^m, 2^{S(\ell(w))}, 2^{S(\ell(w))}, C_{\text{init}})$ on input w , considering $C_{\text{init}} = (q_0, 0, \gamma_{\text{word}}(w))$. By applying some projection, we get the following function $\tilde{f}(2^m, 2^S, y) = \pi_3^3\left(\overline{\text{Exec}}(2^m, S, 2^S, (q_0, 0, y))\right)$ that satisfies the property.

□

The proof of Theorem 5.1.6 is then direct, using similar to those of Chapter 3 for **FPTIME**.

5.2 Characterisation with Continuous ODEs

When considering space complexity, we now consider the case of ODEs with classical derivation. Hence, we want to use only functions over the reals and continuous derivation

(Definition 1.2.1). As mentioned before, for functions over the reals, an important issue is numerical stability. While discussing all these issues, we propose an algebraic characterisation of **PSPACE**, using *continuous* ODEs with the following algebra (\mathbb{RCD} stands for Robust Continuous Differential) (Schema *robust continuous ODE* is formally defined in Definition 5.2.3):

$$\mathbb{RCD} = \left[0, 1, \pi_i^k, +, -, \times, \tanh, \cos, \pi, \frac{x}{2}, \frac{x}{3}; \text{composition}, \text{robust continuous ODE} \right].$$

where \cos is the cosinus function.

Theorem 5.2.1

$$\text{DP}(\mathbb{RCD}) = \mathbf{FPSPACE} \cap \mathbb{N}^{\mathbb{N}}$$

We prove this theorem in Subsection 5.2.6. We also characterise functions over the reals computable in polynomial space, inspired by the characterisation of Section 5.1: we add a limit schema *ELim* to \mathbb{RCD} . If we consider

$$\overline{\mathbb{RCD}} = \left[0, 1, \pi_i^k, +, -, \times, \tanh, \cos, \pi, \frac{x}{2}, \frac{x}{3}; \text{composition}, \text{robust continuous ODE}, \text{ELim} \right]$$

then:

Theorem 5.2.2 (Generic functions over the reals)

$\overline{\mathbb{RCD}} \cap \mathbb{R}^{\mathbb{R}} = \mathbf{FPSPACE} \cap \mathbb{R}^{\mathbb{R}}$. More generally:

$$\overline{\mathbb{RCD}} \cap \mathbb{R}^{\mathbb{N}^d \times \mathbb{R}^{d'}} = \mathbf{FPSPACE} \cap \mathbb{R}^{\mathbb{N}^d \times \mathbb{R}^{d'}}.$$

We prove this theorem in Subsection 5.2.7.

A first step is to study how to solve continuous ODEs and managing space.

5.2.1 Solving efficiently ODEs: a space efficient method

As we saw in Section 2.1, it is not easy to solve ODEs in the general case and it might even be non-computable or with unbounded complexity. We propose an original alternative approach to optimize space complexity: this can be seen as either using a non-deterministic algorithm that “guesses” the correct intermediate positions of the dynamics or, from the proof of Savitch’s theorem approach, as an original recursive method to solve ODEs.

Concretely, from the flow property, a strategy to compute $\Phi(\mathbf{f}_0, T)$ is either to use a particular numerical method if T is small, says smaller than $\Delta > 0$. Otherwise, we know that $\Phi(\mathbf{f}_0, T) = \Phi(\mathbf{z}, T/2)$, the trajectory at time T starting from \mathbf{f}_0 , where $\mathbf{z} = \Phi(\mathbf{f}_0, T/2)$. This always holds, so if we can compute both quantities, we will solve the problem.

The difficulty is that we cannot precisely compute \mathbf{z} in practice, but some numerical approximation $\tilde{\mathbf{z}}$. If the system is numerically stable, we may assume this strategy works. The case when this strategy will not work is if the trajectory starting from $\tilde{\mathbf{z}}$, for the second half of the work from time $T/2$ to T , has a behaviour different from the one starting in \mathbf{z} : in other words, if there is a high instability somewhere, namely in \mathbf{z} .

This leads to the following concept:

Definition 5.2.3 (Robust (continuous) ODE)

A function $\mathbf{f} : \mathbb{R}^{d+1} \rightarrow \mathbb{R}^{d'}$ is robustly ODE definable (from initial condition \mathbf{g} , and dynamic \mathbf{u}) if

1. it corresponds to the solution of the following continuous ODE:

$$\mathbf{f}(0, \mathbf{x}) = \mathbf{g}(\mathbf{x}) \quad \text{and} \quad \frac{\partial \mathbf{f}(t, \mathbf{x})}{\partial t} = \mathbf{u}(\mathbf{f}(t, \mathbf{x}), t, \mathbf{x}), \quad (5.2)$$

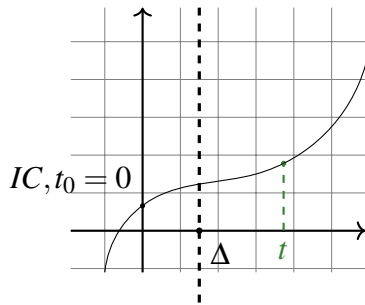
2. and there is some rational $\Delta > 0$, and some polynomial p such that the schema (5.2) is (polynomially) numerically stable on $[0, \Delta]$: for all integer n , considering $\eta(n) = p(n + \ell(\lceil \mathbf{x} \rceil))$ we can compute $\mathbf{f}(t, \mathbf{x})$ at precision 2^{-n} by working a precision $2^{-\eta(n)}$: if you consider any solution of $\tilde{\mathbf{x}} =_{2^{-\eta(n)}} \mathbf{x}$, $\tilde{\mathbf{f}}(0, \tilde{\mathbf{x}}) =_{2^{-\eta(n)}} \mathbf{g}(\mathbf{x})$ and $\frac{\partial \tilde{\mathbf{f}}(t, \tilde{\mathbf{x}})}{\partial t} =_{2^{-\eta(n)}} \mathbf{u}(\tilde{\mathbf{f}}(t, \tilde{\mathbf{x}}), t, \tilde{\mathbf{x}})$ then $\tilde{\mathbf{f}}(t, \tilde{\mathbf{x}}) =_{2^{-n}} \mathbf{f}(t, \mathbf{x})$ when $0 \leq t \leq \Delta$.
3. For $t \geq \Delta$, we can compute $\mathbf{f}(t, \mathbf{x})$ at precision 2^{-n} by computing some approximation $\widetilde{\mathbf{f}(t/2, \mathbf{x})}$ of $\mathbf{f}(t/2, \mathbf{x})$ at precision $2^{-\eta(n)}$, i.e. of $\Phi(\mathbf{g}(\mathbf{x}), t/2)$, and some approximation of $\Phi(\widetilde{\mathbf{f}(t/2, \mathbf{y})}, t/2)$, working at precision $2^{-\eta(n)}$.

NB 5.2.4

For more clarity and conciseness, we will assume in the proofs that $d = d' = 1$, as it can be easily extended to more general cases.

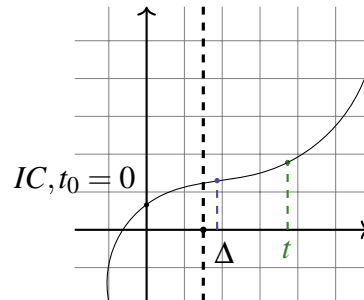
Example 5.2.5

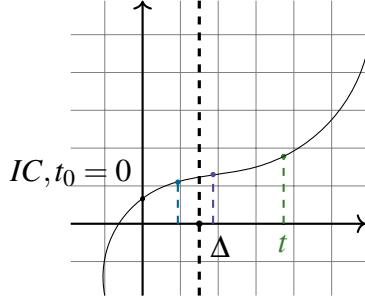
We illustrate the idea of our method. Consider the following dynamic corresponding to solving some ODE $f'(t) = u(f(t))$, where IC stands for “Initial Condition”:



We want to solve it, meaning computing $u(t)$ at t , with $t \notin [0, \Delta]$ at precision 2^{-n} , with initial condition $t_0 = 0$.

For that we do the computation at $\frac{t}{2}$. Here, $\frac{t}{2} \notin [0, \Delta]$.



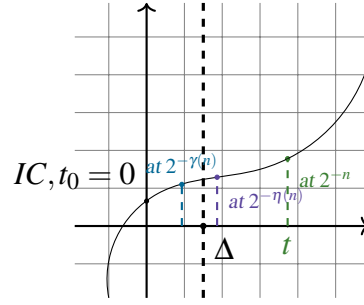


Thus, we do the computation at $\frac{t}{4}$. Here $\frac{t}{4} \in [0, \Delta]$. Then, we can compute $u(t)$ from $u(\frac{t}{2})$ and $u(\frac{t}{4})$.

Concretly:

$$\{0, \frac{x}{4}, \frac{x}{2}, \frac{3x}{4}\}.$$

- Computation at $t = x$ is done at precision 2^{-n} , with IC $t = 0$;
- Computation at $t = t_0 + \frac{x}{2}$ is done at precision $2^{-\eta(n)}$, with IC $t_0 \in \{0, \frac{x}{2}\}$;
- Computation at $t = t_0 + \frac{x}{4}$ is done at precision $2^{-\gamma(n)}$, with IC $t_0 \in$



with $\eta(\cdot)$ and $\gamma(\cdot)$ being polynomials in our context.

Using this principle, we get:

Theorem 5.2.6

Consider an IVP as in the previous definition. If \mathbf{g} and \mathbf{u} are computable in polynomial space, then the solution \mathbf{f} can be computed in polynomial space.

Intuitively, from the definitions and previous arguments, all bits of $\Phi(\mathbf{y}, t)$ can be computed non-deterministically with precision 2^{-n} using computations with precision $\eta(n)$, hence is in **NPSpace**, so in **PSPACE** by Savitch theorem (Theorem 1.3.30). It can also be proved with a deterministic polynomial space recursive algorithm.

The above theorem is the key argument to obtain one direction of our main theorems. We now go in the reverse direction. This requires talking about discrete ODEs, and some previous constructions.

Conversely, we need to simulate TMs with continuous ODEs.

5.2.2 Simulating a discrete ODE using a continuous ODE

We first prove that it is possible to simulate a discrete ODE with a continuous ODE. The underlying basic idea to simulate a discrete dynamic by an ODE can be attributed to Branicky in [Bra95] and has been improved in many ways by several authors. We present here the basic ideas, reformulated in our context. A more precise analysis will come (Proposition 5.2.26).

Definition 5.2.7 (“Ideal iteration trick” [Bra95])

Consider the following initial value problem for a discrete ODE, given by functions \mathbf{g} and \mathbf{u} :

$$\begin{cases} \mathbf{f}(0, \mathbf{x}) = \mathbf{g}(\mathbf{x}) \\ \frac{\delta \mathbf{f}}{\delta t}(t, \mathbf{x}) = \mathbf{u}(\mathbf{f}(t, \mathbf{x}), t, \mathbf{x}) \end{cases} \quad (5.3)$$

Then, let $\mathbf{G}(\mathbf{v}, t, \mathbf{x}) = \mathbf{u}(\mathbf{v}, t, \mathbf{x}) + \mathbf{v}$, and consider the (continuous) IVP:

$$\begin{cases} \mathbf{y}_1(0, \mathbf{x}) = \mathbf{y}_2(0, \mathbf{x}) = \mathbf{g}(\mathbf{x}) \\ \mathbf{y}_1' = c(\mathbf{G}(r(\mathbf{y}_2), r(t), \mathbf{x}) - \mathbf{y}_1)^3 \theta(\sin(2\pi t)) \\ \mathbf{y}_2' = c(r(\mathbf{y}_1) - \mathbf{y}_2)^3 \theta(-\sin(2\pi t)) \end{cases} \quad (5.4)$$

where c a constant, $\theta(x) = 0$ if $x \leq 0$ and $\theta(x) > 0$ if $x > 0$. Here, r is a rounding function: we mean, by construction, G preserves the integers, and r is a function that maps a real value close to some integer to this integer: assume, say, that for $z \in [n - \frac{1}{4}, n + \frac{1}{2}]$, $r(z) = n$, for any integer $n \in \mathbb{Z}$. We abusively write $r(\mathbf{y})$ for applying function $r : \mathbb{R} \rightarrow \mathbb{R}$ componentwise on vector \mathbf{y} .

NB 5.2.8

We do not need to specify what $r(z)$ values for a z not in such an interval: the following reasoning remains correct, whatever it is.

Then,

Proposition 5.2.9

The solution of continuous ODE (5.4) simulates in a continuous way the discrete ODE (5.3).

Indeed, the idea is that \mathbf{y}_1 corresponds to the actual computation of the iterates of \mathbf{G} (and hence computes the successive values of \mathbf{f}) and \mathbf{y}_2 acts as a “memory” equation. Let us detail how it works.

NB 5.2.10

We describe here an “ideal” computation, as $\theta(x)$ is exactly 0 when $x \leq 0$, and $r(z)$ is exactly some integer on suitable domains. Later in the thesis, we will deal with a not-so-ideal θ and r .

Initially, $\mathbf{f}(0, \mathbf{x}) = \mathbf{y}_1(0, \mathbf{x}) = \mathbf{y}_2(0, \mathbf{x}) = \mathbf{g}(\mathbf{x})$. For $t \in [0, 1/2]$, we have

$$\theta(-\sin(2\pi t)) = 0,$$

and hence $\mathbf{y}_2' = 0$, so \mathbf{y}_2 is fixed and kept at value $\mathbf{g}(\mathbf{x})$ for $t \in [0, \frac{1}{2}]$. Consequently, for $t \in [0, \frac{1}{2}]$, $r(\mathbf{y}_2)$ is also fixed and kept at value $\mathbf{g}(\mathbf{x})$, and $r(t)$ is also fixed and kept at value 0. Consequently, on this interval, if we write $C(t) = c\theta(\sin(2\pi t))$, then the dynamics of \mathbf{y}_1 is given by

$$\mathbf{y}_1' = C(t) \left(\mathbf{G}(\mathbf{g}(\mathbf{x}), 0, \mathbf{x}) - \mathbf{y}_1 \right)^3 \quad (5.5)$$

Then, a key is to analyse such ODE:

Lemma 5.2.11 (Analysis of ODE (5.5))

The solution $\mathbf{y}_1(t, \mathbf{x})$ of ODE (5.5) is converging to $G(\mathbf{g}(\mathbf{x}), 0, \mathbf{x})$ for any initial condition. Furthermore, for any initial condition $\mathbf{y}_1(0, \mathbf{x}) \neq G(\mathbf{g}(\mathbf{x}), 0, \mathbf{x})$, we have $\left\| \mathbf{y}_1\left(\frac{1}{2}, \mathbf{x}\right) - \mathbf{G}(\mathbf{g}(x), 0, \mathbf{x}) \right\| \leq \frac{\sqrt{2}}{2\sqrt{\int_0^{\frac{1}{2}} C(z)dz}}$.

Proof of Lemma 5.2.11, Adapted from [Bra95]. If initially, or at any instant:

$$\mathbf{y}_1(0, \mathbf{x}) = G(\mathbf{g}(x), 0, \mathbf{x})$$

then the result holds, as $\mathbf{y}'_1 = 0$, and \mathbf{y}_1 remains constant. Otherwise, we have

$$\frac{\mathbf{y}'_1}{\left(\mathbf{G}(\mathbf{g}(\mathbf{x}), 0, \mathbf{x}) - \mathbf{y}_1\right)^3} = C(t).$$

Integrating this equality between 0 and t , we obtain

$$\frac{1}{2\left(\mathbf{G}(\mathbf{g}(\mathbf{x}), 0, \mathbf{x}) - \mathbf{y}_1(t)\right)^2} - \frac{1}{2\left(\mathbf{G}(\mathbf{g}(\mathbf{x}), 0, \mathbf{x}) - \mathbf{y}_1(0)\right)^2} = \int_0^t C(z)dz,$$

hence $\frac{1}{2\left(\mathbf{G}(\mathbf{g}(\mathbf{x}), 0, \mathbf{x}) - \mathbf{y}_1(t)\right)^2} \geq \int_0^t C(z)dz$. This yields the property. \square

In particular, for any $m \in \mathbb{N}$, we can select constant c such that for any initial condition $\mathbf{y}_1(0, \mathbf{x})$,

$$\left\| \mathbf{y}_1\left(\frac{1}{2}, \mathbf{x}\right) - \mathbf{G}(\mathbf{g}(x), 0, \mathbf{x}) \right\| \leq 2^{-m}$$

Consequently, $\mathbf{y}_1(t, \mathbf{x})$ will approach $\mathbf{G}(\mathbf{g}(\mathbf{x}), 0, \mathbf{x}) = \mathbf{f}(1, \mathbf{x})$ on this interval. Thus, $\mathbf{y}_1(\frac{1}{2}, \mathbf{x}) =_\varepsilon \mathbf{f}(1, \mathbf{x})$ and $\mathbf{y}_2(\frac{1}{2}, \mathbf{x}) = \mathbf{g}(\mathbf{x})$, for some $\varepsilon > 0$, that we can consider less than $\frac{1}{4} = 2^{-2}$, by selecting a big enough constant c (just taking $m = 2$ above). At $t = \frac{1}{2}$, \mathbf{y}_1 will hence have simulated one step of discrete ODE (5.3).

Now, for $t \in \left[\frac{1}{2}, 1\right]$ the roles of \mathbf{y}_1 and \mathbf{y}_2 are exchanged : $\mathbf{y}'_1(t, \mathbf{x}) = 0$, so \mathbf{y}_1 is kept fixed, \mathbf{y}_2 approaches $r(\mathbf{y}_1) = \mathbf{f}(1, \mathbf{x})$, thus $\mathbf{y}_1(1, \mathbf{x}) =_\varepsilon \mathbf{y}_2(1, \mathbf{x}) =_\varepsilon \mathbf{f}(1, \mathbf{x})$.

By induction, from the same reasoning, we obtain that, for all $n \in \mathbb{N}$, $\mathbf{y}_1(n, \mathbf{x}) =_\varepsilon \mathbf{y}_2(n, \mathbf{x}) =_\varepsilon \mathbf{f}(n, \mathbf{x})$, and actually, we also have

$$\mathbf{y}_1\left(t + \frac{1}{2}, \mathbf{x}\right) =_\varepsilon \mathbf{y}_2(t, \mathbf{x}) =_\varepsilon \mathbf{f}(n, \mathbf{x})$$

for all $t \in \left[n, n + \frac{1}{2}\right]$, for any integer n .

To implement such an ODE, we must fix a function $\theta(x)$ with the above property. Taking $\text{ReLU}(x) = \max(0, x)$ would satisfy it, but it is not a derivable function, and hence would not lead to a (classical) ODE. We could then take $\theta(x) = 0$ for $x \leq 0$, and $\exp\left(-\frac{1}{x}\right)$ for $x > 0$. The point is that such a function is not real analytic. The base functions we

consider in our class \mathbb{RCD} are all real analytic, and real analytic functions are preserved by composition, so we cannot get such a function by compositions from our base functions. Furthermore, it is known that a real analytic function that is constant on some interval (we assumed it is 0 for $x \leq 0$) is constant. Hence, the above-considered function $\theta(x)$ cannot be real analytic. So, implementing this trick cannot be done directly using our base functions, using only compositions.

In Proposition 5.2.26, we will do a similar construction, but we improve it to deal with errors and not exact functions $\theta(z)$ and $r(x)$. Furthermore, here the purpose of the function r was to correct errors around integers, i.e. around \mathbb{Z} : this will be possibly around other $\mathbb{Z}\delta = \{n\delta | n \in \mathbb{Z}\}$ for some $\delta > 0$. We will then naturally assume that for $z \in \left[n\delta - \frac{1}{4}\delta, n\delta + \frac{1}{2}\delta\right]$, $r(z) = n$, for any integer $n \in \mathbb{Z}$, to have a similar reasoning as the one above where $\delta = 1$.

Before doing so, we need to revisit some previous constructions.

5.2.3 Revisiting some previous constructions

Our proofs rely on some constructions from Chapter 3. Concretely, we need to simulate the execution of a Turing machine (TM) by some dynamical system over the reals. This requires to encode the configurations of a Turing machine into some real numbers.

We denote by \mathbb{RCD}_* the algebra:

$$[0, 1, \pi_i^k, +, -, \times, \tanh, \cos, \pi, \frac{x}{2}, \frac{x}{3}; \text{composition}].$$

NB 5.2.12

This is close to the class:

$$\mathbb{LDL}^\circ = [0, 1, \pi_i^k, \ell(x), +, -, \tanh, \frac{x}{2}, \frac{x}{3}; \text{composition, linear length ODE}],$$

considered in Chapter 3, but without the function $\ell(x)$, and without the possibility of defining functions using linear length ODE (and with multiplication added).

We will reuse some of the construction from Chapter 3 but avoid systematically any use of linear length ODE and the length function $\ell(x)$. Furthermore, the class considered in Chapter 3 is mixing functions from the integers to the reals, and from the reals to the reals, and we need to keep only functions over the reals. So we need to avoid completely any discrete function that can be considered there, in particular any computation of 2^m from some m .

First, we improve Lemma 5.2.11 and we observe that considering $Y(x, z) = \frac{1 + \tanh(4xz)}{2}$ would yield a function in \mathbb{RCD}_* : we avoid the computation of 2^m by a substitution of a variable, and using a multiplication. We then write $\text{ReLU-}\mathfrak{s}(Y, x)$ for $xY(x, z)$: we have $|\text{ReLU-}\mathfrak{s}(2^m, x) - \text{ReLU}(x)| \leq 2^{-m}$.

In particular, this was used to prove we can uniformly approximate the continuous sigmoid functions (when $\frac{1}{b-a}$ is in \mathbb{LDL}°) defined as $\mathfrak{s}(a, b, x) = 0$ whenever $w \leq a$, $\frac{x-a}{b-a}$ whenever $a \leq x \leq b$, and 1 whenever $b \leq x$.

Lemma 5.2.13 (Uniform approximation of any piecewise continuous sigmoid)

Assume $a, b, \frac{1}{b-a}$ is in \mathbb{RCD}_* . Then there is some function $\mathcal{C}\text{-}\mathfrak{s}(z, a, b, x) \in \mathbb{RCD}_*$ such that for all integer m ,

$$|\mathcal{C}\text{-}\mathfrak{s}(2^m, a, b, x) - \mathfrak{s}(a, b, x)| \leq 2^{-m}.$$

Proof. Take $\mathcal{C}\text{-}\mathfrak{s}(z, a, b, x) = \frac{(x-a)Y(x-a, z2^{1+c}) - (x-b)Y(x-b, z2^{1+c})}{b-a}$. Observing that

$$(b-a)\mathfrak{s}(a, b, x) = \text{ReLU}(x-a) - \text{ReLU}(x-b),$$

From triangle inequality, it will hold, choosing c with $\frac{1}{b-a} \leq 2^c$. \square

In Chapter 3, Theorem 3.2.9, we proved the existence of some function corresponding to a continuous (controlled) approximation of the fractional part function in \mathbb{LDL}° . Here, we prove an approximation of $\{\cdot\}$ in \mathbb{RCD}_* .

Definition 5.2.14 (Real extension)

A real function is a real extension of a function over the integers if they coincide for integer arguments.

It is not clear that we have an extension over the reals of ξ in our algebra \mathbb{RCD}_* , but if we add a real extension of such a function, from the proof of Corollaries 3.2.10, 3.2.11, 3.2.12, 3.2.13, 3.2.1 and 3.2.14, we obtain a similar bestiary of functions.

Definition 5.2.15 ($\mathbb{RCD}_* + \xi$)

We write $\mathbb{RCD}_* + \xi$ for the algebra where some real extension of function ξ is added as a base function.

We prove in Lemma 5.2.28 that such extension exists.

Corollary 5.2.16

There exist $\xi_1, \xi_2 : \mathbb{N}^2 \times \mathbb{R} \mapsto \mathbb{R} \in \mathbb{RCD}_* + \xi$ such that, for all $n, m \in \mathbb{N}$, $\lfloor x \rfloor \in [-2^n + 1, 2^n]$, whenever $x \in \left[\lfloor x \rfloor - \frac{1}{2}, \lfloor x \rfloor + \frac{1}{4} \right]$, $|\xi_1(2^m, 2^n, x) - \{x\}| \leq 2^{-m}$, and whenever $x \in \left[\lfloor x \rfloor, \lfloor x \rfloor + \frac{3}{4} \right]$, $|\xi_2(2^m, 2^n, x) - \{x\}| \leq 2^{-m}$.

Proof. $\xi_1(M, N, x) = \xi\left(M, N, x - \frac{3}{8}\right) - \frac{1}{2}$ and $\xi_2(M, N, x) = \xi\left(N, x - \frac{7}{8}\right)$. \square

Corollary 5.2.17

There exist $\sigma_1, \sigma_2 : \mathbb{N}^2 \times \mathbb{R} \mapsto \mathbb{R} \in \mathbb{RCD}_* + \xi$ such that, for all $n, m \in \mathbb{N}$, $\lfloor x \rfloor \in [-2^n + 1, 2^n]$, whenever $x \in \left[\lfloor x \rfloor - \frac{1}{2}, \lfloor x \rfloor + \frac{1}{4} \right]$, $|\sigma_1(2^m, 2^n, x) - \lfloor x \rfloor| \leq 2^{-m}$, and whenever $x \in I_2 = \left[\lfloor x \rfloor, \lfloor x \rfloor + \frac{3}{4} \right]$, $|\sigma_2(2^m, 2^n, x) - \lfloor x \rfloor| \leq 2^{-m}$.

Proof. $\sigma_i(M, N, x) = x - \xi_i(M, N, x)$. \square

Corollary 5.2.18

There exist $\lambda : \mathbb{N}^2 \times \mathbb{R} \mapsto [0, 1] \in \mathbb{RCD}_* + \xi$ such that for all $m, n \in \mathbb{N}$, $\lfloor x \rfloor \in [-2^n + 1, 2^n]$, whenever $x \in \left[\lfloor x \rfloor + \frac{1}{4}, \lfloor x \rfloor + \frac{1}{2} \right]$, $|\lambda(2^m, 2^n, x) - 0| \leq 2^{-m}$, and whenever $x \in \left[\lfloor x \rfloor + \frac{3}{4}, \lfloor x \rfloor + 1 \right]$, $|\lambda(2^m, 2^n, x) - 1| \leq 2^{-m}$.

Proof. $\lambda(M, N, x) = \mathcal{C}\text{-}\mathfrak{s} \left(2M, 1/4, 1/2, \xi(2M, N, x - 9/8) \right)$. \square

Corollary 5.2.19

There exist $\text{mod}_2 : \mathbb{N}^2 \times \mathbb{R} \mapsto [0, 1] \in \mathbb{RCD}_* + \xi$ such that for all $m, n \in \mathbb{N}$, $\lfloor x \rfloor \in [-2^n + 1, 2^n]$, whenever $x \in \left[\lfloor x \rfloor - \frac{1}{4}, \lfloor x \rfloor + \frac{1}{4} \right]$, $|\text{mod}_2(2^m, 2^n, x) - \lfloor x \rfloor \bmod 2| \leq 2^{-m}$.

Proof. $\text{mod}_2(M, N, x) = 1 - \lambda \left(M, N/2, \frac{1}{2}x + \frac{7}{8} \right)$. \square

Corollary 5.2.20

There exist $\div_2 : \mathbb{N}^2 \times \mathbb{R} \mapsto [0, 1] \in \mathbb{RCD}_* + \xi$ such that for all $m, n \in \mathbb{N}$, $\lfloor x \rfloor \in [-2^n + 1, 2^n]$, whenever $x \in \left[\lfloor x \rfloor - \frac{1}{4}, \lfloor x \rfloor + \frac{1}{4} \right]$, $|\div_2(2^m, 2^n, x) - \lfloor x \rfloor // 2| \leq 2^{-m}$, with $//$ the integer division.

Proof. $\div_2(M, N, x) = \frac{1}{2} (\sigma_1(M, N, x) - \text{mod}_2(M, N, x))$. \square

Similarly, the equivalent of Lemmas 3.2.15, 3.2.17, 3.2.19 still hold in $\mathbb{RCD}_* + \xi$:

Lemma 5.2.21 (Generalisation of Lemma 3.2.15)

There exists $\mathcal{C}\text{-if} \in \mathbb{RCD}_* + \xi$ such that, $l \in [0, 1]$, if we take $|d' - 0| \leq 1/4$, then

$$|\mathcal{C}\text{-if}(2^m, d', l) - 0| \leq 2^{-m}$$

and if we take $|d' - 1| \leq 1/4$, then

$$|\mathcal{C}\text{-if}(2^m, d', l) - l| \leq 2^{-m}.$$

Lemma 5.2.22 (Generalisation of Lemmas 3.2.18 and 3.2.17)

Let $\alpha_1, \alpha_2, \dots, \alpha_n$ be some integers, and V_1, V_2, \dots, V_n some constants. We write

$$\text{send}(\alpha_i \mapsto V_i)_{i \in \{1, \dots, n\}}$$

for the function mapping any $x \in [\alpha_i - 1/4, \alpha_i + 1/4]$ to V_i , for all $i \in \{1, \dots, n\}$.

There is some function in $\mathbb{RCD}_* + \xi$, that we write

$$\mathcal{C}\text{-send}(2^m, \alpha_i \mapsto V_i)_{i \in \{1, \dots, n\}},$$

mapping any $x \in [\alpha_i - 1/4, \alpha_i + 1/4]$ to a real at distance at most 2^{-m} of V_i , for all $i \in \{1, \dots, n\}$.

Lemma 5.2.23 (Generalisation of Lemmas 3.2.19 and 3.2.20)

Let $N \in \mathbb{N}$. Let $\alpha_1, \alpha_2, \dots, \alpha_n$ be some integers, and $V_{i,j}$ for $1 \leq i \leq n$ some constants, with $0 \leq j < N$. We write

$$\text{send}((\alpha_i, j) \mapsto V_{i,j})_{i \in \{1, \dots, n\}, j \in \{0, \dots, N-1\}}$$

for the function that maps any $x \in [\alpha_i - 1/4, \alpha_i + 1/4]$ and $y \in [j - 1/4, j + 1/4]$ to $V_{i,j}$, for all $i \in \{1, \dots, n\}, j \in \{0, \dots, N-1\}$.

There is some function in $\mathbb{RCD}_* + \xi$, that we write:

$$\mathcal{C}\text{-send}(2^m, (\alpha_i, j) \mapsto V_{i,j})_{i \in \{1, \dots, n\}, j \in \{0, \dots, N-1\}},$$

mapping any $x \in [\alpha_i - 1/4, \alpha_i + 1/4]$ and $y \in [j - 1/4, j + 1/4]$ to a real at distance at most 2^{-m} of $V_{i,j}$, for all $i \in \{1, \dots, n\}, j \in \{0, \dots, N-1\}$.

Working with one step of a Turing machine

As in the proof of Lemma 3.2.22, we encode one transition in a Turing machine using the functions of Corollaries 5.2.16, 5.2.17, 5.2.18, 5.2.19, 5.2.20:

Lemma 5.2.24 (Generalisation of Lemma 3.2.21)

We can construct some function $\overline{\text{Next}}$ in $\mathbb{RCD}_* + \xi$ that simulates one step of M , i.e. computing the Next function sending a configuration \overline{C} of Turing machine M to \overline{C}' , where C' is the next one: $\|\text{Next}(2^m, 2^S, \overline{C}) - \overline{C}'\| \leq 2^{-m}$. Furthermore, it is robust to errors on its input, up to space S : considering $\|\tilde{C} - \overline{C}\| \leq 4^{-(S+2)}$, $\|\text{Next}(2^m, 2^S, \tilde{C}) - \overline{C}'\| \leq 2^{-m}$ remains true.

Converting integers and dyadics to words and conversely

In Chapter 3, we defined some functions for converting integers and dyadics to their encoding as words, and conversely. Namely, we consider the following encoding: every digit in the binary expansion of dyadic d is encoded by a pair of symbols in the radix 4

expansion of $\bar{d} \in \mathcal{J} \cap [0, 1]$: digit 0 (respectively: 1) is encoded by 11 (resp. 13) if before the “decimal” point in d , and digit 0 (respectively: 1) is encoded by 31 (resp. 33) if after. For example, for $d = 101.1$ in base 2, $\bar{d} = 0.13111333$ in base 4. Conversely, given \bar{d} , we provided a way to construct d : Lemma 3.2.24 and Lemma 3.2.25.

As for ξ , it is not clear that we have some real extensions of these functions in \mathbb{RCD}_* : we write $\mathbb{RCD}_* + \xi + \text{Decode} + \text{Encode}$ for the algebra where we add some real extension of these functions as a base function.

5.2.4 Constructing the missing functions

We need to construct some substitute of “missing functions” (ξ , Decode and EncodeMul). All of them were defined in Chapter 3 using discrete ODEs. An idea is to use a continuous ODE to simulate the respective discrete ODEs: we hence revisit the construction of the ideal iteration trick of Section 5.2.2, dealing with errors and not exact functions $\theta(z)$ and $r(x)$.

The key is to revisit Lemma 5.2.11 and do a more detailed analysis of possible involved errors in dynamics of the form (5.5). Various authors have studied this dynamic in several articles, including [Bra05, CMC00, GCB05, BGP16a, Goz22]. We use the following statement from [Goz22, Lemma 4.5], [BGGP23, Lemma 5.2], obtained basically by a case analysis of error propagations in Lemma 5.2.11.

Lemma 5.2.25 (Improved error analysis of ODE (5.5), [Goz22], [BGGP23])

Consider a point $b \in \mathbb{R}$, some $\gamma > 0$ some reals $t_0 < t_1$, and a function $\phi : \mathbb{R} \rightarrow \mathbb{R}$ with the property that $\phi(t) \geq 0$ for all $t \geq t_0$ and $\int_{t_0}^{t_1} \phi(t) dt > 0$. Let $\rho, \delta \geq 0$ and let $\bar{b}, E : \mathbb{R} \rightarrow \mathbb{R}$ be functions such that that $|\bar{b}(t) - b| \leq \rho$ and $|E(t)| \leq \delta$ for all $t \geq t_0$. Then the IVP defined by

$$z' = c(\bar{b}(t) - z)^3 \phi(t) + E(t)$$

with the initial condition $z(t_0) = \bar{z}_0$, where $\gamma > 0$ and $c \geq \frac{1}{2\gamma^2 \int_{t_0}^{t_1} \phi(t) dt}$ satisfies

1. $|z(t_1) - b| < \rho + \gamma + \delta(t_1 - t_0)$, independently of the initial condition $\bar{z}_0 \in \mathbb{R}$
2. $\min(\bar{z}_0, b - \rho) - \delta(t_1 - t_0) \leq z(t) \leq \max(\bar{z}_0, b + \rho) + \delta(t_1 - t_0)$ for all $t \in [t_0, t_1]$.

Now, assume \mathbf{G} is almost constant around $\mathbb{N}\delta$ and r is a rounding function around $\mathbb{N}\delta$ for some $\delta > 0$: for $z \in \left[n\delta - \frac{1}{4}\delta, n\delta + \frac{1}{2}\delta\right]$, $r(z) = n$, for any integer $n \in \mathbb{Z}$:

Proposition 5.2.26 (Simulating a discrete ODE by a continuous ODE)

Suppose that, in (5.4), we replace function $\theta(z)$ and function $r(z)$ by some suitable approximations: we take $\theta(x) = \text{ReLU}(x)$, $\theta_{\varepsilon'}(x)$, $r_{\varepsilon'}(z)$ such that $\theta(z) =_{\varepsilon'} \theta_{\varepsilon'}(z)$ and $r(x) =_{\varepsilon'} r_{\varepsilon'}(x)$, and take constant c big enough. Then the solution of the obtained ODE will continuously simulate the discrete ODE (5.3), with the same bounds as in the analysis in Section 5.2.2, i.e. with error at most ε if ε' is taken sufficiently small. To guarantee $\varepsilon = 2^{-n}$, it is sufficient to take $\varepsilon' = 2^{-p(n)}$ and $\theta_{\varepsilon'}(x) = \text{ReLU}_{-s}(2^{p(n)}, x)$ for some polynomial p .

Proof. The key is that the involved errors additively propagate from Lemma 5.2.25. Namely, they are in $\mathcal{O}(\varepsilon')$, but they are corrected from the reasoning in Section 5.2.2: rounding function corrects errors of order ε whenever its argument is at a distance less than $\frac{1}{4}\delta$ of some $n\delta$ exactly as in the reasoning in Section 5.2.2 (where $\delta = 1$, even if now it introduces some error ε' at every step; but the latter is corrected at the next step). Observe that the involved constant c is of order 2^n .

More formally, we claim that for all $n \in \mathbb{N}$,

$$\mathbf{y}_1(n, \mathbf{x}) =_{\varepsilon} \mathbf{y}_2(n, \mathbf{x}) =_{\varepsilon} \mathbf{f}(n, \mathbf{x})$$

and

$$\mathbf{y}_1\left(t + \frac{1}{2}, \mathbf{x}\right) =_{\varepsilon} \mathbf{y}_2(t, \mathbf{x}) =_{\varepsilon} \mathbf{f}(n, \mathbf{x})$$

for all $t \in \left[n, n + \frac{1}{2}\right]$.

For $n = 0$, initially $\mathbf{f}(0, \mathbf{x}) = \mathbf{y}_1(0, \mathbf{x}) = \mathbf{y}_2(0, \mathbf{x}) = \mathbf{g}(\mathbf{x})$. For $t \in \left[n, n + \frac{1}{2}\right]$, we then have $\theta(-\sin(2\pi t)) =_{\varepsilon'} 0$, and hence $\mathbf{y}'_2 =_{\varepsilon'} 0$, so \mathbf{y}_2 is kept close to value $\mathbf{g}(\mathbf{x})$ for $t \in \left[0, \frac{1}{2}\right]$, with an error less than $\frac{1}{2}\varepsilon'$.

Consequently, for $t \in \left[0, \frac{1}{2}\right]$, $r(\mathbf{y}_2)$ is kept close to a constant value $\mathbf{g}(\mathbf{x})$, when an error less than ε' , if we choose $\varepsilon' < \frac{1}{4}\delta$. Meanwhile, $r(t)$ is also at a value close to n with an error lesser than ε' .

Consequently, on this interval, if we write $C(t) = c\theta(\sin(2\pi t))$, then the dynamics of \mathbf{y}_1 is given by a dynamic of the form of Lemma 5.2.25. This lemma states that $\mathbf{y}_1(t, \mathbf{x})$ will approach $\mathbf{G}(\mathbf{g}(\mathbf{x}), 0, \mathbf{x}) = \mathbf{f}(1, \mathbf{x})$ on this interval, with an error of order $\varepsilon' + \varepsilon' + \frac{1}{2}\varepsilon'$.

Here the hypothesis that \mathbf{G} is almost constant around $\mathbb{N}\delta$ means that its value is guaranteed to be at ε' from $\mathbf{G}(\mathbf{g}(\mathbf{x}), 0, \mathbf{x})$ on the interval.

Thus, $\mathbf{y}_1\left(\frac{1}{2}, \mathbf{x}\right) =_{\varepsilon/2} \mathbf{f}(1, \mathbf{x})$, if we choose $\frac{5}{2}\varepsilon' < \frac{\varepsilon}{2}$. At $t = n + \frac{1}{2}$, \mathbf{y}_1 will hence have simulated one step of discrete ODE (5.3), with an error less than $\frac{\varepsilon}{2}$, and \mathbf{y}_2 will be close to $\mathbf{g}(\mathbf{x})$ with an error less than $\varepsilon' < \frac{\varepsilon}{2}$.

Now, for $t \in \left[n + \frac{1}{2}, n + 1\right]$ the roles of \mathbf{y}_1 and \mathbf{y}_2 are exchanged: $\mathbf{y}'_1(t, \mathbf{x}) =_{\varepsilon'} 0$, so \mathbf{y}_1 is kept almost fixed, with a new error less than $\frac{1}{2}\varepsilon'$. In the same time \mathbf{y}_2 approaches $r(\mathbf{y}_1) = \mathbf{f}(1, \mathbf{x})$ by Lemma 5.2.25, with some new error of order less than $\frac{5}{2}\varepsilon' < \frac{\varepsilon}{2}$.

Consequently, we get the property at rank $n + 1$. \square

NB 5.2.27

Somehow, the main idea is that the constructions always replace every function with

another that does not change much locally (i.e. changes in a controlled way). This is the key that provides a robust ODE as in Definition 5.2.3, leading to polynomial space complexity by Theorem 5.2.6.

In other words, whenever we have some discrete ODE as in (5.3) defining some function $\mathbf{f}(t, \mathbf{x})$, we can construct some continuous ODE, using only functions from \mathbb{RCD}_* , such that one of its projection provides a function $\mathbf{f}(z, t, \mathbf{x})$, with the guarantee $\mathbf{f}(2^n, t, \mathbf{x})$ is 2^{-n} close to $\mathbf{f}(n, \mathbf{x})$, whenever t is close (at a distance less than $\frac{1}{4}$) to some integer n .

This works, as we can obtain such a $r_{\varepsilon'}(x)$ from the functions from Corollaries 5.2.16, 5.2.17, 5.2.18, 5.2.19, 5.2.20: consider $r(x, 2^m) = \sigma_2\left(2^m, 2^n, x + \frac{1}{4}\right)$ that works over $\lfloor x \rfloor \in [-2^n + 1, 2^n]$, and observe that this is sufficient to apply the trick for the required functions, from the form of the considered discrete ODE in Chapter 3.

Except that we have a bootstrap problem: ξ was defined using a discrete ODE in Chapter 3 and as the functions from Corollaries 5.2.16, 5.2.17, 5.2.18, 5.2.19, 5.2.20 are constructed above using ξ , we cannot apply this reasoning to get function ξ . But the point is that for the special case of ξ , it is easy to construct a function in \mathbb{RCD} that corresponds to some real extension of ξ , as we have functions such as $\sin(x) = \cos\left(\frac{\pi}{2} - x\right)$ and π .

Lemma 5.2.28

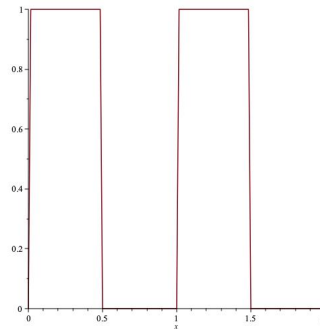
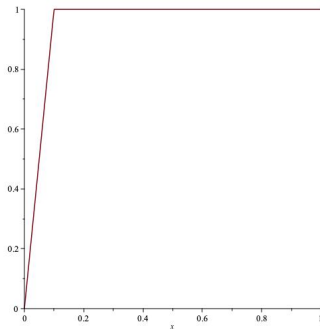
Function ξ has some real extension in \mathbb{RCD}_* .

Proof. If we succeed to obtain $i(2^m, 2^n, x)$ valuing $\lfloor x \rfloor$ whenever $x \in \left[\lfloor x \rfloor, \lfloor x \rfloor + \frac{3}{4}\right]$, we are done, as we have $\xi(2^m, 2^n, x)$ by considering $\xi(2^m, 2^n, x) = x + \frac{7}{8} - i\left(2^m, 2^n, x + \frac{7}{8}\right)$.

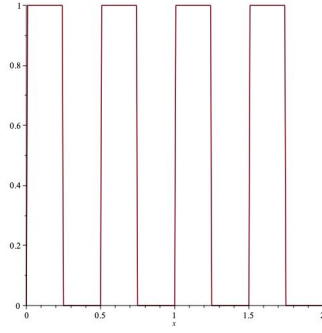
A possible solution is then the following: consider function $R_e(x) := \mathfrak{s}\left(x, 0, \frac{\varepsilon}{2}\right)$, and then $t_e(x) = \left(1 - R_e(\sin(2\pi x))\right)\left(1 - R_e(\sin(4\pi x))\right)$. If we put aside some interval of width $\frac{\varepsilon}{2}$ around $\frac{1}{2}$ and $\frac{7}{8}$ where it takes values in $[0, 1]$, it values 0 on $\left[\lfloor x \rfloor, \lfloor x \rfloor + \frac{7}{8}\right]$, and then 1 on $\left[\lfloor x \rfloor + \frac{7}{8}, \lfloor x \rfloor + 1\right]$. We can then consider $I_e(t) = 8 \int_0^t t_e(x) dx$ (i.e. the solution of ODE $I_e' = 8t_e$), and then $i(t) =_{e,t} I_e(t)$.

It is then sufficient to replace \mathfrak{s} by $\mathcal{C}\text{-}\mathfrak{s}$, in the above expressions, in order to control the error and make it smaller than 2^{-m} . \square

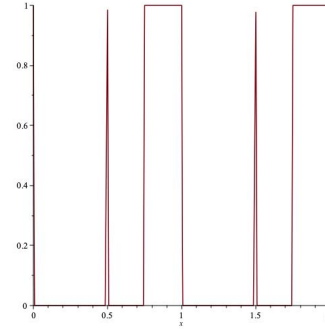
Here is a graphical representation of $R_{\frac{1}{10}}(x)$: Then of $R_{\frac{1}{10}}(\sin(2\pi x))$:



and $R_{\frac{1}{10}}(\sin(4\pi x))$:



We then get $t_{\frac{1}{10}}(x)$:



The integral of $t_{\frac{1}{10}}(\cdot)$ is then close to $\frac{1}{8}\lfloor x \rfloor$ on $\left[\lfloor x \rfloor, \lfloor x \rfloor + \frac{3}{4}\right]$.

Consequently, we can substitute a discrete ODE with a continuous ODE for the required functions *Decode* and *EncodeMul*: just replace ξ in the involved schemas with the above function. Notice that we can also easily get a real extension of the function that maps n to 2^n . Now that we have proved real extensions of the approximations of basic discrete functions, we can show we can encode the execution of a polynomial space Turing machine in \mathbb{RCD} .

5.2.5 Working with all steps of a Turing machines

We can then go from one step of a Turing machine, to arbitrarily many steps. We are following the idea of Chapter 3, but replacing discrete ODEs with continuous ODEs.

Theorem 5.2.29

Consider some Turing machine M that computes some function $f : \Sigma^* \rightarrow \Sigma^*$ in some polynomial space $S(\ell(w))$ on input w . One can construct some function $\tilde{f} : \mathbb{N}^2 \times \mathbb{R} \rightarrow \mathbb{R}$ in \mathbb{RCD} doing the same: if we consider the value of $\tilde{f}(2^m, 2^{S(\ell(w))}, \gamma_{word}(w))$, it is at most 2^{-m} far from $\gamma_{word}(f(w))$.

Proof. We denote by M the Turing machine computing f . The idea is to simulate the discrete ODE encoding the execution of M by a continuous ODE, using Proposition 5.2.26. We can state that there exists a function *Exec* solution of a robust linear discrete ODE (E) that "computes" the execution of M , with C_{init} the initial configuration :

$$(E) : \begin{cases} Exec(2^m, 0, 2^S, C_{init}) = C_{init} \\ \frac{\delta Exec(2^m, t, 2^S, C_{init})}{\delta t} = Next(2^m, 2^S, Exec(2^m, t, 2^S, C_{init})) \\ \quad - Exec(2^m, t, 2^S, C_{init}). \end{cases}$$

For any configuration \bar{C} of M , let write

$$F(\bar{C}) = F(2^m, 2^S, \bar{C}) = Next(2^m, 2^S, \bar{C}) + \bar{C},$$

associated to the righthand side of the above discrete ODE. Denoting by \tilde{C} the errorless encoding of the configuration C , from the constructions of Lemma 5.2.24), it is true that if $|\bar{C} - \tilde{C}| \leq 4^{-(S+2)}$, then $|F(\bar{C}) - F(\tilde{C})| \leq 4^{-(S+2)}$. F does not change much locally on the space of configuration. Denoting by S the space of M , and replacing m by $m + 2S + 4$, we have $|Next(2^m, 2^S, \bar{C}) - \bar{C}| \leq 4^{-(S+2)}$. So at each step of the TM, the error is fixed (and bounded). We can then apply the above arguments (Proposition 5.2.26) to simulate continuously (E), with some controlled error: all involved quantities have encoding polynomials in the size of the inputs. \square

Hence, we prove that *Next* is definable in continuous settings. We now must prove that \mathbb{RCD} characterises **FSPACE** for function over the reals. We start by proving that \mathbb{RCD} characterises **FSPACE** in discrete settings.

5.2.6 Proof of Theorem 5.2.1: the algebra also characterises FSPACE in discrete settings

We prove Theorem 5.2.1, stating that $DP(\mathbb{RCD}) = \mathbf{FSPACE} \cap \mathbb{N}^{\mathbb{N}}$.

Proof. \subseteq : In this direction, we just need to prove that \mathbb{RCD} contains only functions over the reals that are computable in polynomial space. Indeed, then for a function $\mathbf{f}: \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$ sending every integer $\mathbf{n} \in \mathbb{N}^d$ to the vicinity of some integer of $\mathbb{N}^{d'}$, at a distance less than $\frac{1}{4}$, by approximating its value with precision $\frac{1}{4}$ on its input arguments, and taking the closest integer, we will get a function from the integers to the integers, that corresponds to $DP(f)$, and that will be in $\mathbf{FSPACE} \cap \mathbb{N}^{\mathbb{N}}$.

This is indeed the case, since

- all the base functions of \mathbb{RCD} are in **FSPACE**: they are even in **FPTIME** (see [Ko91]);
- $\mathbb{R}^{\mathbb{R}} \cap \mathbf{FSPACE}$ is stable under *composition*;
- stability under *robust continuous ODE* follows from Theorem 5.2.6.

\supseteq : In the other direction, we use an argument similar to Chapter 3: namely, as the function is polynomial space computable, this means that there is a polynomial space computable function $g: \mathbb{N}^{d''+1} \rightarrow \{1, 3\}^*$ so that on $\mathbf{m}, 2^n$, it provides the encoding $\phi(\mathbf{m}, n)$ of some dyadic $\phi(\mathbf{m}, n)$ with $\|\phi(\mathbf{m}, n) - \mathbf{f}(\mathbf{m})\| \leq 2^{-n}$ for all \mathbf{m} . The problem is then to decode, compute and encode the result to produce this dyadic. More precisely, from Theorem 5.2.29, we get \tilde{g} with

$$\left| \tilde{g}(2^e, 2^{p(\max(\mathbf{m}, n))}, Decode(2^e, \mathbf{m}, n)) - \gamma_{word}(g(\mathbf{m}, n)) \right| \leq 2^{-e}$$

for some polynomial p corresponding to the time required to compute g and $e = \max(p(\max(\mathbf{m}, n)), n)$. Then we need to transform the value to the correct dyadic: we mean

$$\tilde{\mathbf{f}}(\mathbf{m}, n) = EncodeMul(2^e, 2^t, \tilde{g}(2^e, 2^t, Decode(2^e, \mathbf{m}, n)), 1),$$

where

$t = p(\max(\mathbf{m}, n))$, $e = \max(p(\max(\mathbf{m}, n)), n)$ provides a solution with $\|\tilde{\mathbf{f}}(\mathbf{m}, 2^n) - \mathbf{f}(\mathbf{m})\| \leq 2^{-n}$. \square

5.2.7 Proof of Theorem 5.2.2: $\overline{\mathbb{RCD}}$ characterises **FPSPACE** for real functions

The aim to prove that: $\overline{\mathbb{RCD}} \cap \mathbb{R}^{\mathbb{R}} = \mathbf{FPSPACE} \cap \mathbb{R}^{\mathbb{R}}$.

Proof. \subseteq : To prove that $\overline{\mathbb{RCD}} \subseteq \mathbb{R}^{\mathbb{R}} \cap \mathbf{FPSPACE}$, we only need to add to the previous arguments that $\mathbb{R}^{\mathbb{R}} \cap \mathbf{FPSPACE}$ is also stable under $ELim$ (from Proposition 3.1.3).

\supseteq : In this direction, we have the same issue as in Chapter 3: the strategy of decoding, working with the Turing machine, and encoding is not guaranteed to work for all inputs. But, we can also solve it by using an adaptative barycenter technique.

We recall the principle here for a function whose domain is \mathbb{R} , but it can be generalised to \mathbb{R}^d . The idea is to construct some function $\lambda : \mathbb{N}^2 \times \mathbb{R} \rightarrow [0, 1]$ definable in \mathbb{RCD}_* as in Corollaries 5.2.16, 5.2.17, 5.2.18, 5.2.19, 5.2.20, but with a continuous ODE : Adapting the proof and using the simulation of ξ in our continuous framework, we can consider $\lambda(2^m, N, x) = \Psi\left(\xi\left(2^{m+1}, N, x - \frac{9}{8}\right)\right)$ where $\Psi(x) = \mathcal{C}\text{-}\mathfrak{s}\left(2^{m+1}, \frac{1}{4}, \frac{1}{2}, x\right)$. In particular, by definition, $\lambda \in \mathbb{RCD}_*$. Thus, by Lemma 5.2.28, if $\lambda(2^m, N, x) =_{2^{-m}} 0$, then:

$$\sigma_2(2^m, N, x) =_{2^{-m}} \lfloor x \rfloor.$$

If $\lambda(2^m, N, x) =_{2^{-m}} 1$, then:

$$\sigma_1(2^m, N, x) =_{2^{-m}} \lfloor x \rfloor$$

and if $\lambda(2^m, N, x) \in (0, 1)$, then:

$$\sigma_1(2^m, N, x) =_{2^{-m}} \lfloor x \rfloor + 1$$

and furthermore:

$$\sigma_2(2^m, N, x) =_{2^{-m}} \lfloor x \rfloor.$$

So,

$$\lambda(\cdot, 2^n, x) \text{Formula}_1(x, u, M, n) + (1 - \lambda(\cdot, 2^n, n)) \text{Formula}_2(x, u, M, n)$$

and we are sure to be close (up to some bounded error) to some 2^{-m} approximation of a function f . \square

5.3 A completeness result on the reachability of dynamical systems

As in Chapter 4, we impose our dynamic functions to be Lipschitz. We are going to use the notion of reduction given in Definition 1.3.65. For $\phi \in \mathbf{Reg}$ an oracle, we define the function $\Phi : \gamma(w) \rightarrow \gamma(\phi(w))$.

We write **PAM** for the algebra of PAMs. We denote by $\mathbf{S} + \Phi$ the algebra where \mathbf{S} is an algebra and Φ is added as a base function.

We define the problem **Discrete Reach Robust Lipschitz**:

Discrete Reach Robust Lipschitz:

Input: A discrete-time polynomially robust dynamical system $\mathcal{H} = (X, g)$, with X a closed rational box, $g : X \rightarrow X \in \text{LDL}^\circ + \Phi$ Lipschitz and $p \in \mathbb{N}$, $\mathbf{x}, \mathbf{y} \in X$

Output: $R_{2^{-p}}^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$

with $R_{2^{-p}}^{\mathcal{H}}(\cdot, \cdot)$ as defined in Subsection 4.2.1. The main difference with the problems considered in [KC12] is that we do not fix the time: there is no fixed number of iterations of g in \mathcal{H} in the input.

We now want a completeness result. We will use the definition of reduction of Definition 1.3.65.

Transition functions for Type-2 machines

The transition function of a Turing machine can be seen as a PAM (by Theorem 2.3.1). Now we can extend Theorem 2.3.1 to deal with real numbers. We must prove we can actually simulate an oracle since we are dealing with an oracle TM.

Proposition 5.3.1

Let $M = (Q, \Sigma, q_{\text{init}}, \Delta_M, Q_{\text{accept}}, Q_{\text{reject}})$ be a machine. We denote by \mathbf{f} the transition function of M with oracle ϕ . We have that \mathbf{f} is in $\mathbf{PAM} + \Phi$.

From Theorem 3.2.3 and Lemma 3.2.21, we can deduce:

Corollary 5.3.2

The transition function of a (Type-2) Turing machine is computable in polynomial time.

Since $g \in \text{LDL}^\circ + \Phi$ in **Discrete Reach Robust Lipschitz**, it is computable in second-order polynomial time.

Hence, with Theorem 4.2.33 and Corollary 5.3.2, we have that **Discrete Reach Robust Lipschitz** $\in \mathbf{PSPACE}^2$.

Discrete Reach Robust Lipschitz is $\mathbf{PSPACE}^2 - \leq_m^2$ -hard

The proofs heavily rely on the fact that we are dealing with Lipschitz functions (see Lemma 2.3.3).

Theorem 5.3.3

Discrete Reach Robust Lipschitz is $\mathbf{PSPACE}^2 - \leq_m^2$ -hard.

First, we state the following lemma:

Lemma 5.3.4

For $w = w_0 w_1 w_2 \dots \in \Sigma^\omega$, for $n \in \mathbb{N}$, $\gamma_{\text{word}}(w_0 w_1 \dots w_n)$ is computable in polynomial time in n .

The proof of this lemma is direct from the definition of γ_{word} .

Proof of Theorem 5.3.3. We are going to reduce from **Basic \mathbf{PSPACE}^2** , that we repeat here:

Basic PSPACE²:

Input: $\langle M, \bar{\mu}, \phi \rangle$, with M a Type-2 Turing machine, $\bar{\mu} \in \mathbf{Reg}$ such that $\bar{\mu}(u) = 0^{\mu(|u|)}$ where $\mu : \mathbb{N} \rightarrow \mathbb{N}$ a non-decreasing polynomial bounding the space of M , $\phi \in \mathbf{Reg}$ and a string u .

Output: Does M with an oracle ϕ accept u ?

Let M, μ, ϕ and u be the inputs of **Basic PSPACE²**. Without loss of generality, we assume that $M^\phi = (Q, \Sigma, \mathbf{f}, q_{init}, Q_{accept}, Q_{reject})$ has a unique accepting state q_{accept} with $0^{\mu(|u|)}$ being the content of the tape. We also assume that, if $\mathbf{f}(C)$, for some configuration C , is not defined, then M^ϕ directly goes to a rejecting state.

We construct the inputs of **Discrete Reach Robust Lipschitz**

Let us now define the functions s and t from Definition 1.3.65.

- The initial configuration is \mathbf{x} and $0^{\mu(|u|)} = \mathbf{y}$.
- $t : \langle M, \mu, \phi \rangle \rightarrow \mathcal{H} = (X, g), p$, where X is the set of all the real numbers encoded by possible configurations of M^ϕ , $g \in X^X$ such that $\mathbf{f}(C) = C'$ if and only if $g(\gamma_{config}(C)) = \gamma_{config}(C')$ and $p = \mu$ is the polynomial bounding the space.
- $s : \phi \in \mathbf{Reg} \rightarrow \Phi$

The function s maps each configuration of M^ϕ to its successor, after one step of M^ϕ , in \mathbb{R} . It is Lipschitz by Lemma 2.3.3 and it is computable in polynomial time by Corollary 5.3.2.

The function t encodes an input string into a representation of a dyadic number (approximating some real number). It is computable in polynomial time by Lemma 5.3.4.

Thus, we have **Discrete Reach Robust Lipschitz** is $\mathbf{PSPACE}^2\text{-}\leq_m^2\text{-hard}$. \square

Corollary 5.3.5

Discrete Reach Robust Lipschitz is $\mathbf{PSPACE}^2\text{-}\leq_m^2\text{-complete}$.

5.4 Chapter Conclusion

In this chapter, we characterised polynomial space using algebraically defined classes of functions, in the spirit of Chapter 3: a finite set of basic functions closed under composition and a schema for defining functions from robust discrete ODEs and robust (continuous) ODEs. We gave an algebraic characterisation of **FSPACE** for functions over the reals, using discrete ODEs.

We also characterised **FSPACE** with continuous ODEs. To do so, we proposed a concept of robust ODEs solvable in polynomial space. As far as we know, this is an original method for solving ODEs optimising space, which was an open problem. It is based on classical constructions such as Savitch's theorem. We extended existing characterisations to a characterisation of functions over the reals and not only over the integers.

Of course, from our statements, adding any **FSPACE**-computable function over the reals among the base functions would not change the class. However, we did not intend to minimise the number of base functions. For example, $\tanh(t)$ is the solution of the ODE

$f' = 1 + f^2$ and $\cos(t)$ can be obtained by the two-dimensional ODE $y_1' = -y_2$, $y_2' = y_1$. Minimising the number of base functions is also left for future work. We believe that even in this setting, proving space complexity corresponds to precision is already significant, independently of this question of a minimal set of base functions.

Using previous algebraic characterisations, we also proved the **PSPACE**²-hardness, for the reduction described in Definition 1.3.65, of the reachability problem for dynamical systems such as in Chapter 4.

Chapter 6

Robustness in Tilings and Subshifts

Vous êtes maximalelement problématiques.

Titouan Carette

NB 6.0.1

This chapter is based on an article co-authored with Nathalie Aubrun and Olivier Bournez. It has been submitted to a conference.

In computer science, the continuum is often seen as a limit of the discrete world. Many other communities view things dually: they see discrete settings as a particular limit of the continuum. For example, the optimal solution of a purely continuous optimal control problem may be a so-called bang-bang solution [SVV64, KK06]. Many approaches, often developed in the continuum, can help with discrete objects. An example where this duality is visible is the context of the verification/control of the hybrid systems, which mix some continuous evolution with discrete steps. This is an example of context where the approach from continuous mathematics, mostly from control theory, meets with a dual approach emanating from computer science, mostly from the verification community [Mal02, Lyg04, AMJ24].

In this chapter, we revisit and extend Chapter 4 and we discuss notions of languages for various objects: the space-time diagram of a Turing machine, the effective subshifts and tilesets (i.e. subshifts of finite type, Definition 1.5.7). For all of them, as in Chapter 4, we prove that decidability holds unless some non-robustness is involved. Stated otherwise, for Turing machines, undecidability holds only for non-robust machines. In particular, it is true for Turing machines running in a space that cannot be bounded by some computable function or for which there is no provable bound on their space. For tilesets, undecidability of the language (e.g. the domino problem) holds only for tilesets (or effective subshifts) involving explicitly a non-robust Turing machine.

In this chapter, we extend the statements of Chapter 4 and we consider more general properties, like invariance properties. Our theory provides a valid and relatively complete proof method (see Section 2.7 and [Win93] for the concept of relative completeness) to decide questions about subshifts, and, in particular, tiling problems like the domino problem. It applies to all known famous examples and explains some observed experimental facts on tilesets, such as the one of [JR21].

Thus, we provide a sound and relatively complete method for proving that a tileset can tile the plane. Our analysis also provides explanations for the similarities between proofs in the literature for various tilesets, as well as of phenomena that have been observed

experimentally in the systematic study of tilesets using computer-assisted methods (e.g. [JR21]).

The relation to (non-)provability and the existence of a proof of non-reachability is original. For example, fundamental discussions stating a clear connection to the non-provability of termination of Turing machines are original. The application to subshift is original. Several of our statements may be new, as we did not find any proof of them. Also, our statements about the soundness and completeness of methods based on finite state abstraction and the resulting fundamental explanation of why all proofs of aperiodicity look so similar are also original. We also prove other notions of robustness related to properties of the transducers associated with tilesets can also be applied.

In Section 6.2, we introduce global properties like invariance kernels or viability kernels. In Section 6.3, we apply this theory to the case of Turing machines. We prove that deciding the language of patterns of the space-time diagram of a given Turing machine is a co-c.e.-complete problem. We prove that for robust Turing machines, this language is decidable. In Section 6.4, we apply the general theory to the case of subshifts and tilings. In Section 6.5, we introduce the Robinson tileset and we prove that it is robust.

6.1 Some representation aspects

As in Chapter 4, we will mainly be interested in the case of dynamical systems over a space $X = \Sigma^\omega$ (the Cantor space, e.g. for the statements related to subshifts, tilings), $X = \mathcal{C}_M = \Sigma^\omega \times Q \times \Sigma^\omega$ (for the statements related to Turing machines) and X a closed rational box (for the statements related to dynamical systems, about the motivation from verification of some of our statements).

In this chapter, we assume the underlying space X to be a metric space with the distance $d(\cdot, \cdot)$. In all these contexts, the dynamics we consider involved by the application context are Lipschitz, the space is compact and we can enumerate a class of basic open sets $(B_i)_{i \in \mathbb{N}}$ generating the topology. In the general case, we can take the balls $B(\alpha(n), 2^{-m})$ for some integers n, m where $\alpha(n)$ is a sequence of elements of X dense in X : for the reals, these can be rational open balls but they could also be dyadic open balls. For the Cantor space, these can be cylinders: subsets of the form $[w] = w\Sigma^\omega$ for $w \in \Sigma^*$. We assume a fixed representation of these basic open sets as words over the finite alphabet Σ .

Let \mathcal{A} denote a finite alphabet and \mathcal{Z} is either \mathbb{Z} or \mathbb{N} , or a finite products of \mathbb{Z} or \mathbb{N} , such as $\mathbb{Z} \times \mathbb{N}$. Given a configuration x , we write $x_{\mathbb{U}}$ for the pattern corresponding to the restriction of x to \mathbb{U} , as in Section 1.5.

6.1.1 About representation of involved sets

We can consider various suitable data structures to represent sets according to the application domain. We take the following minimal assumption about this structure: a closed set K can be represented by its language $L(K) \subset \Sigma^*$, where $L(K)$ corresponds to the representations of all basic open sets intersecting K . Dually, we write $L^-(K)$ for the language corresponding to the representations of all basic open sets, such that its closure does not intersect K . Indeed, closed sets (respectively: open sets) K are characterised by their languages $L(K)$ (respectively: $L^-(K)$).

Proposition 6.1.1

Given closed sets A and B , we have $A = B$ iff $L(A) = L(B)$ iff $L^-(A) = L^-(B)$.

Proof. Assume A and B to be closed. If $A = B$, then $L(A) = L(B)$ and $L^-(A) = L^-(B)$. Conversely, if $A \neq B$, without loss of generality, there must be some point x in A and $x \notin B$.

If $\text{int}(A) \cap B^c = \emptyset$, then A and B differ only by their frontier and, since they are closed, $A = B$.

Thus, we assume that $x \in \text{int}(A) \cap B^c$. Since $\text{int}(A) \cap B^c$ is some open set, it contains some basic open set containing x . It is in $L(A)$ but not in $L(B)$ and hence $L(A) \neq L(B)$. By restricting if necessary this open set to $B(x, d)$ for some small enough d , we can assume that $B(x, d)$ does not intersect B . The latter contains some basic open set. Then the closure of this basic open set does not intersect B , but it intersects A , and hence $L^-(A) \neq L^-(B)$.

□

6.1.2 About allowed inputs

In the coming contexts, we are interested in questions related to a subset of points of X .

NB 6.1.2

We cannot guarantee that some \mathbf{x} (e.g. irrational) leads to “hard-to-decide” questions when robustness holds, but this is not a problem if the inputs we consider are never of this type (e.g. rational \mathbf{x}).

In particular, to use classical computation theory, inputs must necessarily belong to some countable set, e.g. finite words over some finite alphabet or rational numbers. The space X , in all the examples, is not countable, e.g. X is a product of intervals in \mathbb{R}^d .

We write \mathcal{Inputs} for the class of allowed/possible inputs and $S \in \mathcal{Subsets}$ for a class of sets S such that $x \in L(S)$ is decidable for $x \in \mathcal{Inputs}$. For example, when considering dynamical systems, \mathcal{Inputs} could be $\mathcal{Inputs} = \mathbb{Q}^d$, while $X = [0, 1]^d$ is a compact domain of real numbers. For Turing machines, while they evolve on $X = \mathcal{C}_M$, allowing infinite tapes, we often need to reason on $\mathcal{Inputs} = \Sigma^* B^\omega \times Q \times \Sigma^* B^\omega$, corresponding to configurations associated to finite words.

6.2 Viability and Invariance Kernels

We present methods to discuss invariance properties, point-to-set or set-to-set reachability and non-reachability for (possibly non-deterministic) dynamical systems. In the rest of the chapter, we will consider non-deterministic dynamical systems. \mathcal{H} (with no inputs) is now given by a domain X and some set-valued function $F : X \rightarrow \mathcal{P}(X)$. A called trajectory in this framework is some sequence such that $x_{t+1} \in F(x_t)$ for all t . As before, we say x^* (or a set X^*) is reachable from x if there is a run with $x_0 = x$ and $x_t = x^*$ (respectively $x_t \in X^*$) for some t .

6.2.1 Some vocabulary from verification, control and dynamical systems theory

We review some basic concepts from verification, control and dynamical systems theory. The following are discussed, for example, in [Aub91, Lyg04, SP94] for even more general classes of dynamical systems such as continuous-time systems or hybrid systems, and with possibly some inputs. Some of their statements are generalised here, in particular for \mathcal{L} -actions.

Definition 6.2.1 (*Viable trajectory*)

An infinite trajectory of a dynamical system \mathcal{H} is called viable in a set $K \subseteq X$ if $x(t) \in K$ for all t .

Viability kernel

We study the subset of initial points in K from which there exists at least one viable solution. A set $K \subseteq X$ is called viable under a (possibly non-deterministic) dynamical system \mathcal{H} if, for all $\mathbf{x}_0 \in K$, there exists an infinite trajectory starting at \mathbf{x}_0 that is viable in K .

The viability kernel, denoted by $\text{Viab}^{\mathcal{H}}(K)$, or $\text{Viab}^F(K)$ for F a set-valued map, of a set $K \subseteq X$ under a dynamical system \mathcal{H} is the set of states $\mathbf{x}_0 \in X$ for which there exists an infinite trajectory viable in K starting from \mathbf{x}_0 . The viability kernel is also the largest **closed** viability domain contained in K . When $F : X \rightarrow \mathcal{P}(X)$ is a set-valued map, a subset $D \subset X$ is a viability domain of F if $\forall x \in D, F(x) \cap D \neq \emptyset$.

Any infinite trajectory viable in K starting from any initial point $x_0 \in K \setminus \text{Viab}^{\mathcal{H}}(K)$ never meets the viability kernel $\text{Viab}^{\mathcal{H}}(K)$ while it remains in K . Moreover, any infinite trajectory which does not start from $\text{Viab}^{\mathcal{H}}(K)$ must leave K in a finite number of steps.

Proposition 6.2.2 ([SP94])

The viability kernel can be obtained by considering the sequence of subsets $K^0 = K, K^1, \dots, K^n, \dots$ with $K^{n+1} := \{x \in K^n \mid F(x) \cap K^n \neq \emptyset\}$. Write $K_\omega := \bigcap_{n \in \mathbb{N}} K^n$. Assume F is an upper-semicontinuous (Definition 1.1.13) set-valued map with closed values and K is a compact subset of $\text{Dom}(F)$. Then $K_\omega = \text{Viab}^{\mathcal{H}}(K)$.

The viability kernel is insensitive to arbitrary small perturbations:

Definition 6.2.3 ([SP94])

Let $F : X \rightarrow \mathcal{P}(X)$ be a set-valued map and $\varepsilon > 0$. An extension of F with a ball of radius ε is the set-valued map $F^\varepsilon : X \rightarrow \mathcal{P}(X)$ defined by $F^\varepsilon(x) := B(F(x), \varepsilon)$.

Proposition 6.2.4 ([SP94])

Consider the sequence of subsets $(K^{\varepsilon,n})_{n \in \mathbb{N}}$ defined as follows: $K^{\varepsilon,0} = K$ and $K^{\varepsilon,n+1} := \{x \in K^{\varepsilon,n} \mid F^{\varepsilon}(x) \cap K^{\varepsilon,n} \neq \emptyset\}$, $K^{\varepsilon,\infty} := \bigcap_{n=0}^{+\infty} K^{\varepsilon,n}$. If F is an upper-semicontinuous set-valued map, $F^{\varepsilon} : X \rightarrow \mathcal{P}(X)$ is also upper-semicontinuous and from Proposition 6.2.2: $\forall \varepsilon > 0$, $K^{\varepsilon,\infty} = \text{Viab}^{F^{\varepsilon}}(K)$.

When ε decreases to 0, the viability kernel of K under F^{ε} converges to the viability kernel of K under F : Let F be upper-semicontinuous and K a compact subset of X . The following property holds: $\text{Viab}^{\mathcal{H}}(K) = \bigcap_{\varepsilon > 0} \text{Viab}^{F^{\varepsilon}}(K)$.

Invariance kernel

Our purpose is now to study the subset of initial points in K from which all solutions are viable. A set K is called invariant under the dynamical system \mathcal{H} , if for all $\mathbf{x}_0 \in K$ all trajectories starting from \mathbf{x}_0 are viable in K . The invariance kernel, $\text{Inv}^{\mathcal{H}}(K)$ of $K \subseteq X$ under \mathcal{H} is the set of initial states $\mathbf{x}_0 \in X$ for which all runs are viable in K . The invariance kernel is also the largest **closed** invariance domain contained in K : A subset $D \subset X$ is a viability invariance of F if $\forall x \in D, F(x) \subset D$.

Proposition 6.2.5 (adapted from [Lyg04] and [Aub91])

Invariance kernel can be obtained by considering the sequence of subsets $(K^n)_{n \in \mathbb{N}}$ with $K^0 = K$ and $K^{n+1} := \{x \in K^n \mid F(x) \subset K^n \text{ or } F(x) = \emptyset\}$. Write $K_{\omega} := \bigcap_{n \in \mathbb{N}} K^n$. Assume F is a lower-semicontinuous (Definition 1.1.14) set-valued map and K is closed. Then $K_{\omega} = \text{Inv}^{\mathcal{H}}(K)$.

Proof. We reason by double inclusion.

- (\subseteq) : Consider K_{ω} . It contains any closed subset of K invariant under F : every set $L \subseteq K$ which is invariant must be contained in $\text{Inv}^{\mathcal{H}}(K)$, since all runs starting in L stay in L and therefore in K .
- (\supseteq) : K_{ω} is closed: $K^0 = K$ is closed. If K^i is closed, since F is lower-semicontinuous, from the definition of K^{i+1} is closed. By induction, the K^i form a sequence of nested closed sets, and therefore K_{ω} is closed (possibly the empty set).

Consider a point $x_0 \in \text{Inv}^{\mathcal{H}}(K)$ and show that $x_0 \in K_{\omega}$. Assume, for the sake of contradiction, that $x_0 \notin K_{\omega}$. Then there exists $N \geq 0$ such that $x_0 \notin K^N$. If $N = 0$, then $x_0 \notin K^0 = K$, therefore there exists a run starting at x_0 that is not viable in K . This contradicts the assumption that $x_0 \in \text{Inv}^{\mathcal{H}}(K)$. If $N > 0$, we show that there exists a run starting at x_0 that after at most one discrete transition finds itself outside K^{N-1} . The claim then follows by induction on N . Indeed, since $x_0 \notin K^N$, we must have $F(x_0) \not\subset K^{N-1}$ and outside of K^{N-1} . But then there exists a run starting at x_0 that transitions outside K^{N-1} and we reach a contradiction as $x_0 \notin K^{N-1}$.

Now, K_{ω} is equal to $\text{Inv}^{\mathcal{H}}(K)$ as the latest is the largest closed invariance domain contained in K .



We see that a true difficulty is that, unlike the viability kernel, the invariance kernel is possibly sensitive to arbitrary small perturbations.

Indeed, we can consider a similar framework as above, and consider $F^\varepsilon(x)$, and then the sequence of subsets $(K^{\varepsilon,1})_{n \in \mathbb{N}}$ defined as:

$$K^{\varepsilon,0} = K \text{ and } K^{\varepsilon,n+1} := \{x \in K^{\varepsilon,n} \mid F^\varepsilon(x) \subset K^{\varepsilon,n} \text{ or } F^\varepsilon(x) = \emptyset\}, K^{\varepsilon,\infty} := \bigcap_{n=0}^{+\infty} K^{\varepsilon,n}.$$

If F is a lower-semicontinuous set-valued map, $F^\varepsilon : X \rightarrow \mathcal{P}(X)$ is also a lower-semicontinuous, and from Proposition 6.2.5, assuming K closed, we have $\forall \varepsilon > 0$, $K^{\varepsilon,\infty} = \text{Inv}^{F^\varepsilon}(K)$, corresponding to the invariance kernel of F^ε .

We have $\text{Inv}^{\mathcal{H}}(K) \subset \bigcap_{\varepsilon > 0} \text{Inv}^{F^\varepsilon}(K)$. Unfortunately, this is not true in the general case that when ε decreases to 0, the invariance kernel of K under F^ε converges to the invariance kernel of K under F : we might have $\text{Inv}^{\mathcal{H}}(K) \subsetneq \bigcap_{\varepsilon > 0} \text{Inv}^{F^\varepsilon}(K)$. See Example 4.1.13 for Turing machines.

6.2.2 Robustness of dynamical systems

Inspired from, and extending [AB01] and the statements of Chapter 4, it is then very natural to define the following.

Definition 6.2.6 (Language-robust dynamical system)

A dynamical system with a set-valued map is said to be language-robust over K , when $\text{Inv}^{\mathcal{H}}(K) = \bigcap_{\varepsilon > 0} \text{Inv}^{F^\varepsilon}(K)$.

In other words, a system is language robust when there is indeed convergence in the above discussion. As closed sets are characterised by their languages, this can be formulated equivalently:

Proposition 6.2.7

The following are equivalent:

- A dynamical system is language-robust over a closed set K .
- $L(\text{Inv}^{\mathcal{H}}(K)) = \bigcap_{r > 0} L(\text{Inv}^{F^r}(K))$
- $L^-(\text{Inv}^{\mathcal{H}}(K)) = \bigcap_{r > 0} L^-(\text{Inv}^{F^r}(K))$

The proof is direct from the definitions and the previous propositions.

Corollary 6.2.8 (Alternative view)

Robustness holds on some closed sets K iff for all basic open set B_i , B_i is not intersecting $\text{Inv}^{\mathcal{H}}(K)$ iff B_i is not intersecting $\text{Inv}^{F^r}(K)$ for some $r = r(B_i)$.

As in Chapter 4, we will prove that many problems are solvable/decidable for robust systems.

We think it helps to read all these statements in a dual way: **decidability holds unless some non-robust systems are considered**. If one thinks about it, non-robustness

means that the answer to the question of reachability is somehow badly formulated, as it requires talking about arbitrary small perturbations. We will also see that non-robustness is always involving some frontier points, hence somehow limit points. This comes from the Proposition above. We think it helps to understand Example 4.1.13.

While the previous theory is very nice from a mathematical point of view and is mostly about closed sets, if we want to talk about computability and complexity issues, it is more natural to formulate statements in terms of open sets (their complements, or their interior): the reason is that computability is more naturally about open sets than closed sets, as a c.e. set for example is a recursively open set: examine the formalisation of all concepts in [BHW08, Wei00]. This leads to getting closer to considerations done in Chapter 4, which originated from computability issues. This also leads first to relate invariance concepts to reachability issues:

Definition 6.2.9 (Invariance envelope)

The invariance envelope $\text{Env}^F(K)$ of $K \subset X$ is the smallest closed subset invariant under F containing K .

Since the intersection of closed subsets invariant under F is still a closed subset invariant under F , the invariance envelope of a closed subset does exist. We write $R^{\mathcal{H}}(S)$ for all the y reachable from some $x \in S$, and $R^{\mathcal{H}}(x)$ for $R^{\mathcal{H}}(\{x\})$. A map $F : X \rightarrow \mathcal{P}(X)$ is said to be *Lipschitz* if and only if there exists a constant $\lambda > 0$ such that for all $x, x' \in X$, $F(x) \subseteq B(F(x'), \lambda \|x - x'\|)$.

Proposition 6.2.10 (Relating reachability and Invariance Envelope [Aub91])

Assume that $F : X \rightarrow \mathcal{P}(X)$ is Lipschitz with nonempty closed values. Then $\text{Env}^F(K) = \overline{R^{\mathcal{H}}(K)}$.

Proof. The subset $\overline{R^{\mathcal{H}}(K)}$ is contained in any closed invariant subset S containing K and, in particular, in the invariance envelope of K . Conversely, we just need to prove that $\overline{R^{\mathcal{H}}(K)}$ is invariant. If not, there would exist $x_0 \in \overline{R^{\mathcal{H}}(K)}$ and some trajectory $x(t)$ starting from x_0 and T such that $x(T) \notin \overline{R^{\mathcal{H}}(K)}$. By the hypotheses (F is Lipschitz), there exists $\delta > 0$ such that for every $y_0 \in \overline{B(x_0, \delta)}$, one can find a trajectory $y(\cdot)$ starting from y_0 such that $y(T) \in B(x(T), \epsilon) \subset X \setminus \overline{R^{\mathcal{H}}(K)}$. Since x_0 belongs to the closure of $R^{\mathcal{H}}(K)$, one can find some initial state $y_0 \in R^{\mathcal{H}}(K)$ so that there exists $z_0 \in K$, that reaches y_0 at time T_0 . Then the concatenation of that trajectory up to time T_0 , and then the one starting from y_0 is a solution starting from K that remains in $R^{\mathcal{H}}(K)$: we reach a contradiction. \square

But, why such issues are interesting if they cannot happen on a computer? If no input to the problem in *Inputs* can ever lead to a problematic point or issue, why studying this? If something bad can happen with irrational numbers and not rational numbers, should we consider dynamical systems over rationals? Similarly for infinite TM configurations vs finite ones? We hence now relativise the above concept to the considered *Inputs*.

Definition 6.2.11 (Language-robust dynamical system with respect to Inputs)

A dynamical system is said to be language-robust over K with respect to *Inputs*, when $\text{Inv}^{\mathcal{H}}(K) \cap \overline{R^{\mathcal{H}}(\text{Inputs})} = \bigcap_{\epsilon > 0} \text{Inv}^{F^\epsilon}(K) \cap R^{\mathcal{H}}(\text{Inputs})$.

This corresponds to the previous definition, but considering $K \cap \overline{R^{\mathcal{H}}(\mathcal{Inputs})}$, instead of K . Consequently, we continue our discussion with general K . A key is that $\text{INV}_\omega(K) = \bigcap_{\varepsilon > 0} \text{Inv}^{F^\varepsilon}(K)$ can always be determined and obtained by reasoning about a discretisation F .

We say that a (possibly. non-deterministic) dynamical system (X, F) is \mathcal{Inputs} -stable when $F(\mathcal{Inputs}) \subset \mathcal{Inputs}$: all its runs live in \mathcal{Inputs} . We say it is \mathcal{Inputs} -computable when $\mathbf{y} \in F(\mathbf{x})$, for $\mathbf{x}, \mathbf{y} \in \mathcal{Inputs}$.

This is a generalisation of the concept of \mathbb{Q} -systems considered in Chapter 4. We obtain a generalisation of Theorem 4.2.6:

Theorem 6.2.12 (*INV_ω can always be obtained by discretising over a grid*)

Assume X is a compact, and that F is Lipschitz. Then, $\text{INV}_\omega(K) = \bigcap_{\varepsilon > 0} \text{Inv}^{F^\varepsilon}(K)$ is independent of the metric.
In particular, considering the metric to be the sup-norm, F^ε corresponds to some box/griddy (over-estimation) of the discretisation of F . Any such F^ε is a dynamical system over a finite space: it can hence be seen as a finite graph G_ε . Assume F is \mathcal{Inputs} -stable and computable and X is a product of closed intervals with \mathcal{Inputs} -bounds. Then $L(\text{INV}_\omega)$ is co-c.e..

Before proving this theorem, we need two auxiliary lemmas, adapted from Chapter 4 in this context:

Lemma 6.2.13 (*Adaption of Lemma 4.2.8*)

Assume $R_{+\varepsilon}^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ for $\varepsilon = 2^{-n}$. Then, $\mathbf{x} = \mathbf{y}$, or, for all i, j , such that $\mathbf{x} \in \mathcal{V}_i$ and $\mathbf{y} \in \mathcal{V}_j$, $\mathcal{V}_i \xrightarrow{+}_\varepsilon \mathcal{V}_j$: the graph for $\delta = \varepsilon$ always has more trajectories/behaviours than $R_{+\varepsilon}^{\mathcal{H}}$.

The proof of Lemma 6.2.13 is very similar to the proof of Lemma 4.2.8.

Lemma 6.2.14 (*Adaptation of Lemma 4.2.9*)

If $\mathbf{x} = \mathbf{y}$, then $R_{+\varepsilon}^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$. Moreover, for any $\varepsilon = 2^{-n}$, there is some $\delta = 2^{-m}$ so that there exists i, j such that $\mathcal{V}_i \xrightarrow{+}_\delta \mathcal{V}_j$. Thus, $R_{+\varepsilon}^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ whenever $\mathbf{x} \in \mathcal{V}_i$, $\mathbf{y} \in \mathcal{V}_j$: This says that $\neg R_{+\varepsilon}^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ implies $\neg(\mathcal{V}_i \xrightarrow{+}_\delta \mathcal{V}_j)$ when $\mathbf{x} \in \mathcal{V}_i$, $\mathbf{y} \in \mathcal{V}_j$, for the corresponding δ .

Proof of Lemma 6.2.14. If $\mathbf{x} = \mathbf{y}$, the first claim is clear. Let L be the Lipschitz constant.

Consider $\delta = 2^{-m}$ with $\delta < \varepsilon / (2L + 1)$: It is sufficient to prove that when $\mathcal{V}_i \xrightarrow{+}_\delta \mathcal{V}_j$ with $\mathbf{x} \in \mathcal{V}_i$, $\mathbf{y} \in \mathcal{V}_j$, we have $R_{+\varepsilon}^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$: indeed, then by induction, if we have

$$\mathcal{V}_{i=i_0} \rightarrow_\delta \mathcal{V}_{i_1} \dots \rightarrow_\delta \mathcal{V}_{i_t=j},$$

we can then reason on points $\mathbf{y}_1 \in \mathcal{V}_{i_1}, \dots, \mathbf{y}_{t-1} \in \mathcal{V}_{i_{t-1}}$, and obtain inductively that $R_{+\varepsilon}^{\mathcal{H}}(\mathbf{x}, \mathbf{y}_1), R_{+\varepsilon}^{\mathcal{H}}(\mathbf{y}_1, \mathbf{y}_2), \dots, R_{+\varepsilon}^{\mathcal{H}}(\mathbf{y}_{t-1}, \mathbf{y})$, and hence that $R_{+\varepsilon}^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$.

So, assume $\mathcal{V}_i \xrightarrow{+}_\delta \mathcal{V}_j$ with $\mathbf{x} \in \mathcal{V}_i$, $\mathbf{y} \in \mathcal{V}_j$. As $\mathcal{V}_i \xrightarrow{+}_\delta \mathcal{V}_j$ there is some $\bar{\mathbf{y}} \in \mathcal{V}_j$ with $d(\mathbf{F}^\varepsilon(\mathbf{x}_i), \bar{\mathbf{y}}) < (L + 1)\delta$. Then $d(\mathbf{F}^\varepsilon(\mathbf{x}), \bar{\mathbf{y}}) \leq d(\mathbf{F}^\varepsilon(\mathbf{x}), \mathbf{F}^\varepsilon(\mathbf{x}_i)) + d(\mathbf{F}^\varepsilon(\mathbf{x}_i), \bar{\mathbf{y}}) < L\delta + (L + 1)\delta = (2L + 1)\delta < \varepsilon$ and hence $R_{+\varepsilon}^{\mathcal{H}}(\mathbf{x}, \bar{\mathbf{y}})$: we get the promised statement. \square

Proof of Theorem 6.2.12. It is sufficient to prove the statement starting from the second sentence. As \mathbf{F} is Lipschitz *Inputs*-computable and X is compact, we know that: there exists some $\lambda > 0$ so that for all $\mathbf{x}, \mathbf{x}' \in X$, $\mathbf{F}(\mathbf{x}) \subseteq \mathbf{F}(\mathbf{x}') + \lambda \|\mathbf{x} - \mathbf{x}'\|B(0, 1)$. For every $\delta = 2^{-m}$, $m \in \mathbb{N}$, we associate some graph $G_m = (V_\delta, \rightarrow_\delta)$: its vertices, denoted by $(\mathcal{V}_i)_i$, correspond to some finite discretisation and covering of the compact K by rational open balls $\mathcal{V}_i = B(\mathbf{x}_i, \delta_i)$ of radius $\delta_i < \delta$.

There is an edge from \mathcal{V}_i to \mathcal{V}_j in this graph, that is to say $\mathcal{V}_i \rightarrow_\delta \mathcal{V}_j$, iff $\mathbf{F}^{(\lambda+1)\delta}(\mathbf{x}_i) \cap \mathcal{V}_j \neq \emptyset$. With the hypotheses, when F is *Inputs*-stable, such a graph can be effectively obtained from m , considering a suitable discretisation of X .

From Lemma 6.2.13 and Lemma 6.2.14, $\neg R_{+\omega}^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ holds iff for $\mathbf{x} \neq \mathbf{y}$, there exists $\delta = 2^{-m}$, for all i, j , $\mathbf{x} \in \mathcal{V}_i$, $\mathbf{y} \in \mathcal{V}_j$, hence $\neg(\mathcal{V}_i \xrightarrow{*}_\delta \mathcal{V}_j)$. This holds iff for all integer m , $\text{NOPATH}(G_m, \mathcal{V}_i, \mathcal{V}_j)$ for all $\mathcal{V}_i, \mathcal{V}_j$ with $\mathbf{x} \in \mathcal{V}_i$, $\mathbf{y} \in \mathcal{V}_j$, where $\text{NOPATH}(G, u, v)$ is the following decision problem: Given a directed graph $G = (V, \rightarrow)$ and some vertices $u, v \in V$, determine whether there is no path between u and v in G . The latter property is c.e., as it is a union of decidable sets (uniform in m), as $\text{NOPATH}(G_m, \mathcal{V}_i, \mathcal{V}_j)$ is a decidable property over finite graph G_m . \square

An invariance set can be non-minimal: it is minimal if it does not contain a non-empty invariance set. When this holds, it makes sense to reason on a c.e. union of basic open sets on which we have local minimality, when it exists. This will preserve c.enumerability. Consequently, w.l.o.g, we can assume we restrict to some open set U , on which $\text{INV}_\omega \cap U$ is minimal. We continue the discussion by considering this is the case, replacing possibly INV_ω by $\text{INV}_\omega \cap U$ in the coming discussions.

A specific case: when an isomorphism holds

From above discussions, consequently, INV_ω can be described by considering any subsequence $(\varepsilon_n)_{n \in \mathbb{N}}$ going to 0. Then, we have $\text{INV}_\omega = \bigcap_{n \in \mathbb{N}} \text{Inv}^{F^{\varepsilon_n}}(K)$. If we write G_n for the graph G_{ε_n} , as defined in the proof of Theorem 6.2.12, we get that $\text{INV}_\omega = \bigcap_{n \in \mathbb{N}} G_n$. This is the general case: G_n can depend on n in some complicated manner.

An interesting and simple case is when all the involved graphs for various n are isomorphic for the following concept:

Definition 6.2.15 (*Label-preserving graph isomorphism*)

Let S, T be two \mathbb{Z}^d -finite abstractions (Definition 4.1.15) over X . A mapping $\phi : S \rightarrow T$ is a label-preserving graph isomorphism if ϕ is a bijection and

- for all $u, v \in X$, there is an edge between u and v in S iff there is one between $\phi(u)$ and $\phi(v)$ in T
- the generator of the edge between u and v is the generator of the edge between $\phi(u)$ and $\phi(v)$.

Definition 6.2.16

We write in this case this phenomenon as $G_{n+1} \equiv G_n$.

This case actually holds in many examples that we will discuss later. The existence of such a case makes the analysis simple/concise for various dynamical systems, in particular subshifts.

6.2.3 Space-time of dynamical systems and related problems

Space-time diagram of a dynamical system

Consider a dynamical system \mathcal{H} over (semi)-group/time-space \mathcal{Z} . The space-time diagram of an infinite trajectory $(x_t)_{t \in \mathcal{Z}}$ of \mathcal{H} is this sequence as an element of $X^{\mathcal{Z}}$. We write $ST(\mathcal{H})$ for the set of the space-time diagrams of \mathcal{H} : it is a subset of $X^{\mathcal{Z}}$. We write $ST(\mathcal{H}, x)$ for the set of the space-time diagrams of trajectories in \mathcal{H} starting from x . From definitions, the allowed inputs of $ST(\mathcal{H})$ are $\text{Inv}^{\mathcal{H}}(X)$, and they are non-empty iff there is some infinite trajectory for the dynamical system in X .

We write $\mathcal{L}(ST(\mathcal{H}))$ for its set of patterns: $\mathcal{L}(ST(\mathcal{H}, x)) = \{p \sqsubseteq ST(\mathcal{H}, x), p \text{ a pattern}\}$ and $\mathcal{L}(ST(\mathcal{H})) = \{p \sqsubseteq ST(\mathcal{H}, x), p \text{ a pattern}, x \in X\}$.

Definition 6.2.17 (Pattern-robustness)

A dynamical system is pattern-robust over a closed set K if $\mathcal{L}(\text{Inv}^{\mathcal{H}}) = \bigcap_{r>0} \mathcal{L}(\text{Inv}^{F^r}(K))$

Proposition 6.2.18 (Pattern-robustness and language-robustness are the same)

A dynamical system is pattern-robust over a closed set K iff it satisfies all the (equivalent) conditions of Proposition 6.2.7.

Dynamical systems and subshifts

The space-time diagram $ST(\mathcal{H})$ over a dynamical system over X is always a generalised subshift (with respect to shifts over \mathcal{Z}).

Put this in perspective with the discussions in NB 6.2.19: any subshift can be seen as the invariance set of some dynamical systems and conversely, any dynamical system over a compact can be abstracted as a subshift.

NB 6.2.19 (Subshifts vs $\mathbb{N}^d/\mathbb{Z}^d$ -dynamical systems)

Let \mathcal{A} be a finite set. Then (X, σ) with $X = \mathcal{A}^{\mathbb{Z}^d}$ and σ the shift is a particular \mathbb{Z}^d -dynamical system. Conversely, consider a dynamical system, or \mathbb{Z}^d -dynamical system, and some (abstraction) function $a : X \rightarrow \mathcal{A}$. Then, we can consider $\Phi'(t, U) := a(\Phi(t, a^{-1}(U)))$ that is a non-deterministic \mathbb{N}^d -dynamical system over $X' = a(X)$.

Finite dynamical systems, their graph representation, Wang tiles

As discussed in Subsection 2.3.2, we can always see a dynamical system (X, F) as a labelled graph, with possibly not finitely many vertices.

Vertices correspond to elements of X . An edge between \mathbf{x} and \mathbf{x}' labelled by g encodes the fact that $\mathbf{x}' \in f^g(\mathbf{x})$ for some generator g .

Definition 6.2.20 (Dual representation of a dynamical system)

We can dually represent the previous graph by its line graphs. For each generator g , we can consider the graph whose vertices are edges of the previous graph (hence encoding the fact that $\mathbf{x}' \in f^g(\mathbf{x})$). Two vertices are then adjacent if and only if their corresponding edge for the previous graph share a common endpoint (hence they involve the same \mathbf{x} or \mathbf{x}').

When X is finite, determining whether some point is reachable from another, or computing the invariance kernel, etc. can be solved in polynomial time and space, see discussions in Section 2.3. However, in classical complexity, we assume the graph is given as an input.

Furthermore, when X (or equivalently the graph) is finite, such a graph can always be seen as colours of finitely many **Wang tiles** (over \mathcal{X}).

6.2.4 Relating robustness to reachability

From Proposition 6.2.10, reachability questions and invariance kernel/envelope computations are related. Also, considering some dynamical system \mathcal{H} , write $\text{Halt} = \{x | F(x) = \emptyset\}$: invariance kernel computation and Reach Set computation problems from \mathbf{x} reduce one to the other, for non-halting instances, that is to say for $\mathbf{x} \in \text{Inv}^{\mathcal{H}}(X)$. A dynamical system is non language-robust over K , iff there is some $\mathbf{x} \notin \text{Inv}^{\mathcal{H}}(K)$, but $\mathbf{x} \in \text{Inv}^{F^\varepsilon}(K)$ for all ε .

We tackle more general questions than in Chapter 4, such as, given a dynamical system \mathcal{H} , to compute the invariance kernel of \mathcal{H} . Consequently, we need generalisations of Chapter 4 to non-determinism, control or \mathbb{Z}^d actions.

NB 6.2.21

Only over-approximations have been considered in [AB01] and in Chapter 4. In this chapter, we generalise this idea to more general systems.

For any $\varepsilon > 0$, we consider here the ε -perturbed system $\mathcal{H}_{+\varepsilon}$. The flow/action of the obtained non-deterministic system is defined by $\mathbf{y} \in \Phi_{+\varepsilon}(g, \mathbf{x})$ iff $d(\mathbf{y}, \Phi(g, \mathbf{x})) < \varepsilon$ for all the generators $g \in \mathcal{G}$ of the subgroup. As in Chapter 4, all trajectories of a non-perturbed system \mathcal{H} are also trajectories of the ε -perturbed system $\mathcal{H}_{+\varepsilon}$. If $\varepsilon_1 < \varepsilon_2$, then any trajectory of the ε_1 -perturbed system is also a trajectory of the ε_2 -perturbed system. We denote reachability in the corresponding system $\mathcal{H}_{+\varepsilon}$ by $R_{+\varepsilon}^{\mathcal{H}}(\cdot, \cdot)$.

Define $R_{\omega}^{\mathcal{H}^-}(\mathbf{x}, \mathbf{y})$ iff $\exists \varepsilon > 0 R_{-\varepsilon}^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$: this relation encodes reachability with arbitrarily small perturbing noise. From definitions, we have generalisations of Lemma 4.2.5 to non-deterministic systems:

Lemma 6.2.22

For any $0 < \varepsilon_2 < \varepsilon_1$ and any \mathbf{x} and \mathbf{y} the following implications hold: $R^{\mathcal{H}}(\mathbf{x}, \mathbf{y}) \Leftarrow R_{\omega}^{\mathcal{H}^+}(\mathbf{x}, \mathbf{y}) \Leftarrow R_{-\varepsilon_2}^{\mathcal{H}}(\mathbf{x}, \mathbf{y}) \Leftarrow R_{-\varepsilon_1}^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$.

Define $R_{\omega}^{\mathcal{H}^+}(\mathbf{x}, \mathbf{y})$ iff $\forall \varepsilon > 0 R_{+\varepsilon}^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$: this relation encodes reachability with arbitrarily small perturbing noise.

Lemma 6.2.23

For any $0 < \varepsilon_2 < \varepsilon_1$ and any \mathbf{x} and \mathbf{y} the following implications hold: $R^{\mathcal{H}}(\mathbf{x}, \mathbf{y}) \Rightarrow R_{\omega}^{\mathcal{H}^+}(\mathbf{x}, \mathbf{y}) \Rightarrow R_{+\varepsilon_2}^{\mathcal{H}}(\mathbf{x}, \mathbf{y}) \Rightarrow R_{+\varepsilon_1}^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$.

Theorem 6.2.24 (Generalisation of Theorem 4.2.6)

Consider a Lipschitz *Inputs*-computable system, with a compact domain X . Then the relation $R_{\omega}^{\mathcal{H}^+}(\mathbf{x}, \mathbf{y}) \subseteq \text{Inputs} \times \text{Inputs}$ is in the class Π_1 .

Proof. As \mathbf{f} is locally Lipschitz and X is compact, we know that \mathbf{f} is Lipschitz: there exists some $L > 0$ so that $d(\mathbf{f}(\mathbf{x}), \mathbf{f}(\mathbf{y})) \leq L \cdot d(\mathbf{x}, \mathbf{y})$. For every $\delta = 2^{-m}$, $m \in \mathbb{N}$, we associate some graph $G_m = (V_{\delta}, \rightarrow_{\delta})$: its vertices, denoted by $(\mathcal{V}_i)_i$, correspond to some finite discretisation and covering of the compact X by rational open balls $\mathcal{V}_i = B(\mathbf{x}_i, \delta_i)$ of radius $\delta_i < \delta$. There is an edge from \mathcal{V}_i to \mathcal{V}_j in this graph, that is to say $\mathcal{V}_i \rightarrow_{\delta} \mathcal{V}_j$, iff $B(\mathbf{f}(\mathbf{x}_i), (L+1)\delta) \cap \mathcal{V}_j \neq \emptyset$. With our hypothesis on the domain, such a graph can be effectively obtained from m , considering a suitable discretisation of the rational box X .

From Lemma 6.2.13 and Lemma 6.2.14, $\neg R_{\omega}^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ holds iff for some $\delta = 2^{-m}$, $\neg(\mathcal{V}_i \xrightarrow{*}_{\delta} \mathcal{V}_j)$ for some $\mathcal{V}_i, \mathcal{V}_j$ with $\mathbf{x} \in \mathcal{V}_i, \mathbf{y} \in \mathcal{V}_j$. This holds iff for some integer m , $\text{NOPATH}(G_m, \mathcal{V}_i, \mathcal{V}_j)$ for some $\mathcal{V}_i, \mathcal{V}_j$ with $\mathbf{x} \in \mathcal{V}_i, \mathbf{y} \in \mathcal{V}_j$. The latter property is c.e., as it is a union of decidable sets (uniform in m), as $\text{NOPATH}(G_m, \mathcal{V}_i, \mathcal{V}_j)$ is a decidable property over the finite graph G_m . \square

Definition 6.2.25 (Robust reachability relation)

We say that the reachability relation is reachability-robust when $R^{\mathcal{H}} \cap \text{Inputs} = R_{\omega}^{\mathcal{H}^+} \cap \text{Inputs}$.

We get the “robustness conjecture”, from the fact that a c.e and co-c.e. set must be decidable: with the hypotheses of Theorem 4.2.6, if the relation $R^{\mathcal{H}}$ is reachability-robust then it is decidable.

Complexity issues

Assume the dynamical system is robust. Hence, for all $\mathbf{x}, \mathbf{y} \in \text{Inputs}$, there exists ε (depending on \mathbf{x}, \mathbf{y}) such that $R^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ and $R_{\varepsilon}^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ have the same truth value (unchanged by smaller ε).

We can generalise some statements of Chapter 4:

Definition 6.2.26 (Level of robustness ε by s , generalisation of Definition 4.2.31)

Given a function $s : \mathbb{N} \rightarrow \mathbb{N}$, we write $R_{\{s\}}^{\mathcal{H}}$ for the relation defined as: for any points $\mathbf{x} \in \text{Inputs}$ and $\mathbf{y} \in \text{Inputs}$ the relation holds iff $R_{s(\ell(\mathbf{x})+\ell(\mathbf{y}))}^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$.

Definition 4.2.21 still holds in that context.

Theorem 6.2.27 (Generalisation of Theorem 4.2.33)

Consider a system satisfying the hypotheses of Theorem 6.2.24. Given some polynomial p , $R_{\{p\}}^{\mathcal{H}} \in \mathbf{PSPACE}$.

Proof. From Theorem 4.2.6, for all n there exists some m (depending on n), such that $R_n^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ and $R^{G_m}(\mathbf{x}, \mathbf{y})$ have the same truth value, where R^{G_m} denotes reachability in the graph G_m . With the hypotheses, given \mathbf{x} and \mathbf{y} , we can determine whether $R_{\{p\}}^P(\mathbf{x}, \mathbf{y})$, by determining the truth value of $R_n^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$, taking n polynomial in $\ell(\mathbf{x}) + \ell(\mathbf{y})$. From Theorem 4.2.6, the corresponding m is linearly related to n . The analysis of Corollary 2.3.4 shows that the truth value of $R^{G_m}(\mathbf{x}, \mathbf{y})$ can be determined in space polynomial in m . \square

6.2.5 Reachability and Witness of non-reachability

Robustness turns out to be related to the existence of (witness of) proofs of non-reachability in a specific form, namely in the form of a finite abstraction graph. Assume the dynamical system Φ' over X' simulates some dynamical system Φ over X (as in Figure 6.1): by induction, if \mathbf{y} is reachable from \mathbf{x} in Φ , then from any \mathbf{x}' with $R(\mathbf{x}, \mathbf{x}')$ there must exist some \mathbf{y}' with $R(\mathbf{y}, \mathbf{y}')$ reachable from \mathbf{x}' in Φ' .

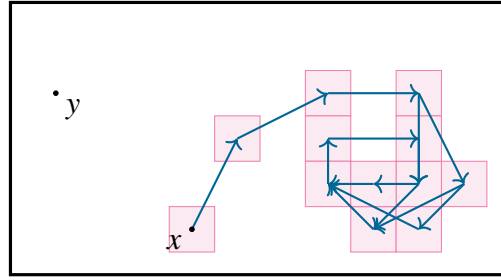


Figure 6.1: A dynamical system simulating the dynamical system of Figure 1.1: its states are indicated by boxes and the possible transitions between states by an arrow. The fact that \mathbf{y} cannot be reached from \mathbf{x} is witnessed by this dynamical system, as trajectories starting from \mathbf{x} will remain forever in color boxes and hence away from \mathbf{y} .

Dually, this can be used as a witness that a point is not reachable from another point:

Definition 6.2.28 (Finite abstraction witness of non-reachability of \mathbf{y} from \mathbf{x})

A graph/dynamical system Φ' over a finite set (i.e a graph) is called a *finite-abstraction witness of non-reachability of \mathbf{y} from \mathbf{x} in Φ* if there is no such \mathbf{y}' is reachable from any such \mathbf{x}' in Φ' .

Theorem 6.2.29 (generalisation of Proposition 4.2.18)

Assume the hypotheses of Theorem 6.2.24. If $R_{\omega}^{\mathcal{H}} = R^{\mathcal{H}}$ then $R^{\mathcal{H}}$ is decidable and whenever $R^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ is false, there always exists some finite abstraction witness of non-reachability of \mathbf{y} from \mathbf{x} . This witness has a non-empty invariance kernel. Furthermore, this witness can be computed effectively.

The proof is the same as the proof of Proposition 4.2.18: finding a finite abstraction to reason about discretisations of the space considered in the proof of Theorem 6.2.12: To every $\delta = 2^{-m}$, $m \in \mathbb{N}$, is associated some graph G_m . Then, the proof consists in showing that the strategy of increasing m until we find a witness in that form must necessarily terminate under the hypotheses. Decidability comes from the fact that a c.e. and co-c.e. decision problem is decidable.

Proof of Theorem 6.2.29. (\Rightarrow): For all $\varepsilon > 0$, $R_\varepsilon^{\mathcal{H}}(\mathbf{x})$ satisfies $\mathbf{x} \in R_\varepsilon^{\mathcal{H}}(\mathbf{x})$ and $\mathbf{f}_\varepsilon(R_\varepsilon^{\mathcal{H}}(\mathbf{x})) \subseteq R_\varepsilon^{\mathcal{H}}(\mathbf{x})$ (this is even the smallest set such that this holds). Let $\mathbf{y} \in \mathbb{Q}^d$, let us assume that $R^{\mathcal{H}}(\mathbf{x}, \mathbf{y}) = R_\omega^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ is not true. Then, there exists ε such that $R_\varepsilon^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ is false, i.e. $\mathbf{y} \notin R_\varepsilon^{\mathcal{H}}(\mathbf{x})$. Consider $\mathcal{R}^* = R_\varepsilon^{\mathcal{H}}(\mathbf{x})$. Then, $\mathbf{x} \in \mathcal{R}^*$ and from the first paragraph $\mathbf{f}_\varepsilon(\mathcal{R}^*) \subseteq \mathcal{R}^*$ and $\mathbf{y} \notin \mathcal{R}^*$.

(\Leftarrow): When $R^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ is true, for all $\varepsilon > 0$, $R_\varepsilon^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ is true, so $R_\omega^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ is. When $R^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ is false, by hypothesis, $R^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ is ε -far from being true for some $\varepsilon > 0$: there exists a set \mathcal{R}^* satisfying $\mathbf{x} \in \mathcal{R}^*$ and $\mathbf{f}_\varepsilon(\mathcal{R}^*) \subseteq \mathcal{R}^*$. As $R_\varepsilon^{\mathcal{H}}(\mathbf{x})$ is the smallest such set, $R_\varepsilon^{\mathcal{H}}(\mathbf{x}) \subseteq \mathcal{R}^*$. As $\mathbf{y} \notin \mathcal{R}^*$, $\mathbf{y} \notin R_\varepsilon^{\mathcal{H}}(\mathbf{x})$. Hence $R_\omega^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ is false. \square

NB 6.2.30 (The necessity of a non-empty kernel and loops)

In particular, the associated graph must have loops: consider any generator g , there must be a path in Φ' corresponding to the sequence $\Phi'(ng, \mathbf{x}')$ for increasing n . The values of this sequence must come from the pigeonhole lemma.

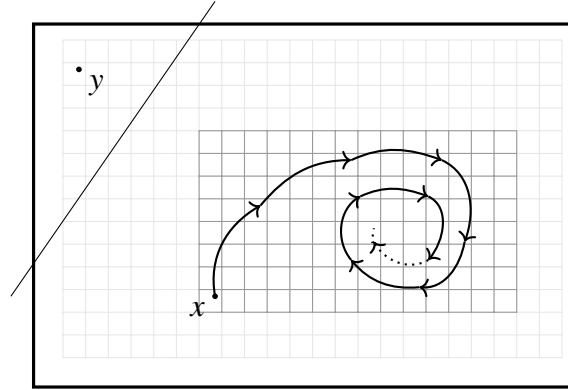


Figure 6.2: In a robust system, when y is not reachable from x , there exists some finite abstraction witness, obtained by discretising the space. It is effectively computed by considering a thinner discretisation until one is found

Reasoning about non-reachability: a topological view

It is possible to go further and use this to obtain more global properties, using a topological explanation of what happens in the case of robustness. Let $NR^{\mathcal{H}}(\mathbf{y})$ be the sets of points not reaching \mathbf{y} (or a class of \mathbf{y} 's): $NR_{+\varepsilon}^{\mathcal{H}}(\mathbf{y}) = \{\mathbf{x} \mid \neg R_{+\varepsilon}^{\mathcal{H}}(\mathbf{x}, \mathbf{y})\}$. The point is that, when there is such a finite abstraction, each vertex of it represents a subset of the space X . Namely, \mathbf{x} belongs to some vertex \mathcal{V}_i . But we also get that all points of $\text{int}(\mathcal{V}_i)$ must also necessarily be points such that $R^{\mathcal{H}}(\mathbf{x}, \mathbf{y})$ is false. We also get that there exists some neighbourhood of \mathbf{x} inside $NR_{+\varepsilon}^{\mathcal{H}}(\mathbf{y})$.

Hence, given some $\delta = 2^{-m}$, the graph G_m provides informations about a subset of points \mathbf{x} not reaching \mathbf{y} : it provides some open O_m included in $NR_{+\varepsilon}^{\mathcal{H}}(\mathbf{y})$. Let $O = \bigcup_{n \in \mathbb{N}} O_n$.

From Corollary 6.2.8, robustness holds iff, for all basic open set B_i , B_i is in $NR^{\mathcal{H}}$ iff B_i is in $L(O_n)$ for some $n = n(B_i)$. It is immediate from Proposition 6.2.7.

If we come back to previous considerations of Subsection 6.2.3, this means, that for all basic open set B_i , there is some $n = n(B_i)$ so that we can reason on some finite graph to tell whether it is in $NR^{\mathcal{H}}$.

We consider $n(B)$, the function of the radius 2^{-n} of the ball B_i . We know from all these arguments that the graph G_{n+1} for a given $n+1$ simulates the graph G_n for n in the general case.

6.2.6 A specific case

An interesting case is when the graph G_n , for various $n \in \mathbb{N}$, is isomorphic for the following concept: we can describe all these graphs in a concise way. We say that this an *isomorphism invariant*, written $G_{n+1} \equiv G_n$ to indicate this invariant holds.

We will see that such graphs provides a sound and complete methods to prove that a given subshift can tile the plane and even to discuss aperiodicity.

6.3 Robustness of Turing machines

We can apply all the previous frameworks to the case of Turing machines, to revisit some of the statements of [AB01] and of Chapter 4.

6.3.1 Space-time diagrams of a Turing machine

We write $ST(M, w)$ for the space-time diagrams of a (possibly non-deterministic) TM M on input w and $ST(M)$ for the set of all its space-time diagrams. With each space-time diagram, we associate its *patterns language*.

The space-time diagram of a Turing machine is not always decidable. It is co-c.e, even co-c.e complete, in the general case. Theorem 6.3.1 remains true even when M is fixed.

Theorem 6.3.1

Given M and p , determining whether a pattern p is in $\mathcal{L}(ST(M))$ is co-c.e. complete.

First, we state and prove the following lemma:

Lemma 6.3.2

Given a Turing machine M , $\mathcal{L}(ST(M))$ is always co-c.e..

Proof. This is a usual argument about tiling (but here on a space-time diagram): pattern p does not appear in $ST(M)$ if and only we can find a sufficiently large rectangle where it is impossible to complete the tiling (this is a compactness argument, see Proposition 1.5.20).

Indeed, if such a rectangle cannot be found, by compactness, we can construct a configuration of $ST(M)$ with pattern p . If such a rectangle is found, we know it is a contradiction. \square

Proof of Theorem 6.3.1. First, it is co-c.e.-hard: this is exactly what the proof of [JV20, Theorem 7] states, which they attribute to Robinson [Rob71] (while omitting the tiles in their proof).

Namely, we can construct a Turing machine M_{univ} that operates on a half-plane bounded by $|$: it starts with $|wq_0$, performs its operations, and never moves to the left of $|$. We design this machine to simulate $M(w)$ on the empty word.

We observe that:

- The pattern p given by $|wq_0$ is part of its patterns if and only if the machine w does not halt on w : by definition.
- Therefore, determining whether a pattern p is in $\mathcal{L}(ST(M_{univ}))$ is co-c.e.-hard.

This is also co-c.e. from Lemma 6.3.2. \square

Pattern-robustness of Turing machines

We can adapt the notion of robustness to space-time diagrams. We introduce:

Definition 6.3.3 (Pattern-robustness)

A Turing machine M is pattern-robust iff $\mathcal{L}(ST(M)) = \mathcal{L}_\omega(ST(M))$, where $\mathcal{L}_\omega(ST(M)) = \bigcap_{n \in \mathbb{N}} \mathcal{L}(ST(M_n))$.

Pattern-robustness for dynamical systems and Turing machines are equivalent. It is also equivalent to the definition of robustness presented in Chapter 4.

It is also possible to quantify pattern-robustness, using the fact that the language of $\mathcal{L}(ST(M_n))$ is decidable in space polynomial in n , we get:

Theorem 6.3.4

Assume M is some polynomial pattern-robust Turing machine. $\mathcal{L}(ST(M_n)) \in \text{PSPACE}$ can be decided in space polynomial in n from the description of M .

Also, on some input $w \in L$ is n -pattern-robust iff there exists $n \in \mathbb{N}$ such that $w \in L_n$. In other words, all the patterns of pattern-robust Turing machine are n -robust for some $n \in \mathbb{N}$. The fact that n can be chosen to be a polynomial in the length of w corresponds exactly to polynomial space languages.

6.4 Robust Subshifts

6.4.1 Computability and uncomputability issues for subshifts

The language of an effectively closed subshift is not decidable in the general case, but it is always co-c.e. (see discussions in Subsection 1.5.1). However, we can state the following: a minimal, effectively closed subshift has a decidable language. We can prove:

Theorem 6.4.1

Assume \mathbf{T}_{em} is some effectively closed subshift, that is essentially minimal (Definition 1.5.14). Then $\min(\mathbf{T}_{em})$ is computably enumerable.

So, in an essentially minimal effectively closed subshift, we always have \mathbf{T}_{em} co-c.e. and $\min(\mathbf{T}_{em})$ is a c.e. subset of it, but possibly undecidable.

Before proving Theorem 6.4.1, we must introduce some notation (Definition 6.4.2) and prove Theorem 6.4.3:

Definition 6.4.2 (Forcing relation \Vdash)

Consider a subshift \mathbf{T} . Write $p \Vdash q$ if there exists some integer n , such that for every pattern U avoiding \mathcal{F} of $[-n, n]^d$, we have $p \sqsubset U \Rightarrow q \sqsubset U$.

Theorem 6.4.3

Consider patterns p and q in $\mathcal{R}ecurrent(\mathbf{T}_{em})$. Then necessarily $p \Vdash q$

Proof. Consider $x \in \min(\mathbf{T}_{em})$. We know that p and q are recurrent in any configuration of $\min(\mathbf{T}_{em})$ (Proposition 1.5.15). So, in particular in x . So they are among patterns of x . Consider $\mathbf{T}_{em} \cap [p]$, that is a closed set. If it was wrong, that means that for all n , we could fill $[-n, n]^d$, in $\mathbf{T}_{em} \cap [p]$, hence with pattern p , and avoiding patterns of \mathcal{F} and q . By compactness, there is consequently a configuration $x' \in \mathbf{T}_{em}$, but without q . Consider $\min(\mathcal{O}(x'))$. It should be avoiding $[q]$, but as \mathbf{T}_{em} is essentially minimal, it should be $\min(\mathbf{T}_{em})$. We reach a contradiction, as q is a pattern of $x \in \min(\mathbf{T}_{em})$. \square

Theorem 6.4.4

Assume $p \in \mathcal{R}ecurrent(\mathbf{T}_{em})$. Assume $p \Vdash q$. Then $q \in \mathcal{R}ecurrent(\mathbf{T}_{em})$.

Proof. Pattern p appears by definition in a configuration $x \in \min(\mathbf{T}_{em})$. From the definition of $p \Vdash q$, q cannot be avoided. So in particular, it cannot be avoided in x . So q appears also in configuration x . \square

Then, we can get the proof:

Proof of Theorem 6.4.1. Either $\min(\mathbf{T}_{em})$ is empty, and then it is computably enumerable. Otherwise: take $p \in \min(\mathbf{T}_{em})$. We know from the two statements of Theorem 6.4.3 and 6.4.4, that $\min(\mathbf{T}_{em}) = \{q : p \Vdash q\}$. The relation \Vdash is computably enumerable from its definition. \square

NB 6.4.5

The proof is “non-constructive”: it requires to know whether $\min(\mathbf{T}_{em})$ is empty or not, and when it is not, it requires some $p \in \min(\mathbf{T}_{em})$.

6.4.2 Robustness of subshifts

We now apply our general theory to the case of subshifts and tilings.

Fix a set of forbidden patterns \mathcal{F} . We write $\mathbf{T}_{[n_1, n_2]}$ for the $x \in \mathcal{A}^S$ avoiding patterns $p \in \mathcal{F}$ over the strip $S_{[n_1, n_2]} = \mathbb{Z} \times [n_1, n_2]$. We write \circ for composition, to mean $\mathbf{T}_{[n_1, n_2]} \circ \mathbf{T}_{[n_2, n_3]} = \mathbf{T}_{[n_1, n_3]}$. We write $\mathcal{L}_{[n, m]}(\mathbf{T}) = \{p \sqsubset \mathbf{T}_{[n, m]}\}$.

Definition 6.4.6 (Robustness of a subshift)

Let $g : \mathbb{N} \rightarrow \mathbb{N}$ be some increasing function, $g(n) \geq n$, such that, for all pattern p of size $\ell(p) = s$, the support of p is included in $[-s, s]^d$, up to some translation. Consider any $n \in \mathbb{N}$ such that $s \leq n$. An effectively closed subshift \mathbf{T} is subshift-robust for g if $p \in \mathcal{L}(\mathbf{T})$ iff $p \in \mathcal{L}_{n, g(n)} = \mathcal{L}_{[-n, n]} \left(\mathbf{T}_{[-g(n), g(n)]} \right)$.

We say an effectively closed subshift \mathbf{T} is subshift-robust if it is for some g .

Definition 6.4.7 (Level of robustness n given by s)

Given a function $g : \mathbb{N} \rightarrow \mathbb{N}$, we write $\mathcal{L}_{\{s\}}$ for the set of patterns defined by $\mathcal{L}_{\{s\}} = \{p \mid p \in \mathcal{L}_{\ell(p), g(\ell(p))}\}$. We say that a subshift, is g -robust, when $\mathcal{L} = \mathcal{L}_{\{g\}}$.

Given a pattern p , compute $n = \ell(p)$, $m = g(n)$ and determine whether $p \in \mathcal{L}_{n, m}$. Thus:

Theorem 6.4.8

A polynomial-robust subshift language \mathcal{L} is decidable in polynomial space.

We illustrate the necessity of this concept, as it holds for many famous tilesets, in the following section.

6.5 The Robinson tileset is polynomially robust

We write $\mathbf{T}_{\text{Robinson}}$ for the SFT corresponding to the Robinson tileset (accordingly to the discussions of Subsection 1.5.1). Its associated graph is in Figure 6.4.

We convert this tileset into a transducer by defining two alphabets for horizontal and vertical colours

$$H := \{ \text{---}, \text{---}, \text{---}, \text{---}, \text{---}, \text{---} \};$$

$$V := \{ \text{---}, \text{---}, \text{---}, \text{---}, \text{---}, \text{---} \}.$$

Note that with this choice for H and V , we forget about the corner types (bumpy or dented). However, this will not be an issue later on, since no two tiles are wearing the same colours from H and V with different corner types. Robinson's tileset can be associated with the transducer $\mathcal{T}_{\text{Robi}}$ depicted in Figure 6.4. For more readability, in this figure, we label a transition from a state q to a state q' by the corresponding tile from Figure 6.3 with the proper orientation.

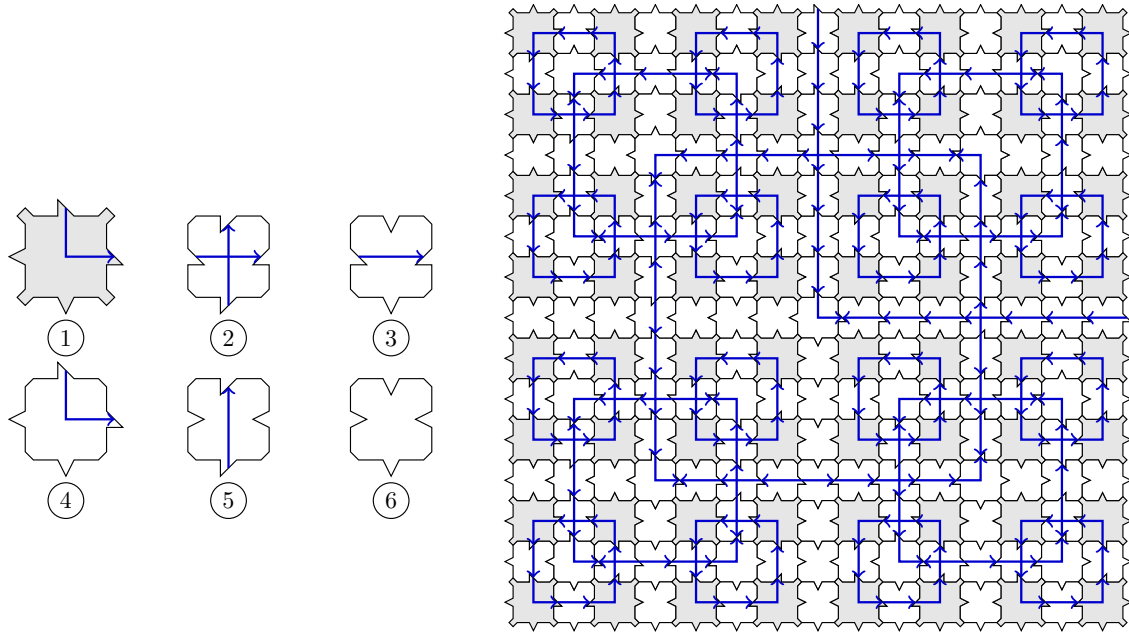


Figure 6.3: To the left is Robinson's tileset, where tiles can be rotated and reflected. To the right a pattern that appears in every tiling by Robinson's tileset.

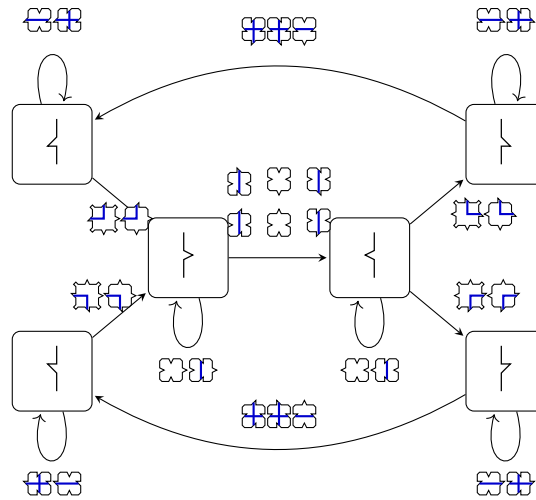






Figure 6.4: The line graph of the graph *Robi* that corresponds to Robinson's tileset. From the symmetry of the tileset, we actually represent only the east edges.

Theorem 6.5.1 (Fundamental remark, technical version from [Fer21])

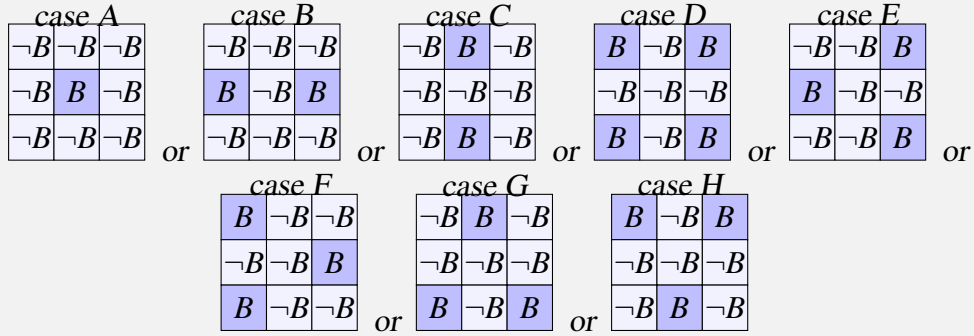
Consider a pattern of size s . Consider $n \in \mathbb{N}$ such that $s \leq g(n)$ and assume the pattern is surrounded by at least $g(s)$ lines above or below. It has either 0, 1, 2, or 3 arms. In the quarter, or half delimited by these arms, we have patterns that are subpatterns of bumpy's.

We must prove some auxiliary lemmas before proving Theorem 6.5.1:
Conventions:

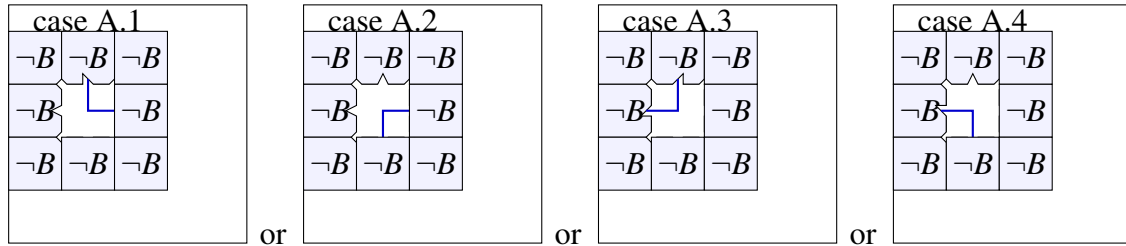
- We write  to mean a bumpy tile.
- We write  to mean a “not bumpy” tile.
- We write  to mean an unspecified tile.
- We write  for a north-east n -bumpy.

Lemma 6.5.2 (Bumpy/NotBumpy Parity)

Consider a 3×2 windows of a configuration of Robinson. It matches either



Proof. We reason by cases. We only show Case A here, the proofs for the other cases are similar. Case A is necessarily of the form



□

A sketch of proof of Theorem 6.5.1 is the following: take a pattern p of size s . Consider n such that $s \leq g(n)$. We establish p is a pattern of $\mathcal{L}(\text{Robinson})$ iff $p \in \mathcal{L}_{[-s,s]}(\text{Robinson}_{[-g(s),g(s)]})$. This follows from a case discussion as in the previous lemma and the discussions of various possible patterns in the Robinson tileset. See e.g. [Fer21].

Thus, we have the following consequences:

Corollary 6.5.3

The Robinson tileset is polynomially robust.

Corollary 6.5.4

The language of the Robinson is in **PSPACE**.

6.5.1 Robustness and invariants

We can now illustrate the observations of Subsection 6.2.5. The graphs mentioned there are described by the following graph that we call H_n (Figure 6.5) and we can restate the observations of Subsection 6.2.5, with the following vocabulary, which may be easier to understand in this context.

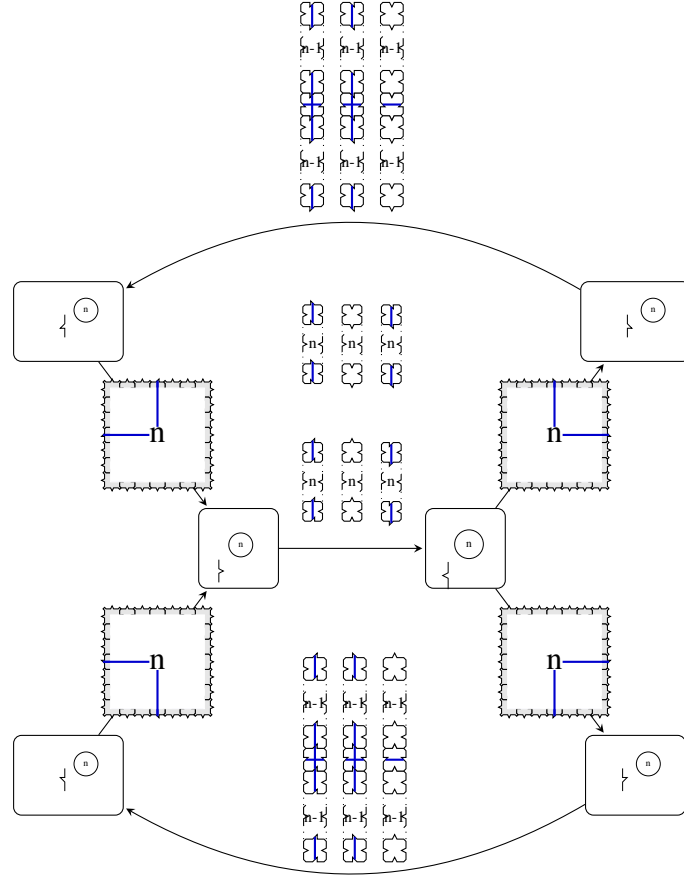


Figure 6.5: Graph H_n

Definition 6.5.5 (Meta-pattern)

Let \mathcal{P} be a set of patterns. Let $\mathbb{U} \subset \mathbb{Z}^d$ be a finite set. A meta-pattern is an element $P \in \mathcal{P}^{\mathbb{U}}$, the support of P is $\text{supp}(P) = \mathbb{U}$.

Notice that a pattern, is a meta-pattern, by considering \mathcal{P} to be \mathcal{A} : size 1 patterns. It describes the graph(s) described in Subsection 6.2.5. Indeed,

1. Its edges are labelled with meta-patterns: that is to say by basic open sets;

2. Each node corresponds to the east-compatibility: any ingoing edge in a node is east-compatible with any outgoing edge from the node: we draw here only the edges with the east label.
3. From the symmetry of the Robinson tileset, we could replace east, with west, south, or north in the above statements (or consider H_n to be an east-graph, while it is also a west, south, or north graph) to get the other edges.
4. The graph has loops, following remark 6.2.30.
5. We are in the case of an isomorphism invariant: consider H_n for various values of n : the meta-pattern s_n in a given vertex are label-isomorphic: if we consider $n = n_1$ and $n = n_2$, then s_{n_1} and s_{n_2} are label-isomorphic.
Notice the following:
6. From 1., and 2., a path in the graph provides a pattern avoiding the forbidden patterns \mathcal{F} . In particular, a simple loop in the graph provides a pattern $p = p_n$. All such patterns p_n appear in meta-patterns of all loops of $H_{n'}$ for some $n' \geq n$.

A sound and complete method for the domino problem:

Finding such a graph(s) provides a sound and complete method to prove that we can tile the plane with a given subshift.

Theorem 6.5.6 (*A sound and complete method for the domino problem*)

- Suppose we have a graph H_n with the Properties 1. to 5. (or d graphs for all d generators of \mathbb{Z}^d). Then, we can tile \mathbb{Z}^d with this subshift.
- Suppose that we have Property 6. Then all the patterns p_n appear inductively, for all n , and the subshift is essentially minimal. Conversely, if the subshift is essentially minimal, then Property 6. holds.

Indeed, we already observed that Properties 1., to 4. are mandatory for robust subshifts. Conversely, if we have Properties 1. to 4., we know that we can tile arbitrarily large supports. By compactness (Definition 1.5.20), we can tile \mathbb{Z}^d .

The *Robinson*(M) tileset is robust iff M is

What we stated holds for many other tilesets. In particular: Robinson described in [Rob71] a method to convert a Turing machine M into a Wang tileset τ_M such that M does not halt on the empty tape iff τ_M tiles the plane.

Theorem 6.5.7

*If the Turing machine M is robust and not terminating, then the tileset τ_M is robust. The associated language is in **PSPACE** iff M is polynomially robust.*

6.6 Application to other tilesets

6.6.1 Extension: Tiling equivalence

In Section 1.5.4 we defined meta-transducers not to get new or stronger results, but rather to simplify notations and proofs. With the same objective in mind, we define a notion of equivalence between tilesets that are not necessarily Wang tilesets.

If τ is a tileset, we say that a tileset $\tilde{\tau}$ is *equivalent* (denoted by \equiv) to τ if (1) each tile in $\tilde{\tau}$ is a valid pattern on τ (2) every tiling x by τ can be decomposed into patterns from $\tilde{\tau}$.

Proposition 6.6.1

Provably robustness and aperiodicity are preserved by tiling equivalence.

Proof. For provably robustness this follows directly from the definitions, since the bijection involved in tiling equivalence is chosen provable. Assume that x is a periodic tiling by τ which is tiling equivalent to τ' through sets of patterns $\tilde{\tau}, \tilde{\tau}'$ and a provable bijection Φ . Then $\Phi(x)$ is a tiling by $\tilde{\tau}'$ that is also periodic. So aperiodicity is also preserved by tiling equivalence. \square

6.6.2 Jeandel-Rao's tileset

In this section, we prove the polynomial robustness of Jeandel Rao's aperiodic tileset \mathcal{T} considered in [JR21].

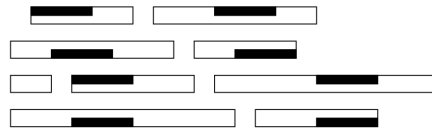
Definition 6.6.2

A tileset τ is tiling equivalent to a tileset τ' if there exist $\tilde{\tau}$ (resp. $\tilde{\tau}'$) equivalent to τ (resp. to τ') and a provable bijection $\Phi : \tau \rightarrow \tau'$ such that x is a tiling by $\tilde{\tau}$ iff $\Phi(x)$ is a tiling by $\tilde{\tau}'$.

Example 6.6.3



is tiling equivalent to the tileset τ_2 pictured below, where not two north (resp. south) black decorations have the same width.



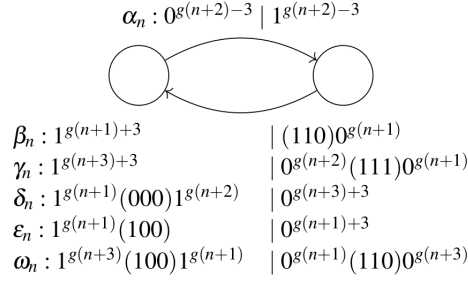
Indeed, from a tiling from the first, by ignoring colours we get a tiling from the second. Conversely, a tiling from the second can be transformed into a tiling from the first, as colours can be recovered by only looking at the width of black decorations.

The two previous tilesets are tiling equivalent to the Jeandel-Rao's Wang tileset [JR21], composed with the 11 tiles pictured below.

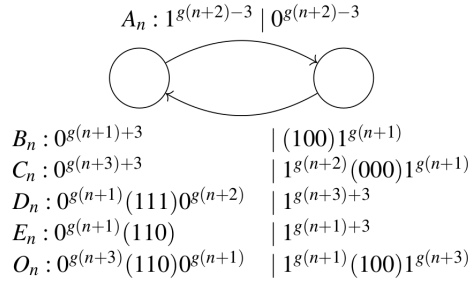


The discussion of [JR21] proves that \mathcal{T} is tiling equivalent to $g(0) = 1$, $g(1) = 2$ and $g(n+2) = g(n) + g(n+1)$ for all n , these finite abstractions are defined by

- for n even:



- for n odd:



The proof of Lemma 9 in [JR21] can be rephrased as a proof that for every n even one has

$$(A_{n+1} \cup C_{n+1}) \circ (\gamma_n \cup \alpha_n \cup \delta_n) \circ (D_{n+1} \cup A_{n+1}) \equiv A_{n+3}.$$

Similar expressions can be established for B_{n+3} , C_{n+3} , D_{n+3} , E_{n+3} , O_{n+3} , from the proof of same lemma. For n odd we can formulate similar relations, mainly by inverting the role of Greek and Latin letters. It follows that $T_{n+1} \circ T_n \circ T_{n+1} \equiv T_{n+3}$. The fact that T_n contains a loop is clear from its definition.

Theorem 6.6.4

Jeandel Rao's aperiodic tileset \mathcal{T} is polynomially robust.

6.6.3 Kari's tilings

Here, we briefly present a technique due to Kari to encode piecewise rational affine maps computations inside Wang tilings. We encourage readers to refer to [Kar07] for details about the construction.

Balanced representation of real numbers

Let $i \in \mathbb{Z}$. We say that a bi-infinite sequence $(x_k)_{k \in \mathbb{Z}}$ of i 's and $(i+1)$'s *represents* a real number $x \in [i, i+1]$ if there exists a sequence of intervals $I_1, I_2, \dots \subseteq \mathbb{Z}$ of increasing lengths $n_1 < n_2 < \dots$ such that $\lim_{k \rightarrow \infty} \frac{\sum_{j \in I_k} x_j}{n_k} = x$, that is to say, the averages of $(x_k)_{k \in \mathbb{Z}}$ over the intervals converge to x .

For a real number x , we write $\lfloor x \rfloor$ for the integer part of x , which is the largest integer that is less than or equal to x .

Definition 6.6.5 (Balanced representation)

Let $x \in \mathbb{R}$. For $k \in \mathbb{Z}$ define $B_k(x) := \lfloor kx \rfloor - \lfloor (k-1)x \rfloor$. The bi-infinite sequence $(B_k(x))_{k \in \mathbb{Z}}$ is called the balanced representation of x . For $\vec{x} \in \mathbb{R}^d$, define $\lfloor \vec{x} \rfloor$ or $B_k(\vec{x})$ coordinatewise.

Clearly $B_k(x) \in \{\lfloor x \rfloor, \lfloor x \rfloor + 1\}$ and $(B_k(x))_{k \in \mathbb{Z}}$ is a representation of x in the sense defined above. A key observation is that balanced representations of irrational $x \notin \mathbb{Q}$ are Sturmian sequences [FBF⁺02], while for rational $x \in \mathbb{Q}$ the sequence is periodic.

Rational piecewise affine maps

A *rational piecewise affine map* is given by finitely many pairs (U_i, f_i) where each U_i are disjoint unit cubes of \mathbb{R}^d with integer corners; f_i are affine maps with rational coefficients.

Each set U_i is the domain where f_i can be applied. The system $(U_i, f_i)_{i=1 \dots n}$ determines a function $f : D \rightarrow \mathbb{R}^d$, its domain is $D = \bigcup_i U_i$ and $f(\vec{x}) = f_i(\vec{x})$ for all $\vec{x} \in U_i$.

Tileset associated with a rational piecewise affine map

Kari's constructions are based on the idea that we can associated with some Wang tileset a rational (piecewise) affine map f .

$$\begin{array}{ccc} & B_k(f_i(\vec{x})) & \\ C_{f_i, k-1}(\vec{x}) & \boxed{} & C_{f_i, k}(\vec{x}) \\ & B_k(\vec{x}) & \end{array}$$

Figure 6.6: Tile τ_{f_i}

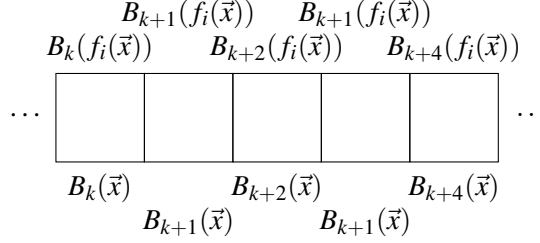
Definition 6.6.6 (Tileset τ_{f_i} associated with f_i)

The tileset τ_{f_i} corresponding to a rational affine map $f_i(\vec{x}) = Y\vec{x} + \vec{b}$ and its domain cube U_i consists of all tiles as shown in Figure 6.6, where $k \in \mathbb{Z}$, $\vec{x} \in U_i$, and $C_{f_i, k}(\vec{x})$ is a rational number depending on f_i, \vec{x} and k (details can be found in [Kar07]).

Every tile as pictured above locally computes f_i with some error since we have

$$f_i(B_k(\vec{x})) + C_{f_i, k-1}(\vec{x}) = B_k(f_i(\vec{x})) + C_{f_i, k}(\vec{x}).$$

Moreover, because f_i is rational, there are only finitely many such tiles (even though there are infinitely many $k \in \mathbb{Z}$ and $\vec{x} \in U_i$). Kari then considers the tileset τ_f obtained as the union of the τ_{f_i} , with additional rules to ensure that only tiles from the same τ_{f_i} can appear on a given row:



Tiling problem and iterations of a rational piecewise map

Let (D, f) be a rational piecewise affine map. A real number $x \in D$ is an *immortal point* for f if for all $k \in \mathbb{N}^*$, $f^{[k]}(x) \in D$. If such an x exists, f is called *immortal*. A non-empty set $I \subseteq D$ is a *rational invariant box* if I is non-empty, $I \cap \mathbb{Q} \neq \emptyset$ and $f(I) \subset I$. Every $x \in I$, where I is a rational immortal box, is an immortal point for f .

Proposition 6.6.7

Let (D, f) be an immortal rational piecewise affine map. Then τ_f can tile the plane. Moreover, if f has a rational invariant box I then τ_f is polynomially robust.

The immortality problem for rational piecewise affine maps is undecidable [BBKT01]. The proof relies on the encoding of any Turing machine M inside a rational piecewise map f_M , such that f_M is immortal iff M does not halt. This extends classical ways to simulate a Turing machine using rational piecewise affine maps [KCG94]. We can then consider the associated tiling τ_{f_M} .

Theorem 6.6.8

Consider a Turing machine M . Then M is polynomially robust and not terminating iff τ_{f_M} is polynomially robust.

6.7 Other types of robustness

We consider another approach to robustness in this section. Transducers and meta-transducers are natural for tilings. We link here the robustness of a tileset to properties on the (meta-)transducers. Adopting this alternative point of view, the statement “tilesets encoding a Turing machine are robust iff the Turing machine is itself robust” remains true in this setting. This approach, using the notion of proof in logic, can also be related to Section 4.6.

Definition 6.7.1 (Notation $\mathcal{T} \asymp \mathcal{T}'$)

Given two transducers \mathcal{T} and \mathcal{T}' , we write $\mathcal{T} \asymp \mathcal{T}'$ iff every edge of the transducer \mathcal{T}' is an edge of \mathcal{T} .

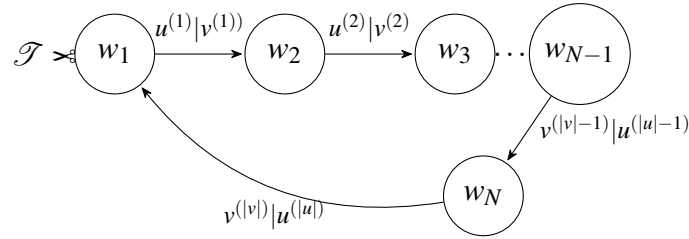
This just means that the local constraints associated with \mathcal{T}' fulfil all the constraints associated with \mathcal{T} . We use a scissor symbol, as this corresponds to deleting (cutting) some edges.

As such, when we write $\mathcal{T} \succ \mathcal{T}'$ and the states of \mathcal{T} are of height n , then \mathcal{T}' is necessarily made of states of height n .

Proposition 6.7.2

Let τ be a tiling and \mathcal{T} the associated transducer. If \mathcal{T} contains a cyclic loop of order N then \mathcal{T}^m contains a periodic loop for some $1 \leq m \leq N$.

Proof. Assume that \mathcal{T} contains a cyclic loop (w, w, uv, vu) with $|uv| = N$. Said otherwise, there exists a sequence of states $w_1 = w, \dots, w_N$ such that



Thus, we have a sequence of loops

$$\mathcal{T} \succ (w_i, w_i, u^{(i)}v^{(i)}, v^{(i)}u^{(i)})$$

for every $i = 1 \dots N$. Since there are N cyclic permutations of uv and we have exactly N words $b_i = u^{(i)}v^{(i)}$ and N words $t_i = v^{(i)}u^{(i)}$, each t_i must also be some $b_{\sigma(i)}$ where $\sigma : [1; N] \rightarrow [1; N]$ is a permutation. Denote m the size of the orbit of 1 by σ , it is the size of the cycle $(1, \sigma(1), \dots, \sigma^{m-1}(1))$ with $\sigma^m(1) = 1$. Using iteratively the rule for composition on the transducer \mathcal{T} , we have that :

$$\mathcal{T}^m \succ (w, w, b_1, t_m)$$

and since $t_m = b_{\sigma^m(1)} = b_1$ this is a periodic loop. \square

Proposition 6.7.3

Consider a transducer \mathcal{T} . Then \mathcal{T}^n contains a loop iff τ there exists a valid tiling by τ of $\mathbb{Z} \times [0; n-1]$ the horizontal strip of height n .

Proof. If \mathcal{T}^n contains a loop, then there is an infinite path in this loop.

Conversely, there exists some $w, w' \in V^{\mathbb{Z}}$ such that $w \mathcal{T}^n w'$. Hence, there exists a bi-infinite sequence of states $q = (q_i)_{i \in \mathbb{Z}} \in H^{\mathbb{Z}}$ such $(q_i, q_{i+1}, w_i, w'_i) \in \mathcal{T}$ that for every $i \in \mathbb{Z}$. As there are finitely many states, there exists some q not appearing finitely many times: we must have $q_i = q_j$ with $i < j$ for some i, j . Then $(q, q, w_i w_{i+1} \dots w_{j-1}, w'_i w'_{i+1} \dots w'_{j-1}) \in \mathcal{T}$. In other words, \mathcal{T} has a loop in state q . \square

This can be reinforced in:

Proposition 6.7.4

Consider a transducer \mathcal{T} . Then \mathcal{T}^n contains a loop of order N iff τ there exists a valid tiling by τ of $\mathbb{Z} \times [0; n-1]$ the horizontal strip of height n using a pattern repeated infinitely with period N .

Proof. This follows from the previous proof, using the fact that a loop is associated with a rectangular pattern, whose width corresponds to its order. \square

Definition 6.7.5 (Composition pattern)

Consider a finite set S of transducers. Consider the signature made of the symbols of S (with arity 0) and the symbol \circ (of arity 2). A composition pattern F over S is a term over this signature.

For example $H \circ \tau \circ H$ is a composition pattern over $\{H, \tau\}$. Such a composition pattern is interpreted as expected: \circ corresponds to composition.

6.7.1 Robinson's tilings described by a recurrence relation

The notions of robustness are motivated by our exploration of Robinson's tileset with transducers. Our result on this particular tileset inspires Theorem 6.7.7, which is important for two main reasons. First, the transducer associated with the tileset τ_{Robi} is quite complex and things get worse as we compute compositions τ_{Robi}^n . So a naive approach would give complex and complicated proof. Yet this theorem expresses in an extremely simple way the fact that a valid tiling exists: the recurrence relation (6.1), which can be read as an invariant, concerning coming discussions. Second Theorem 6.7.7 is not specific to Robinson tileset: an analogous semantic statement can be written for every tileset for which one can prove it admits a valid tiling.

We set $g(n) := 2^n - 1$ for every integer $n \geq 0$, so that in the sequel we are only interested in the meta-transducers $\mathcal{T}_n := \mathcal{T}_{Robi}^{g(n)}$. Note that the sequence $(g(n))_{n \in \mathbb{N}}$ satisfies the recurrence relation $g(n) = 2g(n-1) + 1$. We also define notations for some horizontal and vertical patterns:

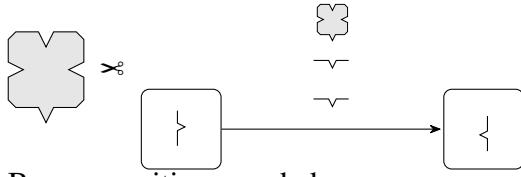
- $u^{(n)} := \text{---}^{g(n-1)} u \text{---}^{g(n-1)}$ for $u \in \{ \text{---}, \text{---}, \text{---} \}$;
- $b^{(n)} := \text{---}^{g(n-1)} b \text{---}^{g(n-1)}$ for $b \in \{ \text{---}, \text{---}, \text{---} \}$.

We also write $\begin{smallmatrix} a \\ \{n\} \\ a \end{smallmatrix}$ (for example, $\begin{smallmatrix} \text{---} \\ \{n\} \\ \text{---} \end{smallmatrix}$) for a pattern made of the tile a repeated $g(n)$ times

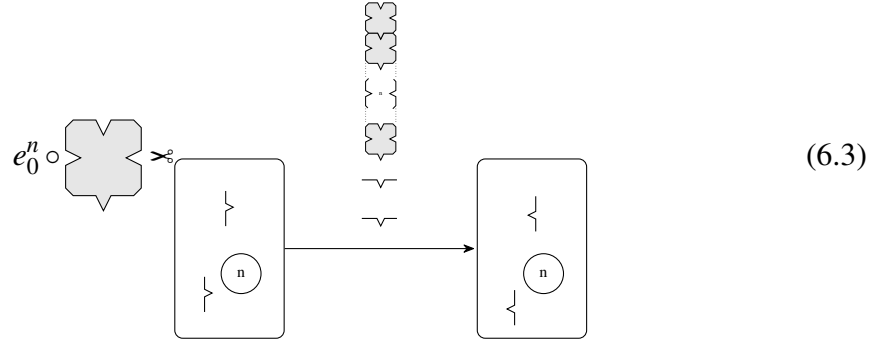
vertically, where a can be either --- , --- , --- , --- , --- or --- .

Studying the entire transducer \mathcal{T}_n would be too tedious (see Figure 6.7 for an insight into the combinatorics of \mathcal{T}_3). We rather consider the meta-transducer H_n defined in Figure 6.5. As we will see (Theorem 6.7.7), this transducer H_n satisfies that $\mathcal{T}_n \approx H_n$ for every $n \in \mathbb{N}^*$. For the sake of clarity, we choose to label the various edges e_j^n with patterns.

We have:



By composition, we deduce:

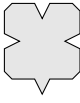

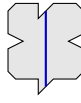
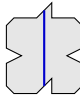
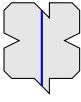



By composition with (6.2) we obtain the lemma. \square

Lemma 6.7.10

For all n , we have

$$\begin{aligned} e_0^n \circ \mathcal{T}_{Robi} \circ e_0^n &\succ_e e_0^{n+1}; \\ e_5^n \circ \mathcal{T}_{Robi} \circ e_5^n &\succ_e e_5^{n+1}. \end{aligned}$$

Proof. Use exactly the same reasoning, but replacing the role of  by , , ,  and  in previous lemma, for the first assertion.

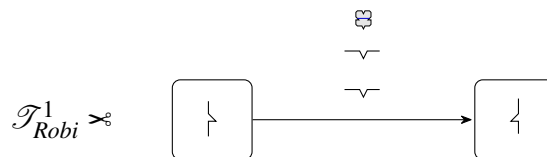
Replace the role of e_0^n by e_5^n in all previous arguments, to get the second assertion. \square

Lemma 6.7.11

For all n , we have

$$\begin{aligned} e_0^n \circ \mathcal{T}_{Robi} \circ e_0^n &\succ_e e_2^{n+1}; \\ e_0^n \circ \mathcal{T}_{Robi} \circ e_0^n &\succ_e e_{-2}^{n+1}. \end{aligned}$$

Proof. From (6.2),



and (6.2) again at rank $n - 1$, we obtain

$$e_0^n \circ \begin{array}{c} \text{---} \\ \text{---} \\ \text{---} \end{array} \circ e_0^n \asymp \begin{array}{c} \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{array}$$

Similarly, we can obtain:

$$e_0^n \circ \begin{array}{c} \text{---} \\ \text{---} \\ \text{---} \end{array} \circ e_0^n \asymp \begin{array}{c} \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{array}$$

and all the other patterns of e_2^{n+1} and e_{-2}^{n+1} . \square

Consequently we only need to prove the following lemma to complete the proof of Theorem 6.7.7.

Lemma 6.7.12

For all n , we have

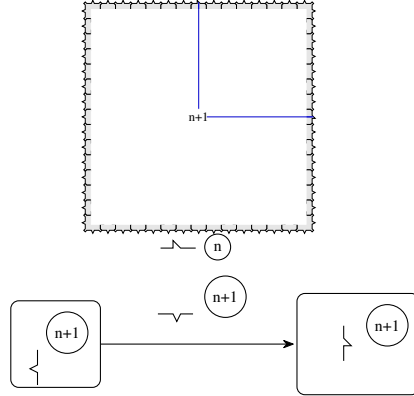
$$\begin{aligned} H_n \circ \mathcal{T}_{Robi} \circ H_n &\asymp e_1^{n+1}; \\ H_n \circ \mathcal{T}_{Robi} \circ H_n &\asymp e_3^{n+1}; \\ H_n \circ \mathcal{T}_{Robi} \circ H_n &\asymp e_{-1}^{n+1}; \\ H_n \circ \mathcal{T}_{Robi} \circ H_n &\asymp e_{-3}^{n+1}. \end{aligned}$$

Proof. We only prove the first assertion as the second is obtained by replacing the role of right and left in the following reasoning.

The third and fourth assertions can be obtained by inverting the role of height 1 and height $g(n)$ in the coming reasoning and by turning a rectangular pattern of some right

angle. So, we only need to prove:

$$H_n \circ \mathcal{T}_{Robi} \circ H_n \succ_{\circ}$$



(6.4)

□

6.7.2 Semantically robust tilesets

Definition 6.7.13

A tileset τ is semantically robust (or inductive) if there exists a family of transducers $(\mathcal{T}_n)_n$, of respective heights $(g(n))_n$, with $g : \mathbb{N}^* \rightarrow \mathbb{N}^*$ increasing and injective with $g(1) = 1$ and some (fixed) composition pattern $F = F(\mathcal{T}_1, \mathcal{T}_{n-k}, \dots, \mathcal{T}_n)$ over $\mathcal{T}_1, \mathcal{T}_{n-k}, \dots, \mathcal{T}_n$ for some integer k , such that:

1. Initialisation: $\tau \succ_{\circ} \mathcal{T}_1, \dots, \tau^{g(k+1)} \succ_{\circ} \mathcal{T}_{k+1}$

2. F provides an invariant:

$$\forall n, \quad F(\mathcal{T}_1, \mathcal{T}_{n-k}, \dots, \mathcal{T}_n) \succ_{\circ} \mathcal{T}_{n+1} \quad (6.5)$$

3. For all n , \mathcal{T}_n contains a loop.

6.7.3 Properties

Proposition 6.7.14

If a tileset τ admits a tiling, then it is semantically robust.

Proof. Consider $g(n) = n$ and $\mathcal{T}_n = \tau^{g(n)}$ for all n . We have 1) from definition, Condition 2) is satisfied, as we have $\mathcal{T}^{n+1} = \mathcal{T}^n \circ \mathcal{T}$ for all $n \geq 0$. Condition 3) comes from Proposition 6.7.3. □

Proposition 6.7.15

If a tiling τ is semantically robust (inductif), then it admits a tiling.

Proof. If τ is semantically robust, then by induction and combining conditions 1) and 2, we have that for all n , $\tau^{g(n)} \succcurlyeq \mathcal{T}_n$. By Proposition 6.7.3, we can tile horizontal stripe of height $g(n)$ for arbitrary large n . By a compactness argument, we conclude that τ tile the plane. \square

6.7.4 Provably robust tilingsets

There might be a strong difference between the fact that $\forall n, F(\mathcal{T}_1, \mathcal{T}_{n-k}, \dots, \mathcal{T}_n) \succcurlyeq \mathcal{T}_{n+1}$ holds for some composition pattern F , written as

$$\models \exists F, \quad \forall n, \quad F(\mathcal{T}_1, \mathcal{T}_{n-k}, \dots, \mathcal{T}_n) \succcurlyeq \mathcal{T}_{n+1}$$

and the fact that we can prove it holds, written

$$\vdash \exists F, \quad \forall n, \quad F(\mathcal{T}_1, \mathcal{T}_{n-k}, \dots, \mathcal{T}_n) \succcurlyeq \mathcal{T}_{n+1}.$$

This is for instance the case if a tilingset admits a tiling, but we cannot prove the existence of any valid tiling. The same phenomenon happens for Turing machines, for which non-termination may hold while there is no proof of it. In the meantime, we introduce the concept of *provably robust* tilingset to add this condition.

Definition 6.7.16

A tilingset τ is provably robust if there exists a family of transducers $(\mathcal{T}_n)_n$, of respective heights $(g(n))_n$, with $g : \mathbb{N}^* \rightarrow \mathbb{N}^*$ increasing and injective, $g(1) = 1$ and some (fixed) composition pattern $F = F(\mathcal{T}_1, \mathcal{T}_{n-k}, \dots, \mathcal{T}_n)$ over $\mathcal{T}_1, \mathcal{T}_{n-k}, \dots, \mathcal{T}_n$ for some integer k ,

1. Initialisation: $\tau \succcurlyeq \mathcal{T}_1, \dots, \tau^{g(k+1)} \succcurlyeq \mathcal{T}_{k+1}$
2. F provides an invariant, and this holds provably:

$$\forall n, \quad F(\mathcal{T}_1, \mathcal{T}_{n-k}, \dots, \mathcal{T}_n) \succcurlyeq \mathcal{T}_{n+1} \tag{6.6}$$

3. For all n , \mathcal{T}_n contains a loop.

Proposition 6.7.17

If a tilingset τ is provably robust then it admits a tiling.

Proof. If we have $\vdash \phi$ for some formula ϕ , then we have $\models \phi$. We did not what notion of proof \vdash we use (this could be provability in ZF set theory or provability in first-order logic starting from Peano's axioms of arithmetic), but of course, we expect it to be sound. Consequently, if τ is provably robust it is semantically robust. \square

Proposition 6.7.18

A periodic tileset is provably robust.

Proof. Let τ be a periodic tileset. Let x be a tiling by τ . Then it admits a period $(a, b) \in \mathbb{Z}^2 \setminus \{(0, 0)\}$ such that for all $(u, v) \in \mathbb{Z}^2$

$$x(u, v) = x(a + u, b + v).$$

Consider \mathcal{T} the transducer associated with τ . Then by periodicity of x the transducer $\mathcal{T}^{|b|}$ contains a cyclic loop. By Proposition 6.7.2 there exists $1 \leq m \leq |b|$ such that $\mathcal{T}_1 := \mathcal{T}^{m \cdot |b|}$ contains a periodic loop. Consequently for every $n \in \mathbb{N}^*$ the transducer $\mathcal{T}_n := \mathcal{T}^{n \cdot m \cdot |b|}$ contains a periodic loop. We are now ready to prove that τ is provably robust. Define $g(n) := n \cdot m \cdot |b|$, $k = 0$ and F the composition pattern $F(\mathcal{T}, \mathcal{T}') := \mathcal{T} \circ \mathcal{T}'$. Then all three conditions of provable robustness are satisfied:

1. $\mathcal{T}^{g(1)} = \mathcal{T}_1 \succ^* \mathcal{T}_1$;
2. for all $n \in \mathbb{N}^*$, $F(\mathcal{T}_1, \mathcal{T}_n) = \mathcal{T}_1 \circ \mathcal{T}_n = \mathcal{T}_{n+1} \succ^* \mathcal{T}_{n+1}$;
3. for all $n \in \mathbb{N}^*$, \mathcal{T}_n contains a periodic loop. \square

Proposition 6.7.19

For all $n \geq 1$, \mathcal{T}_n contains a loop.

Proposition 6.7.20

Robinson tileset is provably robust.

Proof. We prove that the three items of Definition 6.7.13 are satisfied: define $\mathcal{T}_1 := \mathcal{T}_{Robi}$ and $\mathcal{T}_n := H_n$. The first two items follow from Theorem 6.7.7, considering the pattern $F(H_n, \mathcal{T}_1) = H_n \circ \mathcal{T}_1 \circ H_n$. The last item follows from Proposition 6.7.19. \square

Corollary 6.7.21

Robinson's tileset admits a tiling.

Proposition 6.7.22 (Non-provable robustness for valid but non-provable formulas)

Example 4.1.13 involves the existence of some valid sentence of arithmetic whose validity is non-provable. This is a necessary condition.

Proof. If a Turing machine M is non-provably robust, it does not terminate but there is no proof of it on some input w . We can always consider the arithmetic formula $\psi = \gamma(m, w)$ which states that the machine M is not terminating. It is true, but we have no proof of ϕ by hypothesis. In other words, we have some explicit examples of the formula, namely ψ that is valid but not provable. Hence, non-provably robustness necessarily involves some valid formula but non-provable formulas. \square

6.8 Chapter Conclusion

In this chapter, we extended Chapter 4 from robustness for reachability questions to the robustness of invariance properties by adopting a more topological point of view.

By searching in parallel for a proof of non-membership and a finite abstraction yielding a proof of membership (i.e. a c.e and co-c.e language is decidable), we obtain a proper notion of robustness. We now have that a robust system is decidable. This also means that decidability holds unless non-robustness is involved. Of course, there is no free-lunch phenomenon: robustness is undecidable.

We applied all famous common examples are robust:

Theorem 6.8.1

Robust tilesets have a decidable language (domino problem).

The main difference with Chapter 4 is that we generalised the statements to more topological properties, allowing us to talk about more frameworks.

The discussion in Section 6.5.1 can be interpreted that reasoning on finite abstractions such as Figure 6.5 can be seen as a sound and complete method to prove that a given subshift can tile \mathbb{Z}^d , and even aperiodic tiling. It explains that we found similar phenomena of discussions of graphs like [JR21]. The existence of an isomorphism invariant here can be interpreted as the fact that the tiling is substitutive in some particular weak form.

Jeandel-Rao's tileset \mathcal{T} (Section 6.6.2) arose from a systematic computer-assisted proof of tilesets by considering an increasing number of tiles [JR21]. They explain that *it was very easy for a computer to produce the transducer for \mathcal{T}^k , even for large values of k ($k \sim 1000$). In contrast, for almost all other tilesets, we were not able to reach even $k = 30$.* The existence of an invariant seems closely related to their observation that *“This suggested this tileset had some particular structure”*. We somehow provide arguments that the involved “structure” is the existence of an isomorphism invariant.

We prove additional notions of robustness, regarding the properties of the transducers.

Conclusion

-Once you make it to the top of the mountain, what's left for you but lightning?
-Wait, is the lightning a good thing or a bad thing?
-Depends whether you're ready for it or not.
Ted Lasso, season 3, episode 10

In this thesis, we aimed to study the complexity of analogue models of computation. In particular, we were motivated to have a better understanding of time and space in those models. Doing so, we wanted to see if we could compute more and faster in analogue models than in classical digital models. We mainly focused in this work on notions to measure resources like time and space for analogue computations.

For that purpose, we studied time and space for analogue models of computation, including continuous-time and discrete-time models, and characterised them with different types of ODEs. We extended time characterisations to a discrete-time framework, with discrete ODEs.

For continuous-time models, such as the GPAC, it was known that time complexity was related to the length of solutions, summarised by the motto **time = length of solutions**. The question of how to measure space complexity (memory) was still open. It was known that in the general case, dynamical systems could lead to undecidability. However, we proved that decidability holds for numerically stable systems. For such systems, one of the main results of our work is that space corresponds to precision. Hence, we obtain a simple motto for measuring space complexity: **space = precision**.

While doing so, we proposed an unusual way to solve ODEs: we solve them recursively, as inspired by the proof of the Savitch theorem. We proved the link between polynomial space and polynomial precision in the solution of the ODE. The advantage is that continuous ODEs might be more natural than discrete ODEs, as they are studied in more fields of science. To our knowledge, it is the first time we have such a measure for space in analogue models of computation.

These works also leads to drawing more precisely the border between what is decidable and what is not in general dynamical systems. Considering that robustness is unsensitivity to arbitrarily small perturbations, we obtain a whole set of results demonstrating that decidability holds for both reachability and invariance properties of dynamical systems, unless non-robust systems are considered. These results provide a unifying theory explaining various observations of the literature. For example, we related our theory to concepts such as delta-decision in logic, or to various other characterisations of this concept of robustness. Furthermore, we establish that space and time complexity are directly related to the tolerated level of robustness.

We proved other points of view of classes over the reals, to make them more usable

in other fields of science. All of our results heavily rely on different notions of *robustness* and *ODEs*. These works prove that it is possible to program with ODEs and to ensure we can do it efficiently. Considering the very wide use of ODEs in many fields of science, we believe that our results open the way to use tools from computer science, and complexity theory in many fields of applied science. It also links mathematics, computer science and physics, paving the way to a common and interdisciplinary research goal.

Let us start by reviewing what we studied and proved in the different chapters.

Chapter 3 was dedicated to giving algebraic characterisations of polynomial time in the context of computable analysis. Such characterisations existed for continuous-time systems and functions in $\mathbb{N}^{\mathbb{N}}$. Here, we proved characterisations of real sequences (Theorem 3.1.6) and then for real functions (Theorem 3.2.3), using discrete ODEs.

In Chapter 4, we studied notions of the robustness of dynamical systems leading to the decidability of the reachability problem: we prove that if robustness is defined as insensitivity to an arbitrarily small perturbation, the decidability for reachability holds. We showed that decidability is closely related to robustness, with various definitions of what it means. In other words, undecidability does not hold for robust systems. Also, we proved that complexity is directly related to the level of robustness. Doing so, we proved **PSPACE**-membership under some additional hypotheses (Theorem 4.2.25), including properly quantifying the perturbation. We showed **PTIME**-membership for another type of perturbation related to length (Theorem 4.5.5).

We came back to algebraic characterisations of complexity classes in Chapter 5. We algebraically characterised polynomial space in the framework of computable analysis using discrete ODEs (Theorem 5.1.6) and continuous ODEs on continuous-time systems (Theorem 5.2.2). Using all the previous characterisations and the **PSPACE**-membership of the reachability relation for polynomially robust dynamical systems, we proved that it is **PSPACE**-hard, leading to a **PSPACE**-completeness results (Theorem 5.3.3).

Finally, in Chapter 6, we generalised several robustness results and applied them to tiling theory. We extend the results of Chapter 4 from reachability to invariance property by adopting a topological point of view. It allowed us to give topological explanations of the results of Chapter 4. We also applied this to various application fields, including sub shifts, and tilings, providing some explanations of phenomena observed in literature.

We proved that, for robust tilesets, the tiling problem becomes decidable. We applied it to several well-known tilesets, like the Robison tileset (Section 6.5). The framework we develop here unifies the existing proofs regarding the decidability of some tilings.

In the short term, several interesting questions remain:

- It would be interesting to see if the algebras we have in Chapters 3 and 5 are minimal. For example, the division by 3 might not be necessary. Similarly, the cosine might not be needed in $\overline{\mathbb{RCD}}$, as it is the solution of simply-stated ODEs.
- We could also relate more precisely the different notions of robustness we studied here (e.g. robustness and pattern-robustness), to obtain a more unifying theory on the other concepts we have.
- We have **FPTIME** and **FSPACE** for functions over the reals. We could study more thoroughly the whole polynomial hierarchy in the framework of computable analysis, for example the classes in-between polynomial time and polynomial space.

There are several tracks we would like to explore in the long term:

- We proved characterisations of **FPTIME** and **FPSPACE** for functions over the reals $((\mathbb{R}^m)^{\mathbb{R}^n})$ for $n, m \in \mathbb{N}$ using ordinary differential equations (ODE) in Chapters 3 and 5. It would be interesting to study the computation of more general concepts of functions, namely distributions. This would imply developing a model in computable analysis for solving partial differential equations using distributions, following [Sel22].
- We would like to relate our algebraic characterisations of functions over the reals, stated in the framework of computable analysis, to algebraic characterisations of complexity classes over the reals defined with circuits, thus closer to the BSS model. Also, we are interested in having algebraic characterisations of classes that are less than polynomial, based on [KO14]. Eventually, we would be interested in adding continuous ODEs in circuit complexity, based on computable analysis and on [KO14].
- Additionally, it would be interesting to look at probabilistic complexity classes in the framework of computable analysis. We use probabilities in computer science for many problems, including for the training data in neural networks and in quantum computing. An interesting application is also parallel algorithmics and probabilistic algorithmics. Having solid computability and complexity results in those areas would prove the efficiency and feasibility of such algorithms. We would prove algebraic characterisations of probabilistic complexity classes, such as **BPP**, but that depends on the probability distribution we are considering.
- A project would be to prove complexity results for neural network training in the framework of computable analysis. There already exist theorems regarding that subject in the Existential Theory of the Reals (ETR) in [HHKV24]. They prove that the training of neural networks is $\exists\mathbb{R}$ -hard. Neural Networks (NN) are a particular kind of circuit. Under some restrictions regarding computability and feasibility, we want to prove that the neural network training is **NP**-complete. Namely, we could think about a concept of robustness for NN such that the training is **NP**-complete and not just $\exists\mathbb{R}$ -hard.

Bibliography

- [AB01] Eugene Asarin and Ahmed Bouajjani. Perturbed Turing machines and hybrid systems. In *Proceedings of the 16th Annual IEEE Symposium on Logic in Computer Science (LICS-01)*, pages 269–278, Los Alamitos, CA, June 16–19 2001. IEEE Computer Society Press.
- [AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, USA, 1st edition, 2009.
- [Abe71] Oliver Aberth. The failure in computable analysis of a classical existence theorem for differential equations. *Proceedings of the American Mathematical Society*, 30:151–156, 1971.
- [AMJ24] Erika Abraham and Manuel Mazo Jr, editors. *Proceedings of the 27th ACM International Conference on Hybrid Systems: Computation and Control*, 2024.
- [AMP95] Eugene Asarin, Oded Maler, and Amir Pnueli. Reachability analysis of dynamical systems having piecewise-constant derivatives. *Theoretical Computer Science*, 138(1):35–65, February 1995.
- [Aub91] Jean-Pierre Aubin. *Viability Theory*. Systems & Control: Foundations & Applications. Springer, 1991.
- [BB22] Manon Blanc and Olivier Bournez. A characterization of polynomial time computable functions from the integers to the reals using discrete ordinary differential equations. In Jérôme Durand-Lose and György Vaszil, editors, *Machines, Computations, and Universality - 9th International Conference, MCU 2022, Debrecen, Hungary, August 31 - September 2, 2022, Proceedings*, volume 13419 of *Lecture Notes in Computer Science*, pages 58–74. Springer, 2022.
- [BB23] Manon Blanc and Olivier Bournez. A characterisation of functions computable in polynomial time and space over the reals with discrete ordinary differential equations: Simulation of turing machines with analytic discrete odes. In Jérôme Leroux, Sylvain Lombardy, and David Peleg, editors, *48th International Symposium on Mathematical Foundations of Computer Science, MFCS 2023, August 28 to September 1, 2023, Bordeaux, France*, volume 272 of *LIPICs*, pages 21:1–21:15. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

- [BB24a] Manon Blanc and Olivier Bournez. A characterization of polynomial time computable functions from the integers to the reals using discrete ordinary differential equations. *International Journal of Foundations of Computer Science*, 0(0):1–28, 2024.
- [BB24b] Manon Blanc and Olivier Bournez. The complexity of computing in continuous time: Space complexity is precision. In Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson, editors, *51st International Colloquium on Automata, Languages, and Programming, ICALP 2024, July 8-12, 2024, Tallinn, Estonia*, volume 297 of *LIPIcs*, pages 129:1–129:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.
- [BB24c] Manon Blanc and Olivier Bournez. Quantifying the robustness of dynamical systems. relating time and space to length and precision. In Aniello Murano and Alexandra Silva, editors, *32nd EACSL Annual Conference on Computer Science Logic, CSL 2024, February 19-23, 2024, Naples, Italy*, volume 288 of *LIPIcs*, pages 17:1–17:20, Naples, Italy, February 2024. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.
- [BB25] Manon Blanc and Olivier Bournez. Simulation of turing machines with analytic discrete odes: Polynomial-time and space over the reals characterised with discrete ordinary differential equations. *Journal of Logic and Analysis*, 17,FDS5:1–42, February 2025.
- [BBEP10] Marie-Pierre Béal, Jean Berstel, Søren Eilers, and Dominique Perrin. Symbolic dynamics, 2010.
- [BBKT01] Vincent D. Blondel, Olivier Bournez, Pascal Koiran, and John Tsitsiklis. The stability of saturated linear dynamical systems is undecidable. *Journal of Computer and System Science*, 62(3):442–462, May 2001.
- [BC92] Stephen Bellantoni and Stephen Cook. A new recursion-theoretic characterization of the poly-time functions. *Computational Complexity*, 2(2):97–110, 1992.
- [BCD23] Olivier Bournez, Johanne Cohen, and Valentin Dardilhac. On the δ -decidability of decision problems for neural network questions. In *Computability, Continuity, Constructivity - from Logic to Algorithms CCC'23*, 2023.
- [BCdNM03] Olivier Bournez, Felipe Cucker, Paulin Jacobé de Naurois, and Jean-Yves Marion. Computability over an arbitrary structure. sequential and parallel polynomial time. In Andrew D. Gordon, editor, *Foundations of Software Science and Computational Structures, 6th International Conference (FOSSACS'2003)*, volume 2620 of *Lecture Notes in Computer Science*, pages 185–199, Warsaw, 2003. Springer.
- [BCdNM05] Olivier Bournez, Felipe Cucker, Paulin Jacobé de Naurois, and Jean-Yves Marion. Implicit complexity over an arbitrary structure: Sequential and parallel polynomial time. *Journal of Logic and Computation*, 15(1):41–58, 2005.

- [BCdNM06] Olivier Bournez, Felipe Cucker, Paulin Jacobé de Naurois, and Jean-Yves Marion. Implicit complexity over an arbitrary structure: Quantifier alternations. *Information and Computation*, 202(2):210–230, February 2006.
- [BCGSH07] Olivier Bournez, Manuel L. Campagnolo, Daniel Graça, and Emmanuel S. Hainry. Polynomial differential equations compute all real computable functions on computable compact intervals. *Journal of Complexity*, 23(3):317–335, 2007.
- [BCSS98] Lenore Blum, Felipe Cucker, Michael Shub, and Steve Smale. *Complexity and real computation*. Springer, 1998.
- [BD19] Olivier Bournez and Arnaud Durand. Recursion schemes, discrete differential equations and characterization of polynomial time computation. In Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen, editors, *44th Int Symposium on Mathematical Foundations of Computer Science, MFCS*, volume 138 of *LIPICs*, pages 23:1–23:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.
- [BD23] Olivier Bournez and Arnaud Durand. A characterization of functions over the integers computable in polynomial time using discrete ordinary differential equations. *Computational Complexity*, 32(2):7, 2023.
- [Ber66] R. Berger. The undecidability of the domino problem. *Memoirs American Mathematical Society*, 66, 1966.
- [BGGP23] Olivier Bournez, Riccardo Gozzi, Daniel S Graça, and Amaury Pouly. A continuous characterization of PSPACE using polynomial ordinary differential equations. *Journal of Complexity*, 77:101755, august 2023.
- [BGH10] Olivier Bournez, Daniel S. Graça, and Emmanuel Hainry. Robust computations with dynamical systems. In *Mathematical Foundations of Computer Science, MFCS’2010*, volume 6281 of *Lecture Notes in Computer Science*, pages 198–208. Springer, 2010.
- [BGH11] Olivier Bournez, Walid Gomaa, and Emmanuel Hainry. Algebraic characterizations of complexity-theoretic classes of real functions. *IJUC*, 7(5):331–351, 2011.
- [BGP12] Olivier Bournez, Daniel S Graça, and Amaury Pouly. On the complexity of solving initial value problems. In *Proceedings of the 37th International Symposium on Symbolic and Algebraic Computation*, pages 115–121, 2012.
- [BGP16a] Olivier Bournez, Daniel Silva Graça, and Amaury Pouly. Computing with polynomial ordinary differential equations. *Journal of Complexity*, 36:106–140, 2016.
- [BGP16b] Olivier Bournez, Daniel Silva Graça, and Amaury Pouly. Polynomial time corresponds to solutions of polynomial ordinary differential equations of polynomial length: The general purpose analog computer and computable analysis are two efficiently equivalent models of computations. In Ioannis

- Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPICs*, pages 109:1–109:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.
- [BGP17] Olivier Bournez, Daniel S. Graça, and Amaury Pouly. Polynomial Time corresponds to Solutions of Polynomial Ordinary Differential Equations of Polynomial Length. *Journal of the ACM*, 64(6):38:1–38:76, 2017.
- [BHW08] Vasco Brattka, Peter Hertling, and Klaus Weihrauch. A tutorial on computable analysis. In *New computational paradigms*, pages 425–491. Springer, 2008.
- [BP17] Olivier Bournez and Amaury Pouly. A universal ordinary differential equation. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPICs*, pages 116:1–116:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.
- [BP21] Olivier Bournez and Amaury Pouly. A survey on analog models of computation. In *Handbook of Computability and Complexity in Analysis*, pages 173–226. Springer, 2021.
- [Bra95] M. S. Branicky. Universal computation and other capabilities of hybrid and continuous dynamical systems. *Theoretical Computer Science*, 138(1):67–100, 6 feb 1995.
- [Bra96] Vasco Brattka. Recursive characterization of computable real-valued functions and relations. *Theoretical Computer Science*, 162(1):45–77, 1996.
- [Bra05] Mark Braverman. Hyperbolic Julia sets are poly-time computable. *Electronic Notes in Theoretical Computer Science*, 120:17–30, 2005.
- [BRS15] Mark Braverman, Cristobal Rojas, and Jon Schneider. Tight space-noise tradeoffs in computing the ergodic measure. *Sbornik: Mathematics*, 208, 08 2015.
- [Bru19] Henk Bruin. Vo special topics in stochastics: Symbolic dynamic, 2019.
- [BSS89] Lenore Blum, Mike Shub, and Steve Smale. On a theory of computation and complexity over the real numbers; NP completeness, recursive functions and universal machines. *Bulletin of the American Mathematical Society*, 21(1):1–46, jul 1989.
- [CG08] Pieter Collins and Daniel S Graça. Effective computability of solutions of ordinary differential equations the thousand monkeys approach. *Electronic Notes in Theoretical Computer Science*, 221:103–114, 2008.

- [CG09] Pieter Collins and Daniel S. Graça. Effective Computability of Solutions of Differential Inclusions The Ten Thousand Monkeys Approach. *Journal of Universal Computer Science*, 15(6):1162–1185, 2009.
- [Chu36] Alonzo Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58:345, 1936.
- [CK02] Peter Clote and Evangelos Kranakis. *Boolean Functions and Computation Models*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2002.
- [Cla24] Laurent Claessens–Donadello. *Le Frido*. Free Software Foundation, 2024.
- [Clo98] P. Clote. Computational models and function algebras. In Edward R. Griffor, editor, *Handbook of Computability Theory*, pages 589–681. North-Holland, Amsterdam, 1998.
- [CMC00] Manuel L. Campagnolo, Cristopher Moore, and José Félix Costa. Iteration, inequalities, and differentiability in analog computers. *Journal of Complexity*, 16(4):642–660, 2000.
- [CMPS23] Robert Cardona, Eva Miranda, and Daniel Peralta-Salas. Computability and beltrami fields in euclidean space. *Journal de Mathématiques Pures et Appliquées*, 169:50–81, 2023.
- [Cob62] Alan Cobham. The intrinsic computational difficulty of functions. In Y. Bar-Hillel, editor, *Proceedings of the International Conference on Logic, Methodology, and Philosophy of Science*, pages 24–30. North-Holland, Amsterdam, 1962.
- [Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, page 151–158, New York, NY, USA, 1971. Association for Computing Machinery.
- [CR24] Jordan Cotler and Semon Rezchikov. Computational dynamical systems. In *Proceedings - 2024 IEEE 65th Annual Symposium on Foundations of Computer Science, FOCS 2024*, Proceedings - Annual IEEE Symposium on Foundations of Computer Science, FOCS, pages 166–202, United States, 2024. IEEE Computer Society. Publisher Copyright: © 2024 IEEE.; 65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024 ; Conference date: 27-10-2024 Through 30-10-2024.
- [CRBD18] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, pages 6571–6583, 2018.
- [Dem96] J.-P. Demailly. *Analyse Numérique et Équations Différentielles*. Presses Universitaires de Grenoble, 1996.
- [Dev89] R. L. Devaney. *An Introduction to Chaotic Dynamical Systems*. Addison-Wesley, 2nd edition, 1989.

- [FBF⁺02] N Pytheas Fogg, Val     Berth  , S  bastien Ferenczi, Christian Mauduit, and Anne Siegel. *Substitutions in Dynamics, Arithmetics and Combinatorics*. Springer Berlin Heidelberg, 2002.
- [Fer21] Thomas Fernique. Robinson tilings. Course given at Moscow University, 2021.
- [FGBP17] Fran  ois Fages, Guillaume Le Guludec, Olivier Bournez, and Amaury Pouly. Strong turing completeness of continuous chemical reaction networks and compilation of mixed analog-digital programs. In J  r  me Feret and Heinz Koepl, editors, *Computational Methods in Systems Biology - 15th International Conference, CMSB 2017, Darmstadt, Germany, September 27-29, 2017, Proceedings*, volume 10545 of *Lecture Notes in Computer Science*, pages 108–127. Springer, 2017. CMSB’2017 Best Paper Award.
- [FGH14] Hugo F  r    , Walid Gomaa, and Mathieu Hoyrup. Analytical properties of resource-bounded real functionals. *Journal of Complexity*, 30(5):647–671, 2014.
- [Fr  99] Martin Fr  nzle. Analysis of hybrid systems: An ounce of realism can save an infinity of states. In J  rg Flum and Mario Rodr  guez-Artalejo, editors, *Computer Science Logic, 13th International Workshop, CSL ’99, 8th Annual Conference of the EACSL, Madrid, Spain, September 20-25, 1999, Proceedings*, volume 1683 of *Lecture Notes in Computer Science*, pages 126–140. Springer, 1999.
- [GAC12] Sicun Gao, Jeremy Avigad, and Edmund M Clarke. Delta-decidability over the reals. In *Logic in Computer Science (LICS), 2012 27th Annual IEEE Symposium on*, pages 305–314. IEEE, 2012.
- [Gan80] Robin Gandy. Church’s thesis and principles for mechanisms. In Jon Barwise, H. Jerome Keisler, and Kenneth Kunen, editors, *The Kleene Symposium*, volume 101 of *Studies in Logic and the Foundations of Mathematics*, pages 123–148. Elsevier, 1980.
- [GC03] Daniel S. Gra  a and Jos   F  lix Costa. Analog computers and recursive functions over the reals. *jcomp*, 19(5):644–664, 2003.
- [GCB05] Daniel S. Gra  a, Manuel L. Campagnolo, and Jorge Buescu. Robust simulations of Turing machines with analytic maps and flows. In B. Cooper, B. Loewe, and L. Torenvliet, editors, *Proceedings of CiE’05, New Computational Paradigms*, volume 3526 of *Lecture Notes in Computer Science*, pages 169–179. Springer-Verlag, 2005.
- [Gel71] Aleksandr Gelfond. *Calculus of finite differences*. Hindustan Publ. Corp., 1971.
- [GMR08] Marco Gaboardi, Jean-Yves Marion, and Simona Ronchi Della Rocca. A logical account of pspace. In George C. Necula and Philip Wadler, editors, *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles*

of Programming Languages, POPL 2008, San Francisco, California, USA, January 7-12, 2008, pages 121–131. ACM, 2008.

- [Göd31] Kurt Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für Mathematik und Physik*, 38:173–198, 1931. English translation in Martin Davis. *The Undecidable: Basic Papers on Undecidable Propositions, Unsolvable Problems and Computable Functions*. Raven Press, 1965.
- [Goz22] Riccardo Gozzi. *Analog Characterization of Complexity Classes*. PhD thesis, Instituto Superior Técnico, Lisbon, Portugal and University of Algarve, Faro, Portugal, 2022.
- [Gra07] Daniel S. Graça. *Computability with Polynomial Differential Equations*. PhD thesis, Instituto Superior Técnico, 2007.
- [GZ18] Daniel S. Graça and Ning Zhong. *Handbook of Computability and Complexity in Analysis*, chapter Computability of Differential Equations. Springer., 2018.
- [GZB06] Daniel S. Graça, N. Zhong, and J. Buescu. Computability, noncomputability and undecidability of maximal intervals of IVPs. *Transactions of the American Mathematical Society*, 2006. To appear.
- [Har64] Philip Hartman. *Ordinary Differential Equations*. John Wiley and Sons, 1964.
- [HHKV24] Teemu Hankala, Miika Hannula, Juha Kontinen, and Jonni Virtema. Complexity of neural network training and ETR: extensions with effectively continuous functions. In Michael J. Wooldridge, Jennifer G. Dy, and Sriram Natarajan, editors, *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2024, February 20-27, 2024, Vancouver, Canada*, pages 12278–12285. AAAI Press, 2024.
- [HKPV98] Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What’s decidable about hybrid automata? *Journal of Computer and System Sciences*, 57(1):94–124, August 1998.
- [HR00] Thomas A. Henzinger and Jean-François Raskin. Robust undecidability of timed and hybrid systems. In Nancy A. Lynch and Bruce H. Krogh, editors, *Hybrid Systems: Computation and Control, Third International Workshop, HSCC 2000, Pittsburgh, PA, USA, March 23-25, 2000, Proceedings*, volume 1790 of *Lecture Notes in Computer Science*, pages 145–159. Springer, 2000.
- [HSD03] Morris W. Hirsch, Stephen Smale, and Robert Devaney. *Differential Equations, Dynamical Systems, and an Introduction to Chaos*. Elsevier Academic Press, 2003.

- [Imm88] N. Immerman. Nondeterministic space is closed under complementation. In *Structure in Complexity Theory Conference, 1988. Proceedings., Third Annual*, pages 112–115. IEEE, 1988.
- [JR21] Emmanuel Jeandel and Michael Rao. An aperiodic set of 11 wang tiles. *Advances in Combinatorics*, 2021.
- [JV20] Emmanuel Jeandel and Pascal Vanier. The undecidability of the domino problem. In *Substitution and Tiling Dynamics: Introduction to Self-inducing Structures: CIRM Jean-Morlet Chair, Fall 2017*, pages 293–357. Springer, 2020.
- [Kar72] Richard Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, 40:85–103, 01 1972.
- [Kar07] Jarkko Kari. The Tiling Problem Revisited. In *MCU*, pages 72–79, 2007.
- [Kaw09] Akitoshi Kawamura. Lipschitz continuous ordinary differential equations are polynomial-space complete. In *2009 24th Annual IEEE Conference on Computational Complexity*, pages 149–160. IEEE, 2009.
- [KC12] Akitoshi Kawamura and Stephen A. Cook. Complexity theory for operators in analysis. *ACM Trans. Comput. Theory*, 4(2):5:1–5:24, 2012.
- [KCG94] Pascal Koiran, Michel Cosnard, and Max Garzon. Computability with low-dimensional dynamical systems. *Theoretical Computer Science*, 132(1-2):113–128, September 1994.
- [Kid22] Patrick Kidger. On neural differential equations. *CoRR*, abs/2202.02435, 2022.
- [KK06] Igor Kluváněk and Greg Knowles. The bang-bang principle. In *Mathematical Control Theory: Proceedings, Canberra, Australia, August 23–September 2, 1977*, pages 138–151. Springer, 2006.
- [Kle36] Stephen C. Kleene. General recursive functions of natural numbers. *Mathematical Annals*, 112:727–742, 1936. Reprinted in Martin Davis. *The Undecidable: Basic Papers on Undecidable Propositions, Unsolvability Problems and Computable Functions*. Raven Press, 1965.
- [Kle38] Stephen C. Kleene. On notation for ordinal numbers. *The Journal of Symbolic Logic*, 3(4):150–155, 1938.
- [Kle43] Stephen Cole Kleene. Recursive predicates and quantifiers. *Transactions of the American Mathematical Society*, 53:41–73, 1943.
- [KMRZ15] Akitoshi Kawamura, Norbert Müller, Carsten Rösnick, and Martin Ziegler. Computational benefit of smoothness: Parameterized bit-complexity of numerical operators on analytic functions and gevrey’s hierarchy. *Journal of Complexity*, 31(5):689–714, 2015.

- [Ko83] Ker-I Ko. On the computational complexity of ordinary differential equations. *Information and Control*, 58(1-3):157–194, jul/aug/sep 1983.
- [Ko91] Ker-I Ko. *Complexity theory of real functions*, volume 3 of *Progress in theoretical computer science*. Birkhäuser, Boston, 1991.
- [KO14] Akitoshi Kawamura and Hiroyuki Ota. Small complexity classes for computable analysis. In Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik, editors, *Mathematical Foundations of Computer Science 2014 - 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part II*, volume 8635 of *Lecture Notes in Computer Science*, pages 432–444. Springer, 2014.
- [KORZ14] Akitoshi Kawamura, Hiroyuki Ota, Carsten Rösnick, and Martin Ziegler. Computational complexity of smooth differential equations. *Logical Methods in Computer Science*, 10, 2014.
- [Koz97] D. Kozen. *Automata and computability*. Springer Verlag, 1997.
- [KST18] Akitoshi Kawamura, Florian Steinberg, and Holger Thies. Parameterized complexity for uniform operators on multidimensional analytic functions and ode solving. In *International Workshop on Logic, Language, Information, and Computation*, pages 223–236. Springer, 2018.
- [Lei94] Daniel Leivant. Predicative recurrence and computational complexity I: Word recurrence and poly-time. In Peter Clote and Jeffery Remmel, editors, *Feasible Mathematics II*, pages 320–343. Birkhäuser, 1994.
- [Lei95] D. Leivant. Intrinsic theories and computational complexity. In *LCC’94*, number 960 in *Lecture Notes in Computer Science*, pages 177–194, 1995.
- [Lev73] Leonid A. Levin. Universal sequential search problems. *Problemy Peredachi Informatsii*, 9:115–116, 1973.
- [LM93] Daniel Leivant and Jean-Yves Marion. Lambda calculus characterizations of Poly-Time. *Fundamenta Informatica*, 19(1,2):167,184, 1993.
- [LM95] Daniel Leivant and Jean-Yves Marion. Ramified recurrence and computational complexity II: substitution and poly-space. In L. Pacholski and J. Tiuryn, editors, *Computer Science Logic, 8th Workshop, CSL’94*, volume 933 of *Lecture Notes in Computer Science*, pages 369–380, Kazimierz, Poland, 1995. Springer.
- [Lor63] E. N. Lorenz. Deterministic nonperiodic flow. *Journal of the Atmospheric Science*, 20:130–141, 1963.
- [Lyg04] John Lygeros. Lecture notes on hybrid systems. In *Notes for an ENSIETA workshop*, 2004.
- [Mal02] Oded Maler. Control from computer science. *Annual Reviews in Control*, 26(2):175–187, 2002.

- [Mil70] Webb Miller. Recursive function theory and numerical analysis. *Journal of Computer and System Sciences*, 4(5):465–472, oct 1970.
- [Mil85] John Milnor. On the concept of attractor. *Communications in Mathematical Physics*, 99:177–195, 1985.
- [Moo91] Cristopher Moore. Generalized shifts: unpredictability and undecidability in dynamical systems. *Nonlinearity*, 4(3):199–230, 1991.
- [Mos47] Andrzej Włodzimierz Mostowski. On definable sets of positive integers. *Fundamenta Mathematicae*, 34:81–112, 1947.
- [MP08] Jean-Yves Marion and Romain Péchoux. Characterizations of polynomial complexity classes with a better intensionality. In Sergio Antoy and Elvira Albert, editors, *Proceedings of the 10th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, July 15-17, 2008, Valencia, Spain*, pages 79–88. ACM, 2008.
- [Mus00] E. Muselli. Upper and lower semicontinuity for set-valued mappings involving constraints. *Journal of Optimization Theory and Applications*, 106:527–550, 2000.
- [Neu21] Eike Neumann. Decision problems for linear recurrences involving arbitrary real numbers. *Logical Methods in Computer Science*, 17, 2021.
- [Neu23] Eike Neumann. On the complexity of robust eventual inequality testing for C-finite functions. In *International Conference on Reachability Problems*, pages 98–112. Springer, 2023.
- [NRTY21] Keng Meng Ng, Nazanin R Tavana, and Yue Yang. A recursion theoretic foundation of computation over real numbers. *Journal of Logic and Computation*, 31(7):1660–1689, 2021.
- [PBV95] A. Puri, V. Borkar, and P. Varaiya. Epsilon-approximation of differential inclusions. In *Proceedings of the 34th IEEE Conference on Decision and Control*, pages 2892–2897, 1995.
- [PER79] Marian Boykan Pour-El and J. Ian Richards. A computable ordinary differential equation which possesses no computable solution. *Annals of Mathematical Logic*, 17:61–90, 1979.
- [PG16] Amaury Pouly and Daniel Silva Graça. Computational complexity of solving polynomial differential equations over unbounded domains. *Theoretical Computer Science*, 626:67–82, 2016.
- [Pou15] Amaury Pouly. *Continuous models of computation: from computability to complexity*. PhD thesis, Ecole Polytechnique and Universidade Do Algarve, Defended on July 6, 2015. 2015. <https://pastel.archives-ouvertes.fr/tel-01223284>, Prix de Thèse de l’Ecole Polytechnique 2016, Ackermann Award 2017.

- [Pur00] Anuj Puri. Dynamical properties of timed automata. *Discrete Event Dynamic Systems*, 10:87–113, 2000.
- [Rat23] Stefan Ratschan. Deciding predicate logical theories of real-valued functions. In *Symposium on Mathematical Foundations of Computer Science (MFCS'2023)*, 2023.
- [Rob71] Raphael M Robinson. Undecidability and nonperiodicity for tilings of the plane. *Inventiones mathematicae*, 12:177–209, 1971.
- [RS23] Cristobal Rojas and Mathieu Sablik. On the algorithmic descriptive complexity of attractors in topological dynamics. *CoRR*, abs/2311.15234, 2023.
- [Rud87] W. Rudin. *Real and Complex Analysis, 3rd edition*. McGraw Hills, USA, March 1987.
- [Rud91] Walter Rudin. *Functional Analysis*. McGraw-Hill, 1991.
- [Ruo96] Keijo Ruohonen. An effective Cauchy-Peano existence theorem for unique solutions. *International Journal of Foundations of Computer Science*, 7(2):151–160, 1996.
- [Sab12] Mathieu Sablik. Pavages: du local au global, 2012.
- [Sel22] Svetlana Selivanova. *Computational Complexity of Classical Solutions of Partial Differential Equations*, pages 299–312. Springer, 06 2022.
- [Sha41] Claude E. Shannon. Mathematical theory of the differential analyser. *Journal of Mathematics and Physics MIT*, 20:337–354, 1941.
- [Sie99] Hava T. Siegelmann. *Neural Networks and Analog Computation - Beyond the Turing Limit*. Birkhauser, 1999.
- [Sip97] Michael Sipser. *Introduction to the Theory of Computation*. PWS Publishing Company, 1997.
- [SP94] Patrick Saint-Pierre. Approximation of the viability kernel. *Applied Mathematics and Optimization*, 29:187–209, 1994.
- [SS95] Hava T. Siegelmann and Eduardo D. Sontag. On the computational power of neural nets. *Journal of Computer and System Sciences*, 50(1):132–150, February 1995.
- [Str65] C. Strachey. An impossible program. *The Computer Journal*, 7(4):313–313, 01 1965.
- [SVV64] LM Sonneborn and FS Van Vleck. The bang-bang principle for linear control systems. *Journal of the Society for Industrial and Applied Mathematics, Series A: Control*, 2(2):151–159, 1964.
- [Sze88] R. Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta informatica*, 26(3):279–284, November 1988.

- [Tak01] Izumi Takeuti. Effective fixed point theorem over a non-computably separable metric space. In Jens Blanck, Vasco Brattka, and Peter Hertling, editors, *Computability and Complexity in Analysis*, pages 310–322, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.
- [Thi18] Holger Thies. *Uniform computational complexity of ordinary differential equations with applications to dynamical systems and exact real arithmetic*. PhD thesis, University of Tokyo, Graduate School of Arts and Sciences, 2018.
- [Tho72] David B. Thompson. Subrecursiveness: Machine-independent notions of computability in restricted time and storage. *Mathematical systems theory*, 6:3–15, 1972.
- [Tuc02] Warwick Tucker. A rigorous ODE solver and Smale’s 14th problem. *Foundations of Computational Mathematics*, 2:53–117, 2002.
- [Tur37] Alan M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265, 1937. Reprinted in Martin Davis. *The Undecidable: Basic Papers on Undecidable Propositions, Unsolvability Problems and Computable Functions*. Raven Press, 1965.
- [Ulm13] Bernd Ulmann. *Analog computing*. Walter de Gruyter, 2013.
- [Ulm20] Bernd Ulmann. *Analog and hybrid computer programming*. De Gruyter Oldenbourg, 2020.
- [Ver22] Veritasum. Future computers will be radically different (analog computing). Youtube video, 2022.
- [Wan61] Hao Wang. Proving theorems by pattern recognition - ii. *Bell System Technical Journal*, 40(1):1–41, January 1961.
- [Wei00] Klaus Weihrauch. *Computable Analysis - An Introduction*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2000.
- [Win93] Glynn Winskel. *The formal semantics of programming languages: an introduction*. MIT press, 1993.

Titre : Systèmes à Temps Discret et à Temps Continu: Relier la Complexité à la Robustesse, la Longueur, la Précision

Mots clés : Complexité, Analyse Calculable, Systèmes Dynamiques

Résumé : Le monde physique qui nous entoure est continu et décrit par des équations différentielles ordinaires (EDO). Mais ce n'est pas le modèle utilisé dans les ordinateurs actuels: les composants sont contraints de vivre dans un régime vrai/faux. Ainsi, les modèles de calcul, comme les machines de Turing, sont discrets: ils fonctionnent sur des mots d'un alphabet fini, avec des pas, c'est-à-dire un temps discret. Nous avons aussi des modèles mathématiques pour les machines analogiques, et les ordinateurs analogiques, comme le "General Purpose Analog Computer" proposé par Claude Shannon, qui peuvent être vus comme le pendant des machines de Turing pour le monde analogique. Nos travaux portent sur la puissance de calcul de tels modèles: comprendre si nous pourrions calculer plus et plus vite avec eux. Pour comparer les approches, nous avons besoin d'un moyen de mesurer le coût des calculs dans des contextes continus. Nous prouvons des caractérisations de classes de complexité pour les calculs utilisant des nombres réels et des modèles analogiques. En particulier, nous étendons les caractérisations de temps aux systèmes à temps dis-

crets avec des EDO discrètes pour couvrir la calculabilité et la complexité des fonctions sur les réels, alors que cela n'avait été fait que pour les problèmes de décision. Nous prouvons qu'il existe une mesure robuste pour les systèmes à temps continus qui correspond à l'espace pour les modèles digitaux: l'espace, soit la mémoire utilisée par un modèle comme la machine de Turing, correspond à la précision pour les systèmes à temps continu.

Un autre axe de recherche porte sur la robustesse des systèmes dynamiques. Nous prouvons que l'indécidabilité n'arrive pas pour des systèmes robustes et relient directement le niveau de robustesse à leur complexité. Dans le contexte des théories logiques sur les réels, des procédures de décision existent en se concentrant sur des propriétés insensibles à des perturbations arbitrairement petites. Nous proposons une théorie unifiée expliquant dans un cadre uniforme ces énoncés, établis dans des contextes différents. Nous appliquons ce principe aux systèmes dynamiques généraux, incluant un point de vue topologique, ainsi qu'aux pavages.

Title : Discrete-Time and Continuous-Time Systems over the Reals: Relating Complexity with Robustness, Length and Precision

Keywords : Complexity, Computable Analysis, Dynamical Systems

Abstract : The physical world around us is continuous and described by ordinary differential equations (ODEs). But this is not the model used in current computers: the components are forced to live in a true/false regime. Thus, the models of computation, such as Turing machines, are discrete: they operate on words of a finite alphabet, with steps, so in discrete time. However, we also have mathematical models for analogue machines, and analogue computers, such as the "General Purpose Analog Computer" proposed by Claude Shannon, which can be seen as the counterpart of Turing machines for the analogue world. Our work focuses on the computing power of such models: understanding whether we could compute more and faster with them. To compare approaches, we need a way to measure the cost of computations in continuous contexts. We prove complexity class characterisations for computations over real numbers and analogue models. In particular, we extend the time cha-

acterisations to discrete-time systems with discrete ODEs to cover computability and complexity of functions on the reals, whereas this had only been done for decision problems. We prove there exists a robust measure for continuous-time systems corresponding to space for digital models: space, that is, the memory used by a model like the Turing machine, corresponds to precision for continuous-time systems.

Another line of research concerns the robustness of dynamical systems. We show that undecidability does not occur for robust systems and directly relate the level of robustness to their complexity. In the context of logical theories of real numbers, decision procedures exist by focusing on properties insensitive to arbitrarily small perturbations. We propose a unified theory explaining these statements, from different contexts, in a uniform framework. We apply this principle to general dynamical systems, including a topological point of view, as well as to tilings.