Hamilton Institute

Maynooth University
National University of Ireland Maynooth

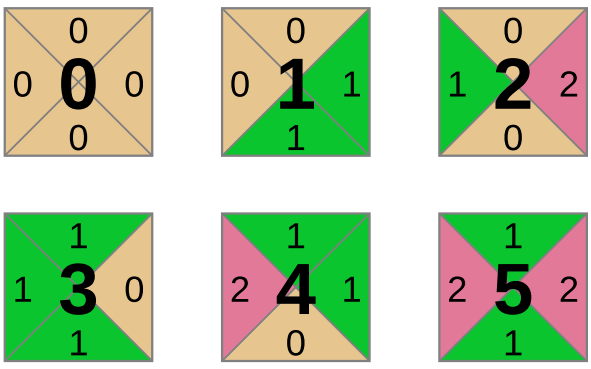
Six Tiles

From Collatz Sequences to Algorithmic DNA Origami

Tristan Stérin

Supervisor
Damien Woods

A thesis submitted in fulfillment of the requirements
for the Ph.D. degree in Computer Science



Hamilton Institute and Department of Computer Science
Maynooth University
Maynooth, Co. Kildare, Ireland

Submitted on October 31st 2022
Defended on June 28th 2023

In loving memory of my grandparents, Elsa, Jacques, Yvette and Raymond

Abstract

We tile a path from the theoretical world of Collatz sequences, which are fascinating, seemingly undecipherable mathematical objects, to the experimental wet lab world of *algorithmic DNA origami*, which is our proposed technique to run algorithms using self-assembling DNA nanostructures. The common thread throughout these seemingly unrelated worlds is the use of tile self-assembly models to reason about the problem at hand.

In the case of Collatz sequences (also known as $3x + 1$ sequences), we study 6 Wang tiles, which give this thesis its name, that assemble Collatz sequences in tilings. These Collatz tilings provide a microscope for studying Collatz sequences symbolically in many bases (base 2, 3, 6, and more) and allow us to interpret known results in our 2D tiling framework, as well as partially characterising the complexity of predicting Collatz iterations, Chapter 1.

We then use these 6 tiles to visualise a conjecture by Erdős on powers of two that we encode as the halting problem (from blank tape) of a 15-state 2-symbol Turing machine. This construction implies that knowing the busy beaver value BB(15) is at least as hard as solving Erdős' conjecture, which seems to put BB(15) out of reach since the conjecture has been open for decades, Chapter 2.

We go on to show that our 6 tiles can simulate arbitrary Boolean circuits in a model of tile assembly that we introduce: the *Maze-Walking Tile Assembly Model*. This is when our journey starts entering the realm of wet lab experiments, as we argue that our Maze-Walking TAM is suited to DNA nanostructure implementation, Chapter 3.

We make this claim concrete in Chapter 4 by introducing the Scaffolded DNA Computer, a thermodynamically favoured model of computation very close in spirit to the Maze-Walking TAM. In the wet lab we implement eight DNA-based programs for this computer (corresponding to over 100 DNA program executions), including a 7-bit adder running *within* a DNA origami, or, as we call it, an *algorithmic* DNA origami.

Résumé

Nous pavons un chemin entre le monde théorique des suites de Collatz, qui sont des objets mathématiques d'une complexité fascinante, et le monde des sciences expérimentales réalisées en laboratoire au travers des *origamis ADN algorithmiques*, la méthode que nous introduisons pour exécuter des algorithmes en utilisant des nanostructures ADN qui s'auto-assemblent. Le fil rouge qui relie ces mondes *a priori* complètement étrangers est l'utilisation des modèles d'auto-assemblage pour traiter les problèmes à résoudre.

Dans le cas des suites de Collatz (aussi connues sous le nom de suites de Syracuse, ou $3x + 1$), nous étudions 6 tuiles de Wang, d'où la thèse tire son nom, qui peuvent assembler les suites de Collatz en des pavages. Ces pavages de Collatz constituent un microscope pour étudier les suites de Collatz symboliquement dans plusieurs bases (base 2, 3, 6, et d'autres) et ils nous permettent aussi d'interpréter des résultats connus dans ce contexte de pavages 2D, ainsi que de caractériser partiellement la complexité de prédire les itérations de Collatz, Chapitre 1.

Ensuite, nous utilisons ces 6 tuiles pour visualiser une conjecture d'Erdős sur les puissances de deux, que l'on encode comme le problème de l'arrêt (depuis ruban vide) d'une machine de Turing à 15 états et 2 symboles. Cette construction implique que déterminer la valeur "busy beaver" BB(15) est aussi dur que de résoudre la conjecture d'Erdős, ce qui semble mettre BB(15) hors de portée de notre savoir étant donné que la conjecture est ouverte depuis plusieurs décennies, Chapitre 2.

Puis, nous montrons que nos 6 tuiles peuvent simuler n'importe quel circuit Booléen dans un modèle d'auto-assemblage que nous introduisons: le *modèle de tuiles auto-assemblantes marcheuses de labyrinthe* ("Maze-Walking Tile Assembly Model"). C'est ici que notre voyage commence son entrée dans le monde des sciences expérimentales réalisées en laboratoire, car nous défendons l'idée que notre modèle est particulièrement adapté pour être implémenté avec des nanostructures ADN, Chapitre 3.

Nous concrétisons cette idée dans le Chapitre 4 en introduisant *l'Ordinateur à Échafaudage ADN* ("Scaffolded DNA Computer"), un modèle thermodynamiquement favorable très proche du "Maze-Walking TAM" précédemment introduit. En laboratoire, nous implémentons huit programmes ADN pour cet ordinateur (correspondant à la réalisation de plus d'une centaine d'expériences), dont en particulier un additionneur 7 bits qui s'exécute *à l'intérieur* d'un origami ADN, ou plutôt, d'un origami ADN *algorithmique*.

Coimriú

Tílimid cosán ó shaol teoiriciúil na seicheamh Collatz, ar nithe iontacha agus matamaiticiúla atá dothuigthe de réir dealraimh iad, go dtí saol saotharlainne fliche eispéireasaí oragámaí DNA algartamaigh, arb é sin an teicníc atá beartaithe againn chun algartaim a rith ag baint úsáid as nanastruchtúir DNA fhéinchóimeála. An snáithe a nascann na saolta sin le chéile, saolta nach bhfuil aon bhaint acu lena chéile de réir dealraimh, ná úsáid a bhaint as samhlacha féinchóimeála tíleanna chun an fhadhb atá i gceist a réasúnú.

I gcás na seicheamh Collatz (ar a dtugtar seichimh 3x + 1 chomh maith), déanaimid staidéar ar 6 thíl Wang, a thugann an t-ainm don tráchtas seo, a bhailíonn seichimh Collatz i dtíleanna. Soláthraítear micreascóp leis na tíleanna Collatz sin chun staidéar a dhéanamh ar sheichimh Collatz ar bhonn siombalach in go leor bonn (bonn 2, 3, 6, agus níos mó) agus cuirtear ar ár gcumas leo torthaí aitheanta inár gcreat tíleála 2T a léirmhíniú, chomh maith leis an gcastacht a bhaineann le hathtriallta Collatz a thuar go páirteach, Caibidil 1.

Úsáidimid na 6 thíl sin ansin chun tuairimíocht ó Erdős a shamhlú ar chumhachtaí dhá cheann a ionchódaímid mar fhadhb stad (ó théip bhán) meaisín 15 staid 2 shiombail Turing. Tugtar le tuiscint leis an bhfoirgníocht sin go bhfuil sé ar a laghad chomh deacair céanna an béabhar gnóthach BB(15) a fháil amach agus atá sé Erdős' a réiteach, a bhfuil an chuma air go ndéantar BB(15) dorochtana leis ós rud é go bhfuil an tuairimíocht oscailte le fiche nó tríocha bliain anuas, Caibidil 2 .

Léirímid ansin gur féidir lenár 6 thíl ciorcaid threallacha Boole a insamhladh i samhail de chóimeáil tíleanna a thugaimid isteach: an Maze-Walking Tile Assembly Model. Is é sin nuair a thosaíonn ár dturas ag dul isteach i réimse na dturgnamh saotharlainne fliche, agus muid ag argóint go bhfuil ár TAM Maze-Walking oiriúnach do chur chun feidhme nanastruchtúr DNA, Caibidil 3.

Neartaímid an t-éileamh sin i gCaibidil 4 tríd an Ríomhaire DNA Scafláilte a thabhairt isteach, samhail theirmidinimiciúil ríomha atá an-chosúil de réir spride leis an Maze-Walking TAM. Sa tsaotharlann fhliuch cuirimid ocht gclár DNA-bhunaithe chun feidhme don ríomhaire seo (a chomhfhreagraíonn do bhreis is 100 cur i gcrích clár DNA), lena n-áirítear suimitheoir 7 ngiotán a ritheann laistigh d'oragámaí DNA, nó, oragámaí DNA algartamach, mar a thugaimid air.

Acknowledgments

I would like to thank my parents, Monique and Gérard Stérin for their unconditional love and support throughout the years. This degree goes to their commitment that we – my sister and I – always received the best education possible. They taught me, from a young age, the importance of asking questions, playing with ideas, and following intuitions. I also thank my sister, Tomoé, for her kind and supportive presence.

Damien, you liberated my approach to science. You've transformed what once was a big, heavy and intimidating canon into a creative, playful, free, exhilarating practice. You've always been there, through thick and thin, mentoring me. I've learnt a lot. But, more than everything, you trusted me tightrope-walking with no safety net during 4 years, and thanks to that trust, I trusted myself and made it to the other side. Thank you for the good craic.

Christopher, I could not have done it without you. You have supported me (et tu m'as supporté) each step of the way. These four years of study also correspond to us going from getting to know each other to being pillars in each other's life, and I will always cherish that. I love you. T'es ma vie. I also thank your family that has always welcomed me heartwarmingly and made me feel at home.

There are many scientists that I want to thank for the time and attention that they have spent with me. In particular, I would like to thank Jose Capco whose work [31] introduced me to the beauty of studying the Collatz process in binary, Olivier Rozier whose work on Collatz also has been of great influence to me and for all the time he spent discussing it with me, Matthew Cook for some of the most inspiring scientific conversations I have ever had and who introduced me to the Collatz tiles, helping conceiving Chapter 1 of this thesis, and Jarkko Kari for teaching me kindly, at the very beginning of my PhD, that we often re-discover results found by others and for his invitation to Finland and all the time he gave me in April 2022 with Johan Kopra. I would like to also specially thank Abeer Eshra for working together so hard on the algorithmic DNA Origami project (see Chapter 4). This project would not have come to light without her involvement and amazing experimental skills.

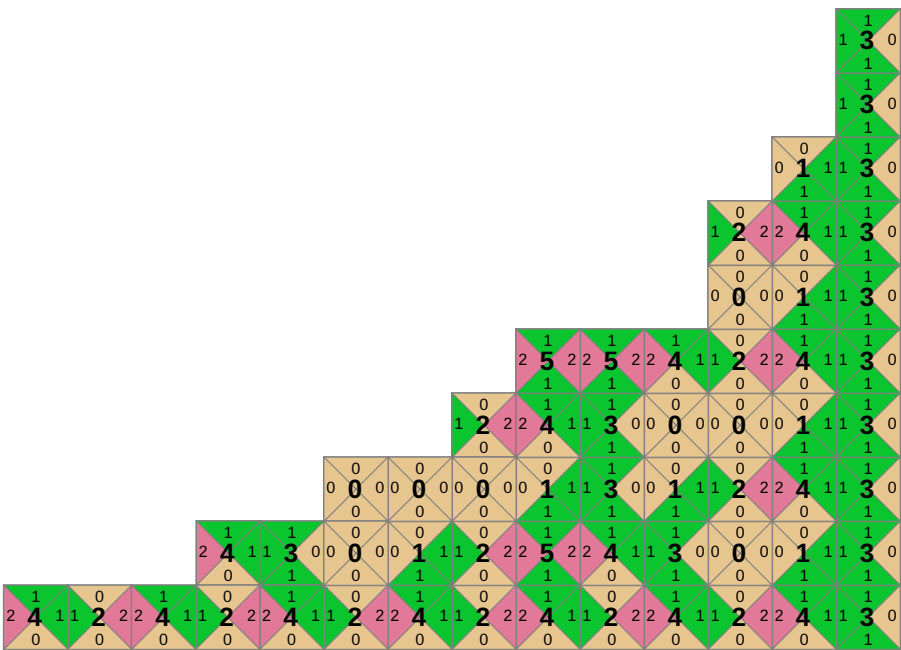
I would also like to thank teachers that have played a crucial role in the development of my passion for mathematics and computer science. In particular, monsieur Bertrand, my professor of mathematics in $3^{\text{ième}}$ (9^{th} grade), telling us about Gödel's incompleteness theorems and stating that mathematics are full of freedom – statement that I did not understand at the time; and Arnaud Basson, my professor of mathematics in first year of *classes préparatoires* (first year of BSc), who made me understand, in practice, monsieur Bertrand's statement.

Many other scientists have helped me in this journey (in rough chronological order): Vincent Gripon,

Nicolas Farrugia, Turlough Neary, Dave Doty, Trent Rogers, Matt Patitz, Constantine Evans (I owe him the latex template of this thesis), Elisa Fadda, David Malone, Anthony Small, Ismael Mullor Ruiz, Josefina López and Peter Stoll, Niall Murphy, Cai Wood, Lulu Qian, Eric Winfree, Ahmed Shalaby, Aurore Guillevic, Nicolas Schabanel, Nicolas Levy, and all *bbchallenge* collaborators. A very special thank goes to the Hamilton's institute administrators Kate Moriarty and Rosemary Hunt.

Then, I really want to thank my dear friends and family for the love and laughter they always bring me: Mathieu, Ana, Christopher-Lloyd (we started and ended our PhDs roughly at the same time, providing each other mutual support in this long endeavour!), Tristan, Vincent, Guilhem, Paul, Michael, Juliette, Clio, Eléonore, Victor, Lysiane, Gilles, Paul, Alma, Yona, Alex, Netha, Raymond, Edna, and many others. I also thank Francis and Maryse and their family for the amazing ski year I have had after my PhD (this thesis was submitted from their chalet!).

Finally, many thanks to my associates Sébastien and Alexandre, who have always trusted and supported me during this journey, and that I have always trusted and been proud of being professionally associated with.



Contents

Publications, software and talks

Publications

1. Tristan Stérin and Damien Woods. "Limitations on counting in Boolean circuits and self-assembly". Technical report, 2020. https://arxiv.org/abs/2005.13581 Not discussed in this thesis

2. Tristan Stérin. "Binary expression of ancestors in the Collatz graph". In: *Reachability Problems*. Ed. by Sylvain Schmitz and Igor Potapov. Springer International Publishing, 2020, pp. 115–130. https://arxiv.org/abs/1907.00775 [154] Results discussed in Chapter 1, Section 1.6 of this thesis

3. Tristan Stérin and Damien Woods. "The Collatz Process Embeds a Base Conversion Algorithm". In: *RP2020: 14th International Conference on Reachability Problems*. Ed. by Sylvain Schmitz and Igor Potapov. Vol. 12448. LNCS. Springer, 2020, pp. 131–147. DOI: 10.1007/978-3-030-61739-4_9. https://arxiv.org/abs/2007.06979 [156] Results discussed in Chapter 1, Section 1.2 of this thesis

4. David Doty, Benjamin L Lee, and Tristan Stérin. "scadnano: A browser-based, scriptable tool for designing DNA nanostructures". In: *DNA 2020: Proceedings of the 26th International Meeting on DNA Computing and Molecular Programming*. Ed. by Cody Geary and Matthew J. Patitz. Vol. 174. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020, 9:1–9:17. ISBN: 978-3-95977-163-4. DOI: 10.4230/LIPIcs.DNA.2020.9. URL: https://drops.dagstuhl.de/opus/volltexte/2020/12962. https://arxiv.org/abs/2005.11841 [58] Software discussed in in Chapter 4 of this thesis

5. Matthew Cook, Tristan Stérin, and Damien Woods. "Small Tile Sets That Compute While Solving Mazes". In: *27th International Conference on DNA Computing and Molecular Programming, DNA 27, September 13-16, 2021, Oxford, UK (Virtual Conference)*. Ed. by Matthew R. Lakin and Petr Sulc. Vol. 205. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 8:1–8:20. DOI: 10.4230/LIPIcs.DNA.27.8. URL: https://doi.org/10.4230/LIPIcs.DNA.27.8. https://arxiv.org/abs/2106.12341 [49] Chapter 3 of this thesis

6. Tristan Stérin and Damien Woods. "On the hardness of knowing busy beaver values BB(15) and BB(5,4)". In submission, 2021. https://arxiv.org/abs/2107.12475 [158] Chapter 2 of this thesis

7. Tristan Stérin*, Abeer Eshra*, Damien Woods. Algorithmic DNA origami: Scaffolded DNA computation in one and two dimensions. *joint first authors. Preliminary version presented at DNA28 (Track B). Manuscript in preparation. Chapter 4 of this thesis

Software

A fundamental aspect of our approach to research – and a common thread to all the chapters of this thesis – is the need to systematically write software that will allow us to creatively explore the research topic at hand (and guide our intuition) or simply, be more efficient. Here are the software that we developed for each part of this thesis:

1. `coreli`: a Python library implementing most of the concepts presented in Chapter 1, see Appendix C[1]. `https://github.com/tcosmo/coreli`
2. `simcqca`: a simulator for the Collatz Quasi Cellular Automaton (CQCA) model that we introduced in [156] (see Chapter 1, Section 1.2). `https://github.com/tcosmo/simcqca`
3. `bbsim`: an online Turing machine simulator that features the Erdős'-conjecture machines that we construct in Chapter 2. `https://dna.hamilton.ie/tsterin/bbsim/`
4. `bbchallenge`: an online collaborative platform dedicated to solving the conjecture $BB(5) = 47,176,870$ [109, 1], see Appendix D. `https://bbchallenge.org` and `https://github.com/bbchallenge`
5. `mawatam`: a simulator for the Maze-Walking aTAM, see Chapter 3. `https://github.com/tcosmo/mawatam`
6. `cosmix`: a library that we created to automatically produce experiments' wet lab mixes in the context of the Algorithmic DNA origami project, see Chapter 4. `https://github.com/tcosmo/cosmix`

Conference and invited talks

During my PhD I was given several opportunities to share the research presented in this thesis:

1. "Tales of the Collatz process in binary, ternary and senary", October 20^{th} 2020, at the 14th International Conference on Reachability Problems, online event. Presenting the results of [154, 156], respectively covered in this thesis by Chapter 1, Section 1.6 and Section 1.2.
2. "Small tile sets that compute while solving mazes", September 16^{th} 2021, at the 27^{th} International Conference on DNA Computing and Molecular Programming, online event. Presenting the results of [49], see Chapter 3.
3. "6 tiles", April 6th 2022, invited talk at Turku University in Finland. Presenting what would then become Chapter 1.
4. "Algorithmic DNA origami: Scaffolded DNA computation in one and two dimensions", joint talk with Abeer Eshra, August 9th 2022, at the 28^{th} International Conference on DNA Computing and Molecular Programming, in Albuquerque, New Mexico. Presenting what would then become Chapter 4.

[1] This library was initially released in the context of our first work on the Collatz process, [154].

Chapter 0

Introduction

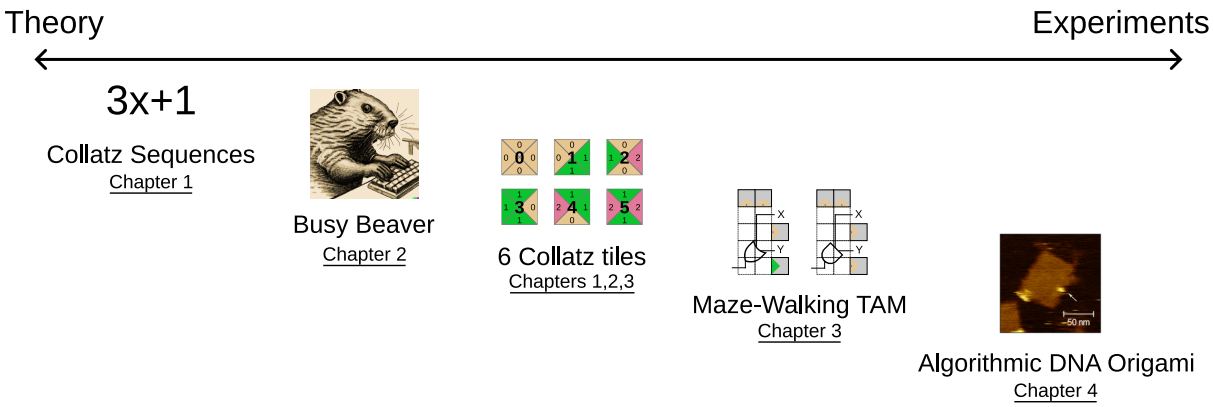


Figure 0.1: From Collatz sequences to algorithmic DNA origami.[2]

One way to introduce this thesis is to distribute its content, a bit naïvely, on an axis separating theory from experiments, Figure 0.1. When doing so, the 6 Collatz tiles (introduced in Chapter 1), that give the name to this thesis, appear to sit right in the middle as they can be considered rather experimental compared to a *pure* mathematical study of Collatz sequences (we use the tiles to perform *experiments* on Collatz sequences) and rather theoretical compared to wet lab experiments on algorithmic DNA origami (Chapter 4).

Arguably, it is not surprising that tiles (more specifically, *Wang tiles*) could occupy such a versatile position in between theory and experiments. Indeed, historically, introduced by Hao Wang in 1961 [170] and consisting of non-rotable square tiles with colors on their sides, Wang tiles have been the center of some rather theoretical questions at the frontier of mathematical logic and computer science where the main object of study were infinite tilings of the plane made of these tiles (two adjacent tiles must have the same color on their shared edge). One theoretical question that was particularly studied has been the search for *aperiodic* tile sets, i.e. tile sets such that any tiling of the plane is not periodic (no translation of the plane gives the same tiling). Over the course of 70 years, many constructions have been given, the first one by Berger in 1966 with 20,426 tiles [14] and the *last* one by Jeandel and Rao in 2015 with 11 tiles and 4 colors [84]. We say the *last* one because Jeandel and Rao show by exhaustive computer search that 10 tiles or 3 colors cannot give aperiodicity. Other constructions with 14 tiles by Kari [85] and, based on the same technique, 13 tiles by Culik [51], had been given prior to the 11 tiles result.

On the experimental side, and in particular, experiments involving DNA-based nanostructures, Wang tiles have played a very important role in the main theoretical models of algorithmic self-assembly. The key idea of self-assembly is to have individual components come together to form a structure solely based

[2]Busy beaver image generated by DALL · E [132]: "Engraving of a beaver working on a computer".

on the local interactions of these components with one another – i.e. there is no centralised *architect* supervising the construction of the structure. DNA is a material that is fit for experimenting with self-assembly in practice since DNA strands can play the role of the individual components and the bases A, T, G, C can be used to encode the local interaction rules using Watson-Crick base-pairing: A binds to T and G binds to C. Algorithmic self-assembly pushes the idea of self-assembly further by having the final assembled structure be the result of a computation that is propagated through the structure while it is self-assembled – imagine a self-assembled Boolean circuit where gates would successively attach to the circuit by self-assembly. In this context, Wang tiles are a formidable tool of abstraction as they serve as a *natural* model for self-assembly: tiles are the individual components and the colors on their sides encode local interactions. In particular, Winfree's abstract Tile Assembly Model (aTAM), introduced in 1998 [174], has been widely studied theoretically [136, 149, 125, 57, 180] and *used* in practice as a crucial abstraction for making algorithmically self-assembling structures in the wet lab [183, 66, 145, 10, 143, 144, 37, 70, 9, 135, 108, 176]. Refinements of the aTAM, such as the kinetic Tile Assembly Model (kTAM) [175], give physically realistic predictions of the kinetics of tile assembly systems and have been routinely used in the wet-lab to simulate these systems prior to implementation or to fit simulations to experimental results [66, 145, 183].

In this thesis, we study a particular set of 6 Wang tiles, that we call the Collatz tile set, which initially came to our attention for their ability to represent a fascinating mathematical problem, the Collatz problem (Chapter 1). The Collatz problem, whose origins traces back to the 1930s, asks a simple question but is still open and has left generations of researchers – including us – baffled by its complexity. Take a positive integer x and play the following game: if the number is even, divide it by 2, and if it is odd replace it by $3x+1$, then repeat. Starting from $x = 77$ we would get 77, 232, 116, 58, 29, 88, 44, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1, ... We eventually reached 1 which loops forever in the cycle $\{4, 2, 1\}$! The Collatz conjecture states that this is no coincidence and that starting from **any** natural number (other than 0) we would have reached 1. Despite tremendous efforts, wonderfully summarised in Jeffrey C. Lagarias's surveys and entire book dedicated on the problem [100, 101, 96], the Collatz problem is still open and its resolution seems very far away. Our main contributions, based on previous work that we did on the topic [154, 156], is to reformulate the Collatz problem in terms of tiles and show that the tiles provide a geometric framework (a *microscope*) for experimenting with the conjecture and reasoning about the computational abilities of the Collatz process, Chapter 1. In more details, we start by studying the arithmetic relations that naturally occur in tilings made with the Collatz tiles and their interpretation in the 2-adic, 3-adic and 6-adic integers (see Appendix A for a survival guide on p-adic integers), then, we show how the 6 tiles can simulate the Collatz process and analyse the complexity of several prediction problems in these tilings, some of which we are to put in circuit complexity class NC^1 and outside AC^0, see Section 1.4. Finally, we apply this framework to reasoning about Collatz cycles and ancestors: (a) we show that the existence of Collatz integer cycles reduces to a simple, yet apparently untractable, geometric question and (b) we provide a tile-based proof of the fact that proving the Collatz conjectures on any set of the form $\alpha + 2^n\mathbb{N}$ with $n \in \mathbb{N}$ and $\alpha < 2^n$ is enough to prove the Collatz conjecture in $\mathbb{N}$ (special case of [117]), as well as establishing the link between constructing Collatz ancestors and solving an (easy) instance of the discrete logarithm problem.

In Chapter 2 (preprint available [158]), we start by using the 6 Collatz tiles to reformulate another number-theoretical conjecture: Erdős' conjecture on powers of two, which has been open since the 1980s [65]. This conjecture is quite simple to state: "for all $n > 8$, there is at least one digit 2 in the base-3 representation of 2^n". Then, our focus deviates a bit from tiles[3], as we encode this conjecture in a *small* Turing machine with 15 states and 2 symbols. In turn, this construction allows us to qualify the hardness of

[3]Although, the busy beaver function has been used in the past in the context of tiling systems [136].

knowing the *busy beaver value* BB(15): it is at least as hard as solving Erdős' conjecture. The busy beaver function $n \mapsto \mathrm{BB}(n)$, introduced by Radó in 1962 [131], is a central object in the theory of computability as it is one of the first reported examples of a *non-computable* function (as well as, arguably, one of the most natural), see Chapter 2 for its definition. Only four of its values are known to date, $\mathrm{BB}(1) = 1$, $\mathrm{BB}(2) = 6$, $\mathrm{BB}(3) = 21$, $\mathrm{BB}(4) = 107$ [19], and it is conjectured that $\mathrm{BB}(5) = 47,176,870$ [109, 1]. It is known that $\mathrm{BB}(6) > 10 \uparrow\uparrow 15$ [113] (a *tetration*, i.e. tower of 15 powers of 10). It is also known how to relate some of its values to hard mathematical problems, for instance, knowing BB(744) is at least as hard as solving Riemann Hypothesis [1]. Our result, which links BB(15) to Erdős' conjecture on powers of two gives the smallest busy beaver value for which a connection to a *natural*, independently studied mathematical problem is known.

Chapter 3 (published in [49]) represents the pivot in this thesis from theoretical concerns (Collatz sequences, busy beaver) to experimental concerns (building DNA-based computing devices in the wet lab). While the initial motivation for this chapter was to understand if there were conditions under which the 6 Collatz tiles could be said to be Turing-complete, we ended up introducing a new model for self-assembly, the *Maze-Walking TAM*, with wet lab implementation in sight (we implemented in the wet lab a variation on this model in Chapter 4). Indeed, in Chapter 1, we failed to find Turing-completeness of the six Collatz tiles *in the wild* (i.e. in the context of Collatz sequences), and we conjectured there may be limits to their computational abilities, but we provided concrete examples of the complex patterns they generate, and computations they preform. Hence, it felt natural to harness this complexity and use it as a programming language. We do so by introducing a new model, the *Maze-Walking TAM*, which allows tiled structures to be disconnected (whereas Collatz-related assemblies need to be connected). In this model we show that the 6 Collatz tiles simulate any Boolean circuit efficiently and we even find a tile set with only 4 tiles that is also Turing-complete. In this work, we make the hypothesis that our model is fit for practical implementation with DNA-based nanostructure by interpreting the disconnected aspect of the model as having our tiles connected within an *underlying* structure such as a DNA origami which is one of the most used methods for the self-assembly of DNA nanostructures [137]. Our Turing-completeness results with the 6 Collatz tiles or even the size-4 tile set are appealing in practice because they suggest that in that model, only very few interactions would need to be designed in the lab in order to get full Turing power.

In Chapter 4 we give a first step towards the wet lab implementation of the model introduced in Chapter 3. More specifically, we implement a slight refinement of that model that we call the *Scaffolded DNA Computer*. This tile-assembly model is Turing complete, but for the sake of experimental feasibility, we implement a simplification that restricts us to Finite State Machines instead of arbitrary Boolean circuits. Nonetheless, we are excited about this experimental work because it features a promising idea: computing *within* a DNA origami, i.e. have the computation be part of the process that assembles the DNA origami itself (rather than computing *on top* of the structure). This idea allows us to inherit a very strong property of DNA origami: the target structure is thermodynamically favored. This is in stark contrast with most algorithmic systems that have been implemented with DNA to date [189, 188, 163, 128, 183] where the computationally correct structure *is not* the equilibrium structure which comes with several experimental challenges (such as algorithmic errors, spurious nucleation or *leak*) which, in principle, we do not face. We successfully implement 6 finite-state programs in a simplified 1D setup, including a 4-bit adder, reporting a 91% average yield and 2 finite-state programs in a 2D origami context, including a 7-bit adder.

Hence, the six Collatz tiles and tiling systems in general represent the common thread throughout this thesis and we regard them as a fascinating tool for studying a broad range of topics, from Collatz sequences to algorithmic DNA origami.

Chapter 1

Breaking Collatz sequences into bits, trits and tiles

1.1 Motivation

The Collatz problem is known under many names such as: the $3n+1$ problem, the $3x+1$ problem, Ulam's conjecture, Thwaites' conjecture, the Syracuse problem, etc. The essence of the problem seems to trace back to Lothar Collatz's notebooks from the 1930s and is known to have circulated in mathematical circles in the 1950s [102]. The first printed occurrence of the problem seems to be a 1971 written version of a lecture by H. S. Coxeter [50, 102]. We highly recommend Jeffrey C. Lagarias's overview article for more history, context and perspective on the problem [102].

Let $\mathbb{N} = \{0, 1, 2, \dots\}$ be the set of natural numbers and $\mathbb{N}^+ = \{1, 2, \dots\}$. The Collatz process is defined on the natural numbers as iterating the map $C(x) = x/2$ when $x \in \mathbb{N}$ is even and $C(x) = 3x+1$ when x is odd. Because $3x+1$ is always even when x is odd, it also natural to consider a slight variant of C, the map T where $T(x) = x/2$ when x is even and $T(x) = (3x+1)/2$ when x is odd. The **Collatz conjecture** states that iterating T (or C) from any natural number $x \in \mathbb{N}^+$ leads to 1. In this work, we only focus on T as it is more convenient to work with in our case[4]. For instance, iterating T from $x = 45$ gives:

$$45, 68, 34, 17, 26, 13, 20, 10, 5, 8, 4, 2, \mathbf{1}, 2, \mathbf{1}, 2, \mathbf{1}, \dots$$

As of 2020, the conjecture had been computer-verified up to 2^{68} [8].

The conjecture can be divided in two seemingly independent conjectures: (*Divergent orbits conjecture*) there are no $x \in \mathbb{N}$ such that $\lim_{n\to\infty} T^n(x) = +\infty$ and (*Nontrivial cycles conjecture*) the only cycle of T in $\mathbb{N}^+$ is $\{1, 2\}$.

The Collatz problem beyond natural numbers: $\mathbb{Z}_2$

Although T and C are defined in $\mathbb{N}$, there are no obstacles to looking at the Collatz map in the negative numbers. Three new cycles show up in the negative numbers: the cycle of -1, the cycle of -5 and the

[4]The choice of focusing on T for convenience of analysis has also been made in several other studies, such as [161, 162, 67, 178, 99, 139].

cycle of -17 [17, 102]. Furthermore, any negative number seems to reach one of these cycles. Would there be a more general conjecture to state about Collatz cycles?

It turns out that there is a set as big as $\mathbb{R}$ and which includes $\mathbb{N}$, $\mathbb{Z}$ and almost all of $\mathbb{Q}$, where the Collatz map is *naturally defined*: the set of 2-adic integers, $\mathbb{Z}_2$, which is the set of binary representations that are infinite on their most significant side. This set comes with a ring structure, i.e. well defined $+$ and $\times$ operations. Maps T and C are well defined in $\mathbb{Z}_2$ since (a) the parity of $x \in \mathbb{Z}_2$ is given by its least significant bit, (b) dividing an even number by 2 consists in removing its least significant 0 and (c) the operation $x \mapsto 3x + 1$ can be computed thanks to the definition of $+$ and $\times$ in $\mathbb{Z}_2$. To know more about $\mathbb{Z}_2$ please refer to Appendix A.

Hence, Collatz sequences are naturally defined in $\mathbb{Z}_2$, which gives a unified framework for representing Collatz iterates in the natural numbers, the integers, the rationals (with odd denominator), and irrational[5] 2-adic numbers. The generalisation of the Collatz conjecture – more specifically of the *divergent orbits conjecture* – in $\mathbb{Z}_2$ is **Lagarias periodicity conjecture** [99] – slightly reformulated: "The Collatz sequence of an eventually periodic 2-adic integer is eventually periodic".

Numbers of $\mathbb{Z}_2$ with eventually periodic representations are exactly the rational numbers with odd denominator[6] and their parity is given by the parity of their numerator (see Appendix A, Theorem A.3). Hence, we can iterate T on $-\frac{1}{23} = (00100001011)^\infty 001 \in \mathbb{Z}_2$ (we can compute this 2-adic expansion thanks to the algorithm given in Appendix A, Remark A.4), which has odd parity, and test the conjecture: $-\frac{1}{23}, \frac{10}{23}, \mathbf{\frac{5}{23}}, \frac{19}{23}, \frac{40}{23}, \frac{20}{23}, \frac{10}{23}, \mathbf{\frac{5}{23}}, \ldots$. We've reached the cycle of $\frac{5}{23}$: the Collatz sequence of $-\frac{1}{23}$ is eventually periodic.

Hence, $\mathbb{Z}_2$ provides a binary symbolic framework which, through Lagarias' periodicity conjecture, *naturally* generalises part of the Collatz conjecture to $\mathbb{Q}$. We will extensively rely on $\mathbb{Z}_2$ (and $\mathbb{Z}_3$, and $\mathbb{Z}_6$) in this work.

The Collatz problem is hard

The Collatz problem is notoriously extremely hard and its solution seems to be still very far away in spite of more than half a century of research, wonderfully summarised in Jeffrey C. Lagarias's surveys and entire book dedicated on the problem [100, 101, 96]. To date, the strongest (arguably) result on the conjecture is due to Terrence Tao and states that, in a certain sense, *almost all* positive integers reach 1 under Collatz iterations [159]. Famously, mathematician Paul Erdős said that "Mathematics is not yet ready for such problems" and, reportedly, he would also have described the problem as "Hopeless. Absolutely hopeless." [102]. As a warning, the Collatz problem is also given as *Problem 2* in Richard Guy's 1983 paper "Don't try to solve these problems!" [76].

Nonetheless, let's lay out our motivation to study such a beast. The question at the heart of this work is: "what is the computational power of the Collatz process?". Indeed, predicting the long-term behavior of Collatz orbits seems to be very hard. This fact allows one to wonder if the Collatz process, given some carefully crafted input, could be able to simulate a general-purpose computer, i.e. be Turing-complete. While it could seem unreasonable that a system with such a simple definition would be Turing-complete, we cannot rule this possibility out, especially in the light of the following points: (a) some other systems with very simple definitions but complex-looking orbits are known to be Turing-complete: Elementary Cellular Automaton "rule 110" being one of the most breathtaking examples [47] and (b) a direct generalisation of the Collatz map, namely, *Generalised Collatz Maps*, is known to be Turing-complete [46, 90, 89].

[5] For instance, the irrational $\sqrt{-7}$ is well defined in $\mathbb{Z}_2$ and we could compute its Collatz sequence, Appendix A.
[6] Rationals with even denominators cannot be represented in $\mathbb{Z}_2$, Appendix A.

Generalised Collatz maps

On this topic we highly recommend reading the extensive review chapter on generalised Collatz functions by Michel and Margenstern [114] in Lagarias' book on the Collatz problem [96]. In 1972, Conway defined *Generalised Collatz Maps* (**GCMs**) [46] and we give an equivalent definition[7] that only uses integer parameters (instead of rationals):

Definition 1.1 (Generalised Collatz Map)**.** A function f is called a Generalised Collatz map if it is defined by $f(x) = f_i(x) = a_i \frac{x-i}{N} + b_i$ when $x \equiv i \bmod N$, with $N \in \mathbb{N}^+$ and $a_i, b_i \in \mathbb{Z}$ for $0 \leq i < N$. The Collatz map T is given by $N = 2$, $a_0 = 1$, $b_0 = 0$, $a_1 = 3$, $b_1 = 2$.

In essence, Conway shows that GCMs simulate a model – which historically was formally introduced later[8] – called FRACTRAN [45]. FRACTRAN programs simulate Minsky's counter machines [115]. Minsky's counter machines can simulate Turing machines – given at least two counters. Hence, GCMs **are Turing-complete**. Koiran and Moore give a slightly different construction that directly simulates any k-instruction 3-counter Minsky machine with $N = 30k$ maps [90]. Using a different reduction to 2-counter machines, Kaščák exhibits an Universal GCM with $N = 396$ maps [89].

In terms of complexity, all known constructions for simulating Turing machines with GCMs are at least exponentially slow because of the use, at some point, of encodings *à la* Gödel – i.e. encoding information using prime factorisation. Interestingly, GCMs of two variables (not defined here) are known to simulate Turing machines efficiently [118]. Whether GCMs of one variable (as presented here) can simulate Turing machines efficiently **is an open problem** [118].

The generalised Collatz problem can be stated for GCMs: "Given a GCM f, can it be decided whether for all integers $x \in \mathbb{N}^+$, there exists $i \in \mathbb{N}$ such that $f^{(i)}(x) = 1$?". Kurtz and Simon have shown this problem is Π_2^0-complete [95].[9]

We argue that there is a problem that is suited to *more* GCMs than the above generalised Collatz problem and that has a flavor that is more computer science than mathematics. Take the GCM (which looks a lot like Collatz!) $x \mapsto 3x/2$ if x is even and $x \mapsto 3(x+1)/2$ if x is odd. The generalised Collatz problem can be trivially answered negatively on this map since any orbit diverges. However, consider the statement: for all $x \in \mathbb{N}$, x eventually reaches two successive odd iterates. If that statement is true then there are no Mahler's Z-numbers which is a problem that has been open for decades, see Appendix B. The parity of the iterates of x is known, in Collatz terminology, as the *parity vector* of x. Mahler's problem asks if the parity vector of any x contains two successive 1s. For Collatz, reaching the trivial cycle from $x \in \mathbb{N}$ corresponds to the parity vector of x reaching the infinitely repeated pattern 01. This notion generalises well to arbitrary GCMs with N maps by considering the sequence of iterates modulo N – it corresponds to which of the N maps has been chosen at each step and can be seen as the sequence of *states* that a GCM traverses while processing an input. The problem, that we loosely define here (see Section 1.4 for more), becomes *parity vector prediction*: "can the parity vector of $x \in \mathbb{N}$ be predicted?". Because GCMs are Turing-complete, we do not expect their parity vectors to be easy to predict in general. However, the question stands for particular maps[10].

[7] The original definition requires $f(x) = r_i x + s_i$ to be integral for all $x \equiv i \bmod N$ and $r_i, s_i \in \mathbb{Q}$. The equivalence with our definition can be seen by writing $x = Ny + i$ and then evaluating in $y = 0$ and $y = 1$ to get that the new coefficients are integers.

[8] FRACTRAN was formally introduced in 1987 while Conway's result on the Turing completeness of GCMs is from 1972.

[9] This means that any mathematical statement of the form "For all x, there exists y, $\phi(x, y)$" with no unbounded quantifiers in predicate ϕ can be reformulated as a Collatz-like question.

[10] Here is a GCM where predicting the parity vector is easy: $x \mapsto x/2$ when x is even and $x \mapsto (x-1)/2$ when x is odd. The parity vector of x is its base-2 representation! See Remark 1.31.

Looking for structure in Collatz iterates

We made the case that the Collatz process could be Turing-complete. However, to date, this possibility is supported by little evidence as it seems very hard to enforce any type of control in the behavior of Collatz iterates. Also, the Collatz conjecture seems to indicate the difficulty of encoding divergent programs in Collatz dynamics, although this remark is purely intuitive and speculative[11]. The Collatz process could very well linger in the mysterious zone of processes that are too complex to be analysed but not complex enough to be programmed.

Whether the Collatz process is Turing-complete or not, asking "What is the computational power of the Collatz process?", which can also be refined in "What is the complexity of predicting Collatz iterates?" remains a motivating question to us as it shifts the point of view on the Collatz process from mathematics to computer science. These questions invite us to study the Collatz process **symbolically** in search of **structure**.

In this chapter, we start by summarising results of our previous work [156] where we show that binary Collatz traces also feature ternary Collatz iterates, Section 1.2. Then we present a more generalised framework based on 6 Wang tiles, which we call "the Collatz tiles", full of arithmetical properties and which allow to reason about Collatz sequences geometrically, Section 1.3. Then, we study the complexity of prediction of various tiling problems that occur *naturally* with these tiles, Section 1.4. Finally, we apply our framework to reason about Collatz cycles, Section 1.5, and Collatz-ancestors, Section 1.6.

We think of the tile-based framework that we introduce as a microscope which allows to study, with clockwork precision, the machinery of the Collatz map on base-2, base-3, base-6 and *mixed base* representations.

We invite the reader to use our Python library `coreli v0.0.3`, see Appendix C, which provides routines to experiment with the ideas that we present here. In particular, all the tilings that illustrate the chapter were computed and outputted by `coreli`.

[11] One could imagine for instance that divergent programs would be encoded with some sort of *fuel* within Collatz and that starting with more fuel would lead to more iterations of the program.

1.2 The Collatz process in base 2 and 3: reading trits in bits

In order to explore the computational abilities of the Collatz process we study it symbolically. We mean that we study the action of the Collatz process on some encoding of numbers. As computer scientists, our heart naturally goes to binary representations of numbers. It turns out that this choice of representation which, at first sight, is arbitrary, is actually quite natural to consider when it comes to the Collatz process and there has been a fruitful trend of studying Collatz iterates in binary [16, 22, 146, 42, 78, 106, 31]. We feel particularly indebted to [31] that initially sparked our interest for studying the Collatz process in binary.

In this section, we summarize the main result of our previous work [156]: somehow, the binary traces of Collatz iterates also contain the ternary representation of these same iterates (Theorem 11 in [156][12]), see Figure 1.4.

In binary, deducing whether a number is even or odd only amounts to looking at its least significant bit (**LSB**) and, applying the $x/2$ branch of the Collatz map merely corresponds to removing a trailing 0 from the binary representation of x. For instance, if we divide number 14, `1110` in binary, by 2 we get 7, which is `111` in binary. Hence, to get an understanding of one application of the Collatz map in binary, we are only left with the task of interpreting the action of $x \mapsto 3x+1$ on binary representations of numbers.

Convention: LSB to the right. As implicitly used above, in this work, we choose the convention of placing the LSB of binary representations (and later, p-adic representations) to the right: `1101` represents thirteen **and not** eleven.

Interpreting $x \mapsto 3x+1$ in binary as a finite state transducer

To make sense of the operation $x \mapsto 3x+1$ in binary let's focus first on the operation $x \mapsto 3x$. Let's then note the seemingly unimportant fact that $3x = x + 2x$. In binary, the latter interprets as "Add x to its left shift". Let's give an example on 3×13, 13 is `1101` in binary:

$$\begin{array}{cccccccc} & & & & 1 & 1 & 0 & 1 \\ + & & \overline{0} & \overline{1} & 1 & 0 & 1 & 0 \\ \hline & & 1 & 0 & 0 & 1 & 1 & 1 \end{array}$$

We get `100111` which is, as expected, the binary representation of $3 \times 13 = 39$. Note that we have used $\overline{0}$ and $\overline{1}$ to signify the propagation of a carry over a 0 or a 1. From this, we understand that, in binary, the operation $x \mapsto 3x$ correspond to **adding each bit of x to its right neighbour and a potential carry** (or add with 0 for the LSB), which can be visualised as follows:

At each step that is depicted in Figure 1.1 the bit inside the blue dashed box gets added to the bit and the potential carry in the magenta box. The result of this addition gets written in the green dashed box and a carry is potentially generated for the next step. For instance, at step 4, bit 1 in dashed blue box gets added to bit 1 in magenta box, which produces a 0 in the dashed green box and generates a carry for step 5. At step 5, bit 0 in the dashed blue box (it is a leading 0 as we have finished to scan the entire input) gets added to bit 1 **and** its carry in the magenta box. This produces a 0 in the dashed green box and generates a carry for step 6.

The logic that is depicted in Figure 1.1 can be reformulated as a Finite State Transducer (**FST**), which is a Finite State Automaton that reads and outputs at each step. Here, the state is the content of the

[12] Theorem 16 in the arxiv version [155].

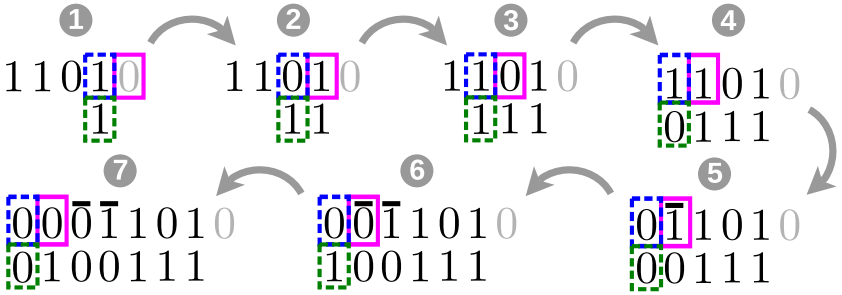
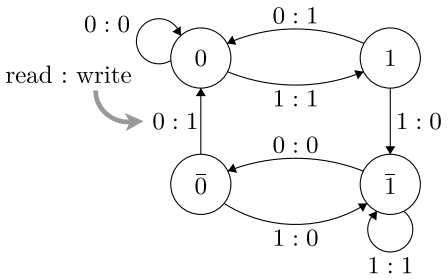
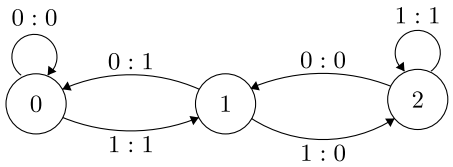


Figure 1.1: Six successive applications of the local rule "**adding each bit of x to its right neighbour and a potential carry**" on $x = 13$, `1101` in binary. This corresponds to computing $3x = 39$, `100111` in binary – see step 6. The LSB is added to an imaginary trailing 0 (in grey). After step 6, leading 0s will be produced for ever, signalling the end of the computation. The content of the magenta box is either 0, 1, $\overline{0}$ or $\overline{1}$ and maintains the state of the computation. The blue (resp. green) dashed box contains the input (resp. output) bit of each step of the computation. Adding two or more ones generates a carry for the next step.

magenta box in Figure 1.1 and can be either 0, 1, $\overline{0}$ or $\overline{1}$ and the input/output alphabet is {0,1}. The input (resp. output) corresponds to the bit in the blue (resp. green) dashed box in Figure 1.1. Here is the FST, which we call the "$3x + i$" FST:



(a) The "$3x + i$" FST.



(b) The "$3x + i$" FST, minimised.

Figure 1.2: The "$3x + i$" finite state transducer. (a) This transducer reads $x \in \mathbb{N}$ in binary, bits by bits (from the LSB) and, given enough leading 0s in the representation of x, outputs the representation of $3x$ or $3x + 1$ or $3x + 2$ depending on initial state. Initial state 0 gives $3x$, initial states $\overline{0}$ or 1 give $3x + 1$ and initial state $\overline{1}$ gives $3x + 2$. (b) States $\overline{0}$ and 1 are redundant and can be merged to give an equivalent minimal 3-state FST (state $\overline{1}$ has been renamed 2). The minimised FST computes $3x + i$ with $i \in \{0, 1, 2\}$ the initial state of the computation.

We realise that starting the computation of Figure 1.1 with $\overline{0}$ (or 1) instead of 0 in the magenta box at step 1 would have computed $3x + 1$ instead of $3x$ since an extra imaginary carry (or bit) would have been added to the output. Hence, after this point, we get an answer to our question about interpreting $x \mapsto 3x + 1$ in binary: it corresponds to processing the binary representation of x with the "$3x + i$" FST (Figure 1.2(a)), starting from initial state $\overline{0}$ or 1 – or just state 1 in the minimised version of Figure 1.2(b).

Nonetheless, a few points about the "$3x + i$" transducer:

1. The FST (Figure 1.2(a)), reads binary representations from their Least Significant Bit and, given enough leading 0s in the input, outputs the binary representation of $3x + i$ where $i \in \{0, 1, 2\}$ depends only on the initial state. If the initial state is 0, then $i = 0$, if the initial state is $\overline{0}$ or 1 then $i = 1$ and finally, if the initial state is $\overline{1}$ then $i = 2$. See [156] for a proof. We can see that in this carry-annotated representation, states 1 and $\overline{0}$ of Figure 1.2(a) are redundant: their read/write behavior is the same and they lead to the same states in each case. Hence, these states can be safely merge to give a minimal equivalent FST with 3 states, Figure 1.2(b), where we removed state $\overline{0}$ and where state $\overline{1}$ has been renamed to 2. In this section, we realise that these three states corresponds to ternary digits (or trits) which is the main result of our previous work [156].

2. The need for having leading 0s in the input merely comes from the fact that often, we need more bits to represent $3x + i$ than we need for x. In fact, because $3x + i \leq 4x$ for all $x \geq 2$ in $\mathbb{N}$ and $i \in \{0, 1, 2\}$, we know that $3x + i$ uses at most 2 more bits than x, hence running any binary input with two extra leading 0s is a safe way to always get the complete output of the FST. Another, beautiful, way to consider this issue is to add an infinite amount of leading 0s to any finite input and let the FST run forever which will produce an infinite amount of leading 0s in the output as well. In that context, inputs and outputs of the FST are semi-infinite binary strings (infinite on the most significant side), i.e. 2-adic integers (see Appendix A). Point 1 generalises beautifully to 2-adic integers as we get that, for all $x \in \mathbb{Z}_2$, the output of the "$3x + i$" FST is $3x + i \in \mathbb{Z}_2$, with i given by the initial state[13] (see Point 1).

3. The FST is reversible, in the sense that inversing read:write instructions yields a FST that is still deterministic. In the context of Figure 1.1 this idea corresponds to reading input bits in the green dashed box instead of the blue one and writing output bits in the blue dashed box instead of the green one. Mathematically, this corresponds to computing division by 3 instead of multiplication by 3 in binary, more precisely, in $\mathbb{Z}_2$, the reversed FST computes $(x - i)/3$ with i corresponding to the initial state and is defined in the same way as in Point 1. For instance, running the *reversed* "3x+1" FST (for ever) from input $0 \in \mathbb{Z}_2$ and state 1 outputs $y = \ldots$`01010101`, which is the 2-adic representation[14] of $-1/3$ (see Appendix A).

[13] Indeed, one can show that for all n we have $3x_n + i = s_n 2^n + y_n$ with $x_n \in \mathbb{N}$ (resp. $y_n \in \mathbb{N}$) the natural number represented in binary by the n least significant bits of the input (resp. output), $s_n \in \{0, 1, 2\}$ the state of the minimised "$3x + i$" FST after n steps. When $n \to \infty$ we get $3x + i = y$ as needed since $\lim_{n\to\infty} s_n 2^n = 0$ and $\lim_{n\to\infty} x_n = x$, $\lim_{n\to\infty} y_n = y$ in $\mathbb{Z}_2$ (see Appendix A). In the context of the tiles (Section 1.3), the relation $3x_n + i = s_n 2^n + y_n$ is the particular case of a rectangle of width n and height 1 with x_n given on the north side and i given on the east side, see Corollary 1.16.

[14] To get convinced that $y = \ldots$`01010101` represents $-1/3$ remark that $2y + y = \ldots$`11111111` which is the representation of -1, hence $3y = -1$.

Given the interpretations that we have given of operations $x \mapsto x/2$ and $x \mapsto (3x+1)/2$ in binary, we can start iterating the Collatz map in binary and introduce the *carry-annotated Collatz trace* of a number[15]. Here is the carry-annotated trace of 99 (1100011 in binary, magenta):

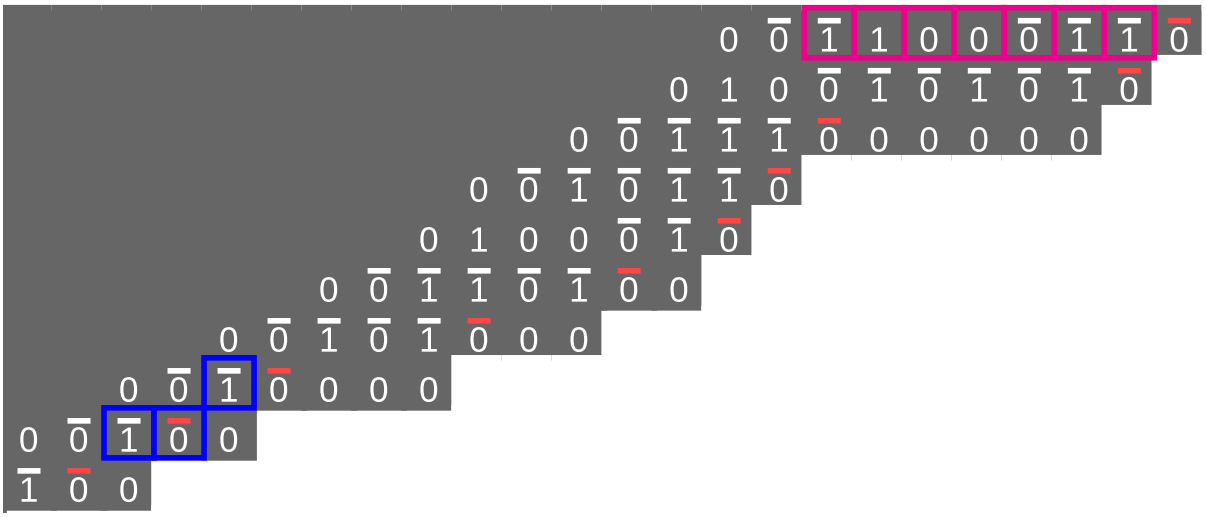


Figure 1.3: *Carry-annotated Collatz trace* of 99, 1100011 in binary (magenta) as introduced[15] in [156]. Each row of the trace corresponds to an odd iterate of the Collatz sequence of 99, even iterates are placed on the same row as they correspond to trailing 0s. Imaginary carries (red) are placed on top of the first trailing 0 of each row, which corresponds to where the "3x+i" FST starts running (in state $\overline{0}$). In blue is shown when the iterates reach the cycle 1,2 for the first time.

The trace is constructed by applying the "3x+i" FST (Figure 1.2) on each successive odd Collatz iterate: there is one odd iterate per row – rows are read in binary, ignoring the value of the carries. For instance, the first 3 rows correspond to 99 (1100011 in binary, magenta in Figure 1.3), 149 (10010101 in binary) and 7 (111 in binary) which are indeed the first three odd iterates in the Collatz sequence of 99 (odd iterates in bold):

99, **149**, 224, 112, 56, 28, 14, **7**, **11**, **17**, 26, **13**, 20, 10, **5**, 8, 4, 2, **1**, 2, ...

Successive even iterates are placed on the same row of the trace as they only differ in a trailing 0, for instance the third row contains the representations of 224, 112, 56, 28, 14 and 7, depending on where the last trailing 0 is placed. In red, we highlight the imaginary carry that is placed right after the last trailing 1 of a row, signifying the starting position of the "3x+i" FST in state $\overline{0}$ which computes the operation $3x+1$ on that row's odd iterate. Highlighted in blue the first occurrence of the cycle 1,2 which here occurs after 7 applications of the "$3x+i$" FST (i.e. after 8 rows in the trace and equivalently, 8 odd iterates).

The main result of our previous work [156] is that *vertical* columns of these traces can be interpreted in ternary where 0,$\bar{0}$,1 and $\bar{1}$ (that we call base 3') respectively correspond to ternary digits[16] $0, 1, 1, 2$ (i.e. the states of the minimised "$3x+i$" FST, Figure 1.2(b)). More specifically, we obtain a base conversion result where, for each position of the trace that is to the left of the most significant non-zero bit of x, interpreting the vertical column directly north of the position in ternary gives the same number as interpreting the horizontal row directly west of the position in binary – in the context of the Collatz tiles (Section 1.3), this result is "only" a special case of Corollary 1.16, see Remark 1.19:

From this result, we conclude that columns also iterate the Collatz process (function T) but, in base 3. Note the nice symmetry: in base 3, the simple operation is $x \mapsto 3x+1$ since it only corresponds

[15] Formally, these traces are space-time diagrams of the *Collatz Quasi-Cellular Automaton* as described in [156].
[16] Conversely, [156] gives a simple rule for knowing which of $\bar{0}$ or 1 is used to represent ternary digit 1.

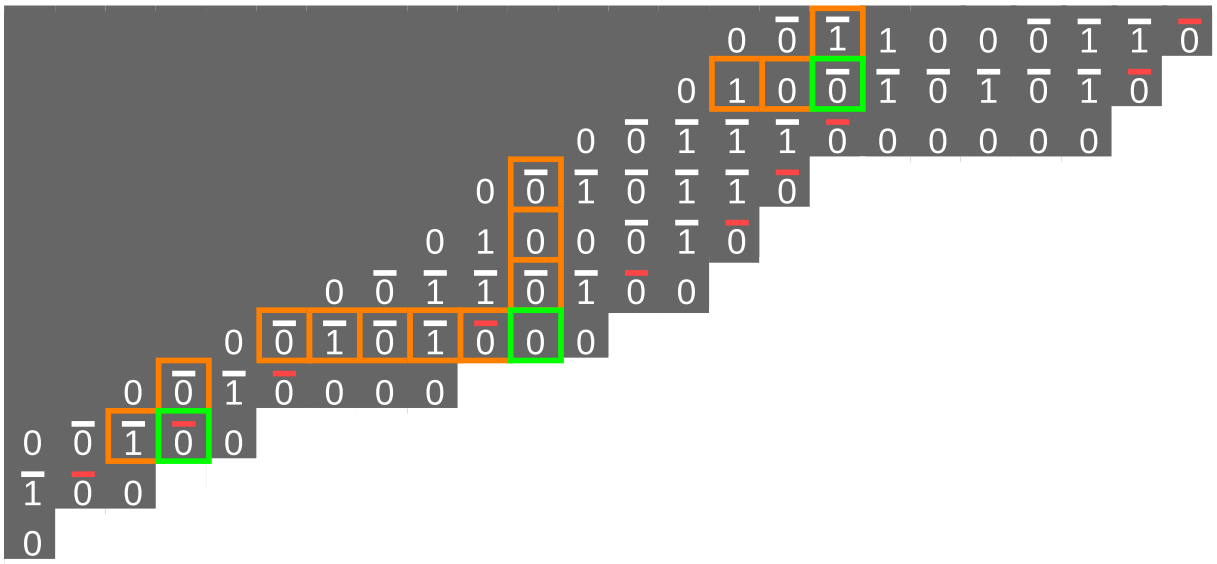
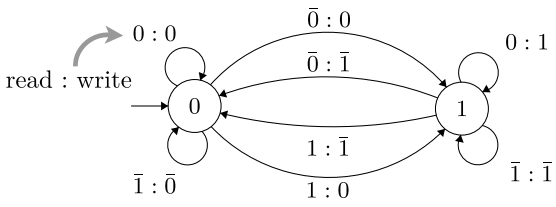
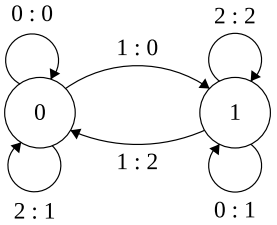


Figure 1.4: Three instances of the base-conversion result of [156]: three orange L-shaped corners such that north and west of green represent the same number respectively in base 3' (i.e. base 3 but where there are two encodings of ternary digit 1, $\bar{0}$ and 1, and where digit 2 is $\bar{1}$) and in base 2 (where carries are ignored). Right orange L-shaped corner: we read $[\![\bar{1}]\!]_{3'} = [\![2]\!]_3 = 2$ to the north of the green cell and $[\![\bar{1}0]\!]_2 = 2$ to the west of the green cell. Centre orange L-shaped corner: we read $[\![\bar{0}0\bar{0}]\!]_{3'} = [\![101]\!]_3 = 10$ to the north of the green cell and $[\![\bar{1}010]\!]_2 = 10$ to the west of the green cell. Left orange L-shaped corner: we read $[\![\bar{0}]\!]_{3'} = [\![1]\!]_3 = 1$ to the north of the green cell and $[\![\bar{1}]\!]_2 = 1$ to the west of the green cell.

to adding ternary digit 1 at the end of the representation, which happens within the same column in carry annotated traces (base 3' symbol $\bar{0}$ with red-coloured carry is appended at the end of odd iterates), whereas computing $x \mapsto x/2$ is harder. This computation is performed, from least significant symbol to most significant symbol, by another finite state transducer, the "$(x-i)/2$" transducer, see Figure 1.5(a).



(a) The "$(x-i)/2$" FST.



(b) The "$(x-i)/2$" FST, minimised.

Figure 1.5: The "$(x-1)/2$" finite state transducer. (a) This transducer reads $x \in \mathbb{N}$ in ternary (actually, base 3′, see above), trits by trits (from the least significant digit) and, outputs the representation of $(x-i)/2$ depending on initial state i. (b) minimised FST once we have merged transitions that are identical when base 3' symbols $0, \bar{0}, 1, \bar{1}$ are converted to base 3 symbols $0, 1, 1, 2$.

Contrarily to the "3x+i" FST, there is no need to use leading 0s since the representation of $(x-i)/2$ uses less space that the one of x. Similarly to the "3x+i" FST, inputs and outputs can be interpreted on semi-infinite ternary strings, which are known as 3-adic integers whose set is denoted $\mathbb{Z}_3$, see Appendix A. For all $x \in \mathbb{Z}_3$, the "$(x-i)/2$" FST gives the 3-adic representation of $(x-i)/2$ when processing x (and running for infinitely many steps). The "$(x-i)/2$" FST is also reversible as inverting its read/write instructions yields a deterministic machine that computes $x \mapsto 2x+i$. Finally, in [156] we had remarked that the "$3x+i$" and "$(x-i)/2$" FSTs are *dual*, in the sense that the symbols of one are the states of the other (and vice versa) and that there is a transition $q_0 \to q_1$, $r : w$ in one machine iff there is a transition $r \to w$, $q_0 : q_1$ in the other, see Figure 1.6.

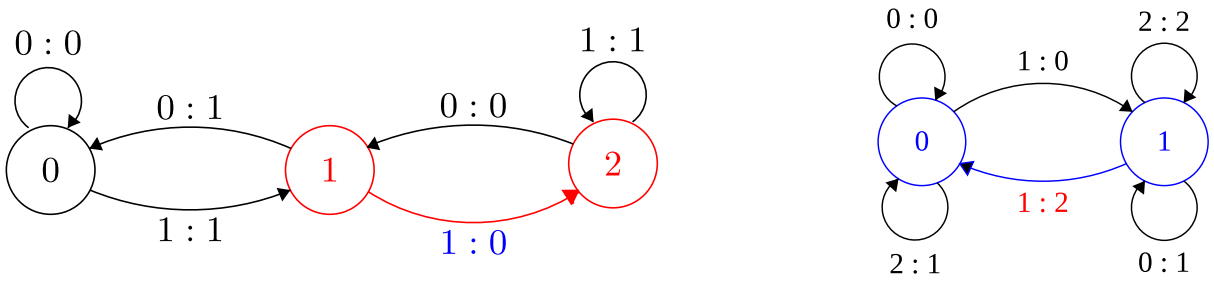


Figure 1.6: The binary "$3x + i$" FST (left) and ternary "$(x - i)/2$" FST (right) are *dual*: the symbols of one are the states of the other (and vice versa) and that there is a transition $q_0 \to q_1$, $r : w$ in one machine iff there is a transition $r \to w$, $q_0 : q_1$ in the other. An example is given in blue: $1 \to 2$, $1 : 0$ in the "$3x + i$" FST (left) gives $1 \to 0$, $1 : 2$ in the "$(x - i)/2$" FST (right).

The fact that the two FSTs are dual should have revealed the 6 Collatz tiles to our eyes (Section 1.3) since pairs of FSTs that are dual while processing their inputs in orthogonal directions can always be converted into Wang tile sets[17] and vice versa, see Section 2.2 in [84]. But, we did not notice them until Matthew Cook brought them to our attention and opened the discussion of converting the base conversion result that we had on carry-annotated in terms of tiles. This theory revealed to be more general (making an analogy with microscopes, one could also say that the tiles give a "better resolution" than the carry-annotated traces) and we present this theory, that sprouted from our discussions with Matthew Cook, in the next section.

[17]The size of the corresponding tile set is the product of the number of states of each machine, here: $6 = 3 \times 2$.

1.3 The Collatz process in base 6 and beyond: Collatz tilings

In this section, we study a set of 6 Wang tiles, that we call "the Collatz tile set", Figure 1.7, that can represent Collatz sequences in base 2, 3, 6 (and many others). We acknowledge Matthew Cook for bringing this tile set to our attention and initiating the discussion on how our results of [156] would be interpreted, and generalised, with the tiles. We also remark that the tiles can be deduced from Korec's base 6 Collatz Cellular Automaton [93]. This tile set can be used to work on other problems than Collatz, such as Erdős conjecture on powers of two (see Chapter 2) or Mahler's 3/2 problem (see Appendix B) and has been implicitly studied in the case of Mahler's problem [87, 91].

Our main innovations in this study are (a) the interpretation of the color of each tile's side as a digit in base 2 or 3 (b) the focus on finite paths in tilings, interpreted as affine functions (Theorem 1.13) (c) the interpretation of infinite horizontal/vertical/diagonal paths as respectively 2-adic, 3-adic and 6-adic integers which are transformed by the functions of each finite path (Theorem 1.25). We then proceed to demonstrate how these tiles can assemble Collatz sequences, Theorem 1.35, and use this result to reason about Collatz cycles (Section 1.5) and Collatz ancestors (Section 1.6). These results generalise the base conversion property seen in binary Collatz traces that we highlighted in our previous work [156] (see Remark 1.19).

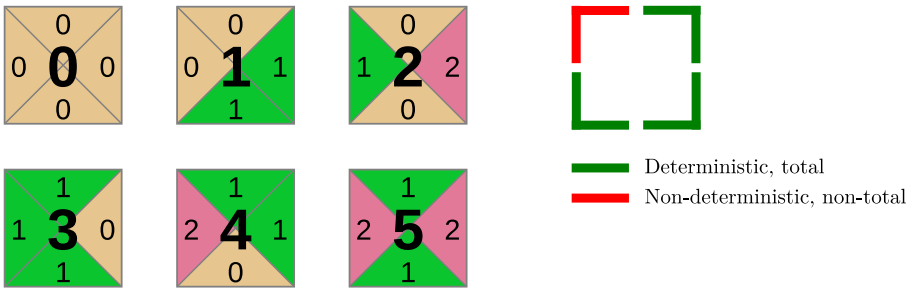


Figure 1.7: The Collatz tile set, or (2,3)-CRT tile set. Colors on horizontal edges are labelled in $\{0,1\}$, colors on vertical edges are labelled in $\{0,1,2\}$. North-east, south-east and south-west corners are deterministic and total, Definition-Lemma 1.2. The north-west corner is non-deterministic and non-total, Remark 1.3.

Definition-Lemma 1.2 (Collatz tile set, or (2,3)-CRT tile set)**.** The Collatz tile set, or (2,3)-CRT tile set, is the set of six Wang tiles given in Figure 1.7. We denote this tile set by $\mathcal{C}$. Tile $t \in \{0,1,2,3,4,5\}$ is labelled by glues $n, s \in \{0,1\}$ on the North and South and $e, w \in \{0,1,2\}$ on the East and West such that:

$$t = 3n + e = 2w + s$$

Hence, corners $(n,e) \in \{0,1\} \times \{0,1,2\}$ and $(s,w) \in \{0,1\} \times \{0,1,2\}$ of the tiles are deterministic and total, i.e. there is exactly one distinct tile for each possible choice of these corners. Secondly, tile t is the only solution in $\{0,1,2,3,4,5\}$ of the system of equations for the corner $(s,e) \in \{0,1\} \times \{0,1,2\}$:

$$t \equiv s \bmod 2$$
$$t \equiv e \bmod 3$$

In other words, the corner (s,e) solves the Chinese Remainder Theorem (**CRT**) in $\mathbb{Z}/2 \times \mathbb{Z}/3 \simeq \mathbb{Z}/6$, and hence (s,e) is a third deterministic and total corner for the tile set.

Proof. All the properties can be manually checked for each tile. For instance on tile 5 we get $(n,e,s,w) = (1,2,1,2)$ and: $5 = 3 \times 1 + 2 = 2 \times 2 + 1$. Furthermore, 5 is the only number in $\{0,1,2,3,4,5\}$ such that $5 \equiv 1 \bmod 2$ and $5 \equiv 2 \bmod 3$.

□

Remark 1.3 (North-west corner is **not deterministic**)**.** The only corner of the tiles that is not deterministic (and not total) is north-west, $(n, w) \in \{0, 1\} \times \{0, 1, 2\}$. Indeed: (i) $(n, w) = (0, 0)$ is used by both tiles 0 and 1, (ii) $(n, w) = (1, 2)$ is used by both tiles 4 and 5 and (iii) no tile has $(n, w) = (1, 0)$ nor $(n, w) = (0, 2)$.

Reminder on the Chinese Remainder Theorem (CRT). The Chinese Remainder Theorem can be stated in different ways, but perhaps its most general statement is that rings $\mathbb{Z}/n_1 \times \mathbb{Z}/n_2 \cdots \times \mathbb{Z}/n_k$ and $\mathbb{Z}/n_1 n_2 \dots n_k$ are isomorphic (noted $\simeq$) if and only if $n_1, \dots, n_k \in \mathbb{N}^+$ are pairwise coprime. One of the consequences of that statement is that for such n_i, the system of k equations $x \equiv a_i \bmod [n_i]$ with $0 \leq a_i < n_i$ always admits a unique solution in $\{0, \dots, n_1 n_2 \dots n_k - 1\}$ and that this solution is different for each possible $n_1 n_2 \dots n_k$ assignments of the a_i. Another consequence is that arithmetical operations $+$ and $\times$ in $\mathbb{Z}/n_1 n_2 \dots n_k$ can be performed in parallel in each $\mathbb{Z}/n_i$ and then *reassembled* in order to get the result in $\mathbb{Z}/n_1 n_2 \dots n_k$ thanks to the ring isomorphism between $\mathbb{Z}/n_1 \times \mathbb{Z}/n_2 \cdots \times \mathbb{Z}/n_k$ and $\mathbb{Z}/n_1 n_2 \dots n_k$, a property that we will use, see Remark 1.26.

Remark 1.4 ((n, m)-CRT tile set)**.** We have seen that the South-East corner of the Collatz tiles is solving the Chinese Remainder Theorem for $\mathbb{Z}/2 \times \mathbb{Z}/3 \simeq \mathbb{Z}/6$. More generally, for any pair of coprimes $(n, m) \in \mathbb{N}^2$ we can construct the (n, m)-CRT tile set which has nm tiles, one per possible South-East corner in $\mathbb{Z}/n \times \mathbb{Z}/m$. The label of each tile is the solution of the CRT in $\mathbb{Z}/n \times \mathbb{Z}/m \simeq \mathbb{Z}/nm$, meaning that tile $t \in \{0, \dots, nm - 1\}$ is the only one such that $t \equiv s \bmod n$ and $t \equiv e \bmod m$ with $(s, e) \in \mathbb{Z}/n \times \mathbb{Z}/m$ the South-East corner. The two other sides of the tiles, North and West, are respectively given by the quotient of t by m and the quotient of t by n. Most of what we say in this chapter, especially Theorem 1.13, can be generalised to these (n, m)-CRT tile sets. The tiles can also be generalised to higher dimensions as well as beyond coprimes [92].

Now that we have our Collatz tiles, we can consider tilings made of these tiles. More precisely, by *tilings* we mean partial tilings:

Definition 1.5 (Partial tilings)**.** Let $\tilde{\mathbb{Z}} = \mathbb{Z} + \frac{1}{2}$. A partial tiling of the Collatz tile set, or *partial tiling* for short, is a partial function $\mathcal{T} : \tilde{\mathbb{Z}}^2 \mapsto \mathcal{C}$ where, if defined, $\mathcal{T}(x, y)$ is one of the six Collatz tiles (Definition 1.2) present at position $(x, y) \in \tilde{\mathbb{Z}}^2$. The corners of the tiles in $\mathcal{T}$ are positioned in $\mathbb{Z}^2$. In this work we use the expressions *partial tiling*, *tiling*, and *assembly* **interchangeably**.

We need a notion of what is a *valid* partial tiling, also see Figure 1.8:

Definition 1.6 (Valid tilings)**.** A *valid* partial tiling $\mathcal{T}$ meets the following three conditions: (i) There are no color mismatches in $\mathcal{T}$, i.e. adjacent colors always match, (ii) $\mathcal{T}$ is 8-connected, i.e. any tile is reachable from any other by walking across tiles that share an edge or a corner[18], (iii) Any horizontal row and vertical column of $\mathcal{T}$ is 2-connected, i.e. there must be no empty position separating two tiles that are defined on the same row/column.

1.3.1 Valued paths in tilings

The central element of this work is to consider *finite* paths in tilings and interpret them as *functions*. The main result of this section is that in a valid tiling, the function computed by a path only depends on its starting and ending position, Theorem 1.13. From this result we get an arithmetical interpretation of any

[18] By lifting this constraint of connectivity, we lose the connection to these arithmetical problems, but we gain the ability to simulate arbitrary Boolean circuits with the Collatz tiles, Chapter 3.

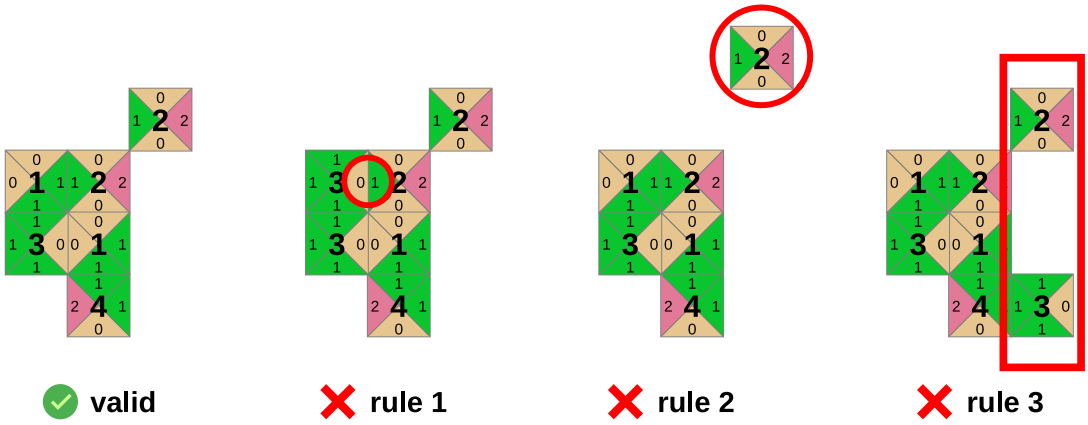


Figure 1.8: A valid partial tiling and three invalid tilings, each one respectively in breach of each rule of Definition 1.6.

valid rectangular tiling, Corollary 1.16. In the next Section 1.3.2, we show how to concretely compute the output of the function of a path on any given input.

Definition 1.7 (Base interpretation functions)**.** Let $[\![\cdot]\!]_2 : \{0,1\}^* \to \mathbb{N}$ the function that interprets finite binary strings to base-2 represented natural numbers. Similarly, define $[\![\cdot]\!]_3 : \{0,1,2\}^* \to \mathbb{N}$ and $[\![\cdot]\!]_6 : \{0,1,2,3,4,5\}^* \to \mathbb{N}$ for base 3 and base 6.

Definition 1.8 (Valued path)**.** A *valued path* is a finite list of *valued edges* $(e_0, v_0), \ldots, (e_{n-1}, v_{n-1})$ where edge e_i is one of $\{\rightarrow, \leftarrow, \downarrow, \uparrow, \searrow, \nwarrow, \swarrow, \nearrow\}$ and where valuation $v_i \in \{0,1\}$ if e_i is horizontal, $v_i \in \{0,1,2\}$ if e_i is vertical and $v_i \in \{0,1,2,3,4,5\}$ if e_i is diagonal. Let $\mathcal{P}$ be the set of valued paths. In the plane, each edge of a path is drawn at the tip of the previous one.

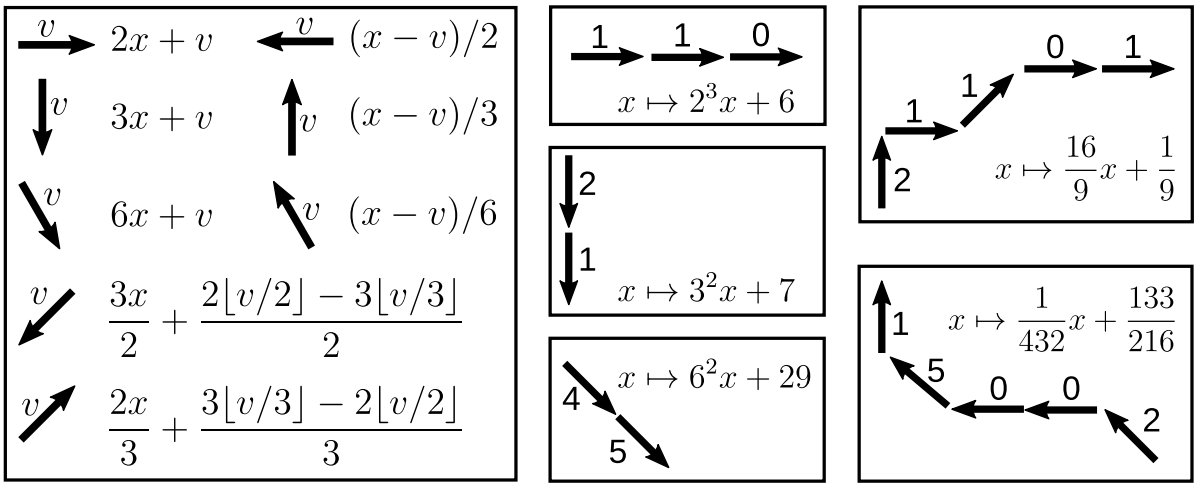


Figure 1.9: Left: functions computed by individual valued edges. Centre: horizontal, vertical and south-east diagonal paths respectively correspond to base 2, base 3 and base 6 (Lemma 1.10), here: $6 = [\![110]\!]_2$, $7 = [\![21]\!]_3$ and $29 = [\![45]\!]_6$. Right: functions computed by two other more complex paths.

Definition 1.9 (Function of a path)**.** Let $\mathcal{A}$ be the set of affine functions with rational coefficients, seen as formal objects: we do not define the domains of these functions or run them on any inputs (yet) but just allows to compose them. We define the function of a path by the morphism $\phi : \mathcal{P} \to \mathcal{A}$ which is

defined on individual valued edges, see Figure 1.9 left:

$$\begin{aligned}
\phi(\rightarrow, v) &= 2x + v & \phi(\leftarrow, v) &= (x - v)/2 \\
\phi(\downarrow, v) &= 3x + v & \phi(\uparrow, v) &= (x - v)/3 \\
\phi(\searrow, v) &= 6x + v & \phi(\nwarrow, v) &= (x - v)/6 \\
\phi(\swarrow, v) &= \frac{3}{2}x + \frac{2\lfloor v/2 \rfloor - 3\lfloor v/3 \rfloor}{2} & \phi(\nearrow, v) &= \frac{2}{3}x + \frac{3\lfloor v/3 \rfloor - 2\lfloor v/2 \rfloor}{3}
\end{aligned}$$

Then, for path $p = (e_0, v_0), \ldots, (e_{n-1}, v_{n-1}) \in \mathcal{P}$ we set $\phi(p) = \phi(e_{n-1}, v_{n-1}) \circ \phi(e_{n-2}, v_{n-2}) \circ \cdots \circ \phi(e_0, v_0)$, and we say that p computes $\phi(p)$. Note that edges that go in opposite directions with the same valuation compute inverse functions to one another, e.g. $\phi(\rightarrow, v) \circ \phi(\leftarrow, v) = x \mapsto x$.

Lemma 1.10 (Remarkable paths: base 2, base 3, base 6)**.** Horizontal paths correspond to base 2, vertical path to base 3 and south-east diagonal path to base 6 – Figure 1.9 centre:

1. A horizontal path $\rightarrow \cdots \rightarrow$ of length n valued $v \in \{0, 1\}^n$ computes the function $x \mapsto 2^n x + [\![v]\!]_2$
2. A vertical path $\downarrow \ldots \downarrow$ of length n valued $v \in \{0, 1, 2\}^n$ computes $x \mapsto 3^n x + [\![v]\!]_3$
3. A diagonal path $\searrow \cdots \searrow$ of length n valued $v \in \{0, 1, 2, 3, 4, 5\}^n$ computes $x \mapsto 6^n x + [\![v]\!]_6$

Proof. Immediate consequence of Definition 1.8. For instance, the function computed by a valued horizontal path $(\rightarrow, v_{n-1}), \ldots, (\rightarrow, v_0)$ of length n is $x \mapsto 2^n x + \sum_{i=0}^{n-1} 2^i v_i = x \mapsto 2^n x + [\![v]\!]_2$. □

Example 1.11. Figure 1.9 right gives the functions computed by two paths, using the composition rule given in Definition 1.8.

Reading paths in tilings. So far, we have defined paths abstractly but their definition is motivated by tilings made of the Collatz tile set. Indeed, in a tiling, we can follow such paths and read their valuations in the color of tiles' edges for horizontal/vertical edges and in tiles' names for diagonal edges, Figure 1.10(b) the valuation of the brown path in is 0,3,0,0,2,1,0,5.

Looking at paths at the scale of just one tile makes us realise that our tiles *commute*, Figure 1.10(a):

Lemma 1.12 (The tiles *commute*)**.** On each of the 6 Collatz tiles, from the top left corner, the paths $\rightarrow\downarrow$, $\downarrow\rightarrow$ and $\searrow$ compute the same function. Similarly, from the top right corner, the paths $\leftarrow\downarrow$, $\downarrow\leftarrow$ and $\swarrow$ compute the same function. Same holds for these paths with edges in the reversed direction.

Proof. This property can be verified individually on each tile. Figure 1.10(a) shows the result for tile 4: path $\rightarrow\downarrow$ valued $(1, 1)$ computes the function $x \mapsto 3(2x + 1) + 1$ and path $\downarrow\rightarrow$ valued $(2, 0)$ computes $x \mapsto 2(3x + 2)$. In both cases this corresponds to the function $x \mapsto 6x + 4$. □

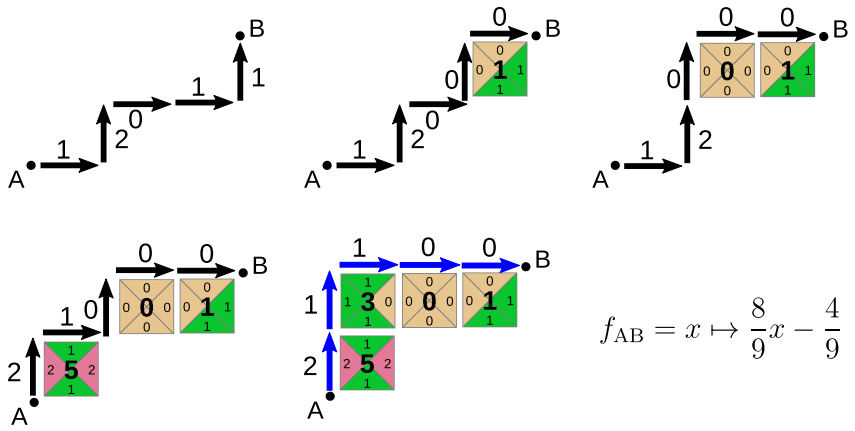
From this local commutation property, we get one of our main results, illustrated in Figure 1.10(b):

Theorem 1.13. Let $\mathcal{T}$ be a valid partial tiling of the Collatz tile set and A and B two points in $\mathbb{Z}^2$. Then, all valued paths in $\mathcal{T}$ from A to B compute the same function: f_{AB}.

Proof. Without loss of generality we can suppose that paths do not contain diagonal edges as they can be replaced by two edges which will compute the same function, Lemma 1.12. For instance, by using tile 4, valued edge $(\nearrow, 4)$ can be replaced by $(\uparrow, 2)(\rightarrow, 1)$ and both paths compute the function $x \mapsto \frac{2}{3}x + \frac{3\lfloor 4/3 \rfloor - 2\lfloor 4/2 \rfloor}{3} = x \mapsto \frac{2}{3}x - \frac{1}{3}$.

The essence of the theorem can be first understood by looking at the special case of *monotonic* paths, i.e. paths that only ever use horizontal (resp. vertical) edges in the same direction. If points A and B are

one the same column/row then there is only one monotonic path from A to B and the result is obvious. The following illustrates the case where the slope from A to B is positive:

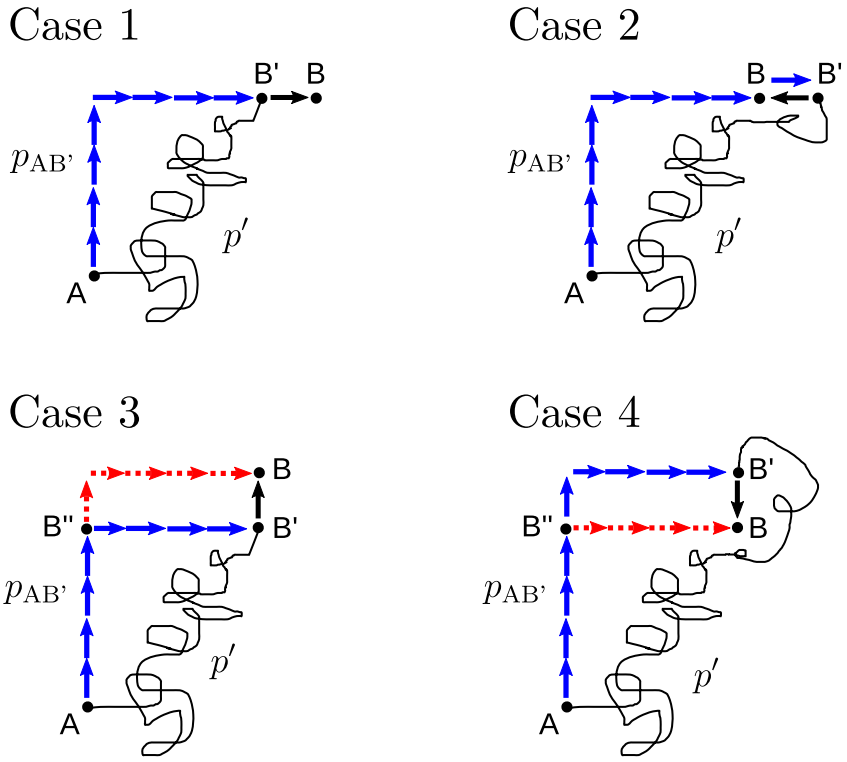


In that case, by determinism and totality of the south-east corner of the tiles, there is a unique way to tile the region north-west from the path. Because of Lemma 1.12, each successive tiling step can be used to update the path in a way that does not change the function computed by the path, i.e. each successive updated path computes the same function, $f_{AB} = x \mapsto \frac{8}{9}x - \frac{4}{9}$ in the above example. Indeed, when the path uses the motif $\rightarrow\uparrow$ we can make it commute to $\uparrow\rightarrow$ by changing the valuation depending on which tile is placed without changing the function locally computed by these edges. At the end of this process, we have transformed our initial path into its *canonical* version (in blue) which is of the form $\uparrow^n\rightarrow^m$ with $n+m$ the Manhattan distance between A and B and whose valuation is given after the tiling process is complete. Hence, when the slope from A to B is positive, all monotonic path from A to B can be transformed into the same canonical path, and they all compute the same function, f_{AB}. If the slope from A to B is negative, the same argument can be used with the canonical path $\rightarrow^n\downarrow^m$ (with $n+m$ the Manhattan distance between A and B) that is obtained by tiling north-east to the path (using the determinism and totality of the south-west corner of the tiles). Note that we have implicitly used the condition that the tiling in which the path is located is valid (Definition 1.6), especially rule 1 which ensures that there are no color mismatches (mismatches would invalidate the uniqueness of each edge's valuation) and rule 3 (Figure 1.8) which ensures that each successive tiling steps will not create color mismatches – which would invalidate the commutation property of Lemma 1.12 – since two tiles cannot "bump into each other" from opposite directions on a given row or column (which could create a mismatch if the disagreed on their shared edge). Hence, we have the result for monotonic paths.

We treat the case of arbitrary paths by induction on the length of the path. More precisely, our induction hypothesis is, for $n \in \mathbb{N}^+$, $H(n)$: "For all A and B in $\mathbb{Z}^2$, for all valued path p in $\mathcal{T}$ from A to B of length n, then the function computed by p is the same as the function computed by the canonical path p_{AB}". Where the canonical path p_{AB} is of the form $\rightarrow^n$ (resp. $\uparrow^n$) if A and B are on the same row (resp. column) with n the distance between A and B; or of the form $\uparrow^n\rightarrow^m$ (resp. $\rightarrow^n\downarrow^m$) with $n+m$ the Manhattan distance between A and B if the slope between A and B is positive (resp. negative). The valuation of the canonical path p_{AB} is given by extending the tiling $\mathcal{T}$ until the canonical path is fully tiled which is always possible (and uniquely defined) by using the determinism and totality of the south-east and south-west corners of the tiles (and, using the same argument than in the monotonic case, relying on rule 1 and rule 3 of Definition 1.6, Figure 1.8, for no color mismatches being initially present or introduced during the tiling process). Let's prove the induction:

Base case $n = 1$. If $n = 1$ then the induction hypothesis is vacuously true if the Manhattan distance from A to B is strictly more than 1 (or equal to 0). If A and B are at distance of 1 from each other then there is a unique path from A to B of length 1 , which is the canonical path p_{AB}, and the hypothesis holds.

Inductive step. Let $n \in \mathbb{N}^+$ and let's assume that $H(n)$ is true for all $1 \leq k \leq n$. Now we consider $H(n+1)$. The hypothesis is vacuously true if $M \in \mathbb{N}$, the Manhattan distance from A to B is strictly more than $n+1$ (or equal to 0). Let's suppose that that $0 < M \leq n+1$. We treat the case where the slope from A to B is positive, all other cases (A and B on the same row/column or with negative slope) can be treated similarly. Consider a valued path p in $\mathcal{T}$ from A to B of length $n+1$. There are 4 cases to study for the last edge of p (as it is either $\rightarrow$, $\leftarrow$, $\uparrow$ or $\downarrow$) and we call B' the point before B and that edge, the path p' from A to B' is of length n and by induction hypothesis p' computes the same function as its canonical version $p_{\mathrm{AB'}}$ (in blue):



Let $\phi : \mathcal{P} \to \mathcal{A}$ the morphism that converts paths to functions (Definition 1.8), we want to show that $\phi(p) = \phi(p_{\mathrm{AB}})$ and we know that $\phi(p') = \phi(p_{\mathrm{AB'}})$ by induction hypothesis ($\phi(p_{\mathrm{AB}})$ is f_{AB} from the theorem statement). Let $+$ be the concatenation operation on paths, and let's denote the last valued edge of p by $\overrightarrow{\mathrm{B'B}}$, then we have $p = p' + \overrightarrow{\mathrm{B'B}}$.

1. Case 1: $\overrightarrow{\mathrm{B'B}}$ is $\rightarrow$ (with any valuation). Then by definition of canonical paths, as illustrated above in this case we have $p_{\mathrm{AB}} = p_{\mathrm{AB'}} + \overrightarrow{\mathrm{B'B}}$. Using Definition 1.8, we have $\phi(p_{\mathrm{AB}}) = \phi(\overrightarrow{\mathrm{B'B}}) \circ \phi(p_{\mathrm{AB'}}) = \phi(\overrightarrow{\mathrm{B'B}}) \circ \phi(p') = \phi(p' + \overrightarrow{\mathrm{B'B}}) = \phi(p)$, as needed.

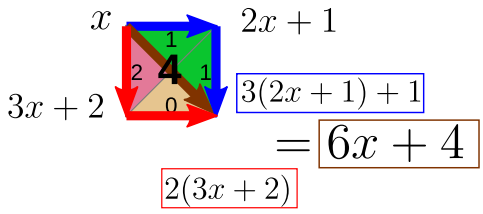
2. Case 2: $\overrightarrow{\mathrm{B'B}}$ is $\leftarrow$ (with any valuation). Then by definition canonical paths, as illustrated above in this case we have $p_{\mathrm{AB}} + \overrightarrow{\mathrm{BB'}} = p_{\mathrm{AB'}}$. Hence, $\phi(\overrightarrow{\mathrm{BB'}}) \circ \phi(p_{\mathrm{AB}}) = \phi(p')$. We have $\phi(\overrightarrow{\mathrm{B'B}}) \circ \phi(\overrightarrow{\mathrm{BB'}}) \circ \phi(p_{\mathrm{AB}}) = \phi(p' + \overrightarrow{\mathrm{B'B}})$ which gives $\phi(p_{\mathrm{AB}}) = \phi(p)$ because $\phi(\overrightarrow{\mathrm{BB'}}) \circ \phi(\overrightarrow{\mathrm{B'B}}) = x \mapsto x$ (for instance take $\overrightarrow{\mathrm{B'B}}$ to be $(\leftarrow, 1)$ then $\overrightarrow{\mathrm{BB'}}$ is $(\rightarrow, 1)$ and their respective functions are $x \mapsto (x-1)/2$ and $x \mapsto 2x+1$ which are indeed compositional inverses to one another).

3. Case 3: $\overrightarrow{\text{B'B}}$ is $\uparrow$ (with any valuation). Call B" the point where the canonical path $p_{\text{AB'}}$ switches from using $\uparrow$ to $\rightarrow$ (illustrated above). Point B" is always defined because we supposed that there is a positive slope from A to B. Then, because we have proven the result on monotonic paths, we know that the paths from B" to B respectively highlighted in dashed red and blue plus the black $\overrightarrow{\text{BB'}}$ edge in the above illustration compute the same function. Hence, from point B", we can replace the end of path $p_{\text{AB'}}$ by the dashed red portion, which gives us p_{AB} by definition of canonical paths and that still computes the same function as $p_{\text{AB'}} + \overrightarrow{\text{BB'}}$ which is the function computed by $p' + \overrightarrow{\text{BB'}} = p$. Hence, we have $\phi(p) = \phi(p_{\text{AB}})$.

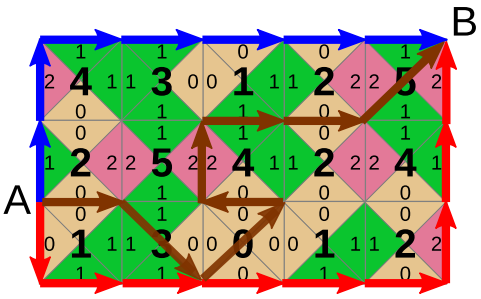
4. Case 4: $\overrightarrow{\text{B'B}}$ is $\downarrow$ (with any valuation). Call B" the point just before the canonical path $p_{\text{AB'}}$ switches from using $\uparrow$ to $\rightarrow$ (illustrated above) – it is always defined because of the positive slope between A and B. The monotonic paths between B" and B' in blue in the above illustration and dashed red path plus edge $\overrightarrow{\text{BB'}}$ compute the same function (we proved the theorem in the case of monotonic paths in the first part of this proof). Hence, we can edit $p_{\text{AB'}}$ to use the dashed red path plus edge $\overrightarrow{\text{BB'}}$ and we still get the function $\phi(p_{\text{AB'}})$. Hence we have $\phi(p_{\text{AB}} + \overrightarrow{\text{BB'}}) = \phi(p_{\text{AB'}})$. Hence $\phi(\overrightarrow{\text{BB'}}) \circ \phi(p_{\text{AB}}) = \phi(p_{\text{AB'}}) = \phi(p')$ and then $\phi(\overrightarrow{\text{B'B}}) \circ \phi(\overrightarrow{\text{BB'}}) \circ \phi(p_{\text{AB}}) = \phi(p' + \overrightarrow{\text{B'B}})$ which gives $\phi(p_{\text{AB}}) = \phi(p)$ because $\phi(\overrightarrow{\text{B'B}}) \circ \phi(\overrightarrow{\text{BB'}}) = x \mapsto x$ by the same argument as in Case 2.

In all cases we get the result when there is a positive slope from A to B. The other possibilities, i.e. negative slope from A to B or having A and B on the same column or row are proved similarly.

□



(a) Tile 4 *commutes* (Lemma 1.12) in the sense that paths $\rightarrow\downarrow$, $\downarrow\rightarrow$ and $\searrow$ compute the same function $x \mapsto 6x + 4$.



(b) All paths from A to B compute the same function (Theorem 1.13), here: $f_{AB} : x \mapsto \frac{32}{9}x + \frac{1}{9}$.

Figure 1.10: An important property of the Collatz tiles is that two paths that share the same starting and ending position compute the same function, Theorem 1.13.

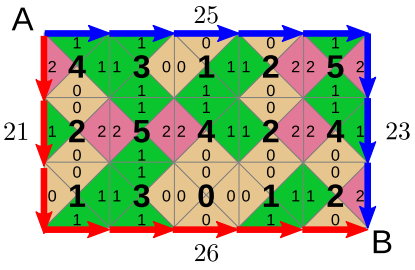
Corollary 1.14. In a valid partial tiling any valued loop – i.e. valued path that starts and ends at the same point – computes the function $x \mapsto x$.

Proof. By Theorem 1.13, a path that starts and end at position A computes the same function as the loop around one of the tiles of which A is a corner of, which can be verified to be $x \mapsto x$ for each of the six tiles. □

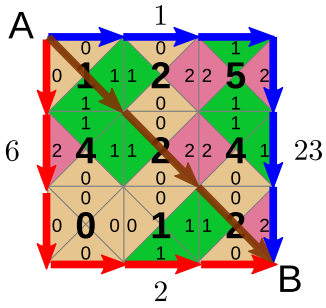
Corollary 1.15. Let $\mathcal{T}$ be a valid partial tiling such that their exists a valued path from A to B. Then we have $f_{AB} = f_{BA}^{-1}$ meaning $f_{AB} \circ f_{BA} = x \mapsto x$.

Proof. By Definition 1.6, a valid partial tiling is 8-connected, hence there is always a path between points A and B of the tiling. By Theorem 1.13, any path from A to B computes the same function hence f_{AB} is

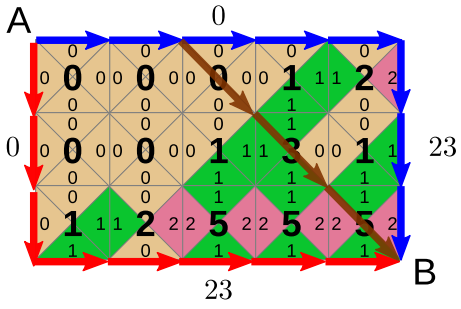
well-defined. By composing a path from A to B and from B to A we get a loop, which computes $x \mapsto x$ by Corollary 1.14, hence we have $f_{AB} \circ f_{BA} = f_{AA} = x \mapsto x$. □



(a) $3^3[\![11001]\!]_2 + [\![212]\!]_3 = 2^5[\![210]\!]_3 + [\![11010]\!]_2$, giving $27 \times 25 + 23 = 32 \times 21 + 26 = 698$.



(b) $50 = [\![122]\!]_6 = 3^3[\![001]\!]_2 + [\![212]\!]_3 = 2^3[\![020]\!]_3 + [\![010]\!]_2$.



(c) An instance of base conversion: $[\![10111]\!]_2 = [\![212]\!]_3 = [\![035]\!]_6 = 23$, Corollary 1.18.

Figure 1.11: The sides of rectangular assemblies satisfy $3^h[\![v_{\text{north}}]\!]_2 + [\![v_{\text{east}}]\!]_3 = 2^w[\![v_{\text{west}}]\!]_3 + [\![v_{\text{south}}]\!]_2$, Corollary 1.16. In the case of squares we also read this equality on the south-east diagonal interpreted in base 6, $[\![v_{\text{south-east}}]\!]_6$. When $[\![v_{\text{north}}]\!]_2 = 0$ and $[\![v_{\text{west}}]\!]_3 = 0$ we get base conversion, Corollary 1.18.

Rectangular tilings and base conversion. A direct consequence of Theorem 1.13 is the following Corollary that allows us to arithmetically interpret any rectangular tiling of the Collatz tile set:

Corollary 1.16 (Rectangles)**.** Any valid rectangular tiling of width w and height h of the Collatz tile set satisfies the following equation: $3^h[\![v_{\text{north}}]\!]_2 + [\![v_{\text{east}}]\!]_3 = 2^w[\![v_{\text{west}}]\!]_3 + [\![v_{\text{south}}]\!]_2$. Where v_{north}, v_{south}, v_{west} and v_{east} are the respective valuations of the paths along each side of the rectangle. In the special case of squares, the previous values are also equal to $[\![v_{\text{south-east}}]\!]_6$ with $v_{\text{south-east}}$ the valuation along the south-east diagonal. See Figure 1.11(a) and 1.11(b).

Proof. This is a direct application of Theorem 1.13 by considering the paths $\rightarrow^w\downarrow^h$ (in blue in Figure 1.11(a) and 1.11(b)) and $\downarrow^h\rightarrow^w$ (in red in Figure 1.11(a) and 1.11(b)) from the north-west corner to the south-east corner of the rectangle. Indeed, using Lemma 1.10, the function computed by the blue path is $f = x \mapsto 2^w3^hx + 3^h[\![v_{\text{north}}]\!]_2 + [\![v_{\text{east}}]\!]_3$ and the function computed by the red path is $f' = x \mapsto 3^h2^wx + 2^w[\![v_{\text{west}}]\!]_3 + [\![v_{\text{south}}]\!]_2$, by Theorem 1.13 we have $f = f'$ which gives $3^h[\![v_{\text{north}}]\!]_2 + [\![v_{\text{east}}]\!]_3 = 2^w[\![v_{\text{west}}]\!]_3 + [\![v_{\text{south}}]\!]_2$ by identification of polynomial coefficients, as needed. In the case of a square ($h = w$), using Lemma 1.10, the function computed along the north-west south-east diagonal is $f'' = x \mapsto 6^hx + [\![v_{\text{south-east}}]\!]_6$ and since $f = f' = f''$ by Theorem 1.13, we get the result. □

Remark 1.17 (Chinese Remainder Theorem (again) and macrotiles)**.** The south-east corner of width-w height-h rectangular tilings is solving the CRT in $\mathbb{Z}/2^w \times \mathbb{Z}/3^h \simeq \mathbb{Z}/2^w3^h$. Hence, a rectangular assembly

can be seen as a digit in base 2^w3^h (or base 6^k in the particular case of squares) and can be considered as a *macrotile*: by changing scale, we can change the base in which we are reading the tilings.

Corollary 1.18 (Base conversion)**.** Any valid rectangular tiling such that $[\![v_{\text{north}}]\!]_2 = 0$ and $[\![v_{\text{west}}]\!]_3 = 0$ satisfies $[\![v_{\text{south}}]\!]_2 = [\![v_{\text{east}}]\!]_3 = [\![v_{\text{south-east}}]\!]_6$ with v_{north}, v_{south}, v_{west} and v_{east} the respective valuations of each side of the rectangle and $v_{\text{south-east}}$ the valuation of the diagonal path that ends in the south-east corner of the rectangle. In other words, the same number is being converted between base 3, 6 and 2 within this rectangle. See Figure 1.11(c).

Proof. This is a direct consequence of Corollary 1.16: $3^h[\![v_{\text{north}}]\!]_2+[\![v_{\text{east}}]\!]_3 = 2^w[\![v_{\text{west}}]\!]_3+[\![v_{\text{south}}]\!]_2$ becomes $[\![v_{\text{south}}]\!]_2 = [\![v_{\text{east}}]\!]_3$ when $[\![v_{\text{north}}]\!]_2 = 0$ and $[\![v_{\text{west}}]\!]_3 = 0$ and we also get $[\![v_{\text{south}}]\!]_2 = [\![v_{\text{east}}]\!]_3 = [\![v_{\text{south-east}}]\!]_6$ since the valuation of the north side is 0 (brown path in Figure 1.11(c)). □

Remark 1.19. In essence, Corollary 1.18 is the same result as Theorem 11 of our previous work [156] – Theorem 16 in the arxiv version [155]. The CQCA model introduced in that work is *effective* (it is a CA augmented with the (easily, linear-time, computable) feature of quickly doing iterated divisions by 2 on an even numbers until they become odd). This in turn makes the CQCA well-suited to simulation of Collatz (and some related systems), as well as to stating and proving Collatz and Collatz-like prediction problems that rely on considering base 2 and 3 expansions. However, the more general Theorem 1.13 that we have stated here and subsequent Theorem 1.25 would have been awkward to derive in the *CQCA* framework of [156] as CQCA traces are slightly too coarse-grained compared to tiles that have glue information that clearly advertises the structure of paths. Additionally, tiles are fundamentally more general than the model of [156] because they are not restricted to the study of the Collatz problem: the Collatz process is only one particular type of assemblies, see Theorem 1.35. In this thesis we show how two other open problems relate to the tiles: Erdős' conjecture on powers of two (Chapter 2) and Mahler's 3/2 problem (Appendix B).

1.3.2 Reading the output of a path in a tiling

In Section 1.3.1 we have interpreted valued paths in tilings as abstract functions but we have not shown how to read input/output mappings of these functions in tilings. This is the goal of the current section. The types of numbers on which our functions are going to operate are the p-adic integers, and more precisely the 2-adic, 3-adic and 6-adic integers: $\mathbb{Z}_2, \mathbb{Z}_3$ and $\mathbb{Z}_6$. Although they can seem foreign, we are going to see that these numbers are particularly suited to our tilings and that they appear *naturally* when considering Collatz cycles, Section 1.5.

We refer the reader to Appendix A for more background on $\mathbb{Z}_p$, the ring of p-adic integers. We can briefly mention that $\mathbb{Z}_p$ (here $\mathbb{Z}_2$, $\mathbb{Z}_3$, $\mathbb{Z}_6$) is the ring of base-p strings that are infinite on their most significant side. In $\mathbb{Z}_p$, we have[19] $\lim_{n\to\infty} p^n = 0$. $\mathbb{Z}_p$ is uncountable and contains $\mathbb{N}$ (numbers with infinitely many leading 0s), $\mathbb{Z}$ (numbers with infinitely many leading digit $p-1$) and almost all[20] of $\mathbb{Q}$ (numbers with eventually repeating representation). There is no ring isomorphism ($\simeq$) from $\mathbb{Z}_p$ to $\mathbb{R}$. For example, $\frac{1}{2} \notin \mathbb{Z}_2$ but $\sqrt{-7} \in \mathbb{Z}_2$, two reasons for which $\mathbb{Z}_2 \not\simeq \mathbb{R}$. Similarly, $\mathbb{Z}_p \not\simeq \mathbb{Z}_q$ when $p \neq q$, meaning that $\mathbb{Z}_2$, $\mathbb{Z}_3$ and $\mathbb{Z}_6$ represent different *kinds* of numbers, see Appendix A.

Definition 1.20 (2-adic, 3-adic and 6-adic value of a point $A \in \mathbb{Z}^2$)**.** Let $\mathcal{T}$ be a valid partial tiling and $A = (x, y) \in \mathbb{Z}^2$. Figure 1.12 illustrates the following:

[19] This – at first – strange-looking equality can be interpreted visually: p^n in base p consists of a 1 followed by n 0s. Hence, the bigger n is the higher in the digits of p^n you need to look to differentiate it from 0, which is the meaning of $\lim_{n\to\infty} p^n = 0$.

[20] $\mathbb{Z}_p$ contains exactly the subset of $\mathbb{Q}$ of fractions whose denominator is not a multiple of p.

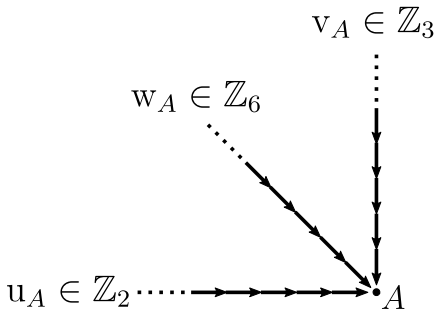


Figure 1.12: 2-adic, 3-adic and 6-adic value of a point $A \in \mathbb{Z}^2$ in a tiling, Definition 1.20.

- If for all $i \in \mathbb{N}$ the edge $(x-i-1, y)$ to $(x-i, y)$ is valued in $\mathcal{T}$ by $b_i \in \{0,1\}$, then the *2-adic value of* A is $\mathrm{u}_A = \ldots b_2 b_1 b_0 \in \mathbb{Z}_2$.
- If for all $i \in \mathbb{N}$ the edge $(x, y-i-1)$ to $(x, y-i)$ is valued in $\mathcal{T}$ by $t_i \in \{0,1,2\}$, then the *3-adic value of* A is $\mathrm{v}_A = \ldots t_2 t_1 t_0 \in \mathbb{Z}_3$.
- If for all $i \in \mathbb{N}$ the edge $(x-i-1, y-i-1)$ to $(x-i, y-i)$ is valued in $\mathcal{T}$ by $s_i \in \{0,1,2,3,4,5\}$, then the *6-adic value of* A is $\mathrm{w}_A = \ldots s_2 s_1 s_0 \in \mathbb{Z}_6$.

Remark 1.21. Definition 1.20 can be interpreted in terms of convergence of sequences of functions. Indeed, if we take a length-n finite suffix of the left-infinite horizontal path that defines the 2-adic value $v \in \mathbb{Z}_2$ of some point in a tiling we get the path $\to \cdots \to$ valued $b_{n-1} \ldots b_0 \in \{0,1\}^n$. By Lemma 1.10, we know that the associated function is $f_n = x \mapsto 2^n x + [\![b_{n-1} \ldots b_0]\!]_2$. The sequence of functions $(f_n)_{n\in\mathbb{N}}$ converges *uniformly*[21] in $\mathbb{Z}_2$ to the constant function $f = x \mapsto v$, which justifies mapping this left-infinite path to the value $v \in \mathbb{Z}_2$. In other words, while a finite horizontal path can be seen as the action of appending a binary suffix to a number, a left-infinite horizontal path corresponds to an entire number of $\mathbb{Z}_2$ itself. The same remark goes for vertical paths and $\mathbb{Z}_3$ and south-east diagonal paths and $\mathbb{Z}_6$.

Remark 1.22 (Where are the reals?)**.** The theory that is developed here has a strong focus on p-adic integers, i.e. numbers with an infinite base p expansion on the most significant side. It is legitimate to wonder if there is room for the more familiar real numbers in our tilings and it turns out that there is! Let $\alpha = 0.b_0 b_1 b_2 \cdots \in [0,1]$ be a unit-interval real expressed with binary "decimals". Then, the right-infinite path where each edge $(i+1, y)$ to (i, y) for $i \in \mathbb{N}$ is valued by b_i can be interpreted as being α, in a very similar way to Remark 1.21. The same goes vertically with base 3 expansions of reals and diagonally for base 6 expansions; we call these representations of the reals $\mathbb{R}_2$, $\mathbb{R}_3$ and $\mathbb{R}_6$, Figure 1.13. Complications arise when looking at reals in tilings: (a) representation is not unique, for instance, $0.01111\ldots = 0.10000\ldots$ in $\mathbb{R}_2$ and (b) these representations involves use of the north-west corners of the tiles which is not deterministic. For these reasons, and especially (b), we do not try to extend the theory to reals. Nonetheless, $\mathbb{R}_6$ is used in a beautiful way to reformulate Mahler's 3/2 problem with the six tiles, Appendix B.

After choosing some $A \in \mathbb{Z}^2$, the numbers/strings $\mathrm{u}_A \in \mathbb{Z}_2$ and $\mathrm{v}_A \in \mathbb{Z}_3$ are two degrees of freedom that one can use to uniquely define the north-west quadrant of a tiling anchored by its south-east corner A. In particular, because the south-east corners of the six tiles are both deterministic and total (see Figure 1.7), any choice of $(\mathrm{u}_A, \mathrm{v}_A) \in \mathbb{Z}_2 \times \mathbb{Z}_3$ yields a valid, unique and complete tiling of the north-west quadrant.

[21] The result comes from the fact that for all $x \in \mathbb{Z}_2$ and $n \in \mathbb{N}$, $f_n(x) - f(x)$ ends with n zeros. See Chapter 8 in [3] which is about real functions but that also applies in $\mathbb{Z}_2$ (or more precisely, in $\mathbb{Q}_2$, see Appendix A) in our case. Uniform convergence of a sequence of functions is stronger than the more intuitive *pointwise* convergence which asks that $\lim_{n\to\infty} f_n(x) = f(x)$ for all x in the domain of f and here directly comes from $\lim_{n\to\infty} 2^n = 0$ in $\mathbb{Z}_2$.

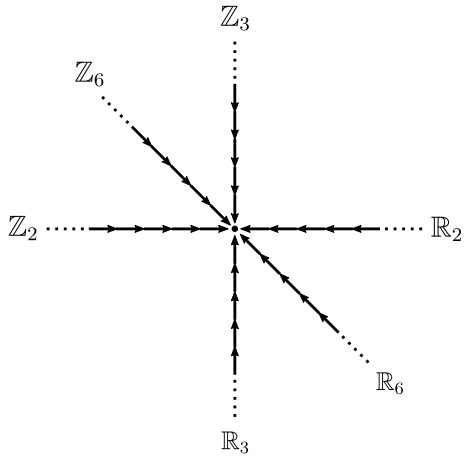


Figure 1.13: Reals in base 2,3 and 6 can also be read in tilings.

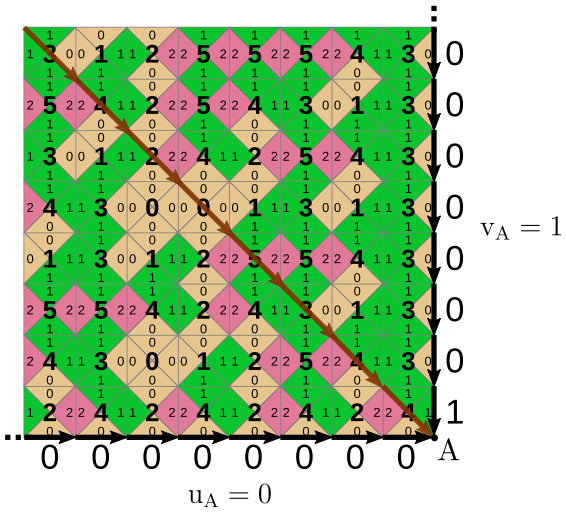
Reciprocally, if in the north-west quadrant we set the semi-infinite south-east-pointing diagonal $\mathrm{w}_A \in \mathbb{Z}_6$ (ending at A), we get a valid, unique, and complete tiling of the quadrant and thus well-defined values of $(\mathrm{u}_A, \mathrm{v}_A) \in \mathbb{Z}_2 \times \mathbb{Z}_3$: that is because the south-west and north-east corners of the tiles are deterministic too (i.e., by tiling to the north-east and south-west from the diagonal w_A). This fact is yet another manifestation of the Chinese Remainder Theorem generalised to $\mathbb{Z}_2 \times \mathbb{Z}_3 \simeq \mathbb{Z}_6$ (ring isomorphism). We concretise this fact in the following:

Lemma 1.23. Let $\mathcal{T}$ be a valid partial tiling and $A \in \mathbb{Z}^2$ be such that all the positions in the north-west quadrant anchored at A are tiled. Then A has a 6-adic value $\mathrm{w}_A \in \mathbb{Z}_6$ if and only if it has 2-adic and 3-adic values $(\mathrm{u}_A, \mathrm{v}_A) \in \mathbb{Z}_2 \times \mathbb{Z}_3$. The mapping $\psi : \mathbb{Z}_2 \times \mathbb{Z}_3 \to \mathbb{Z}_6$ such that $\psi(\mathrm{u}_A, \mathrm{v}_A) = \mathrm{w}_A$ is a ring isomorphism, i.e. $\mathbb{Z}_2 \times \mathbb{Z}_3 \simeq \mathbb{Z}_6$.

Proof. If A has a 6-adic value then, by determinism and totality of the south-west and north-east corners of the tiles, we can reconstruct its uniquely corresponding 2-adic and 3-adic values. Reciprocally, if A has a 2-adic and a 3-adic value, by totality and determinism of the the south-east corner of the tiles, we can reconstruct its uniquely corresponding 6-adic value. The fact that the mapping $\psi : \mathbb{Z}_2 \times \mathbb{Z}_3 \to \mathbb{Z}_6$ such that $\psi(\mathrm{u}_A, \mathrm{v}_A) = \mathrm{w}_A$ is a ring isomorphism, which means that $\psi(1,1) = 1$ and $\psi(\mathrm{u} + \mathrm{u}', \mathrm{v} + \mathrm{v}') = \psi(\mathrm{u}, \mathrm{v}) + \psi(\mathrm{u}', \mathrm{v}')$ and $\psi(\mathrm{u} \times \mathrm{u}', \mathrm{v} \times \mathrm{v}') = \psi(\mathrm{u}, \mathrm{v}) \times \psi(\mathrm{u}', \mathrm{v}')$, is directly obtained by extending to the limit the Chinese Remainder Theorem which gives ring isomorphisms $\mathbb{Z}_{2^k} \times \mathbb{Z}_{3^k} \simeq \mathbb{Z}_{6^k}$ for all $k \in \mathbb{N}$ [80]. □

Remark 1.24 (Aperiodicity of 6-adic values)**.** In practice, it is easier to work with the pair $(\mathrm{u}_A, \mathrm{v}_A) \in \mathbb{Z}_2 \times \mathbb{Z}_3$ rather than with their associated 6-adic value $\mathrm{w}_A \in \mathbb{Z}_6$ (Lemma 1.23) because w_A is typically *aperiodic*, hence hard to describe. For instance, if both u_A and v_A are eventually periodic, i.e. $\mathrm{u}_A \in \mathbb{Z}_2 \cap \mathbb{Q}$ and $\mathrm{v}_A \in \mathbb{Z}_3 \cap \mathbb{Q}$, then w_A is eventually periodic if and only if $\mathrm{u}_A = \mathrm{v}_A$. Said otherwise, as soon as $\mathrm{u}_A \neq \mathrm{v}_A$ (i.e. u_A and v_A do not represent the same rational number), we have that w_A is aperiodic, or equivalently, $\mathrm{w}_A \notin \mathbb{Q}$. See Appendix A for a proof. The implication in terms of tilings is that simple choices of u_A and v_A can lead to complex tilings of the north-west quadrant, for instance choosing $\mathrm{u}_A = 0$ (south-most border) and $\mathrm{v}_A = 1$ (east-most border) gives[22]:

[22] Matthew Cook pointed out that we give here an algorithm for computing OEIS sequence `https://oeis.org/A055620` and that choosing $\mathrm{u_A} = 1$ and $\mathrm{v_A} = 0$ gives `https://oeis.org/A054869`.



We can see that despite its simple south and east borders – which uniquely determine the tiling because of the determinism and totality of the south-east corner of the tiles – this tiling is complex, and its south-east diagonal (brown), which corresponds to $w_A \in \mathbb{Z}_6$, is not periodic (no matter how much we extend the tiling by adding leading 0s on the south and east sides).

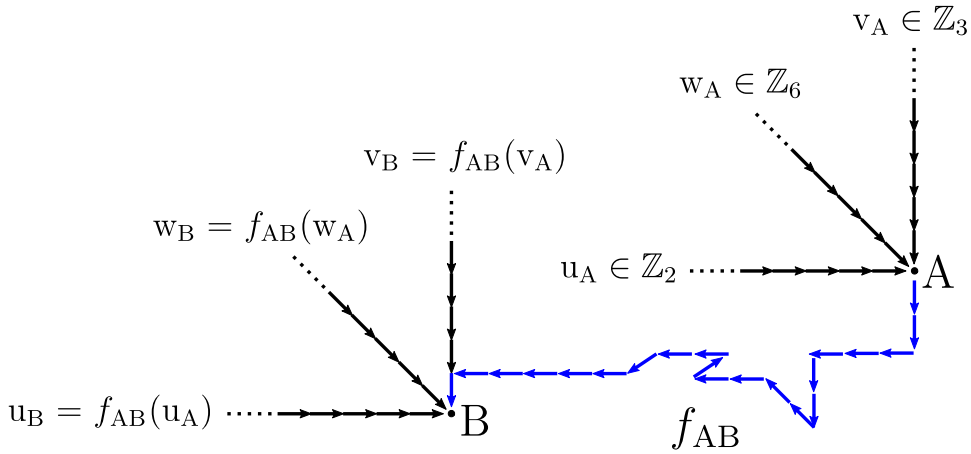


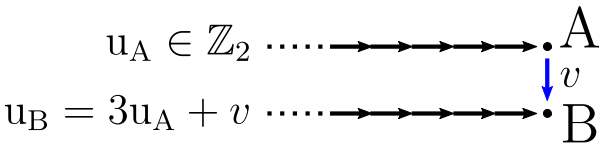
Figure 1.14: Computing with the blue valued path from A to B in an (invisible) tiling (Theorem 1.25). The function of that path, f_{AB}, is independent from the path (Theorem 1.13).

Theorem 1.25. Let $\mathcal{T}$ be a valid partial tiling and A and B two points in $\mathbb{Z}^2$ such that there exists a valued path in $\mathcal{T}$ from A to B computing f_{AB} (see Theorem 1.13). Assume that for any $p \in \{2, 3, 6\}$, the p-adic values z_A and z_B, of A and B, are defined in $\mathcal{T}$. Then, $z_B = f_{AB}(z_A)$. See Figure 1.14.

Proof. It is enough to prove the result for $p \in \{2, 3\}$ since, by Lemma 1.23, thanks to the ring isomorphism $\mathbb{Z}_2 \times \mathbb{Z}_3 \simeq \mathbb{Z}_6$, performing operations in $\mathbb{Z}_6$ is equivalent to performing operations in parallel in $\mathbb{Z}_2$ and $\mathbb{Z}_3$, i.e. if we are given a 6-adic value for points A and B we can compute their corresponding 2-adic and 3-adic values and work with them directly: the ring isomorphism ensures that operations performed on them will also apply to their associated 6-adic values. Because of path composition (Definition 1.8) it is enough to show the result for paths consisting of only one valued edge. Furthermore, we can further restrict to edges $\leftarrow$, $\rightarrow$, $\uparrow$ and $\downarrow$ since diagonal edges can be obtained by composing horizontal and vertical edges. Cases where $p = 2$ and the edge is horizontal (resp. $p = 3$ and the edge is vertical) are immediate since by definition, the operation of f_{AB} merely consists in adding/removing a trailing binary symbol (resp. ternary symbol). The cases that remain to be studied are when $p = 2$ and the edge is vertical and

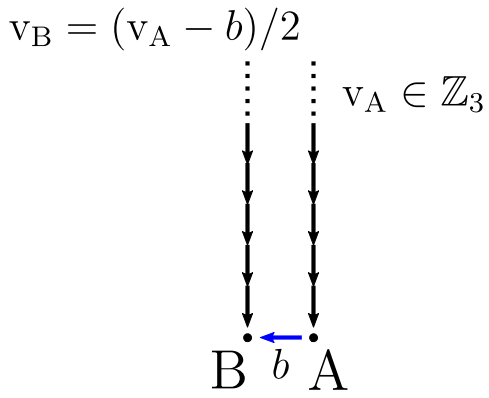
when $p = 3$ and the edge is horizontal. By reversibility of the operations computed by edges we are left with only 2 cases to study, where the vertical edge is $\downarrow$ and the horizontal edge is $\leftarrow$:

- **Case 1:** $p = 2$ and the valued edge between A and B is $(\downarrow, v)$ with $v \in \{0, 1, 2\}$. We need to show that $\mathrm{u_B} = f_{\mathrm{AB}}(\mathrm{u_A}) = 3\mathrm{u_A} + v$ with $\mathrm{u_A}$ the 2-adic value of A and $\mathrm{u_B}$ the 2-adic value of B:



  Call $v_0 = v$ and v_n for $n \in \mathbb{N}^+$ the valuation of the n^{th} vertical edge parallel to the west of v. Call $\mathrm{u_A}^{(n)}$ the last n bits of $\mathrm{u_A}$ and $\mathrm{u_B}^{(n)}$ the last n bits of $\mathrm{u_B}$. Then, by Corollary 1.16 we have: $3\mathrm{u_A}^{(n)} + v = 2^n v_n + \mathrm{u_B}^{(n)}$. Knowing that $\lim_{n\to\infty} 2^n v_n = 0$ in $\mathbb{Z}_2$ (cf. Appendix A) we get $3\mathrm{u_A} + v = \mathrm{u_B}$ when $n \to \infty$, as needed. Note: the sequence of trits v_n gives the succession of states that the minimised "$3x + i$" finite state transducer, Figure 1.2(b), will traverse when processing input $\mathrm{u_A}$ (which outputs $\mathrm{u_B}$).

- **Case 2:** $p = 3$ and the valued edge between A and B is $(\leftarrow, b)$ with $b \in \{0, 1\}$. We need to show that $\mathrm{v_B} = f_{\mathrm{AB}}(\mathrm{v_A}) = (\mathrm{v_A} - b)/2$ with $\mathrm{v_A}$ the 3-adic value of A and $\mathrm{v_B}$ the 3-adic value of B:



  Call $b_0 = b$ and b_n for $n \in \mathbb{N}^+$ the valuation of the n^{th} horizontal edge parallel to the north of b. Call $\mathrm{v_A}^{(n)}$ the last n trits of $\mathrm{v_A}$ and $\mathrm{v_B}^{(n)}$ the last n trits of $\mathrm{v_B}$. Then, by Corollary 1.16 we have: $3^n b_n + \mathrm{v_A}^{(n)} = 2\mathrm{v_B}^{(n)} + b$ which rewrites into $\mathrm{v_B}^{(n)} = (3^n b_n + \mathrm{v_A}^{(n)} - b)/2$. When $n \to \infty$ we get $\mathrm{v_B} = (\mathrm{v_A} - b)/2$ since $\lim_{n\to\infty} 3^n b_n = 0$ in $\mathbb{Z}_3$ (cf. Appendix A), as needed. Note: the sequence of bits b_n gives the succession of states that the minimised "$(x - i)/2$" finite state transducer, Figure 1.5(b), will traverse when processing input $\mathrm{v_A}$ (which outputs $\mathrm{v_B}$).

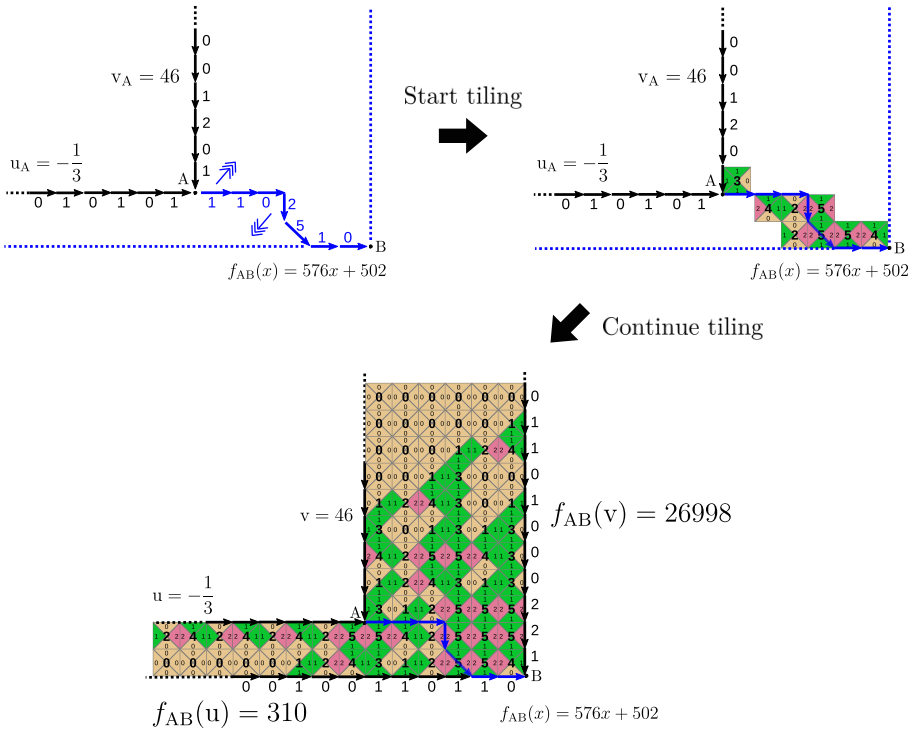
In all the cases we get the result.

□

Remark 1.26 (Two birds one stone)**.** In Theorem 1.25, we see that f_{AB} is applied in parallel on both the 2-adic and the 3-adic value of point A (or equivalently, on its 6-adic value). This idea is put in practice in Example 1.27.

Example 1.27. Although Theorem 1.25 is a statement about tilings that are already formed, we can use it in practice to compute the output of the function of a given path on some given 2-adic/3-adic/6-adic inputs. Let's compute the output of function $f_{\mathrm{AB}} : x \mapsto 576x + 502$, defined by the blue path

between A and B below (valued 1,1,0,2,5,1,0), on 2-adic input $u_A = -1/3 = \dots 010101 \in \mathbb{Z}_2$ (i.e. pattern 01 repeated forever[23]), and 3-adic input $v_A = 46 = \dots 001201 \in \mathbb{Z}_3$ positioned at point A. Specifying these allows for a unique tiling grown north-east and south-west of the path (we are only using corners of the tiles that are deterministic and total). By Theorem 1.25, we expect to read outputs $f_{AB}(-1/3) = 310 = \dots 00100110110 \in \mathbb{Z}_2$ and $f_{AB}(46) = 26998 = \dots 01101000221 \in \mathbb{Z}_3$ once the tiling reveals the 2-adic and 3-adic values of point B:[24]



For the sake of brevity, we do not explore all the possible paths that one could consider computing with. Important remarks are that (a) one can focus on monotonic paths only (i.e. the sequence of coordinates along the path are monotonic in x in y), such as the blue path above, since, if there is a path from A to B in a valid tiling then there is also a monotonic path and they compute the same function (Theorem 1.13) and (b) some monotonic paths may constrain their inputs: a canonical example is that the path consisting only of valued edge $(\leftarrow, 0)$ constraints a 2-adic input to have its last bit equal to 0 since, otherwise, the operation $x \mapsto x/2$ is not defined[25] in $\mathbb{Z}_2$. These constraints can be computed with the tiles and are "only" fixing the last n digits of an input to a particular value with n the length of the path, i.e. the domain of the function computed by the path if of the form $\alpha + p^n\mathbb{Z}_p$, with $p \in \{2, 3, 6\}$ and $\alpha < p^n$, with p depending on the case considered. For instance, in Figure 1.16, the given path of length 12 is forcing 2-adic inputs to end in 100001011001.

[23] To get convinced that $x = \dots 010101$ represents $-1/3$, note that $2x = \dots 0101010$, hence $x + 2x = \dots 1111 = -1$. We can also get this 2-adic expansion thanks to the algorithm given in Appendix A, Remark A.4.

[24] Note that we have not bothered to tile the region north-west of A, as it is "only" computing the (aperiodic) 6-adic value of A associated to $(-1/3, 46) \in \mathbb{Z}_2 \times \mathbb{Z}_3$, Lemma 1.23 and Remark 1.24.

[25] Indeed, $\mathbb{Z}_2$ is not a field meaning that not all divisions are allowed, see Appendix A.

1.3.3 Tiling Collatz sequences

We finally get to the moment where we relate our 6 tiles to Collatz sequences. We do so through the concept of a *parity vector* which has been central in the study of the Collatz process [161, 178, 139]:

Definition 1.28 (Parity vectors)**.** A *parity vector* is a finite sequence of bits, i.e. an element of $\{0,1\}^n$ for some $n \in \mathbb{N}$. An *infinite parity vector* is an infinite sequences of bits, i.e. an element of $\{0,1\}^{\mathbb{N}}$. The *infinite Collatz parity vector* of $x \in \mathbb{Z}_2$ is the infinite sequence of bits $T^n(x) \bmod 2$ for $n \in \mathbb{N}$ and T the Collatz map $T(x) = x/2$ if $x \in \mathbb{Z}_2$ is even and $T(x) = (3x+1)/2$ if it is odd.

Example 1.29. The infinite Collatz parity vector of $x = 73 \in \mathbb{Z}_2$ starts with $[1,0,1,1,1,0,0,1,1,1,1,\dots]$ since the first 11 T-iterates of x are: $[73, 110, 55, 83, 125, 188, 94, 47, 71, 107, 161, \dots]$.

Definition 1.30 ($Q : \mathbb{Z}_2 \to \mathbb{Z}_2$)**.** The infinite Collatz parity vector of $x \in \mathbb{Z}_2$ can be identified with the 2-adic integer sharing the same sequence of bits and is denoted $Q(x) \in \mathbb{Z}_2$ in that case. In other words, the function $Q : \mathbb{Z}_2 \to \mathbb{Z}_2$ maps a 2-adic integer to its infinite Collatz parity vector, seen as a 2-adic integer.

The following is known about Q [139]:

1. Q is a bijection. This means that distinct inputs in $\mathbb{Z}_2$ to the Collatz process will have distinct infinite parity vectors and that for any infinite parity vector, there exists an input in $\mathbb{Z}_2$ whose Collatz sequence follows that parity vector.
2. If x and y in $\mathbb{Z}_2$ share their last n bits, i.e. $x \equiv y \bmod 2^n$, then $Q(x)$ and $Q(y)$ also share their last n bits, i.e. $Q(x) \equiv Q(y) \bmod 2^n$. This implies that Q is continuous everywhere in $\mathbb{Z}_2$.
3. Q is nowhere differentiable [15].
4. If $Q(x)$ is eventually periodic then the Collatz sequence of x is also eventually periodic (i.e. reaches a cycle in $\mathbb{Z}_2$).
5. If $Q(x)$ is eventually periodic then x is also eventually periodic, i.e. $Q(x) \in \mathbb{Q} \Rightarrow x \in \mathbb{Q}$.
6. Lagarias' periodicity conjecture (c.f. Section 1.1) states that the Collatz sequence of an eventually periodic number in $\mathbb{Z}_2$ is eventually periodic. That is equivalent to the converse of Point 5 together with Point 4: the conjecture states that $Q(\mathbb{Q} \cap \mathbb{Z}_2) \subseteq \mathbb{Q}$.

Remark 1.31 (Are parity vectors *numbers*?)**.** In Appendix A, Remark A.4, we make the point that the 2-adic representation of a rational 2-adic $x \in \mathbb{Z}_2$ (or in fact *any* 2-adic) can be seen as being the parity vector of x under *the* "trivial" Collatz-like map $x \mapsto x/2$ if x is even (least significant bit is 0) and $x \mapsto (x-1)/2$ if x is odd (least significant bit is 1) – this map is also known as the *shift map*[26]. This, plus the fact that Q is bijective, forces us to wonder if Collatz parity vectors could be thought of as numbers expressed in the "Collatz-base". The main problem of this analogy is that we do not know how to perform any operation in that "base", i.e. $Q(x+y)$ and $Q(x \times y)$, in general, do not seem to be related to $Q(x)$ and $Q(y)$.

Remark 1.32 (Predicting parity vectors)**.** The infinite Collatz parity vector of $x \in \mathbb{Z}_2$ can be thought as giving which of the maps $x \mapsto x/2$ or $x \mapsto (3x+1)/2$ is used at each step of the Collatz sequence of x. Infinite parity vectors can be easily generalised to Generalised Collatz Maps (Section 1.1) by looking at the sequence giving which of the N maps of a GCM is used at each step of processing a particular input. Computationally speaking, infinite parity vectors are *natural* objects to study since they

[26] This is because it *shifts* the representation of a 2-adic integer by one to the right, i.e. it gets rid of the least significant bit. It is known that $T = Q^{-1} \circ \mathcal{S} \circ Q$ [15, 139].

represent the sequence of *states* in which a GCM will be while processing a given input. Since GCMs are Turing-complete, we know that predicting parity vectors in general is arbitrarily hard. In the specific case of Collatz, although it is in P, the exact complexity of predicting parity vectors arbitrarily far in the future is unknown, see Section 1.4, Remark 1.32. Nonetheless, it is expected to be non-trivial as it is not even known if an eventually periodic x produces an eventually periodic infinite parity vector (Lagarias' periodicity conjecture).

We are going to associate parity vectors to a particular kind of infinite valued paths in a straightforward way:

Definition 1.33 (Path representation of a parity vector)**.** Consider a parity vector, finite or infinite (Definition 1.28). Replace bits equal to 0 with valued edge $(\leftarrow, 0)$ and bits equal to 1 with valued edge $(\swarrow, 4)$, the resulting sequence defines a valued path that we call the path representation of a parity vector and is denoted $p \in \{0,1\}^*$ in the finite case and $\mathfrak{p} \in \{0,1\}^{\mathbb{N}}$ in the infinite case. From now on, we identify Collatz parity vectors with their path representation. Similarly, from now on we define the function Q to be from $\mathbb{Z}_2$ to $\{(\leftarrow, 0), (\swarrow, 4)\}^{\mathbb{N}}$.

Remark 1.34. The function computed by $(\leftarrow, 0)$ is $x \mapsto x/2$ and the function computed by $(\swarrow, 4)$ is $x \mapsto (3x+1)/2$, that is T to a T!

Theorem 1.35 (Tiling Collatz sequences)**.** Let $\mathfrak{p} \in \{(\leftarrow, 0), (\swarrow, 4)\}^{\mathbb{N}}$ be the path representation of the infinite Collatz parity vector of some $z \in \mathbb{Z}_2$ (Definition 1.33). Call $A_n = (\mathrm{x}_n, \mathrm{y}_n) \in \mathbb{Z}_2$ each point along $\mathfrak{p}$ for $n \in \mathbb{N}$ and let $A_0 = (0,0)$. Then, there is a unique valid tiling of the region north-west from $\mathfrak{p}$ delimited to the north by $y = 0$, i.e. $\{(x,y) \mid y \leq 0 \text{ and } x \leq \mathrm{x}_{|y|}\} \subseteq \mathbb{Z}^2$. In that tiling, $\mathrm{u}_n \in \mathbb{Z}_2$ the 2-adic value of A_n is defined for all n and we have: $u_0 = z = Q^{-1}(\mathfrak{p})$ and $u_n = T^n(u_0)$. In that sense, we have tiled the Collatz sequence of z, starting from its parity vector $\mathfrak{p} \in \{(\leftarrow, 0), (\swarrow, 4)\}^{\mathbb{N}}$. See Figure 1.15.

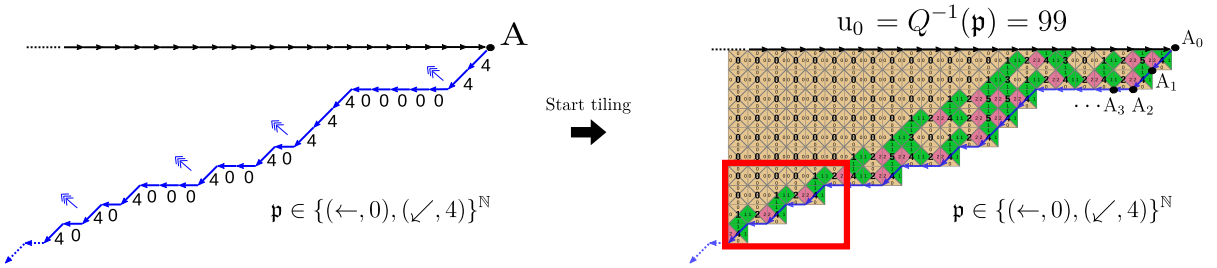
Proof. The fact that there is a unique valid tiling of the region north-west from $\mathfrak{p}$ delimited to the north by $y = 0$ directly comes from the determinism and totality of the south-east corner of the tiles.

Once we have the tiling, the result directly follows from Theorem 1.25 since the edge $(\leftarrow, 0)$ computes the function $x \mapsto x/2$ and forces the last bit of x to be 0, i.e. is applied when x is even and $(\swarrow, 4)$ computes the function $x \mapsto (3x+1)/2$ and forces the last bit of x to be 1, i.e. is applied when x is odd. Hence, by iteration, rows after rows, we read u_n (the 2-adic value of A_n) which gives the successive Collatz-iterates of $u_0 = z = Q^{-1}(\mathfrak{p})$ in $\mathbb{Z}_2$. □

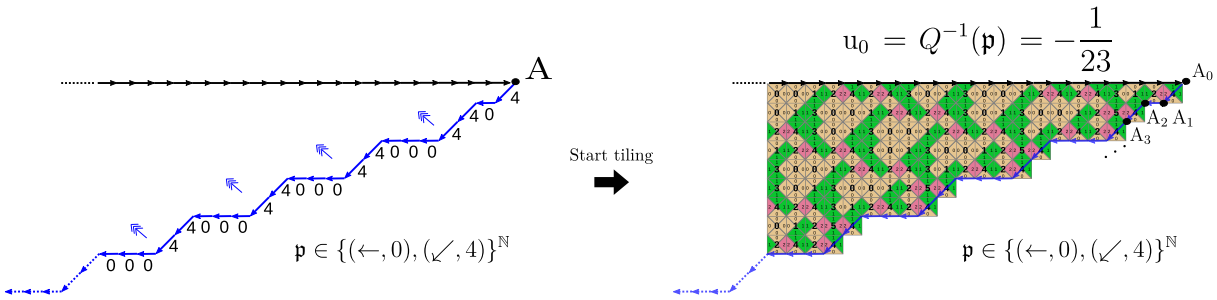
Finite parity vectors can be interpreted in terms of finite Collatz sequences, restricted to the natural numbers, they will prove useful when thinking about Collatz cycles, Section 1.5:

Corollary 1.36. Let $p \in \{(\leftarrow, 0), (\swarrow, 4)\}^n$ be the path representation of a finite parity vector (Definition 1.33). Then, there is a unique tiling of the finite triangular region north-west from p, and we have $\beta = T^n(\alpha)$ with α and β natural numbers respectively given in binary and ternary on the north and west sides of that triangular region, see Figure 1.16.

Proof. Uniqueness of the tiling is immediate by determinism and totality of the south-east corner of the tiles. The fact that $\beta = T^n(\alpha)$ is a consequence of Theorem 1.35: if we embed this finite tiling in a bigger tiling where α has been prepended with an infinite amount of 0s, we know that each row gives successive Collatz iterate in $\mathbb{Z}_2$, and even $\mathbb{N}$ since $\alpha \in \mathbb{N}$. Now, by base conversion (Corollary 1.18), the last row (which is $T^n(\alpha)$) is also given in base-3 by the column where we read β. □



(a) Tiling the Collatz sequence of 99 from $\mathfrak{p} \in \{(\leftarrow, 0), (\swarrow, 4)\}^{\mathbb{N}}$, the path representation of its infinite Collatz parity vector (Definition 1.33). Intuitively, the parity vector defines a path upon which we can tile the Collatz iterations of 99, but backwards (the input is the parity vector, the output is 99). We get $u_0 = [\![1100011]\!]_2 = 99$ and, for instance, $u_3 \in \mathbb{Z}_2$, the 2-adic value of point A_3, satisfies $u_3 = [\![1110000]\!]_2 = 112 = T^3(99)$. There is one odd iterate per horizontal row. The Collatz sequence of 99 eventually reaches the trivial cycle $2, 1$ of which we see 3 repetitions here (highlighted in red). The trivial cycle's path repeats the pattern $(\leftarrow, 0), (\swarrow, 4)$ for ever, corresponding to alternating even/odd parities.



(b) Tiling the Collatz sequence of $-\frac{1}{23}$ from $\mathfrak{p} \in \{(\leftarrow, 0), (\swarrow, 4)\}^{\mathbb{N}}$, the path representation of its infinite Collatz parity vector (Definition 1.33). We get $u_0 = (00100001011)^{\infty}001 \in \mathbb{Z}_2 = -\frac{1}{23}$ (we can compute this 2-adic expansion thanks to the algorithm given in Appendix A, Remark A.4), and, for instance, $u_3 \in \mathbb{Z}_2$, the 2-adic value of point A_3, satisfies $u_3 = (00010110010)^{\infty}1 \in \mathbb{Z}_2 = \frac{19}{23} = T^3(-\frac{1}{23})$. There is one odd (i.e. last significant bit 1) iterate per horizontal row. The Collatz sequence of $-\frac{1}{23}$ eventually reaches the cycle $\frac{5}{23}$, $\frac{19}{23}$, $\frac{40}{23}$, $\frac{20}{23}$, $\frac{10}{23}$ of which we see 4 repetitions here. This cycle's path repeats the pattern $(\swarrow, 4), (\swarrow, 4), (\leftarrow, 0), (\leftarrow, 0), (\leftarrow, 0)$ for ever, corresponding to odd/odd/even/even/even parities.

Figure 1.15: Tiling the Collatz sequences of 99 and $-\frac{1}{23}$ from their respective infinite Collatz parity vector, see Theorem 1.35.

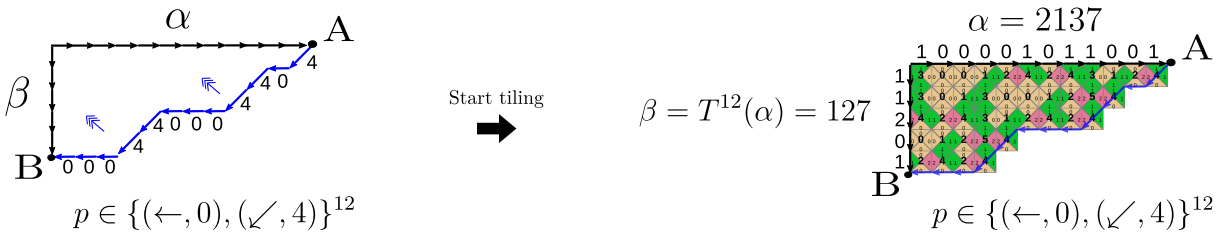


Figure 1.16: Tiling north-west from the path representation of length-12 parity vector $p = [1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0]$ gives $T^{12}(2137) = 127$ with 2137 and 127 respectively given in binary and ternary on the north and west sides of the final assembly. The first 12 Collatz iterates of 2137 follow the parities given by p.

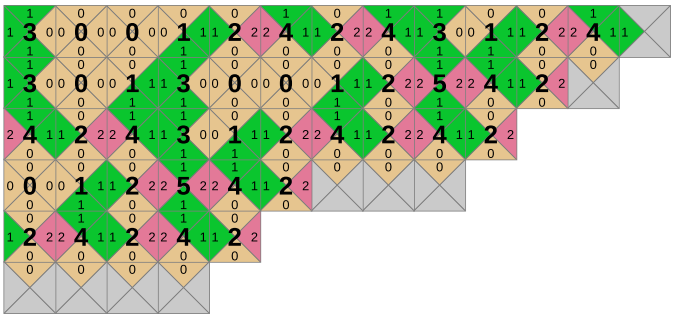
Remark 1.37 (`coreli`)**.** You can compute the pair (α, β) for an arbitrary parity vector using our library `coreli v0.0.3` (see Appendix C), same example as Figure 1.16:

```
>>> from coreli import ParityVector
>>> pv = ParityVector([1,0,1,1,0,0,0,1,1,0,0,0])
>>> pv.first_occurrence()
(2137, 127)
>>> pv.first_occurrence(symbolic=True)
('100001011001', '11201')
```

You can also get the tiling corresponding to a parity vector, see Appendix C:

```
>>> from coreli import ParityVector
>>> pv = ParityVector([1,0,1,1,0,0,0,1,1,0,0,0])
>>> tiling = pv.to_tiling()
>>> tiling.all_steps()
>>> tiling.draw_svg().saveSvg("tiling.svg")
```

Will get you the following image which is (almost[27]) the same as Figure 1.16:



Remark 1.38 (Explicit formulae for α and β)**.** While Corollary 1.36 gives a way to reconstruct the α and β of a parity vector (Figure 1.16) via the tiles, there are explicit formulae that are known in the literature [178, 139]. If we call $(p_0, \ldots, p_{n-1}) \in \{0,1\}^n$ the parity vector and $\sigma_i = \sum_{j=0}^{i} p_j$ its *partial sums* and $k = \sigma_{n-1}$ the number of 1s in the parity vector (i.e. the number of odd terms), then we have [178, 139]:

$$\alpha \equiv -\sum_{i=0}^{n-1} p_i 2^i 3^{-\sigma_i} \bmod 2^n \tag{1.1}$$

$$\beta \equiv \sum_{i=0}^{n-1} p_i 2^{-(n-i)} 3^{k-\sigma_i} \bmod 3^k \tag{1.2}$$

Note that these formulae respectively give us the last n bits of α and last k trits of β, which is the same that we get by tiling since n is the width of the assembly and k its height (Figure 1.16) – our α and β are the smallest representants of these congruence classes. Let's interpret the formula of α, that we slightly rewrite $\alpha \equiv \sum_{i=0}^{n-1} p_i 2^i (-3^{-\sigma_i}) \bmod 2^n$. Let's first interpret the term $(-3^{-\sigma_i})$. The 2-adic representation of $-3^{-\sigma_i}$ is given by filling an assembly whose south border is the 2-adic representation of $-3^0 = -1$ (i.e. …1111) and east border is valued with σ_i 0s as each row will correspond to iterating the operation $x \mapsto x/3$ starting from $-3^0 = -1$. We will get the 2-adic representation of $-3^{-\sigma_i}$ on the north side of this assembly (Theorem 1.25), its last n bits give $(-3^{-\sigma_i}) \bmod 2^n$. Then, the formula for α asks us to sum these representations with i extra trailing 0s when bit p_i is not zero.

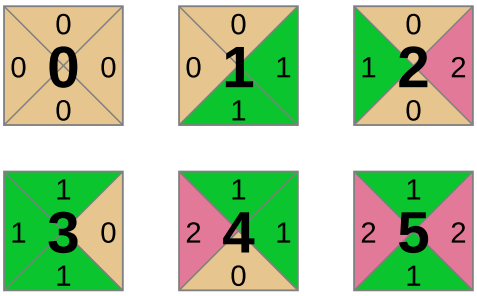
Remark 1.39 (Function computed by the path of a parity vector)**.** Let $p \in \{(\leftarrow, 0), (\swarrow, 4)\}^n$ be the path representation of a finite parity vector (Definition 1.33) with $n \in \mathbb{N}^+$. We know that the function f that it computes is affine, of the form $f(x) = ax + b$, by Definition 1.9 we get that $a = 3^k/2^n$ with k the number of edges $(\swarrow, 4)$ in p, we also get that $b \geq 0$. We remark that α and β given by Corollary 1.36 finish to characterise f since we have $f(\alpha) = \beta$ (Theorem 1.25). We get, $b = \beta - \frac{3^k}{2^n}\alpha$. And since $b \geq 0$, we learn that $2^n\beta \geq 3^k\alpha$. Function f can be understood as computing $x \mapsto T^n(x)$, when $x \equiv \alpha \bmod 2^n$. In Section 1.5, where we look at Collatz cycles, what we will do will amount to solving $f(x_0) = x_0$ with the tiles (Theorem 1.58), which gives $x_0 = b/(1-a) = \frac{2^n\beta - 3^k\alpha}{2^n - 3^k}$ since $a = \frac{3^k}{2^n} \neq 1$. Note that we get that

[27] The difference is that `coreli` has put some tiles 2 "below" the parity vector as it is the only tile that has a 1 on the west and a 0 on the north, a constraint that appears when $\swarrow$ follows $\leftarrow$.

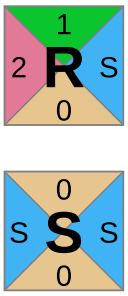
$x_0 \geq 0$ iff $a < 1$, meaning $3^k < 2^n$ which gives the condition $k < n \log_3(2)$ in a positive integer cycle, with n the length of the cycle and k the number of odd iterates in the cycle (k is the height of path p and there is one odd iterate per row). The function f of a parity vector, or more specifically f^{-1}, called v_s in [178, 117], has been important in the historical study of the Collatz problem and the results of this remark can be found in [178, 117].

Remark 1.40 (But we don't know the parity vectors)**.** Theorem 1.35 gives the link between the six tiles and Collatz sequences. However, in order to tile the Collatz sequence of $x \in \mathbb{Z}_2$ we first need to know its infinite Collatz parity vector, and to the best of our knowledge, the only way to know the infinite Collatz parity vector of x in general is to compute the Collatz sequence of x, which puts us in a difficult position. This does not impact us for the rest of this work as Section 1.6 and Section 1.5 feature parity vectors as their main objects of study, but, if we wanted to compute the Collatz iterates of x, Theorem 1.35 does directly not help us.

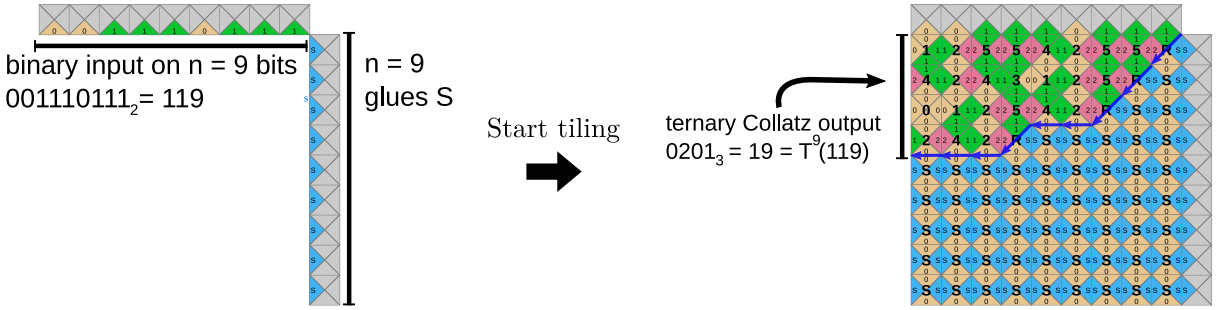
To remediate this issue, we introduce two additional tiles, R and S, that will allow us to compute parity vectors at the same time as the Collatz sequence of an input $x \in \mathbb{Z}_2$, see Figure 1.17. Tile S is responsible for detecting trailing 0s of each horizontal row of the tiling and tile R, which is almost identical to tile 4 except for its east color, is responsible for triggering the $x \mapsto (3x+1)/2$ operation when the trailing 1 of a row is reached. Fig 1.17(c) shows how to use these tiles to compute the first 9 steps of the Collatz sequence of 119: starting from a north-east 9×9 square border, Corollary 1.36 allows us to conclude that in the reconstructed assembly, the westmost side gives the ternary expression of $T^9(119) = [\![0201]\!]_3 = 19$. The corresponding constructed finite parity vector is highlighted in blue. The same technique can be used to get the Collatz sequence of an arbitrary $x \in \mathbb{Z}_2$ by specifying an infinite south-going column of glue S instead of the finite one used here. When focusing on natural numbers only, we are not limited to giving the input in binary: by base conversion, Corollary 1.18, we can for instance specify our input in base 3, Figure 1.17(d). We could also have done in base 6, or in the many bases that we can work with with these tiles, Remark 1.17.



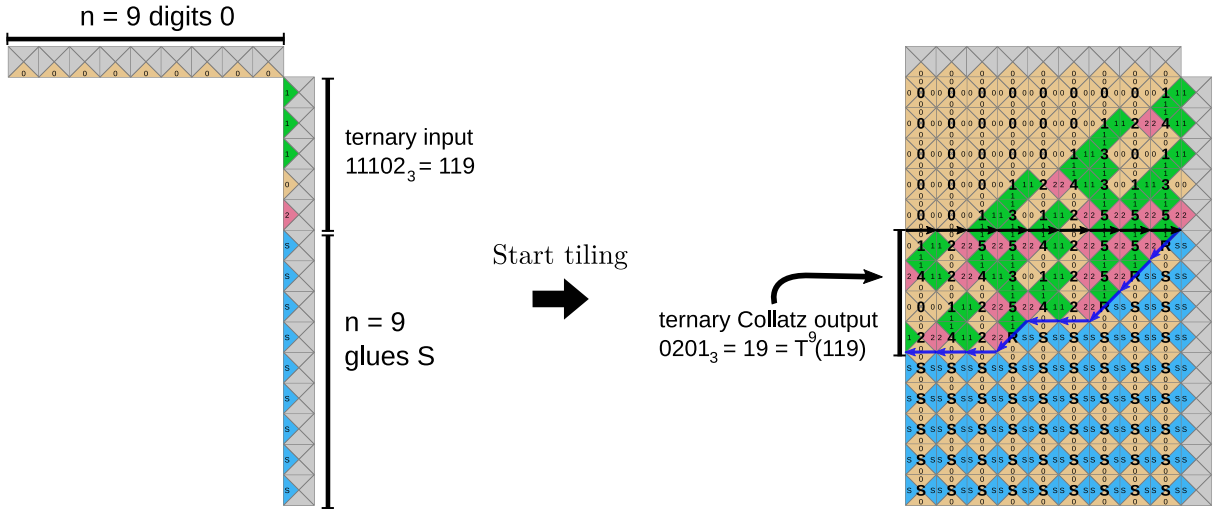
(a) The Collatz tile set, or (2,3)-CRT tile set.



(b) The two additional Collatz tiles: R and S.

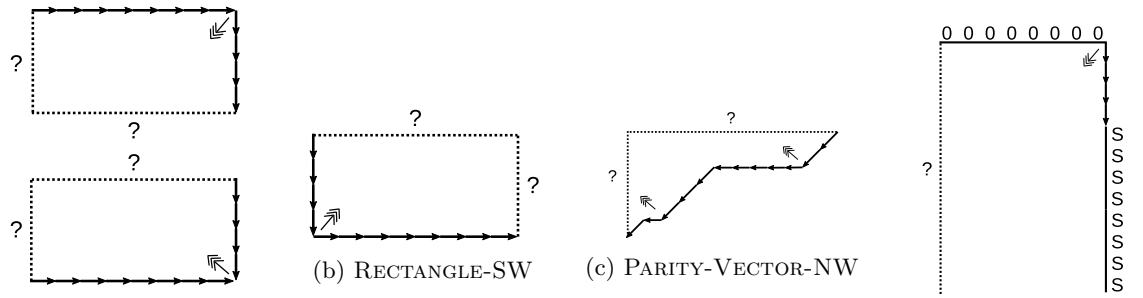


(c) Computing the 9$^{\text{th}}$ Collatz iterate of 119, input in binary, output in ternary. The dark blue curve highlights the first 9 edges of the Collatz parity vector of 119 that has been constructed by the tiles.



(d) Computing the 9$^{\text{th}}$ Collatz iterate of 119, input in ternary, output in ternary. The dark blue curve shows the parity vector, and the black curve (line) highlights the instance of base conversion (Corollary 1.18): 119 has been converted from base 3 to 2.

Figure 1.17: Tiles R and S, in addition to the six Collatz tiles allow us to compute Collatz sequences without having to know the parity vector in advance.



(a) Rectangle-NE-SE: one of two regions to be tiled.

(b) Rectangle-SW

(c) Parity-Vector-NW

(d) Collatz-tile-prediction

Figure 1.18: Four Collatz-related prediction problems: in each case a finite valued border is given (solid lines with arrows), and an area that can be uniquely tilted is delimited by the dashed lines. We want to know the valued border along the dashed lines.

1.4 Complexity of predicting Collatz tilings

In Section 1.1 we asked: what is the complexity of predicting Collatz iterates? Indeed, the hardness of the Collatz conjecture seems to be intrinsically linked to our inability to predict the long-term behavior of Collatz iterates so it is natural to wonder about the complexity of this prediction problem. Our focus on the digit expansions of Collatz iterates is exploited here as a kind of microscope to peer into the mechanics of the Collatz function.

Overview of Section 1.4 First, we introduce relevant background from computational complexity theory, including some preliminary observations about the four tiling problems studied in this section. In Section 1.4.1, we introduce several natural, time bounded, prediction problems for GCMs, and an open problem which aims at potentially separating the difficulty of bounded-time prediction of 2D GCMs, 1D GCMs, and the Collatz function. Then in Section 1.4.2, using tiles, we reformalise Collatz function prediction as Collatz-tile-prediction, as well as stating three closely related tiling problems (Rectangle-NE-SE, Rectangle-SW, Parity-Vector-NW). All four tiling problems are of the form: given some finite region of the plane, bordered by a path that is partially valued ("input") such that the region inside can be *uniquely* tiled, what values ("output") does the tiling place on the remaining unvalued border? See Figure 1.18. What we care most about mathematically measuring *how difficult these problems are*, which we tackle using computational complexity theory. We make use of the arithmetical relations that we've shown to exist within tilings (Section 1.3.3) to show that some of these problems are easy (Rectangle-NE-SE and Rectangle-SW are in NC^1, but not in AC^0). Likewise, tiling a bounded region from a parity vector provided as input, (Parity-Vector-NW, is in NC^1. However, despite the structure we've found, we leave a number of questions open. For the problem we leave open, Collatz-tile-prediction, we have at least narrowed the possibilities to where the difficulty lies.

Complexity classes and immediate observations Since the four prediction problems (Figure 1.18) delimit regions that are deterministically tiled (see Figure 1.7), it is immediate that all four problems are in the complexity class P;[28] it takes merely $O(n^2)$ tiling steps to tile an area with border length $O(n)$.

To better understand, and tease apart, the complexity of these problems we look within the fine-grained structure of P. The hardest such problems we consider are P-complete: problems that are in P and that

[28] P is the class of problems solved by Turing Machines that run in time polynomial in input size, or equivalently by uniform[31] families of Boolean circuits of polynomial size (and depth).

any problem in P can be reduced[29] to them. It is worth noting that it would be a groundbreaking result to show that Collatz prediction (in any reasonable form) is P-complete as it would imply that the tiles and/or Collatz itself could run any polynomial algorithm efficiently. If the reader is skeptical that a tile set as small as the 6 tiles could have P-complete prediction problem, we remark that there is a 7-tile simulation[30] of cellular automaton rule 110, hence there is a tile set with only 7 tiles with a P-complete prediction problem. Within P, we focus on classes in the NC and AC hierarchies:

Definition 1.41 (Complexity class AC)**.** For $i \in \mathbb{N}$, AC^i is the class of problems solved by uniform[31] Boolean circuits of unbounded-fanin AND, OR gates, and fanin-1 NOT gates, of depth $O(\log n)^i$ and size $n^{O(1)}$ in input length n. Also, $\mathrm{AC} = \cup_{i\in\mathbb{N}} \mathrm{AC}^i$.

Definition 1.42 (Complexity class NC)**.** For $i \in \mathbb{N}$, NC^i is the class of problems solved by uniform Boolean circuits, of fanin-2 AND, OR gates, and fanin 1 NOT gates, of depth $O(\log n)^i$ and size $n^{O(1)}$ in input length n. Also, $\mathrm{NC} = \cup_{i\in\mathbb{N}} \mathrm{NC}^i$.

Remark 1.43. NC and AC and both contained in P, since polynomial size circuits are easily simulated by polynomial time Turing Machines. Also, since a size-$O(k)$ tree of bounded fanin gates can simulate a k-fanin gate we get that $\mathrm{AC}^i \subset \mathrm{NC}^{i+1}$, meaning that the (infinite) AC and NC hierarchies interleave, and that NC = AC. Problems in class that are low (small i) in the NC and AC hierarchies are intuitively, some sometimes provably, simple. For example, NC^1 corresponds to Boolean formulae [116, 27, 26], and AC^0 is even simpler, being famous for a problem it does *not* contain: PARITY, binary strings with an odd number of 1s [82, 118]. Since PARITY $\in \mathrm{NC}^1$ we know $\mathrm{AC}^0 \subsetneq \mathrm{NC}^1$, but we do not know which of the following inclusions are strict $\mathrm{NC}^1 \subset \mathrm{NC}^2 \subset \mathrm{NC}^3 \subset \cdots \subset \mathrm{NC} \subset \mathrm{P}$. It is widely conjectured [73, 118] that $\mathrm{NC} \subsetneq \mathrm{P}$, hence by putting Collatz-related tiling problems in low complexity classes such as NC^1 we are lending weight to the notion that these are simple problems, almost certainly much simpler than analogous P-complete tiling problems.[30] In particular, we would be proving that these problems have such a simple combinatorial structure that they do not require explicit sequential simulation and, moreover, that they can be decomposed (parallelised) to give significant shortcuts to prediction of long-term dynamics.

1.4.1 What complexity should we expect? The case of 1D and 2D GCMs

As mentioned in Section 1.1, GCMs in one and two dimensions simulate single-tape deterministic Turing Machines. In 2D, the standard construction [90] has one map application per Turing Machine step. Furthermore, any 2D GCM can be simulated on a Turing Machine, in time polynomial in the number of input bits and number of steps (the digit expansion of the two variables grows in length by at most an additive constant per map application). Hence, 2D-GCM-PREDICTION is P-complete:

[29] Logarithmic space reductions are commonly used when comparing (say) L, NL and P [118]. For classes (conjectured to be) strictly contained in L, tighter notions of reduction may be needed (we don't want to the reduction providing 'too much power').

[30] In SI A, [183], Figure S4(b), gates g and f can be used to simulate Rule 110, which in turn can be simulated by 4 tiles each. These 8 tiles can be further optimised to 7 tiles by sharing one glue type between both half-layers.

[31] A note is warranted on the notion of uniformity for Boolean circuits. A Boolean circuit family, that decides a language, is a set of circuits $C = \{c_n | c_n \text{ has } n \in \mathbb{N} \text{ input gates and one output gate}\}$, one circuit for each input length n. Without further qualification on the definition, there are families that decide the halting problem or other undecidable/hard problems (c_n outputs 1 iff the nth (Turing machine, input) pair, encoded in unary, halts). Hence, in this work, we require that there is an algorithm that given n constructs (an encoding of) c_n, and in particular, that algorithm 'should not be too powerful': the powerfulness of the algorithm is the strength of the uniformity condition. For intuition, when thinking about problems outside of P, polynomial time uniformity, or 'P-uniform circuits', is sufficient since the uniformity condition is weaker than the problem itself so can not possibly be used to 'cheat'. Here, since we deal with complexity classes within P, even stronger uniformity conditions are needed (meaning, weaker circuit-generating algorithms). Generally, the gold standard is DLOGTIME-uniformity [12], as it is known to be equivalent to other seemingly natural notions yet easy enough to work with [77]. A DLOGTIME-uniform circuit family has an associated deterministic logarithmic time Turing Machine, that has random access to its input (a suitably simple encoding of the circuit as a string), which can be repeatedly applied to verify the circuit structure component-wise. Giving a full proof of uniformity spans from checking mundane details to solving multi-year open problems [77].

Problem: 2D-GCM-PREDICTION
Input: 2D GCM G, two values $x, y \in \mathbb{N}$ written in binary, and $t \in \mathbb{N}$ in unary[32]
Output: What is the value of $G^t(x, y)$?

Since the Collatz function is a 1D GCM, let's consider:

Problem: 1D-GCM-PREDICTION
Input: 1D GCM G, value $x \in \mathbb{N}$ written in binary, and $t \in \mathbb{N}$ in unary
Output: What is the value of $G^t(x)$?

Certainly 1D-GCM-PREDICTION is in P, but what is its exact complexity? The best known simulation technique for 1D GCMs simulating Turing machines is singly-exponentially slow [46, 90]. We don't know if such a drastic slowdown is required, a fascinating open problem [90]. If an exponential slowdown is required for a reasonable notion of simulation, we could place prediction in NC:

Open problem 1.44. Is 1D-GCM-PREDICTION in NC?

The 2-map Collatz function, which, if anything, might be easier to predict than many-map 1D GCMs:

Problem: COLLATZ-FUNCTION-PREDICTION
Input: A value $x \in \mathbb{N}$ written in binary, and $t \in \mathbb{N}$ in unary
Output: What is $T^t(x)$, the tth iterate of the Collatz function?

Open problem 1.45. Is 1D-GCM-PREDICTION, for the special case of the Collatz function, in NC?

1.4.2 Complexity of Collatz tiling prediction problems

We reformulate COLLATZ-FUNCTION-PREDICTION in terms of the 8-tile extended Collatz tile (Figure 1.17) set with ternary input and output:

Problem: COLLATZ-TILE-PREDICTION (Figure 1.18(d), example in Figure 1.17(d))
Input: A number $x \in \mathbb{N}$, with x given in binary or ternary.
Output: Using the extended Collatz tile set (Figure 1.17): if x is in ternary (written using m trits): The output is the west side values of the tiled $n \times (m + n)$ rectangle defined by: an east side of length $m + n$, with ternary x written (from north to south) on east followed by n 'S' glues, and a number of '0' glues along the north side equal to n the length of the binary string encoding x. Else if x is in binary (written using n bits): The output is the west side values of the tiled $n \times n$ rectangle defined by: a north side of length n with the binary string encoding x (from west to east), and with n 'S' glues on the east side.

It can be seen that COLLATZ-FUNCTION-PREDICTION and COLLATZ-TILE-PREDICTION are essentially identical, and that their hardness-of-prediction is identical (up to binary/ternary base conversion, and the fact that the former asks for prediction of t iterates of the Collatz function T and the latter asks for n in *odd iterates* of T). Before discussing COLLATZ-TILE-PREDICTION, we first define and characterise three related prediction problems.

[32] For finite-time prediction problems, the time bound t is typically given in unary, for the following reason. What we care about is the difficulty of predicting t time steps into the future on some, typically binary/ternary/etc., input string w. If we also gave t as a binary word, $t' \in \{0,1\}^{\Theta(\log t)}$, it would mean that running our prediction algorithm for merely t steps would takes time at least exponential $|t'|$, so for short input strings, such as $|w| \in O(\log t)$, we would have unnecessarily slow running time in terms of the *overall* input length which is $|w| + |t'|$.

Problem: RECTANGLE-NE-SE, see Figure 1.18(a)
Input: Rectangle: north and east valued sides (resp. south and east)
Output: South and west valued sides (resp. north and west)

Theorem 1.46. RECTANGLE-NE-SE is in NC^1 and not in AC^0.

Proof. First, to show that RECTANGLE-NE-SE is not in AC^0 we reduce it to PARITY which is not in AC^0, as noted in Remark 1.43. It can be easily verified that each of the 6 tiles compute parity in the following two senses: a north glue is the parity of the bit pair (east mod 2, south), and the south side is the parity of (east mod 2, north). Consider the $1 \times n$ rectangle, with 0 on its single north edge and a length-n ternary string $w \in \{0,1,2\}^n$ along the east. The rectangle has a unique tiling (Figure 1.7 and Definition-Lemma 1.2) that outputs the parity of the number of 1s in w as its single south edge, as shown in Figure 4.10, left, hence if binary input is given on east (i.e. ternary without 2s), predicting the south glue gives a decision procedure for PARITY. Similarly, if input is given on south and east, we can set the south glue of the rectangle to 0, and, after uniquely tiling the rectangle, the parity of w appears as the north glue of the rectangle (Figure 4.10, right).

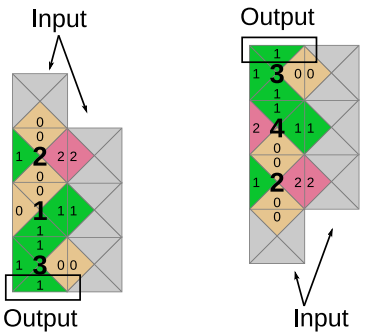


Figure 1.19: Two examples of tiles computing PARITY. In both cases, the input is $w = 210$, which has an odd number of 1s, hence the output is 1.

Next we show that RECTANGLE-NE-SE is in NC^1. If the north and east sides are given, by Corollary 1.16, we have the following arithmetical relation: $3^h[\![\text{north}]\!]_2 + [\![\text{east}]\!]_3 = 2^w[\![\text{west}]\!]_3 + [\![\text{south}]\!]_2$ with w, h the width and height of the rectangle. We can compute the number $K = 3^h[\![\text{north}]\!]_2 + [\![\text{east}]\!]_3$ in NC^1 (proof: writing 3^h down as a ternary string $100\ldots0$ of length $h+1$ is computable by a constant depth circuit since h is essentially given in unary in the problem statement; addition is in AC^0 by a simple carry propagation formula [29]; multiplication is also in NC^1 [29]; and more recently base conversion was shown to be in NC^1 [77]).[33,34] Then (in either ternary or binary) we compute the quotient q of $K/2^w$ as well as the remainder $m = K \mod 2^w$, i.e. we are doing Euclidean division using the fact that $m < 2^w$), and write q out in base 3 to give the west side, and m in base two to give the south.

If the east and south sides are given, by Remark 1.17, the tiling of the rectangle solves the Chinese Remainder Theorem (CRT) in $\mathbb{Z}/3^h \times \mathbb{Z}/2^w$ in the following sense: since $[\![\text{east}]\!]_3 < 3^h$, $[\![\text{south}]\!]_2 < 2^w$, and $3^h, 2^w$ are coprime, the CRT gives a unique positive integer $K < 3^h 2^w$ such that $K \equiv [\![\text{east}]\!]_3 \mod 3^h$ and $K \equiv [\![\text{south}]\!]_2 \mod 2^w$. Converting from a CRT representation $([\![\text{south}]\!]_2, [\![\text{east}]\!]_3)$ to K is in NC^1 (Theorem 3.3 in [39] shows it is in L-uniform NC^1, and Theorem 4.1 of [77] improves that to DLOGTIME-uniform NC^1). In the tiled rectangle, we get $3^h[\![\text{north}]\!]_2 + [\![\text{east}]\!]_3 = 2^w[\![\text{west}]\!]_3 + [\![\text{south}]\!]_2 = K$ (Corollary 1.16), and we can extract north and west by computing the quotient of K by 3^h and 2^w, which is in NC^1 [77], and convert to respectively base 2 and base 3 to get north and west. □

Problem:	RECTANGLE-SW, see Figure 1.18(b)
Input:	Rectangle: south and west valued sides, given in binary and ternary, respectively
Output:	North (in binary) and east (in ternary) valued sides

Theorem 1.47. RECTANGLE-SW is in NC^1.

Proof. By Corollary 1.16, we have the arithmetical relation $3^h[\![\text{north}]\!]_2 + [\![\text{east}]\!]_3 = 2^w[\![\text{west}]\!]_3 + [\![\text{south}]\!]_2$ with w, h the width and height of the rectangle. We will use the technique from the proof of Theorem 1.46: compute the number $K = 2^w[\![\text{west}]\!]_3 + [\![\text{south}]\!]_2$ in NC^1 [77]. Then, since $[\![\text{east}]\!]_3 < 3^h$, we can use Euclidean division to compute $K/3^h = q$ where quotient $q = [\![\text{north}]\!]_2$, and $K \mod 3^h = m$ where reminder $m = [\![\text{east}]\!]_3$ (convert K to base 3 and take the last h trits), all in NC^1 [77]. Finally, since base conversion is in NC^1 [77], write out q in base 2 as the north value, and and m in base 3 as east. □

Intuitively, we believe that RECTANGLE-SW is not in AC^0, but we don't have a proof! Hence, we leave the following open:

Open problem 1.48 (RECTANGLE-SW $\notin \mathrm{AC}^0$?)**.** Is RECTANGLE-SW in AC^0?

Similarly, we would tend to believe that reconstructing the α and β of a parity-vector (see Corollary 1.36) is in NC^1 and not in AC^0, but we have no proof:

Problem:	PARITY-VECTOR-NW, see Figure 1.18(c)
Input:	A parity vector $p \in \{0,1\}^n$ (Definition 1.28)
Output:	North and west valued sides of the corresponding triangular assembly (respectively representing α and β of Corollary 1.36)

[33]The uniformity[31] of these constructions is straightforward to show, except for base conversion (and division, and Chinese remaindering) which has a significant history [77].

[34]A technicality is that these arithmetical problems are function problems, not decisions problems, which means we are abusing notation by saying they are in NC^1 or AC^0. However, there are straightforward mappings between the two. For example, when we say addition is in AC^0 we either (equivalently) mean that it is in the function analog of AC^0 (sometimes called FAC^0), or that we are computing the ith bit of the sum s, for any $0 \leq i < |s|$).

Theorem 1.49. PARITY-VECTOR-NW is in NC^1.

Proof. Let n be the length of the parity vector. As noted in Remark 1.38, α is defined by the summation in Equation (1.1), each of the terms of which correspond to an instance of RECTANGLE-NE-SE (with south side $111\ldots1$ and east $000\ldots0$ as inputs), and so by Theorem 1.46 each is NC^1 computable. It turns out[35] that there are uniform TC^0 circuits for iterated addition (polynomial in n, n-bit additions) [77, 88] and thus that problem is in NC^1. The composition of two $O(\log n)$ depth circuits is gives the statement. Similar approach for β using Equation (1.2). □

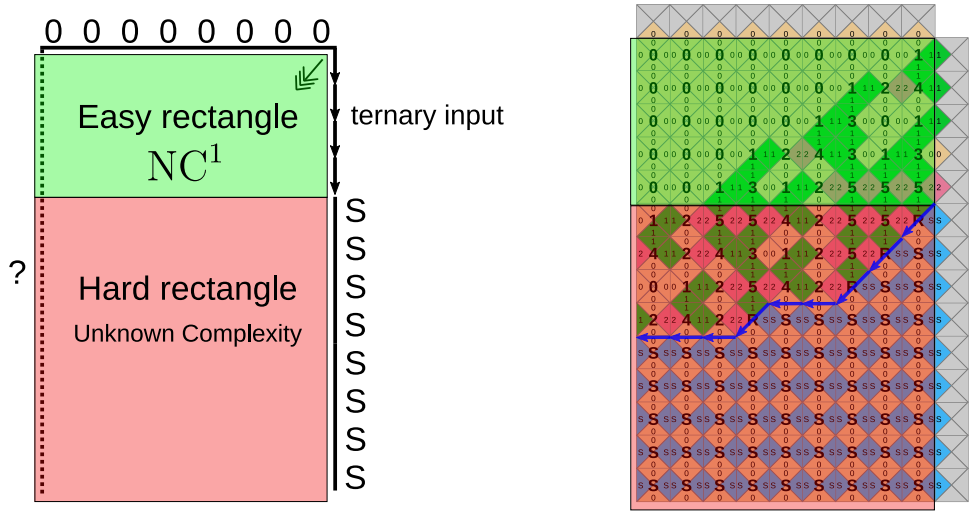
Open problem 1.50. Is PARITY-VECTOR-NW outside of AC^0?

In the beginning of this section, we defined COLLATZ-TILE-PREDICTION, and argued that it is a natural tiling-based encoding of the problem COLLATZ-FUNCTION-PREDICTION. Despite the tools developed in this section we leave its complexity open, putting it in NC would certainly imply strong results on the structure of Collatz iterations:

Open problem 1.51. Is COLLATZ-TILE-PREDICTION in NC? Is it outside of AC^0?

Remark 1.52. With respect to membership in classes such as NC, or NC^1, or P-complete, the question of predicting merely a single bit of the parity vector, specifically, the last bit of the parity vector, is as hard as writing out the entire parity vector. Also, the complexity of producing the first (least significant) bit of $T^t(x)$ is as hard as computing the LSB for all $T^i(x), 1 \leq i \leq t$. The reason is straightforward: given an algorithm that predicts the first bit of $T^t(x)$, in other words the t^{th} bit of the parity vector, that either runs in parallel (e.g. in NC) or sequentially (e.g. in P), respectively, we can easily generalise that algorithm to produce the entire parity vector by running it t times, to get the first bit of $T^i(x)$ for all $1 \leq i \leq t$, (in parallel or sequentially, respectively).

Remark 1.53 (Easy rectangle, hard rectangle, predicting parity vectors)**.** One thing that we can say – and was also pointed at in our earlier work, see Figure 4 in [156] – about COLLATZ-TILE-PREDICTION is that part of it is simple (i.e. in NC^1) as it reduces to RECTANGLE-NE-SE and more precisely, base 3 to 2 conversion (Corollary 1.18):



[35] To get a more intuitive, but deeper, NC^2 algorithm it suffices to observe that the sum has $\leq n$ terms, each a $\leq n$ bit number, hence their addition is performed using a circuit of depth $O(\log^2 n)$: the circuit is a tree of depth $O(\log n)$ nodes where each node is a $O(\log n)$ depth circuit (with fanin ≤ 2 gates, and noting that addition is in AC^0, and hence in NC^1).

On the example on the right we see that the easy rectangle is "only" carrying-out the computation of converting $[\![11102]\!]_3$ from base 3 to base 2 $[\![1110111]\!]_2$ (Corollary 1.18), which is an instance of Rectangle-NE-SE thus in NC^1.

It is in what we call the "hard rectangle", whose complexity is open (Open problem 1.51), that the Collatz parity vector of $[\![1110111]\!]_2$ emerges (highlighted in blue). This setting leaves open the complexity of predicting parity vectors, which we believe must be somewhat challenging to settle since it is not even known if an eventually periodic binary inputs produces an eventually periodic parity vector (Lagarias' periodicity conjecture, see Section 1.3.3 and [99, 139]).

Remark 1.54 (Hardness is in the parity vector)**.** With Parity-Vector-NW we know that reconstructing a Collatz sequence once we have its parity vector is *easy*: its in NC^1. Hence, while we do not know the exact complexity of Collatz-tile-prediction, we know that, assuming that predicting the parity vector is NC^1-hard, then Collatz-tile-prediction is as hard as predicting the parity vector. In other words: hardness is in the parity vector. A practical implication of the complexity of Collatz-tile-prediction being open is that in a Collatz tiling, parts far from the parity vector are easy to predict but for parts close to the parity vector hardness of prediction still contains some mystery. In particular this means that, assuming $\mathrm{NC}^1 \neq \mathrm{P}$, essentially the only hope to simulate Turing machines efficiently within Collatz traces is if that simulation exploits some hardness of prediction property of the parity vector, and in does not solely rely on the high-order bits of each iterate (i.e. NC^1 part of the diagram). In future work, we intend to continue researching the complexity of predicting Collatz parity vectors with the hope to place it in NC or L, which would procure strong evidence that Collatz iterates cannot efficiently simulate Turing machines *at all*.

1.5 Application to Collatz cycles

A fundamental question on the Collatz process is: "what are the x such that $x = T^k(x)$ for some $k \in \mathbb{N}$?". Generally, the question is stated in the natural numbers, i.e. $x \in \mathbb{N}$, and only 2 cycles are known: the cycle containing 0 and the cycle containing 1. When extending the question of Collatz cycles to the negative numbers, 3 more cycles are known: the cycle containing -1, the cycle containing -5 and the cycle containing -17. When extending the question to $\mathbb{Z}_2$, we get infinitely[36] many more cycles, such as the cycle containing $\frac{5}{23}$, see Section 1.1. Famously, the following has been conjectured [17, 102]:

Conjecture 1.55 (Nontrivial cycles conjecture)**.** There is only one cycle of T in $\mathbb{N}^+$, the cycle {1,2}, also called *the trivial cycle*.

Remark 1.56. More generally, in $\mathbb{Z}$, it is conjectured that T has only five cycles: respectively containing 1, 0, -1, -5, and -17 [17, 102].

In this section, we aim at reformulating questions about Collatz cycles in the context of our six tiles. In particular, we see that Conjecture 1.55 admits simple geometric reformulations in terms of tilings, Theorem 1.61 and Theorem 1.68. But first, let's highlight some results that are known on Collatz cycles and that we will either use or illustrate in our study:

1. A Collatz cycle on the positive integers must contain more than 10,439,860,591 elements and at least 6,586,818,670 odd iterates [64, 102].
2. In a Collatz cycle on the positive integers of size n, all elements are strictly less than 2^n [140, 64].
3. There are no cycles (called *circuits*) on the positive integers whose iterates modulo 2 repeat a pattern of the form $1^m0^{m'}$ with $m, m' \in \mathbb{N}$ [153, 138]. See Remark 1.63.
4. In $\mathbb{Z}_2$, all Collatz cycles are rational. Furthermore, any 2-adic rational that is in a Collatz cycle is also in $\mathbb{Z}_3$ and $\mathbb{Z}_6$ [178]. See Theorem 1.58 below.

In order to reformulate Conjecture 1.55 in terms of tilings, we will make great use of the notion of parity vector introduced in Definition 1.28 and Definition 1.33.

Definition 1.57 (Support of a cycle)**.** Let $n \in \mathbb{N}$. A parity vector $U \in \{0,1\}^n$ (Definition 1.28) is the support of a Collatz cycle if there exists $x \in \mathbb{Z}_2$ such that $T^n(x) = x$ and $U_k = T^k(x) \bmod 2$ for $0 \leq k < n$. In that case, we also say that $p \in \{(\leftarrow, 0), (\swarrow, 4)\}^n$, the path representation of U, is the support of this Collatz cycle.

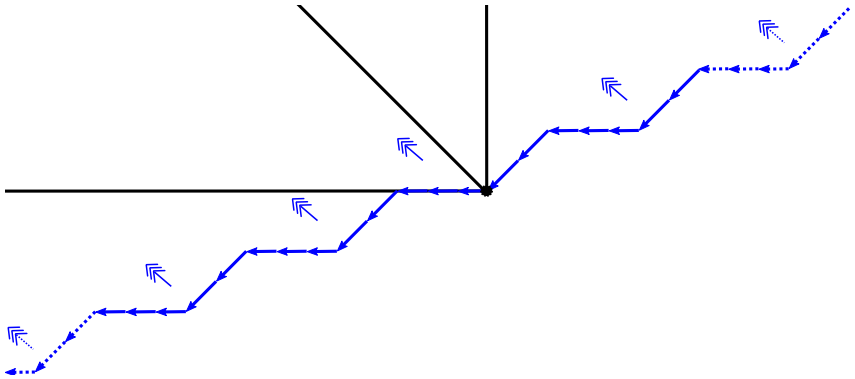
Theorem 1.58 (Rational cycles are in $\mathbb{Z}_2 \cap \mathbb{Z}_3 \cap \mathbb{Z}_6$)**.** Let $U \in \{0,1\}^n$ be a parity vector. Then there exists a unique $x \in \mathbb{Q} \cap \mathbb{Z}_2 \cap \mathbb{Z}_3 \cap \mathbb{Z}_6$ such that U is the support of the Collatz cycle of x.

Proof. This theorem can be deduced easily from [178] 2.12 and 2.13, since we get the following explicit formula for $x \in \mathbb{Q}$:

$$x = \frac{\sum_{i=0}^{k-1} 3^{k-1-i} 2^{d_i}}{2^n - 3^k}$$

With n the length of U, k its number of 1s d_i the position of the i^{th} 1 in U (starting from 0). We see that x is a 2-adic, 3-adic and 6-adic rational because $2^n - 3^k$ cannot be a multiple of 2, 3, or 6. Hence $x \in \mathbb{Q} \cap \mathbb{Z}_2 \cap \mathbb{Z}_3 \cap \mathbb{Z}_6$.

[36] This will be justified in this section.



However, we can also show this result **visually** with the above illustration: repeat the path representation of U in both direction infinitely to the south-west and north-east in the plane (blue in the above illustration). There is a unique valid tiling that can be constructed north-west from it and by Corollary 1.36 we know that it reconstructs the Collatz sequence of x. Also, by construction, the tiling is cyclic since we have repeated the same parity vector in both directions (tiling constraints are the same at each repetition of the parity vector). Hence, we will read the 2-adic, 3-adic, and 6-adic values of x respectively horizontally, vertically and in diagonal, in this tiling (solid black in above illustration). By construction, these representations are eventually periodic and x follows the parity sequence given by U under Collatz iterations. Hence we have the result and $x \in \mathbb{Q} \cap \mathbb{Z}_2 \cap \mathbb{Z}_3 \cap \mathbb{Z}_6$.

□

Remark 1.59 (`coreli`)**.** For a given parity vector U, you can compute the rational number that cycles under Collatz iterations while following the parities given by U, by using our library `coreli` (see Appendix C):

```
>>> from coreli import ParityVector
>>> pv = ParityVector([0,1,1,0,1])
>>> pv.cyclic_rational()
46/5
```

We can verify that this result is correct by iterating Collatz over this rational and confirm that they have the parities given by `[0,1,1,0,1]`:

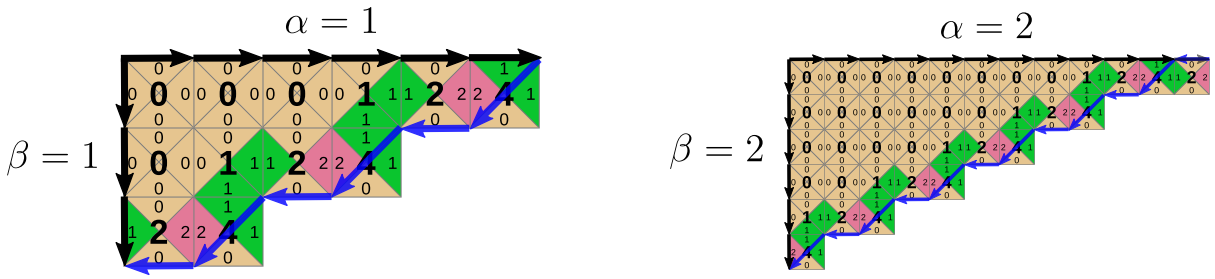
```
>>> from coreli import T, iterates
>>> from sympy import Rational
>>> iterates(T, len(pv), Rational(46,5))
[46/5, 23/5, 37/5, 58/5, 29/5, 46/5]
```

We get the immediate reformulation of Conjecture 1.55 in terms of parity vectors:

Theorem 1.60. Conjecture 1.55 is equivalent to the following: the parity vectors that support a positive integer cycle are only those of the form $(01)^n$ or $(10)^n$ for some $n \in \mathbb{N}$. See Figure 1.20.

Proof. $(01)^n$ or $(10)^n$ correspond to the parity of different length instances of the trivial cycle either starting from 2 or 1: 2,1,2 and 1,2,1. □

With the 6 tiles, together with Point 2, we get the following reformulation:



(a) The parity vector $[1,0,1,0,1,0]$ supports the cycle $\{1,2\}$, starting at 1.

(b) The parity vector $[0,1,0,1,0,1,0,1,0,1,0,1]$ supports the cycle $\{1,2\}$, starting at 2.

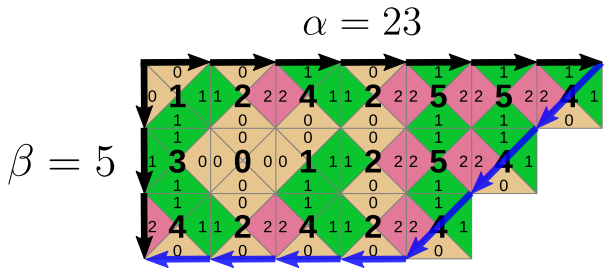
Figure 1.20: Example of parity vectors that support the trivial cycle $\{1,2\}$, Theorem 1.60.

Theorem 1.61. Let $p \in \{(\leftarrow,0),(\swarrow,4)\}^*$ be the path representation of some parity vector. Then p is the support of a positive integer cycle if and only if $\alpha = \beta$ with $\alpha, \beta \in \mathbb{N}$ in the reconstructed triangular assembly north-west from p as given by Corollary 1.36.

Proof. If $\alpha = \beta$ then p is the support of a positive integer cycle. Suppose that p is the support of a positive integer cycle and $\alpha \neq \beta$. Then, $x \in \mathbb{Q} \cap \mathbb{Z}_2 \cap \mathbb{Z}_3 \cap \mathbb{Z}_6$ given by Theorem 1.58 that cycles along p is of the form $c2^n + \alpha$ with $c > 0 \in \mathbb{N}$, by Section 1.3.3, Point 2. Hence $x > 2^n$ which is in contradiction with this section's Point 2: in a positive cycle of length n all elements are less than 2^n. □

Example 1.62. Figure 1.20(a) shows that parity vector $[1,0,1,0,1,0]$ supports the trivial cycle $\{1,2\}$, we have $\alpha = \beta = 1$. Figure 1.20(b) shows that parity vector $[0,1,0,1,0,1,0,1,0,1,0,1]$ also supports the trivial cycle $\{1,2\}$, we have $\alpha = \beta = 2$. Figure 1.16, shows that parity vector $[1,0,1,1,0,0,0,1,1,0,0,0]$ is not the support of a cycle since $\alpha = 2137$ and $\beta = 127$, $\alpha \neq \beta$.

Remark 1.63. It is quite mesmerizing that it is so hard to prove, in general, that as soon as we deviate from the simple staircase geometry of the trivial cycle's parity vector (Figure 1.20), such as in Figure 1.16, we never get $\alpha = \beta$. Some restricted families of parity vectors have been proven not to give $\alpha = \beta$, such as *circuits* defined in Point 3, which we can illustrate on parity vector $1^3 0^4$ where $\alpha \neq \beta$:



We can also reformulate Theorem 1.61 in terms of a natural tile to look at in the triangular assembly of a parity vector: the most complex tile, i.e. the north-most west-most tile which has the property that it will be deduced last when tiling the region. An integer cycle requires that the most complex tile is the same for all rotations of the parity vector concatenated to itself, Theorem 1.68 and Figure 1.21.

Definition 1.64 (Most complex tile)**.** Let $p \in \{(\leftarrow,0),(\swarrow,4)\}^n$ be the path representation of some parity vector of length n. Then the most complex tile of p, $\mathrm{mct}(p) \in \{0,1,2,3,4,5\}$, is the north-most west-most tile in the triangular assembly reconstructed north-west of p (see Corollary 1.36). The most complex tiles of some parity vectors are highlighted in red in Figure 1.21.

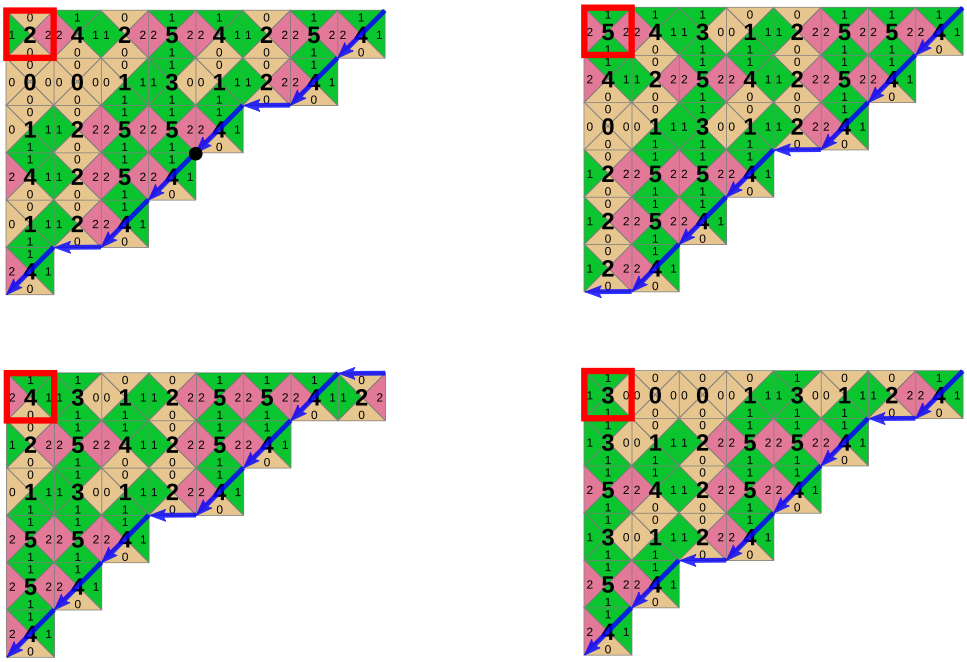


Figure 1.21: Four successive rotations of $2p$ with p the path representation of parity vector $[1, 1, 0, 1]$. The two repetitions of p are delimited by a black dot in the first (top-left) assembly. Their respective most complex tile is 2, 5, 4, 3 (Definition 1.64), highlighted in red. Because these most complex tiles are not all-0 or all-5, we know that p is not the support of an integer cycle, Theorem 1.68.

.

Remark 1.65 (`coreli`)**.** We can compute the "most complex tile" of a parity vector with `coreli` (Appendix C), using examples of Figure 1.21:

```
>>> from coreli import ParityVector
>>> ParityVector([1,1,0,1]*2).most_complex_tile()
2

>>> ParityVector([1,1,0,1]*2).rotate()
[1, 0, 1, 1, 1, 0, 1, 1]
>>> ParityVector([1,1,0,1]*2).rotate().most_complex_tile()
3

>>> ParityVector([1,1,0,1]*2).rotate(-1)
[1, 1, 1, 0, 1, 1, 1, 0]
>>> ParityVector([1,1,0,1]*2).rotate(-1).most_complex_tile()
5
```

Remark 1.66 (The most complex tile is not *that* complex)**.** Based on the arithmetic formulae for the north and west sides of the triangular assembly reconstructed from a parity vector (Remark 1.38) we know that computing the most complex tile is feasible in NC^1 which is widely believed to be a strict subset of P, see Theorem 1.49, Section 1.4.

Definition 1.67 (Rotation of a parity vector)**.** Let $p \in \{(\leftarrow, 0), (\swarrow, 4)\}^n$ be the path representation of some parity vector of length n. Then $\mathcal{R}(p)$ is the rightward rotation of p. For instance, if p corresponds to parity vector $[0, 1, 0, 0]$ then $\mathcal{R}(p)$ corresponds to $[0, 0, 1, 0]$, $\mathcal{R}^2(p)$ to $[0, 0, 0, 1]$ and $\mathcal{R}^3(p)$ to $[1, 0, 0, 0]$.

Theorem 1.68. Let $p \in \{(\leftarrow, 0), (\swarrow, 4)\}^n$ and let $2p$ be p concatenated to itself. Then, integer cycles are characterised in terms of most complex tiles as follows:

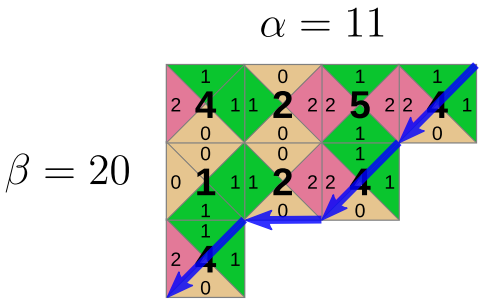
- p is the support of a cycle in $\mathbb{N}^+$ if and only if the most complex tile of all rotations of $2p$ is always 0, i.e. $\text{mct}(\mathcal{R}^i(2p)) = 0$ for $0 \leq i < n$
- p is the support of a cycle in $\mathbb{Z}^-$ if and only if the most complex tile of all rotations of $2p$ is always 5, i.e. $\text{mct}(\mathcal{R}^i(2p)) = 5$ for $0 \leq i < n$

Proof. If $\text{mct}(\mathcal{R}^i(2p)) = 0$ for $0 \leq i < n$ then p is the support of a cycle in $\mathbb{N}^+$ since the statement implies that there is a translation of the parity vector, north-west to the original one, that is entirely valued with 0s and hence makes all elements of the cycles positive integers (note that mct is not well-defined for flat parity vectors of the form 0^n which is why we exclude 0 here by working in $\mathbb{N}^+$). Same reasoning shows that if $\text{mct}(\mathcal{R}^i(2p)) = 5$ for $0 \leq i < n$ then p is the support of a cycle in $\mathbb{Z}^-$.

The other direction crucially relies on this section Point 2 as if p is the support of a cycle in $\mathbb{N}^+$ and $\text{mct}(\mathcal{R}^i(2p)) \neq 0$ for all $0 \leq i < n$ we'd get an iterate bigger than 2^n which is impossible for a cycle of length n (Point 2). Same argument in the negative case.

□

Example 1.69. Figure 1.21 illustrates Theorem 1.68 in the case of parity vector $[1, 1, 0, 1]$ with path representation $p \in \{(\leftarrow, 0), (\swarrow, 4)\}^4$. Indeed, the first four rotations of $2p$ give most complex tiles 2,5,4,3 which is not all-0 or all-5. Hence $[1, 1, 0, 1]$ is not the support of an integer cycle and indeed, it has $\alpha = 11$ and $\beta = 20$, $\alpha \neq \beta$ (Theorem 1.61):



Remark 1.70 (Most complex tiles distribution is close to random: integer cycles are unlikely)**.** We do the following experiment: we take all 1024 parity vectors[37] of length 10 and, similarly to the setting of Theorem 1.68, for each p we compute the most complex tile of the first 10 rotations of $2p$. This gives us, for each parity vector a distribution of tiles in $\{0, 1, 2, 3, 4, 5\}$. We consider the Shannon entropy of each of these distributions, i.e. a number between 0 and 1 that is 1 if the distribution is uniform and 0 if the distribution is concentrated on only one tile. We get an average entropy of 0.82, which is quite close to 1, meaning that in average the most complex tile of the rotations of a parity vector is close to random. We also get max entropy of 0.98 for parity vector $[1, 1, 1, 0, 1, 0, 0, 0, 0, 0]$ and expected min entropy of 0 for the three parity vectors corresponding to integer cycles (by Theorem 1.68 their distribution of most complex tiles is concentrated on only one tile: 0 or 5), which are $(10)^5$ and $(01)^5$ for the trivial cycle $\{1, 2\}$ and $(1)^{10}$ for the negative cycle $\{-1\}$. This small experiment illustrates how unlikely integer cycles are: for instance, by Point 1, the parity vector of a positive cycle must be of length at least roughly 10 billion and then, we need all of its billions of rotations to place only tile 0 as most complex tile which is in stark contrast with the acquired belief that the most complex tiles of rotations of parity vectors are essentially random. In future work it would be interesting to run this kind of experiment on larger parity vectors and see if the corresponding distributions pass the randomness statistical test suites such as NIST used for evaluating pseudorandom number generators [11, 13].

[37] We actually exclude the completely flat parity vector $(0)^{10}$, which makes 1023 parity vectors in total.

1.6 Application to Collatz ancestors

A rather natural question to ask about the Collatz process is: "what are the *Collatz-ancestors* of $x \in \mathbb{N}$?" – where a *Collatz-ancestor* of x is a $y \in \mathbb{N}$ such that $T^n(y) = x$ for some $n \in \mathbb{N}$. Indeed, the Collatz conjecture itself can be reformulated as: "The set of Collatz-ancestors of 1 is $\mathbb{N}^+$". Another justification to study Collatz-ancestors is a line of research that looks for *sufficient sets*, i.e. subsets of $\mathbb{N}$ on which proving the Collatz conjecture is sufficient to get the conjecture on all $\mathbb{N}$ [94, 6, 117, 31]. The most general result known to us states that it is enough to prove the Collatz conjecture on $A + B\mathbb{N}$ for any $A, B \in \mathbb{N}$ and $B > 0$ [117]. To prove this result, the author shows that for all $x \in \mathbb{N}$ there is a Collatz-ancestor of some descendant of x in $A + B\mathbb{N}$.

A case – perhaps *the only case* – where the set of Collatz-ancestors is simple to describe is when x is a multiple of three:

Lemma 1.71 (Multiples of three have no odd Collatz-ancestors)**.** Let $x \in \mathbb{N}$. If x is a multiple of three, i.e. $x \in 3\mathbb{N}$, then the set of Collatz-ancestors of x is $\{2^n x \mid n \in \mathbb{N}\}$, i.e. x has no odd Collatz-ancestor. If x is not a multiple of three, i.e. $x \in \mathbb{N} \setminus 3\mathbb{N}$, then x has infinitely many odd Collatz-ancestors.

Proof. Let $x = T(y)$ with $y \in \mathbb{N}$ odd. Then $2x = 3y + 1$. We get $2x \equiv 1 \bmod 3$, whose unique solution is $x \equiv 2 \bmod 3$. If $x \equiv 0 \bmod 3$ then $x_n = 2^n x \equiv 0 \bmod 3$ for all $n \in \mathbb{N}$ meaning no y odd can satisfy $x_n = T(y)$, meaning that x has no odd Collatz-ancestors and that its set of Collatz-ancestors is exactly $\{2^n x \mid n \in \mathbb{N}\}$. If $x =\in \mathbb{N} \setminus 3\mathbb{N}$ then $x_n = 2^n x$ alternates parity 1 and 2 (or 2 and 1) modulo 3, giving infinitely many $x_n \equiv 2 \bmod 3$. Take any such $x_n = 3z + 2$ with $z \in \mathbb{N}$, inverting $x_n = (3y + 1)/2$ we get $2(3z + 2) = 3y + 1$ giving $y = 2z + 1$ which is odd and which is a Collatz-ancestor of x since $T^{n+1}(y) = T^n(x_n) = x$. □

We can refine the question on Collatz-ancestors a bit more by asking, for some $k \in \mathbb{N}$: "what are the Collatz-ancestors of $x \in \mathbb{N}$ at *odd-distance* k?", where a Collatz-ancestor y of x is at odd-distance k if its trajectory to x meets exactly k odd iterates. We call $\mathcal{E}\mathrm{Pred}_k(x)$ the set of binary representations of Collatz-ancestors at odd-distance k. This question is somehow natural because, in binary, it is asking for ancestors whose representation has been *significantly* changed k times: only odd iterates get a significant change in their binary representation since even iterates only get their trailing 0 removed. In our tilings, k corresponds to the number of rows (or height) of the triangular assembly corresponding to $x = T^n(y)$ with n the number of T-steps separating y from x, in Figure 1.16 for instance we have $y = \alpha = 2137$, $x = \beta = 127$, $n = 12$ and $k = 5$.

Using a simple argument, Shallit and Wilson have shown that $\mathcal{E}\mathrm{Pred}_k(x)$ is **a regular language**[38] [146]. In our work[39] [154], we have refined this result by giving a concrete algorithm to produce a regular expression, $\mathbf{reg}_k(x)$, for $\mathcal{E}\mathrm{Pred}_k(x)$. The size of $\mathbf{reg}_k(x)$ is exponential in k (as soon as $x \notin 3\mathbb{N}$), which is a **doubly-exponential improvement** on what a naïve conversion of the finite state machines of [146] to regular expressions would give. Our work also generalised [42, 78, 72] which focused on special cases. As a concrete example, here is $\mathbf{reg}_3(14)$, the regular expression of $\mathcal{E}\mathrm{Pred}_3(14)$ consisting of the Collatz-ancestors of 14 at odd-distance 3, given in binary:

[38] The set $\mathcal{E}\mathrm{Pred}_k(x)$ is infinite as soon as $k > 0$ and $x \not\equiv 0 \bmod 3$ [146, 154].

[39] Unfortunately and confusingly for the reader, in our paper [154], symbols $\leftarrow$ and $\downarrow$ respectively correspond to symbols $\swarrow$ and $\leftarrow$ used in this thesis.

$$
\begin{aligned}
\mathbf{reg}_3(14) = {} & 100(000111)^*00(01)^*01(0)^* \mid \\
& 10000(100011)^*100(01)^*01(0)^* \mid \\
& 1000010010111110(110001)^*1100(01)^*01(0)^* \mid \\
& 10000100101(111000)^*11100(01)^*01(0)^* \mid \\
& 100001001(011100)^*011100(01)^*01(0)^* \mid \\
& 10000100101111011(001110)^*0(01)^*01(0)^* \mid \\
& 100(000111)^*0001(10)^*1(0)^* \mid \\
& 10000(100011)^*10001(10)^*1(0)^* \mid \\
& 1000010010111110(110001)^*110001(10)^*1(0)^* \mid \\
& 10000100101(111000)^*1(10)^*1(0)^* \mid \\
& 100001001(011100)^*01(10)^*1(0)^* \mid \\
& 10000100101111011(001110)^*001(10)^*1(0)^*
\end{aligned}
$$

By sampling the first line, for instance, we get that 100 (000111000111000111) 00 (0101) 01 (0000), representing 4,414,271,824 in binary, is at odd-distance 3 from 14 and indeed, in its Collatz sequence there are exactly 3 odd iterates (in bold below) before it reaches 14:

4414271824, 2207135912, 1103567956, 551783978, **275891989**,
413837984, 206918992, 103459496, 51729748, 25864874, **12932437**,
19398656, 9699328, 4849664, 2424832, 1212416, 606208, 303104, 151552, 75776, 37888, 18944, 9472, 4736, 2368, 1184, 592, 296, 148, 74, **37**, 56, 28, 14

With this result, we get a sense that the set of Collatz-ancestors of $x \in \mathbb{N}$, which is $\text{Pred}(x) = \cup_{k\in\mathbb{N}}\mathcal{E}\text{Pred}_k(x)$, bears some complexity as it is structured in layers of regular languages whose shortest-length description known to date is exponential[40] in k [154]. Intuitively, we do not expect Collatz-ancestors sets to have much simpler descriptions in general since the Collatz problem asks whether $\text{Pred}(1) = \mathbb{N}^+$ or not, and, having a simple description of $\text{Pred}(1)$ would probably settle the question.

In this section, coming back to sufficient sets, we give a visual proof that $\alpha + 2^n\mathbb{N}$ is sufficient[41] for all $n \in \mathbb{N}$ and $\alpha < 2^n$ (Corolary 1.74) using ideas that were central to [154]. Showing this result amounts to constructing an ancestor of x that is in $\alpha + 2^n\mathbb{N}$ for all $x \in \mathbb{N} \setminus 3\mathbb{N}$ (and treat multiples of three separately). This is a special case of [117] but we will refine the construction by giving, not just one ancestor but infinitely many, one per layer $\mathcal{E}\text{Pred}_k(x)$ for k large enough, Theorem 1.73. Also, before the results of [117], it was only known that $\alpha + 2^n\mathbb{N}$ is sufficient for $\alpha = 1$ and $n \leq 4$ and was left as an open problem [6]. We will also discuss the complexity of our construction – subject not studied in [117] – which is surprisingly less hard than expected, Remark 1.75.

The following Lemma is central to our construction and is a visual reformulation, with the tiles, of a well known number-theoretical result: 2 is a primitive root of $(\mathbb{Z}/3^k)^\times$ for all $k \in \mathbb{N}^+$ [81] (see proof below for context).

Lemma 1.72. Consider $\mathcal{K}_k$, the set of ternary words of length $k \in \mathbb{N}^+$ that do not end with digit 0, i.e. $\mathcal{K}_k = \{t_0, \ldots t_{k-1} \in \{0,1,2\}^k \mid t_{k-1} \neq 0\}$. The cardinal of $\mathcal{K}_k$ is $2 \times 3^{k-1}$. Let $z \in \mathcal{K}_k$. Then, the columns of the rectangular tiling of height k and width $2 \times 3^{k-1}$ constructed from valued west (or east)

[40] It takes roughly 3 pages to print $\mathbf{reg}_4(1)$ in [154].
[41] This means that proving the Collatz conjecture in $\alpha + 2^n\mathbb{N}$ is enough to prove the conjecture in $\mathbb{N}$.

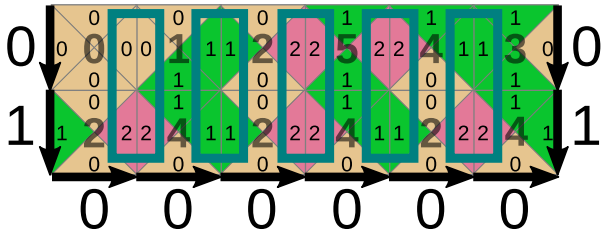


Figure 1.22: The columns of the rectangular assembly of height $k = 2$ and width $2 \times 3^{k-1} = 6$, constructed from west (or east) side valued 01 and south side valued 000000 enumerate – in a custom order – all strings of $\mathcal{K}_2 = \{01, 02, 11, 12, 21, 22\}$ (length-2 ternary strings not ending in 0), highlighted in teal color, Lemma 1.72.

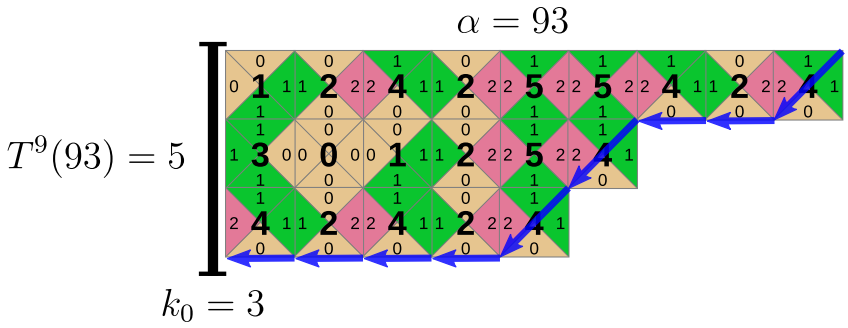
side z of length k and valued south side $0 \dots 0$ of length $2 \times 3^{k-1}$ enumerate all the elements of $\mathcal{K}_k$ and both west-most and east-most columns are z – Figure 1.22.

Proof. The multiplicative subgroup[42] of $\mathbb{Z}/3^k$ is denoted $(\mathbb{Z}/3^k)^\times$ and contains exactly any $x \in \mathbb{Z}/3^k$ that is not a multiple of 3. The cardinal of $(\mathbb{Z}/3^k)^\times$ is $2 \times 3^{k-1}$. The set $\mathcal{K}_k$ gives the ternary representations of elements of $(\mathbb{Z}/3^k)^\times$. Element 2 is a primitive root of $(\mathbb{Z}/3^k)^\times$ [81], meaning that successive powers of 2 modulo 3^k enumerate $(\mathbb{Z}/3^k)^\times$, i.e. any $x \in (\mathbb{Z}/3^k)^\times$ can be written $2^i \bmod 3^k$ with some $i \in \mathbb{N}$. Let i_0 be such that $[\![z]\!]_3 \equiv 2^{i_0} \bmod 3^k$. The rectangular tiling constructed from valued west (or east) side $W = z$ of length k and valued south side $S = 0 \dots 0$ is uniquely defined thanks to the determinism and totality of the south-west (or south-east) corners of the tiles. Then, column at distance $0 \leq i \leq 2 \times 3^k$ from W is giving $E_i \in \mathcal{K}_k$, the ternary valuation of $2^{i+i_0} \bmod 3^k$. Indeed, by Corollary 1.16, in the sub-rectangle of height k and width i, delimited to the east by E_i, and to the west by W, with $N_i \in \{0,1\}^i$ and $S_i \in \{0,1\}^i$ the north and south sides of this sub-rectangle, we have $3^k[\![N_i]\!]_2 + [\![E_i]\!]_3 = 2^i[\![W]\!]_3 + [\![S_i]\!]_2$ which gives $3^k[\![N_i]\!]_2 + [\![E_i]\!]_3 = 2^i \times [\![z]\!]_3 + 0$, which modulo 3^k, becomes $[\![E_i]\!]_3 = 2^i 2^{i_0} = 2^{i+i_0} \bmod 3^k$. Hence, since 2 is primitive in $(\mathbb{Z}/3^k)^\times$, we have the result that successive E_i are enumerating $\mathcal{K}_k$. Furthermore, by Lagrange's theorem, $2^{2\times 3^{k-1}+i_0} = 2^{i_0} = [\![z]\!]_3$ in $(\mathbb{Z}/3^k)^\times$, meaning that the first and last column of the assembly are the same, equal to z. We've shown the result by tiling from the west (i.e. multiplied by 2), if we had tiled from the east (i.e. divided by 2) we would have been multiplying by 2^{-1} (the multiplicative inverse of 2 in $(\mathbb{Z}/3^k)^\times$) which is also a primitive root of $(\mathbb{Z}/3^k)^\times$ and the result would be the same. □

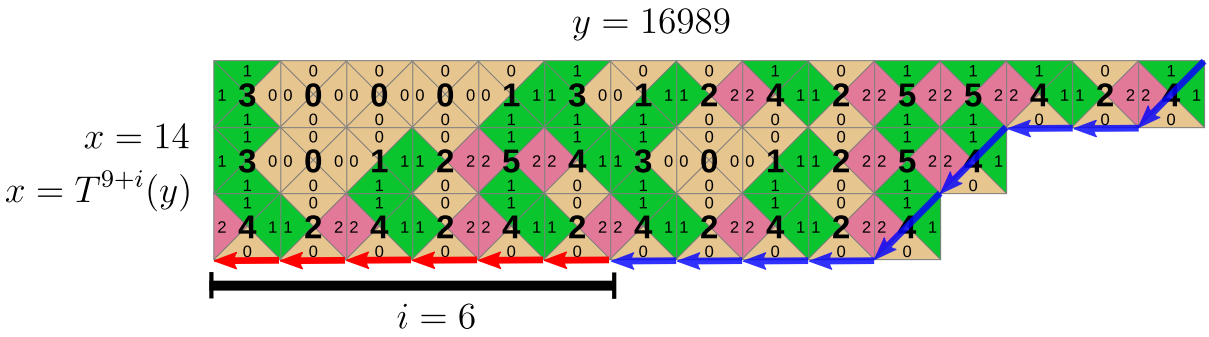
Theorem 1.73. Let $n \in \mathbb{N}^+$, $w \in \{0,1\}^n$, and $\alpha = [\![w]\!]_2$. Let $x \in \mathbb{N} \setminus 3\mathbb{N}$. Let k_0 be the number of odd iterates in $[\alpha, T(\alpha), \dots, T^{n-1}(\alpha)]$ and $k_1 = \lfloor \log_3(x) \rfloor$. Then, for all $k \geq \max(k_0, k_1)$, there is a Collatz-ancestor y at odd-distance k of x such that $y \in \alpha + 2^n\mathbb{N}$, i.e. the last n bits of y are given by w.

Proof. Note that we have to choose $x \in \mathbb{N} \setminus 3\mathbb{N}$ because of Lemma 1.71: multiples of 3 have no ancestors at odd-distance greater than 0. Let $p \in \{(\leftarrow, 0), (\swarrow, 4)\}^n$ be the path representation of the parity vector of $[\alpha, T(\alpha), \dots, T^{n-1}(\alpha)]$. Then, by Corollary 1.36, there is a unique height-k_0, width-n, triangular assembly constructed north-west from p, situation that we can illustrate with the following example where $n = 9$, $w = \texttt{001011101}$, $\alpha = 93$, $\beta = T^9(93) = [\![012]\!]_3 = 5$, $k_0 = 3$:

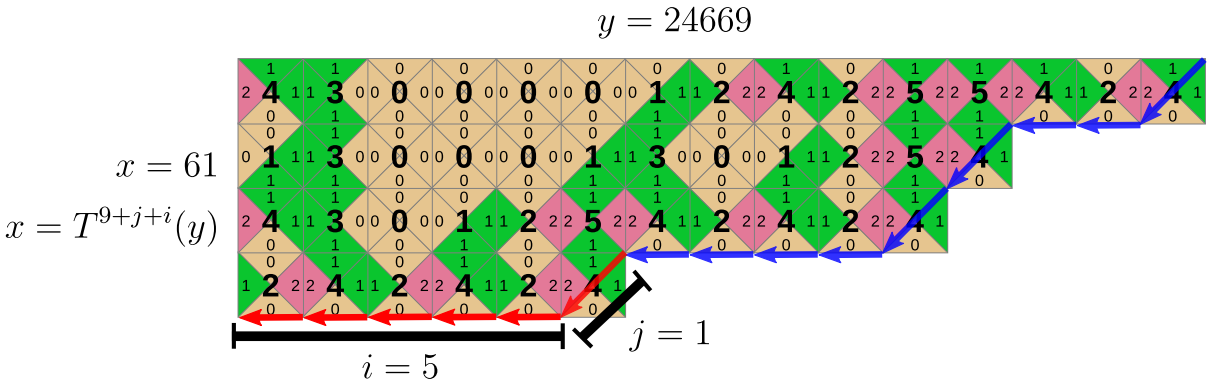
[42] The multiplicative subgroup of $\mathbb{Z}/3^k$ consists of the elements of $\mathbb{Z}/3^k$ that can be inverted, i.e. the set of $x \in \mathbb{Z}/3^k$ such that there is $y \in \mathbb{Z}/3^k$ such that $xy = 1$.



Assume that $x < 3^{k_0}$, then by Lemma 1.72, because $x \not\equiv 0 \bmod 3$, there is $i \in \mathbb{N}$ such that continuing the parity vector with i edges $(\leftarrow, 0)$ will give x on the west-most side of the assembly, for instance, in the above example, if $x = [\![112]\!]_3 = 14$ we can take $i = 6$:



By construction, on the north side of the extended assembly, we read y in binary such that $x = T^{n+i}(y)$ (Corollary 1.36), the last n bits of y are given by w and y is at odd-distance k_0 of x. If $x \geq 3^{k_0}$, i.e. $k_1 > k_0$, we can extend the original assembly first with $j = k_1 - k_0$ valued edge $(\swarrow, 4)$ (or in fact any parity vector of height j) and then add the right amount of edges $(\leftarrow, 0)$ to reach x, for instance if $x = [\![2021]\!]_3 = 61 \leq 3^3$ we have $j = 1$ then $i = 5$:



By construction, on the north side of the extended assembly, we read y in binary such that $x = T^{n+j+i}(y)$ (Corollary 1.36), the last n bits of y are given by w and y is at odd-distance k_1 of x.

Similarly, for any $k > \max(k_0, k_1)$, we can keep increasing j until we get the total height of the assembly to be k. That way, we proved what we wanted: we constructed a Collatz-ancestor of x at odd-distance k for all $k \geq \max(k_0, k_1)$. □

Corollary 1.74 ($\alpha + 2^n\mathbb{N}$ is sufficient)**.** Let $n \in \mathbb{N}^+$ and $\alpha < 2^n$. If the Collatz conjecture is true for all $x \in \alpha + 2^n\mathbb{N}$ then the Collatz conjecture is true for all $x \in \mathbb{N}$, i.e. the set $\alpha + 2^n\mathbb{N}$ is sufficient for the Collatz conjecture.

Proof. If $x \in \mathbb{N} \setminus 3\mathbb{N}$, by Theorem 1.73 we get y, an ancestor of x in $\alpha + 2^n\mathbb{N}$ and if the Collatz conjecture is true for an ancestor of x it is true for x. If $x \in 3\mathbb{N}$, if x is even, iterating T on x will eventually make it odd. Once we get to $x' \in 3\mathbb{N}$ that is odd, we have $2T(x') = 3x + 1$ which gives $2T(x') \equiv 1 \bmod 3$ which gives $T(x') \equiv 2 \bmod 3$. Hence we can apply Theorem 1.73 on $T(x')$ and we have found the ancestor of a descendant of x for which the Collatz conjecture is true, hence the Collatz conjecture is true for x. □

Remark 1.75 (Discrete logarithm is not hard (in our case!))**.** In the construction of Theorem 1.73 we essentially need to find i such that $x \equiv 2^i \bmod 3^k$ for given $x \in \mathbb{N} \setminus 3\mathbb{N}$ and $k \in \mathbb{N}$. This is always possible because 2 is a primitive root (or *generator*) of $(\mathbb{Z}/3^k)^\times$ (see proof of Lemma 1.72). Finding such an i is an instance of a general problem known as *discrete logarithm*:

Problem:	Discrete-logarithm
Input:	G a group, g a generator[43] of G and $x \in G$
Output:	The smallest $i \in \mathbb{N}$ such that $x = g^i$ in G

The discrete logarithm problem is a notoriously hard problem whose hardness is key to the security of various cryptographic protocols [157]. When formulated as a decision problem, discrete logarithm is known to be in NP and co-NP but is in the rare situation that it is suspected not to be in P **and** not to be NP-complete [21]. Interestingly, the problem is in BQP, meaning that it can be solved in polynomial time by a quantum computer, using Shor's algorithm [148]. Studying the complexity of this problem in various settings is an entire field of research [75].

However, surprisingly (to us), in our case, the problem is not hard! Indeed, in our case $G = (\mathbb{Z}/3^k)^\times$ which has $N_k = 2 \times 3^{k-1}$ elements (see Lemma 1.72). Numbers N_k are 3-smooth meaning that their highest prime factor is 3, which is a small prime. In such a case, Pohlig-Hellman algorithm [127, 173] computes discrete logarithm using only $O(k)$ multiplications – as opposed to $O(3^k)$ multiplications using naïve bruteforce search.

Hence, the construction of Theorem 1.73 can be performed using $O(k)$ multiplications (apart from computing discrete logarithm the construction only requires to compute $O(k)$ Collatz steps which can be done using $O(k)$ multiplications) and $O(3^k)$ space (the number of bits of the ancestors – north side of the assembly – is $O(i)$ and $i < 2 \times 3^{k-1}$). If we were only concerned by getting a specific bit of the ancestor, we intuitively believe that less than $O(3^k)$ space would be required and we leave this to future work.

1.7 Conclusion and future work

In this chapter, we have introduced the 6 Collatz tiles (Section 1.3, Figure 1.7), which we connected to our previous work [156] in Section 1.2. Then, we studied valid partial tilings made of the 6 Collatz tiles by associating functions to paths in tilings. We showed that every path between two given points compute the same function, Theorem 1.13. This result allowed us to arithmetically interpret rectangular tilings, Corollary 1.16, and to show that base conversion can occur naturally in these tilings, Corollary 1.18. Then, we saw how to apply the function of a path to a 2-adic, 3-adic or 6-adic integer, Theorem 1.25. From that point on, we focused more specifically on Collatz sequences and showed how to *tile* Collatz sequences, Theorem 1.35. We studied the complexity of predicting various types of Collatz tilings in Section 1.4 and

[43] A generator of a group, if it exists, is $g \in G$ such that powers of g enumerate the group.

showed that most of the hardness of predicting Collatz iterates is concentrated in predicting *parity vectors*, i.e. the sequence of parities of the Collatz-iterates of a number, Remark 1.54. We then studied Collatz cycles and showed that integer cycles can be characterised by a simple geometric property related to their *most complex tile*, Theorem 1.68. Finally, we reasoned about Collatz-ancestors and gave a visual proof that for all $n \in \mathbb{N}$ and $\alpha < 2^n$ proving the Collatz conjecture on the set $\alpha + 2^n\mathbb{N}$ is enough to prove the Collatz conjecture on all $\mathbb{N}$, Theorem 1.73[44].

We view this chapter, and the tiling framework that it introduces, as a base camp from which we will be able to pursue further research on the action of the Collatz process on symbolic representations of numbers.

There are several topics on which we want to work on in the future:

- **Interpreting more Collatz results.** Throughout this chapter, we have visually interpreted known Collatz results with the tiles. For instance, with the visual proof of Theorem 1.58 which characterises rational cycles, or with Theorem 1.73 on the sufficiency of sets $\alpha + 2^n\mathbb{N}$ (a special case of [117]). We wish to continue this effort of visualising known results with the six tiles. For instance, we would like to interpret the complete result – not just the special case – of [117] which says that any arithmetical progression is sufficient for the Collatz conjecture. We would also like to interpret Elihaou's work on cycles [64] as well as Tao's result on the Collatz conjecture being true for *almost all* positive integers [159].

- **Characterising the complexity of Collatz prediction.** In Section 1.4, we left several questions open for future work concerning lower and upper complexity bounds of Collatz-related prediction problems, most importantly to us, Open problem 1.45 (and Open problem 1.51, its tiling version) which asks whether predicting the t^{th} Collatz-iterate of some $x \in \mathbb{N}$ is in NC or not.

- **Base 3/2.** A south-west-going diagonal of edges $\swarrow$ can be interpreted in base 3/2, a non-integral base that has been studied in the past [4, 5]. One *natural* property of base 3/2 is that the biggest number that can be encoded on k digits is less than $(3/2)^k$. Hence, in Collatz tilings, one can simultaneously read the base 6 and 3/2 representations of a number by respectively looking at the north-east-going and north-west-going diagonal, but base 3/2 being a lot more inefficient encoding than base 6 in number of digits used. We believe that this encoding in fact creates constraints that could play a role in understanding why there are no non-trivial positive integer cycles.

- **Arbitrary border conditions.** In this chapter we particularly focused on a specific type of south-west-going border conditions for our assemblies, corresponding to parity vectors, see Section 1.3.3: horizontal segments of edges $\leftarrow$ are valued with 0, there cannot be more than one edge $\swarrow$ at once, and they are valued 4. In general, we could break free of these constraints and study arbitrary south-west-going border conditions. Although we move away from Collatz, questions such as characterising "most complex tiles" (Definition 1.64) remain valid and we believe that studying this more general context could shine some light on better understanding the special case of Collatz parity vectors. Our intuition is that south-west-going paths in tilings can be interpreted as some kind of *zigzadic* numbers with their own arithmetical rules.

[44]This theorem is a refinement of a special case of [117], see Section 1.6. While we're in footnote 44, here's the Collatz sequence of 44 (iterating map T, see Section 1.1): $[44, 22, 11, 17, 26, 13, 20, 10, 5, 8, 4, 2, 1]$.

Chapter 2

Hardness of busy beaver value BB(15)

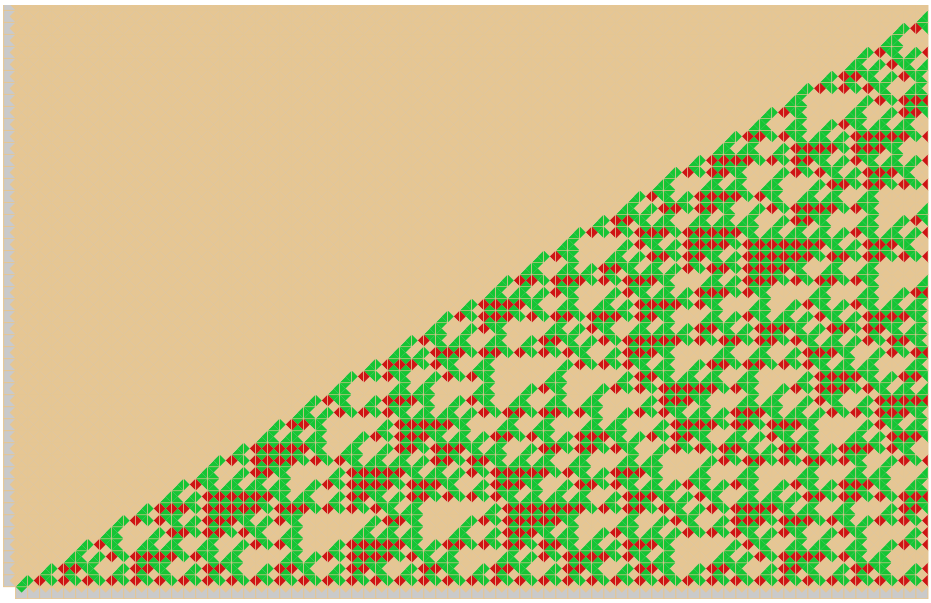


Figure 2.1: The first 75 powers of two assembled in base 3 by the 6 Collatz tiles introduced in Chapter 1. Reading left-to-right, each column of glues (colours) corresponds to a power of two: beige glues represent ternary digit 0, green glues 1 and red glues 2. For instance, the rightmost column encodes (from top to bottom) $110210021020202011202012000020112001021021222022_3 = 2^{75}$. The complexity of the patterns illustrates the complexity of answering Erdős' conjecture on powers of 2: "for all $n > 8$, there is at least one digit 2 in the base 3 representation of 2^n" which amounts to asking if each glue-column (except for the few first) has at least one red glue.

2.1 Foreword

Consider the rectangular infinite assembly of the 6 Collatz tiles from Figure 1.7 where all positions of the south border have been set to 0 and where the south border has been set to $\ldots 001$, Figure 2.1. By Chapter 1, Theorem 1.25, we know that each successive column of this assembly gives a successive power of 2, in base 3. Knowing if, for all $n > 8$, the base 3 representation of 2^n contains at least one digit 2 is an open problem known as Erdős' conjecture on powers of 2. In terms of tiles, it asks if all columns of Figure 2.1 contain a red glue (except for the first few columns). In this chapter, we encode this conjecture as the halting problem (from blank tape) of two Turing machines, first with 5 states and 4 symbols and then with 15 states and 2 symbols. In turn, these machines shed light on the difficulty of knowing *busy beaver values* BB(15) and BB(5,4).

2.2 INTRODUCTION

In the theory of computation, there lies a strange and complicated beast, the busy beaver function. This function tracks a certain notion of algorithmic complexity, namely, the maximum number of steps any halting algorithm of a given program-size may take. Although formulated in terms of Turing machines, the underlying notion of "maximum algorithmic bang for your buck" that it captures could be defined in any reasonable programming language [1].

The busy beaver function $n \mapsto \mathrm{BB}(n)$ was introduced by Tibor Radó in 1962 and corresponds to the maximum number of steps made by a halting deterministic Turing machine with n states and 2 symbols starting from blank input [131]. It was generalised by Brady [18, 111] to machines with k symbols; $\mathrm{BB}(n, k)$. Busy beaver functions, i.e. $n, k \mapsto \mathrm{BB}(n, k)$, or $n \mapsto \mathrm{BB}(n, k)$ for fixed $k \geq 2$, are not computable. Otherwise the Halting problem on blank tape would be computable: take any machine with n states and k symbols, run it for $\mathrm{BB}(n, k) + 1$ steps; if it has not halted yet, we know that it will never halt. They also dominate any computable function [1].

To date, only four non-trivial busy beaver values are known: $\mathrm{BB}(2) = 6$, $\mathrm{BB}(3) = 21$, $\mathrm{BB}(4) = 107$ and $\mathrm{BB}(2, 3) = 38$ [111, 112]. Machines that halt without input after so many steps that they beat previously known record-holders, with the same n and k, are called busy beaver *champions* and they give lower bounds for $\mathrm{BB}(n)$ or $\mathrm{BB}(n, k)$. It is conjectured [1] that $\mathrm{BB}(5) = 47{,}176{,}870$ as there is an explicit 5-state 2-symbol champion [109] that halts after 47,176,870 steps[45]. Since May 2022, it is known [113] that $\mathrm{BB}(6) \geq 10 \uparrow\uparrow 15$, which is a tower $10^{10^{10^{\cdots}}}$ of height 15 (*tetration*) – well beyond the estimated number of atoms in the observable universe. This new bound on BB(6) beats the previously known bounds on BB(7) and leaves one baffled by how big BB(7) must be.

A recent trend, that we follow here, uses theory to address the question: "How hard is it to know $\mathrm{BB}(n, k)$", for various pairs n, k. One way to answer that question is to relate values of $\mathrm{BB}(n, k)$ to notoriously hard mathematical problems. For instance, one can imagine designing a Turing machine that, starting from blank tape, will halt if and only if it finds a counterexample to Goldbach's conjecture (every even integer greater than 2 is the sum of two primes). Such a machine was actually built, using 4,888 states and 2 symbols [186]. This result implies that knowing the value of BB(4,888) would allow us to computably decide Goldbach's conjecture: run the machine for $\mathrm{BB}(4{,}888) + 1$ steps – which must be an unbelievably huge number – and if it has halted before, then the conjecture is false otherwise it is true. The result was later claimed to be improved to a 27-state 2-symbol machine [1], which, subject to being proved correct, would be to date the smallest busy beaver value that relates to a *natural* mathematical problem. Similar efforts led to the construction of a 5,372-state 2-symbol machine that halts if and only if the Riemann hypothesis is false [186]; with a later claimed improvement to 744 states [1], hence knowing the value of BB(744) is at least as hard as solving the Riemann hypothesis.

An even more drastic way to establish hardness is to find an n for which $\mathrm{BB}(n)$ independent of some standard set of axioms such as PA, or ZFC. In that spirit, a 7,910-state machine whose halting problem is independent of ZFC was constructed [186] and this was later claimed to be improved to 748 states [1]. The 748-state machine explicitly looks for a contradiction in ZFC (such as a proof of $0 = 1$) which is, by Gödel's second incompleteness theorem, independent of ZFC. Aaronson [1] conjectures that BB(10) is independent of PA and BB(20) is independent of ZFC meaning that the frontier between knowable and unknowable busy beaver values could be as low as BB(10) if we limit ourselves to typical inductive proofs.

[45]

	A (init)	**B**	**C**	**D**	**E**
0	1 R **B**	1 R **C**	1 R **D**	1 L **A**	Halt
1	1 L **C**	1 R **B**	0 L **E**	1 L **D**	0 L **A**

Current busy beaver champion [109] for machines with 5 states (**A**–**E**) and 2 symbols (0,1), it halts in 47,176,870 steps starting from all-0 input and state **A**, which gives the lowerbound $\mathrm{BB}(5) \geq 47{,}176{,}870$.

2.2.1 Results and discussion

Here, we continue the approach of relating small busy beaver values to hard mathematical problems towards obtaining insight into the location of the frontier between knowable and unknowable busy beaver values. Our particular approach gives upper bounds on the smallest counterexample to such problems. We do this by giving machines that search for counterexamples to an open, Collatz-related conjecture formulated in 1979 by Erdős:

Conjecture 2.1 (Erdős [65])**.** For all natural numbers $n > 8$ there is at least one digit 2 in the base 3 representation of 2^n.

In Section 2.2.2 we discuss the relationship between the Erdős, Collatz and weak Collatz conjectures.

The main technical contribution of this chapter is to prove the following theorem:

Theorem 2.2. There is an explicit 15-state 2-symbol Turing machine that halts if and only if Erdős' conjecture is false.

That Turing machine, called $M_{15,2}$, is given in Figure 2.4 and proven correct in Section 2.5. The proof shows that $M_{15,2}$ *simulates*, in a tight (linear time) fashion, an intuitively simpler machine with 5 states and 4 symbols, that we call $M_{5,4}$ (Figure 2.3) and whose behaviour is proven correct in Section 2.4:

Theorem 2.3. There is an explicit 5-state 4-symbol Turing machine that halts if and only if Erdős' conjecture is false.

Why are these theorems important? Because they give a concrete sense of the hardness of knowing, i.e. getting of a proof of, busy beaver values BB(15) and BB(5, 4). Indeed, the only method known to prove the value of BB(n) for some n is to meticulously study all n-state machines (up to isomorphism and other immediate symmetries [109]) and show whether they halt or not from blank tape and, from this analysis, deduce BB(n). Since our results exhibit specific 15-state 2-symbol and 5-state 4-symbol machines whose halting problem from blank tape reduces to Erdős' conjecture, we get that solving BB(15) and BB(5, 4) requires first solving that hard problem that has been open for decades. Before these results, it was known that BB(15) and BB(5, 4) must be unbelievably huge numbers (since already BB(6) $> 10 \uparrow\uparrow 15$ [113]), but our results add to that empirical knowledge by showing that it would require a qualitatively different kind of hard mathematical work, solving an open conjecture, to know these numbers.

One corollary of Theorems 2.2 and 2.3 is that the infinite conjecture can be replaced by a finite one (short proof in Section 2.5.3):

Corollary 2.4. Erdős' conjecture is equivalent to the following conjecture over a finite set: for all $8 < n \leq \min(\text{BB}(15), \text{BB}(5, 4))$ there is at least one digit 2 in the base 3 representation of 2^n.

It should be emphasised that, although finite, this bound is certainly so large as not to be of any practical use for proving or disproving the conjecture by explicit enumerative search for potential counterexamples. A second implication of our results is that discovering a counterexample to Erdős conjecture gives a lower bound to the values of BB(15) and BB(5,4):

Corollary 2.5. Let $x \in \mathbb{N}$ be the smallest counterexample to Erdős conjecture, if it exists. Then we have: $\text{BB}(15) \geq \log_2 x$ and $\text{BB}(5, 4) \geq \log_2 x$.

Our results make BB(15) the smallest busy beaver value linked to a natural, hard, open problem in mathematics, a significant improvement on the BB(4,888) and BB(5,372) results cited earlier (that have

a proof of correctness), and the BB(27) and BB(744) results (that as yet lack a published proof of correctness). Perhaps this lowers the hope that we will ever know the value of BB(5, 4) or BB(15), as it implies a computable procedure to solve Erdős' conjecture.

As we've noted, some claimed results on the busy beaver function come with proofs, but some do not [1]. We advocate for proofs, although we acknowledge the challenges in providing human-readable correctness proofs for small programs that are, or almost are, program-size optimal.[46] Here, the proof technique for correctness of our BB(15) candidate $M_{15,2}$ amounts to proving by induction that $M_{15,2}$ *simulates* (in a tight sense via Definition 2.9 and Lemma 2.11) another Turing machine $M_{5,4}$ (giving Theorem 2.2). $M_{5,4}$ exploits the existence of a tiny finite state transducer (FST), in Figure 2.2, for multiplication by 2 in base 3. We directly prove $M_{5,4}$'s correctness by induction on its time steps (giving Theorem 2.3).

A Turing machine simulator, `bbsim`, was built in order to test our constructions[47]. The reader is invited to run the machines of this chapter through the simulator.

2.2.2 Erdős' conjecture and its relationship to the Collatz and weak Collatz conjectures

A first obvious fact about Conjecture 2.1 is that the bound $n > 8$ is set so as to exclude the three known special cases: 1, 4 and 256 whose ternary representations are respectively 1, 11 and 100111. Secondly, since having no digit 2 in ternary is equivalent to being a sum of distinct powers of three, Conjecture 2.1 can be restated as: for all $n > 8$, the number 2^n is not a sum of distinct powers of 3.

The conjecture has been studied by several authors. Notably, Lagarias [97] showed a result stating that, in some sense, the set of powers of 2 that omits the digit 2 in base three, is small. In [63], the authors showed that, for p and q distinct primes, the digits of the base q expansions of p^n are equidistributed on average (averaging over n) which in our case suggests that digits $0, 1, 2$ should appear in equal proportion in the base 3 representation of 2^n. Dimitrov and Howe [56] showed that 1, 4 and 256 are the only powers of 2 that can be written as the sum of at most twenty-five distinct powers of 3.

The Collatz conjecture states that iterating the Collatz function T on any $x \in \mathbb{N}$ eventually yields 1, where $T(x) = x/2$ if x is even and $T(x) = (3x+1)/2$ if x is odd. The weak Collatz conjecture (or nontrivial cycles conjecture) states that if $T^k(x) = x$ for some $k \geq 1$ and x a natural number then $x \in \{0, 1, 2\}$.

Although solving the weak Collatz conjecture, given current knowledge, would not directly solve Erdős' conjecture, intuitively, Erdős' conjecture seems to be the simpler problem of the two. Indeed, Tao [160] justifies calling Erdős' conjecture a "toy model" problem for the weak Collatz conjecture by giving the number-theoretical reformulation that there are no integer solutions to $2^n = 3^{a_1} + 3^{a_2} + \cdots + 3^{a_k}$ with $n > 8$ and $0 \leq a_1 < \cdots < a_k$, which in turn seems like a simplification of a statement equivalent to the weak Collatz conjecture, also given in [160] (Conjecture 3; Reformulated weak Collatz conjecture).

The three conjectures have been encoded using the 6 Collatz tiles (Figure 1.7). As Figure 2.1 shows, a simple assembly made of these tiles illustrates the complexity of the patterns occurring in ternary representations of powers of 2 which gives a sense of the complexity underlying Erdős' conjecture. Beyond making complex, albeit pretty, pictures there is deeper connection here: the 6 Collatz tiles can be shown to simulate the base 3, **mul2**, Finite State Transducer that we introduce in Section 2.4, Figure 2.2, and our small Turing machines use it to look for counterexamples to Erdős' conjecture. The tile set also simulates the *inverse* of the **mul2** FST (which computes the operation $x \mapsto x/2$ in ternary) which was used to build a 3-state 4-symbol non-halting machine that runs the Collatz map on any ternary input

[46] The situation can be likened to the hunt for small, and fast, universal Turing machines [182], or simple models like Post tag systems, with earlier literature often missing proofs of correctness, but some later papers using induction on machine configurations to do the job, for example refs [120, 181].

[47] Simulator and machines available here: `https://github.com/tcosmo/bbsim`

[110] and finally, it also simulates the *dual* of that FST (which computes the operation $x \mapsto 3x + 1$ in binary) which can be used to simultaneously compute the Collatz map both in binary and ternary, see Chapter 1, Section 1.2 and [156]. So these three closely-related FSTs are all encoded within that small tile set, that in turn encodes the three conjectures (other conjectures can be encoded with the Collatz tile set, such as Mahler's 3/2 problem, see Appendix B).

2.2.3 Future work

This chapter opens a number of avenues for future work.

We've given a finite bound on the first counterexample to Erdős' conjecture, an obvious future line of work is to improve that cosmologically large bound by shrinking the program-size of our 15-state machine.

It would be interesting to design small Turing machines that look for counterexamples to the weak Collatz conjecture. That way we could relate the fact of knowing busy beaver values to solving that notoriously hard conjecture. We have already found 124-state 2-symbol, and 43-state 4-symbol, machines that look for such counterexamples[48] but we would like to further optimise them (i.e. reduce their number of states) before formally proving that their behaviour is correct.

There are certain other open problems amenable to BB-type encodings. For instance, the Erdős–Mollin–Walsh conjecture states that there are no three consecutive powerful numbers, where $m \in \mathbb{N}$ is *powerful* if $m = a^2b^3$ for some $a, b \in \mathbb{N}$. It would not be difficult to give a small Turing Machine, that enumerates unary encoded natural numbers, i.e. candidate m and a, b values, and runs the logic to check for counterexamples. Algorithms that bounce back and forth on a tape marking off unary strings have facilitated the finding of incredibly small universal Turing machines [120, 134, 182], so stands a chance to work well here too, although we wouldn't expect it to beat our main results in terms of program-size.

In fact, any problem that can be expressed as a Π_1 sentence [1] should be relatable to a value of BB in the same way that we did for Erdős' conjecture, although many problems would presumably yield unsuitably large program-size. It is also worth noting that some problems seem out of reach of the busy beaver framework: for instance the Collatz conjecture, as to date, there are no known ways for an algorithm to recognise whether or not an arbitrary trajectory is divergent, making it seemingly impossible for a program to search for a divergent counterexample[49].

In future work we also intend to make progress on settling the value of BB(5), conjectured to be 47,176,870 [1, 109], and we believe that this research goal is inherently collaborative which is why we have created the platform `https://bbchallenge.org` dedicated to it, see Appendix D.

2.3 Definitions: busy beaver Turing machines

Let $\mathbb{N} = \{0, 1, 2 \dots\}$ and $\mathbb{Z} = \{\dots, -1, 0, 1, \dots\}$. We consider Turing machines that are deterministic, have a single bi-infinite tape with tape cells indexed by $\mathbb{Z}$, and a finite alphabet of k tape symbols that includes a special 'blank' symbol (we use # or b). For readability, we use bold programming-style names, for example **check_halt**. We use '(init)' to denote the initial state. Each state has k transitions, one for each of the k tape symbols that might be read by the tape head, and a transition is either (1) the halting instruction 'Halt' or (2) a triple: write symbol, tape head move direction (L or R), and next state. In the busy beaver setting used throughout this chapter, we start machines in their initial state on an all-blank

[48] These machines are also available here: `https://github.com/tcosmo/bbsim`

[49] Said otherwise: given current knowledge there are no reason to believe that the set of counterexamples to the Collatz conjecture is recursively enumerable.

tape and, at each time step, according to what symbol is read on the tape, the specified transition is performed, until 'Halt' is encountered (if ever).

Let $\mathrm{TM}(n,k)$ be the set of such n-state, k-symbol Turing machines. Given a machine $M \in \mathrm{TM}(n,k)$, let $s(M)$ be the number of transitions it executes before halting, including the final Halt instruction[50] and let $s(M) = \infty$ if M does not halt. Then, $\mathrm{BB}(n,k)$ is defined [1] by

$$\mathrm{BB}(n,k) = \max_{M \in \mathrm{TM}(n,k),\ s(M)<\infty} s(M)$$

In other words, $\mathrm{BB}(n,k)$ is the number of steps by a machine in $\mathrm{TM}(n,k)$ that runs the longest without halting. By convention, $\mathrm{BB}(n) = \mathrm{BB}(n,2)$, this being the most classic and well-studied busy beaver function.

Some conventions: A busy beaver candidate is a Turing machine for which we don't currently *know* whether it halts or not on blank input. A busy beaver contender is a Turing machine that halts on blank input. A busy beaver champion is a Turing machine that halts on blank input in more steps than any other *known* machine with the same number of states and symbols.

A Turing machine configuration is given by: the current state, the tape contents, and an integer tape head position. We sometimes write configurations in the following condensed format: **state_name**, $\cdots * * \underline{*} * * * \ldots$ with $*$ any tape symbol of the machine and $\underline{\ }$ for tape head position. Let c_1 and c_2 two configurations of some machine M and let $k \in \mathbb{N}$, we write $c_1 \vdash_M^k c_2$ to mean that M transitions from c_1 to c_2 in k steps. We write $\vdash$ (without M) if M is clear from the context.

2.4 FIVE STATES, FOUR SYMBOLS TURING MACHINE

In this section we prove Theorem 2.3. To do this we define, and prove correct, a 5-state 4-symbol Turing machine that searches for a counterexample to Erdős' conjecture, the machine is given in Figure 2.3. The construction begins with the **mul2** finite state transducer (FST) in Figure 2.2.

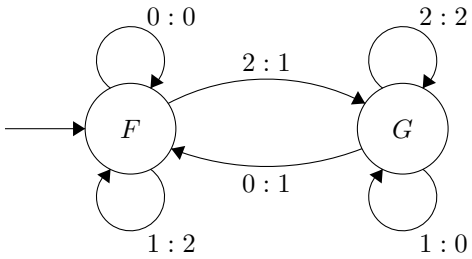


Figure 2.2: The **mul2** Finite State Transducer that multiplies a reverse-ternary represented number (base-3 written in reverse digit order) by 2. For example, the base 10 number 64_{10} is 2101_3 in base 3, which we represent in reverse-ternary with a leading zero to give the input 10120, which in turn yields the FST output 20211; the reverse-ternary of 128_{10}. Transition arrows are labelled $r : w$ where r is the read symbol and w is the write symbol.

A similar FST, and its 'dual', can be used to compute iterations of the Collatz map [156] (Appendix B). The fact that there is an FST that multiplies by 2 in base 3 is not surprising, since there is one for any affine transformation in any natural-number base ≥ 1 [2]. However, in this section, we will exploit **mul2**'s small size. We begin by proving its behaviour is correct (by reverse-ternary we mean the base-3 representation written in reverse digit order):

[50] As in [1], we took the liberty of not defining a symbol to write and a direction to move the tape head to when the halting instruction is performed as this does not change the number of transitions that the machine executed.

Lemma 2.6. Let $x \in \mathbb{N}$ and let $w = w_1 \dots w_n 0 \in \{0,1,2\}^n 0$ be its reverse-ternary representation, with $n \geq 0$, and a single leading 0. Then, on input w the **mul2** FST outputs $\gamma = \gamma_1 \gamma_2 \dots \gamma_{n+1} \in \{0,1,2\}^{n+1}$ which represents $y = 2x$ in reverse-ternary.

Proof. We give an induction on ternary word length n, with the following induction hypothesis: given a word $w = w_1 \dots w_n 0$ that represents $x \in \mathbb{N}$ (as in the lemma statement), if the FST reads from state F then the operation $x \mapsto 2x$ is computed in reverse-ternary, and if the FST reads from state G then the operation $x \mapsto 2x + 1$ is computed in reverse-ternary.

For the base case, when $n = 0$ we have $w = 0$ and, when started from state F the FST outputs 0 and when started from state G the FST outputs 1 which corresponds to respectively applying $x \mapsto 2x$ and $x \mapsto 2x + 1$.

Let's assume that the induction hypothesis holds for n and consider the base 3 word $w = w_1 \dots w_{n+1} 0 \in \{0,1,2\}^{n+1} 0$ that represents some $x \in \mathbb{N}$. We first handle state F. There are three cases for the value of the least significant digit $w_1 \in \{0,1,2\}$. If $w_1 = 0$, from the FST state F the first transition will output 0 and return to state F. Then, from state F, by applying the induction hypothesis to the length-m word $w_2 \dots w_{m+1}$, which represents the number $(x - w_1)/3 = x/3$ in base 3, we get that the FST output (on $w_2 \dots w_{n+1}$) is the representation of $2\frac{x}{3}$.

Then, by including the first output 0, the complete output of the FST represents the number $0 + 3 \cdot 2\frac{x}{3} = 2x$ which is what we wanted. Similarly, if $w_1 = 1$ the FST outputs the representation of the number $2 + 3 \cdot 2\frac{x-1}{3} = 2x$, or if $w_1 = 2$ the FST outputs the representation of $1 + 3 \cdot (2\frac{x-2}{3} + 1) = 2x$ since it moves to state G after the first transition. Likewise, if we start the FST in state G the FST outputs: $1 + 3 \cdot 2\frac{x}{3} = 2x + 1$ if $w_0 = 0$, or $0 + 3 \cdot (2\frac{x-1}{3} + 1) = 2x + 1$ if $w_0 = 1$, or $2 + 3 \cdot (2\frac{x-2}{3} + 1) = 2x + 1$ if $w_0 = 2$. In all the cases we get the result. □

	mul2__F	**mul2__G** (init)	**find__2**	**rewind**	**check__halt**
0	0 R **mul2__F**	1 R **mul2__F**	0 L **find__2**	0 L **rewind**	1 R **rewind**
1	2 R **mul2__F**	0 R **mul2__G**	1 L **find__2**	1 L **rewind**	2 R **rewind**
2	1 R **mul2__G**	2 R **mul2__G**	2 L **rewind**	2 L **rewind**	Halt
# (blank)	# L **find__2**	1 R **mul2__F**	# L **check__halt**	# R **mul2__F**	0 R **rewind**

Figure 2.3: 5-state 4-symbol Turing machine $M_{5,4}$ that halts if and only if Erdős' conjecture is false. The initial state of the machine is **mul2__G**, denoted '(init)'. The blank symbol is # and, since this is a busy-beaver candidate, the initial tape is empty: $\dots ##\underline{#}### \dots$ (tape head underlined). Example 2.7 shows the initial 333 steps of $M_{5,4}$. States **mul2__F** and **mul2__G** implement states F and G of the "mul2" FST in Figure 2.2, that multiplies a reverse-ternary number by 2. The other states check whether the result is a counterexample to, or one of the three special cases of, Erdős' conjecture.

Intuitively, the 5-state 4-symbol Turing machine $M_{5,4}$ in Figure 2.3 works as follows. Starting from all-# tape and state **mul2__G**, $M_{5,4}$ constructs successive natural number powers of 2 in base 3 (in fact in reverse-ternary) by iterating the **mul2** FST, which is embedded in its states **mul2__F** and **mul2__G**. Then, for each power of two, $M_{5,4}$ checks that there is at least one digit 2 using state **find__2**. If at least one digit 2 is found, then, using state **rewind** the machine goes back to the start (left) and iterates on to the next power of 2. If no digit 2 is found, such as in the three known special cases $1_{10} = 1_3$, $4_{10} = 11_3$ and $256_{10} = 100111_3$ (giving the condition $n > 8$ in Erdős' conjecture), then a counter is incremented on the tape (using state **check__halt**) and if this counter goes beyond value three the machine halts. The machine halts, iff we have found a counterexample to Erdős' conjecture, a fact we prove formally below in Theorem 2.3.

Example 2.7. Here, we highlight 11 out of the first 334 configurations of $M_{5,4}$. Five of the first 16 configurations are shown on the left (read from top to bottom), and 6 out of configurations 17 to 334 are

shown on the right.[47] From step 5 onwards, the content of the tape is of the form $c\#w_1 \dots w_n\#$ with $c, w_i \in \{0, 1, 2\}$, where c represents a single-symbol counter keeping track of the 3 special cases of Erdős conjecture and $w_1 \dots w_n$ is the reverse-ternary representation of a power of two. For instance, in the final configuration below we have 1101011202221 as the reverse-ternary representation of $2^{20} = 122021101011_3$.

	mul2_G,	...###**#**##...	$\vdash^1$	**rewind**,	...#1**#**11#...
$\vdash^5$	**rewind**,	...##0**#**1#...	$\vdash^6$	**rewind**,	...#1**#**22#...
$\vdash^4$	**rewind**,	...##0**#**2#...	$\vdash^{40}$	**rewind**,	...#1**#**20211#...
$\vdash^4$	**find_2**,	...#0#1**1**#...	$\vdash^8$	**find_2**,	...#1#11100**1**#...
$\vdash^3$	**check_halt**,	...#**0**#11#...	$\vdash^8$	**rewind**,	...#2**#**111001#...
			$\vdash^{254}$	**rewind**,	...#2**#**1101011202221#...

Theorem 2.3. There is an explicit 5-state 4-symbol Turing machine that halts if and only if Erdős' conjecture is false.

Proof. The machine is called $M_{5,4}$ and is given in Figure 2.3. We index tape positions by integers in $\mathbb{Z}$. Initially (at step 0), the tape head is at position 0 and each position of the tape contains the blank symbol #. The construction organises the tape as follows: position -1 holds a counter to keep track of the 3 known special cases of the Erdős' conjecture (1, 4 and 256 in base 10 which are 1, 11 and 100111 in base 3), position 0 always contains a blank # to act as a separator, and positive positions will hold reverse-ternary representations of powers of two, i.e. least significant digit at position 1 and most significant at position $\lceil \log_3(2^n) \rceil$. States **mul2_F** and **mul2_G** reproduce the logic of the "**mul2**" FST (Figure 2.2) where, in state **mul2_G** the blank symbol behave like ternary digit 0 while in **mul2_F** it triggers the end of the multiplication by 2.

Starting the Turing machine in state **mul2_G** appropriately initialises the process, the reader can verify that at step 5 the machine is in state **rewind**, the tape head is at position 0 and the tape content between positions -1 to 2 included is: 0#1# (tape head position underlined) and all other positions are blank.[51] From there we prove the following result (IH) by induction on $n \in \mathbb{N}$: at step $s_n = 5 + c_n + \sum_{k=1}^{n} 2 * (\lceil \log_3(2^k) \rceil + 1)$ either (a) $M_{5,4}$ has halted, the tape head is at position -1 and the reverse-ternary represented number written on the positive part of the tape (with digits in reverse order) is a counterexample to Erdős' conjecture, i.e. it has no digit equal to 2 and is of the form 2^{n_0} with $n_0 > 8$, or (b) $M_{5,4}$ is in state **rewind**, the tape head is at position 0 which contains a blank symbol and the reverse-ternary represented number by the digits between position 1 and $\lceil \log_3(2^n) \rceil$ is equal to 2^n and all positions coming after $\lceil \log_3(2^n) \rceil$ are blank. The number c_n in the expression of s_n accounts for extra steps that are taken by $M_{5,4}$ to increment the counter at position -1 which keeps track of the three known special cases of the conjecture: $1 = 1_3$, $4 = 11_3$ and $256 = 100111_3$. In practice, c_n is defined by $c_n = 0$ for $n \leq 1$, $c_n = 2$ for $2 \leq n \leq 7$ and $c_n = 4$ for $n \geq 8$.

For $n = 0$ we have $s_0 = 5$ and case (b) of the induction hypothesis (IH) is verified as $1 = 2^0$ is represented on the positive part of the tape as seen above.

Let's assume that the result holds for $n \in \mathbb{N}$. If $M_{5,4}$ had already halted at step s_n, then case (a) of IH still holds and nothing is left to prove. Othewise, by case (b) of IH, at step s_n then $M_{5,4}$ is in state **rewind** at position 0 which contains a # and between positions 1 and $\lceil \log_3(2^n) \rceil + 1$ the following word is written: $w = w_1 \dots w_{\lceil \log_3(2^n) \rceil}\#$ such that $w_1 \dots w_{\lceil \log_3(2^n) \rceil}$ is the reverse-ternary representation of 2^n. At step $s_n + 1$, tape position 0 still contains the blank symbol #, $M_{5,4}$ is in state **mul2_F** and the tape head is at position 1 where it begins simulating the **mul2** FST on w. By having the final # of w encode a

[51] State **mul2_G** was not designed to kick-start the process, but starting with it happens to give us what we need.

0 then Lemma 2.6 applies which means that, after scanning w, the positive part of the tape contains the reverse-ternary expression of $2 * 2^n = 2^{n+1}$. Either $M_{5,4}$ was in state **mul2__F** when it read the final # of w, in which case it will stop simulating the FST and jump to state **find__2** and move the tape head to the left. Otherwise it was in state **mul2__G** and read the # in which case it jumps to state **mul2__F** and moves the tape head to the right where, by IH, a # will be read bringing us back to the previous case. In both cases, $\lceil \log_3(2^{n+1}) \rceil + 1$ symbols have been read since s_n, $M_{5,4}$ is in state **find__2** and all positions after and including $\lceil \log_3(2^{n+1}) \rceil + 1$ are still blank.

State **find__2** will scan to the left over the reverse-ternary expression that was just computed to search for a digit equal to 2. If a 2 is found, then it switches to state **rewind** and will reach position 0 at step $s_n + 2 * (\lceil \log_3(2^{n+1}) \rceil + 1)$. In that case, we also have $c_{n+1} = c_n$ as by definition of c_n, one can verify that $c_{n+1} \neq c_n$ implies that 2^{n+1} has no ternary digit equal to 2. Hence, if a 2 was found the machine reaches tape position 0 at step $s_{n+1} = 5 + c_{n+1} + \sum_{k=1}^{n+1} 2 * (\lceil \log_3(2^k) \rceil + 1)$, position 0 contains the blank symbol, positions 1 to $\lceil \log_3(2^{n+1}) \rceil$ hold the reverse-ternary representation of 2^{n+1} and all positions after $\lceil \log_3(2^{n+1}) \rceil$ are still blank, which is what we wanted under case (b) of IH. If no 2s were found then $M_{5,4}$ will reach position -1 in state **check__halt** and if $n+1 = 2$ or $n+1 = 8$ it will respectively read a 0 or 1 which will respectively be incremented to 1 or 2, then $M_{5,4}$ goes back to position 0 in state **rewind** and case (b) of IH is verified as the value c_n accounts for the extra steps that were taken to increment the counter in those cases. However, if $n+1 > 8$ then $M_{5,4}$ will read a 2 at position -1 and consequently halt with positions 1 to $\lceil \log_3(2^{n+1}) \rceil$ giving the base three representation of 2^{n+1} containing no digit equal to two: a counterexample to Erdős' conjecture has been found and it is consequently false and case (a) of IH holds.

From this proof, if $M_{5,4}$ halts then Erdős' conjecture is false. For the reverse direction, note that $M_{5,4}$ will test every single successive power of two until it finds a potential counterexample meaning that if Erdős' conjecture is false $M_{5,4}$ will find the smallest counterexample and stop there. Hence the 5-state, 4-symbol Turing machine given in Figure 2.3 halts if and only if Erdős' conjecture is false. □

2.5 Fifteen states, two symbols Turing machine

	mul2_F_sim
a	a R **mul2_F_a**
b	b R **mul2_F_b**

	mul2_G_sim (init)
a	a R **mul2_G_a**
b	a R **mul2_G_b**

	find_2_sim
a	a L **find_2_a**
b	b L **find_2_b**

	rewind_sim
a	a L **rewind_a**
b	b L **rewind_b**

	check_halt_sim
a	b L **check_halt_a**
b	a R **rewind_b**

	mul2_F_a
a	b R **mul2_G_sim**
b	a R **mul2_F_sim**

	mul2_G_a
a	a R **mul2_G_sim**
b	a L **mul2_G_extra**

	find_2_a
a	a L **rewind_sim**
b	b L **find_2_sim**

	rewind_a
a	a L **rewind_sim**
b	b L **rewind_sim**

	check_halt_a
a	Halt
b	a R **mul2_G_extra**

	mul2_F_b
a	a R **mul2_F_sim**
b	b L **find_2_a**

	mul2_G_b
a	b R **mul2_F_sim**
b	b R **mul2_F_sim**

	find_2_b
a	a L **find_2_sim**
b	b L **check_halt_sim**

	rewind_b
a	a L **rewind_sim**
b	b R **mul2_G_b**

	mul2_G_extra
a	b R **mul2_G_a**
b	b R **rewind_b**

Encoding E

0	ba
1	ab
2	aa
#	bb

Figure 2.4: 15-state 2-symbol Turing machine $M_{15,2}$ that halts if and only if Erdős' conjecture is false. The initial state is **mul2__G__sim**, denoted '(init)'. The blank symbol is b, and, since this is a busy-beaver candidate, the initial tape is empty: $\ldots bb\underline{\mathbf{b}}bb\ldots$ (tape head position bold and underlined). States are organised into 5 columns, one for each state of $M_{15,2}$ in Figure 2.3, and inherit name prefixes from $M_{15,2}$. In Lemma 2.11 we prove that $M_{15,2}$ simulates $M_{5,4}$.

In this section, we prove Theorem 2.2. To do this we define the 15-state 2-symbol Turing machine $M_{15,2}$ in Figure 2.4 and then prove that it halts if and only if it finds a counterexample to Erdős' conjecture. We begin with some remarks to guide the reader.

2.5.1 Intuition and overview of the construction

Turing machines with two symbols can be challenging to reason about and prove correctness of for two reasons: (1) the tape is a difficult-to-read stream of bits and (2) simple algorithmic concepts need to be distributed across multiple states (because of the low number of symbols). We took the approach of first designing $M_{5,4}$ (Figure 2.3, relatively easy to understand), and then designing $M_{15,2}$ to simulate $M_{5,4}$. (With the caveat that both machines have a few program-size optimisations.)

In order to avoid confusion, $M_{15,2}$ uses alphabet $\{a, b\}$ (with b blank) which is distinct to that of $M_{5,4}$. In Lemma 2.11 we prove that $M_{15,2}$ *simulates* $M_{5,4}$, via a tight notion of simulation given in Definition 2.9 and the Lemma statement. Symbols of $M_{5,4}$ are encoded by those of $M_{15,2}$ using the encoding function $E : \{\#, 0, 1, 2\} \to \{a, b\}^2$ defined by $E(\#) = bb$, $E(0) = ba$, $E(1) = ab$, $E(2) = aa$. Intuitively, the 15 states of $M_{15,2}$ are partitioned into 5 sets: the idea is that each of the five states of $M_{5,4}$ is simulated by one of the five corresponding *column of states* in Figure 2.4.

The naming convention for states in $M_{15,2}$ is as follows: for each state of $M_{5,4}$, there is a state with the same name in $M_{15,2}$, followed by **__sim** (short for 'simulate'), that is responsible for initiating the behaviour of the corresponding state in $M_{5,4}$. Then, from each **__sim** state in $M_{15,2}$ control moves to one of two new states, suffixed by **__a** and **__b**, after reading symbol a or symbol b. At the next step, two consecutive letters have been read and $M_{15,2}$ knows which of the 4 possible cases of the encoding function E it is considering and has sufficient information to simulate one step of $M_{5,4}$. However, in many cases, for program-size efficiency reasons, $M_{15,2}$ makes a decision before it has read both symbols of the encoding.

There are two exceptions to the $M_{15,2}$ state naming rule: there is no state **check__halt__b** as our choice of encoding function E meant not needing to consider that case when simulating **check__halt**, and the state **mul2__G__extra** which is involved in simulation of both **mul2__G** and **check__halt**.

Example 2.8. Here, we highlight 11 of the first 742 configurations of $M_{15,2}$. The example reads from top to bottom, then left to right. These configurations correspond represent the simulation (via encoding E) of those highlighted in Example 2.7. For instance, in the second configuration shown we have $bb\ ba\ bb\ ab\ bb\ = E(\#)E(0)E(\#)E(1)E(\#)$ which corresponds to the tape content of the second configuration shown in Example 2.7. Simulation comes with a (merely linear) cost in time as here, the first 333 steps of $M_{5,4}$ are simulated in 741 steps in $M_{15,2}$.

	mul2__G__sim,	$\ldots bb\ bb\ \underline{\mathbf{b}}b\ bb\ bb \ldots$	$\vdash^3$	**rewind__b**,	$\ldots bb\ ab\ \underline{\mathbf{b}}b\ ab\ ab\ bb \ldots$
$\vdash^{10}$	**rewind__b**,	$\ldots bb\ ba\ \underline{\mathbf{b}}b\ ab\ bb \ldots$	$\vdash^{13}$	**rewind__sim**,	$\ldots bb\ ab\ b\underline{\mathbf{b}}\ aa\ aa\ bb \ldots$
$\vdash^9$	**rewind__sim**,	$\ldots bb\ ab\ b\underline{\mathbf{b}}\ aa\ bb \ldots$	$\vdash^{92}$	**rewind__sim**,	$\ldots bb\ ab\ b\underline{\mathbf{b}}\ aa\ ba\ aa\ ab\ ab\ bb \ldots$
$\vdash^{10}$	**find__2__sim**,	$\ldots bb\ ba\ bb\ ab\ a\underline{\mathbf{b}}\ bb \ldots$	$\vdash^{22}$	**find__2__sim**,	$\ldots bb\ ab\ bb\ ab\ ab\ ab\ ba\ ba\ a\underline{\mathbf{b}}\ bb \ldots$
$\vdash^6$	**check__halt__sim**,	$\ldots bb\ b\underline{\mathbf{a}}\ bb\ ab\ ab\ bb \ldots$	$\vdash^{15}$	**rewind__b**,	$\ldots bb\ aa\ \underline{\mathbf{b}}b\ ab\ ab\ ab\ ba\ ba\ ab\ bb \ldots$
			$\vdash^{561}$	**rewind__sim**,	$\ldots bb\ aa\ b\underline{\mathbf{b}}\ ab\ ab\ ba\ ab\ ba\ ab\ ab\ aa\ ba\ aa\ aa\ aa\ ab\ bb \ldots$

2.5.2 Proof of correctness

We define what we mean by 'simulates'; the definition couples the dynamics of two machines so that the tape content of one machine is mapped in a straightforward way to that of the other.

Definition 2.9 (simulates)**.** Let M and M' be single-tape Turing machines with alphabets Σ and Σ'. Then, M' *simulates* M if there exists $m \in \mathbb{N}$, a function $E : \Sigma \to \Sigma'^m$ and a computable *time-scaling* function $f : \mathbb{N} \to \mathbb{N}$, such that for all time steps $n \in \mathbb{N}$ of M, then for step $f(n)$ of M', either both M and M' have already halted, or else they are both still running and, if M has tape content $\ldots t_{-1}^n t_0^n t_1^n \ldots$ (where $t_i \in \Sigma$) then M' has tape content $\ldots E(t_{-1}^n)E(t_0^n)E(t_1^n)\ldots$.

Definition 2.10 ($M_{15,2}$'s encoding function E)**.** Let $E : \{\#, 0, 1, 2\} \to \{a, b\}^2$ be $M_{15,2}$'s encoding function, where $E(\#) = bb$, $E(0) = ba$, $E(1) = ab$ and $E(2) = aa$.

Our main technical lemma in this section states that $M_{15,2}$ (Figure 2.4) simulates $M_{15,2}$ (Figure 2.3):

Lemma 2.11. Machine $M_{15,2}$ simulates $M_{5,4}$, according to Definition 2.9, with encoding function E (Definition 2.10) and time-scaling function f recursively defined by: $f(0) = 0$ and $f(n+1) = f(n) + g((q_n, d_n), \sigma_n)$ with: q_n being the state of $M_{5,4}$ at step n, $d_n \in \{\mathrm{L}, \mathrm{R}\}$ the tape head direction at step $n-1$ (where $d_0 = \mathrm{R}$), $\sigma_n \in \{\#, 0, 1, 2\}$ the read symbol at step n, and g the partial function defined in Figure 2.5.

Proof. Let $S = \{\textbf{mul2\_F}, \textbf{mul2\_G}, \textbf{find\_2}, \textbf{rewind}, \textbf{check\_halt}\}$ be the set of 5 states of $M_{5,4}$ and S' be the set of 15 states of $M_{15,2}$ given in Figure 2.4. We prove the result by induction on $n \in \mathbb{N}$, the number of steps taken by the *simulated* machine $M_{5,4}$. Let $k_n = (q_n, d_n) \in S \times \{\mathrm{L}, \mathrm{R}\}$ be $M_{5,4}$'s state at step n together with the tape-head direction at the *previous* step $n-1$ (we take the convention $d_0 = \mathrm{R}$), let $\sigma_n \in \{0, 1, 2, \#\}$ be $M_{5,4}$'s read symbol at step n, and let $i_n \in \mathbb{Z}$ be $M_{5,4}$'s tape head position at step n. We note that by inspecting Figure 2.3 the set of possible values for k_n is $K = \{(\textbf{mul2\_F}, \mathrm{R}), (\textbf{mul2\_G}, \mathrm{R}), (\textbf{find\_2}, \mathrm{L}), (\textbf{check\_halt}, \mathrm{L}), (\textbf{rewind}, \mathrm{L}), (\textbf{rewind}, \mathrm{R})\}$, observing that **rewind** is the only $M_{5,4}$ state reachable from both left and right tape head moves.

The induction hypothesis (IH) is the definition of simulation (Definition 2.9) instantiated with E (Definition 2.10), f as defined in the lemma statement, and the following: at step $f(n)$ machine $M_{15,2}$'s head is at position $2i_n + \pi(d_n)$ with $\pi(L) = 1$ and $\pi(R) = 0$ and the machine is in state $h(k_n)$ with $h : K \to S'$ defined by $h(\textbf{mul2\_F}, \mathrm{R}) = \textbf{mul2\_F\_sim}$, $h(\textbf{mul2\_G}, \mathrm{R}) = \textbf{mul2\_G\_sim}$, $h(\textbf{find\_2}, \mathrm{L}) = \textbf{find\_2\_sim}$, $h(\textbf{check\_halt}, \mathrm{L}) = \textbf{check\_halt\_sim}$, $h(\textbf{rewind}, \mathrm{L}) = \textbf{rewind\_sim}$ and $h(\textbf{rewind}, \mathrm{R}) = \textbf{rewind\_b}$.

If $n = 0$, we have $f(0) = 0$ and both tapes are entirely blank: $\ldots\#\#\#\ldots$ for $M_{5,4}$, and $\ldots bbb\ldots$ for $M_{15,2}$ which is consistent with $E(\#) = bb$. Additionally, $k_0 = (\textbf{mul2\_G}, R)$, the tape head of $M_{15,2}$ is at position $0 = 2i_0 + 0$ with $i_0 = 0$ being the tape head position of $M_{5,4}$ and finally, $M_{15,2}$ is in state $h(k_0) = \textbf{mul2\_G\_sim}$ (which is also its initial state).

Let's assume that the induction hypothesis (IH) holds for $n \in \mathbb{N}$. If at step n for $M_{5,4}$, and step $f(n)$ for $M_{15,2}$, both machines have already halted they will still have halted at any future step and the IH still holds. Let's instead assume from now on, IH holds and we are in the second case of Definition 2.9 (both are still running and tape contents are preserved under the encoding function E).

If $k_n = (\textbf{mul2\_F}, \mathrm{R})$ and $\sigma_n = 2$ then the configuration of $M_{5,4}$ at step n is **mul2_F**, $\ldots\underline{\mathbf{2}}\ldots$ with tape head at position i_n. By Figure 2.3, at step $n+1$ the configuration becomes **mul2_G**, $\ldots 1\underline{*}\ldots$ and $i_{n+1} = i_n + 1$ (with $*$ being whatever symbol is at position $i_n + 1$) and $k_{n+1} = (\textbf{mul2\_G}, \mathrm{R})$. By IH, at step $f(n)$ the tape head position of $M_{15,2}$ is $2i_n + \pi(\mathrm{R}) = 2i_n$ and the configuration of $M_{15,2}$ is **mul2_F_sim**, $\ldots\underline{\boldsymbol{a}}a\ldots$, since $E(2) = aa$ and $h(k_n) = \textbf{mul2\_F\_sim}$. By Figure 2.4, at step $f(n+1) = f(n) + g((\textbf{mul2\_F}, \mathrm{R}), 1) = f(n) + 2$, the configuration of $M_{15,2}$ is **mul2_G_sim**, $\ldots ab\underline{*}\ldots$ with tape head at position $2i_n + 2$. We have $E(1) = ab$ (and no other tape positions than $2i$ and $2i+1$ were modified), $2i_n + 2 = 2(i_n + 1) = 2i_{n+1} + \pi(\mathrm{R}) = 2i_{n+1} + \pi(d_{n+1})$ and **mul2_G_sim** $= h(k_{n+1})$ which is everything we needed to satisfy IH at step $n+1$. We leave verifications of cases $\sigma_n = 0$ and $\sigma_n = 1$ to the reader as they are very similar. If $\sigma_n = \#$ then $M_{5,4}$ writes $\#$ then goes left to state **find_2**, which gives $k_{n+1} = (\textbf{find\_2}, \mathrm{L})$ and $i_{n+1} = i_n - 1$. By IH, at step $f(n)$ the tape head position of $M_{15,2}$ is $2i_n + \pi(\mathrm{R}) = 2i_n$ and the configuration of $M_{15,2}$ is **mul2_F_sim**, $\ldots\underline{\boldsymbol{b}}b\ldots$. After $g((\textbf{mul2\_F}, \mathrm{R}), \#) = 3$ steps, the configuration becomes **find_2_sim**, $\ldots\underline{*}bb\ldots$. This is consistent with having $E(\#) = bb$, moving the tape head to position $2i_n - 1 = 2(i_n - 1) + 1 = 2i_{n+1} + \pi(d_{n+1})$ and having **find_2sim** $= h(k_{n+1})$ which is all we needed.

If $k_n = (\textbf{mul2\_G}, \mathrm{R})$ and $\sigma_n = 1$ then the configuration of $M_{5,4}$ at step n is **mul2_G**, $\ldots\underline{\mathbf{1}}\ldots$ with tape head at position i_n. By Figure 2.3, at step $n+1$ the configuration becomes **mul2_G**, $\ldots 0\underline{*}\ldots$

	(**mul2_F**,R)	(**mul2_G**,R)	(**find_2**,L)	(**check_halt**,L)	(**rewind**,L)	(**rewind**,R)
0	2	2	2	3	2	-
1	2	4	2	1	2	-
2	2	2	2	2	2	-
#	3	2	2	1	3	2

Figure 2.5: The partial function g of Lemma 2.11 that defines how many steps are needed by $M_{15,2}$ to simulate one step of $M_{5,4}$, for each (state, move-direction-on-previous-step), and symbol, of an $M_{5,4}$ step. The function is partial, defined on only one entry of the final column, as the proof of Theorem 2.3 shows that only symbol # can be read if $M_{5,4}$ reaches state **rewind** coming from the left (hence by moving tape head in direction R).

and $i_{n+1} = i_n + 1$ and $k_{n+1} = (\textbf{mul2\_G}, \mathrm{R})$. By IH, at step $f(n)$ the tape head position of $M_{15,2}$ is $2i_n + \pi(\mathrm{R}) = 2i_n$ and the configuration of $M_{15,2}$ is **mul2_G_sim**, $\ldots \underline{\boldsymbol{a}} b \ldots$ as $E(1) = ab$ and $h(k_n) =$ **mul2_G_sim**. In that case, the machine will need $g((\textbf{mul2\_G}, \mathrm{R}), 1) = 4$ steps as it will have to bo back one step to the left after having scanned the second symbol of $E(1) = ab$ in order to write the first symbol $E(0) = ba$. This is realised by the first transition of intermediate state **mul2_G_extra** as we are assured to read symbol a in that case. Altogether, we get that at step $f(n+1) = f(n) + 4$ the configuration of $M_{15,2}$ is **mul2_G_sim**, $\ldots ba\underline{*} \ldots$ with **mul2_G_sim** $= h(k_{n+1})$ and tape head at position $2i_n + 2 = 2i_{n+1} + \pi(d_{n+1})$ which is everything we need. We leave cases $\sigma_n \in \{0, 2, \#\}$ to the reader.

If $k_n = (\textbf{find\_2}, \mathrm{L})$, the simulation is straightforward for any $\sigma_n \in \{0, 1, 2, \#\}$ as the content of the tape is not modified and the tape head always moves to the left, hence we leave those cases to the reader.

If $k_n = (\textbf{rewind}, \mathrm{L})$ and $\sigma_n = \#$ then the configuration of $M_{5,4}$ at step n is **mul2_G**, $\ldots \underline{\boldsymbol{\#}} \ldots$ with tape head at position i_n. By Figure 2.3, at step $n+1$ the configuration becomes **mul2_F**, $\ldots \#\underline{*} \ldots$ and $i_{n+1} = i_n + 1$ and $k_{n+1} = (\textbf{mul2\_F}, \mathrm{R})$. By IH, at step $f(n)$ the tape head position of $M_{15,2}$ is $2i_n + \pi(\mathrm{L}) = 2i_n + 1$ and the configuration of $M_{15,2}$ is **rewind_sim**, $\ldots b\underline{\boldsymbol{b}} \ldots$ as $E(\#) = bb$ and $h(k_n) =$ **rewind_sim**. By Figure 2.4, at step $f(n+1) = f(n) + g((\textbf{rewind}, \mathrm{L}), \#) = f(n) + 3$, the configuration of $M_{15,2}$ is **mul2_F_sim**, $\ldots bb\underline{*} \ldots$ with $bb = E(\#)$, **mul2_F_sim** $= h(k_{n+1})$ and tape head at position $2i_n + 2 = 2i_{n+1} + \pi(d_{n+1})$, which is everything that we need. We leave cases $\sigma_n \in \{0, 1, 2\}$ to the reader as in those cases the tape head always moves to the left and the tape content is also not modified.

If $k_n = (\textbf{rewind}, \mathrm{R})$, by the proof of Theorem 2.2, we know that necessarily $\sigma_n = \#$, since in that case, the tape head of $M_{5,4}$ is at position $i_n = 0$ which always holds a # (the machine is just after incrementing the '3 special cases of Erdős' conjecture' counter at position -1), hence the configuration of $M_{5,4}$ at step n is **rewind**, $\ldots \underline{\boldsymbol{\#}} \ldots$ and at step $n+1$ it becomes **mul2_F**, $\ldots \#\underline{*} \ldots$ (Figure 2.3) and $i_{n+1} = i_n + 1$ and $k_{n+1} = (\textbf{mul2\_F}, \mathrm{R})$. By IH, at step $f(n)$ the tape head position of $M_{15,2}$ is $2i_n + \pi(\mathrm{R}) = 2i_n$ and the configuration of $M_{15,2}$ is **rewind_b**, $\ldots \underline{\boldsymbol{b}} b \ldots$ as $E(\#) = bb$ and $h(k_n) =$ **rewind_b**. By Figure 2.4, at step $f(n+1) = f(n) + g((\textbf{rewind}, \mathrm{R}), \#) = f(n) + 2$, the configuration of $M_{15,2}$ is **mul2_F_sim**, $\ldots bb\underline{*} \ldots$ with $bb = E(\#)$, **mul2_F_sim** $= h(k_{n+1})$ and the tape head is at position $2i_n + 2 = 2i_{n+1} + \pi(d_{n+1})$, which is everything that we need.

If $k_n = (\textbf{check\_halt}, \mathrm{L})$ then note that for $\sigma_n \in \{0, 1, \#\}$ machine $M_{5,4}$ will 'increment' the value of σ_n then transition to $k_{n+1} = (\textbf{rewind}, \mathrm{R})$ moving its head to $i_{n+1} = i_n + 1$. Cases $\sigma_n \in \{\#, 1\}$ are arguably where we make the most use of the encoding function E in order to use very few states in $M_{15,2}$ to simulate the behavior of $M_{5,4}$. Indeed, if $\sigma_n = \#$ or $\sigma_n = 1$ then $M_{5,4}$ must respectively write 1 and 2. This means that encodings must go from $E(\#) = bb$ to $E(1) = ba$ and from $E(1) = ab$ to $E(2) = aa$. By IH, machine $M_{15,2}$ is currently reading the second symbol of the encoding (because the head is at position $2i_n + \pi(\mathrm{L}) = 2i_n + 1$), which is symbol b, then it is enough for $M_{15,2}$, using

only one step, just to turn that b into an a which deals with both cases and then go right to state $h(k_{n+1}) =$ **rewind_b** while moving the head to $2i_n + 2 = 2i_{n+1} + \pi(d_{n+1})$ which is what we need. This gives $g((\textbf{check\_halt}, \mathrm{L}), \#) = g((\textbf{check\_halt}, \mathrm{L}), 1) = 1$. We let the reader verify the case $\sigma_n = 0$ which gives $g((\textbf{check\_halt}, \mathrm{L}), 0) = 3$.

If $k_n = (\textbf{check\_halt}, \mathrm{L})$ and $\sigma_n = 2$ then, at step $n{+}1$ machine $M_{5,4}$ will halt. By IH, at step $f(n)$ the tape head position of $M_{15,2}$ is $2i_n{+}\pi(\mathrm{L}) = 2i_n{+}1$ and the configuration of $M_{15,2}$ is **check_halt_sim**, $\ldots a\underline{\boldsymbol{a}}\ldots$ as $E(2) = aa$ and $h(k_n) =$ **check_halt_sim**. By Figure 2.4, at step $f(n{+}1) = f(n){+}g((\textbf{check\_halt}, \mathrm{L}), 2) = f(n) + 2$ machine $M_{15,2}$ halts. Hence if $M_{5,4}$ halts then $M_{15,2}$ halts. Machine $M_{15,2}$ has only one halting instruction and this proof shows that it is reached only in the case where $k_n = (\textbf{check\_halt}, \mathrm{L})$ and $\sigma_n = 2$ meaning that if $M_{15,2}$ halts then $M_{5,4}$ halts. Hence the machines are either both running at respectively step $n + 1$ and $f(n + 1)$ or both have halted.

In all cases, the induction hypothesis is propagated at step $n + 1$ and we get the result: machine $M_{15,2}$ simulates $M_{5,4}$ according to Definition 2.9. □

2.5.3 Main result and corollaries

Using our previous results, the proofs of Theorem 2.2, and Corollary 2.4 and 2.5 are almost immediate:

Theorem 2.2. There is an explicit 15-state 2-symbol Turing machine that halts if and only if Erdős' conjecture is false.

Proof. By Lemma 2.11, $M_{15,2}$ simulates $M_{5,4}$ which in turns means that $M_{15,2}$ halts if and only if $M_{5,4}$ halts. By Theorem 2.3 this means that $M_{15,2}$ halts if and only if Erdős' conjecture is false. □

Corollary 2.12. Erdős' conjecture is equivalent to the following conjecture over a finite set: for all $8 < n \leq \min(\mathrm{BB}(15), \mathrm{BB}(5, 4))$ there is at least one digit 2 in the base 3 representation of 2^n.

Proof. From the proof of Theorem 2.3, If we run $M_{5,4}$ then, at step $\mathrm{BB}(5, 4) + 1 \in \mathbb{N}$ we know if Erdős' conjecture is true or not: it is true if and only if $M_{5,4}$ is still running. If $M_{5,4}$ is still running, and because the machine outputs fewer than one power of 2 per step, at step $\mathrm{BB}(5, 4) + 1$ the power of 2 written on the tape is at most $2^{\mathrm{BB}(5,4)}$. Analogously, $M_{15,2}$ writes fewer than one power of 2 per step, hence, at step $\mathrm{BB}(15) + 1 \in \mathbb{N}$ the power of 2 written on the tape is at most $2^{\mathrm{BB}(15)}$. In either case, by then, we know whether the conjecture is true or not. Hence, it is enough to check Erdős' conjecture for all $n \leq \min(\mathrm{BB}(15), \mathrm{BB}(5, 4))$ and we get the result. □

Corollary 2.13. Let $x \in \mathbb{N}$ be the smallest counterexample to Erdős conjecture, if it exists. Then we have: $\mathrm{BB}(15) \geq \log_2 x$ and $\mathrm{BB}(5, 4) \geq \log_2 x$.

Proof. By Theorems 2.2 and 2.3, if Erdős' conjecture has a counterexample x, then we have explicit busy beaver contenders for BB(15) and BB(5, 4), respectively, i.e. machines $M_{15,2}$ and $M_{5,4}$. These machines both halt with $x = 2^{n_0}$, $n_0 \in \mathbb{N}$, written on their tape in base 3 (by the proofs of Theorems 2.2 and 2.3). $M_{15,2}$ and $M_{5,4}$ enumerate less than one power of 2 per time step, hence their running time is $\geq n_0 = \log_2 x$, giving the stated lowerbound on BB(15) and BB(5, 4). □

Chapter 3

Small tile sets that compute while solving mazes

3.1 Foreword

In Chapter 1, we did not manage to show how to run arbitrary computations with the 6 Collatz tiles. We have shown that many natural prediction problems with these tiles are in NC^1 (Chapter 1, Section 1.4), meaning that it is very unlikely that the tiles could simulate Turing machines *efficiently* since it is widely believed that $P \neq NC^1$. However, we know that with these tiles complex patterns occur easily (for instance, Chapter 2, Figure 2.1), and one is tempted to harness them in order to simulate Boolean logic. That is exactly what we do in this chapter, we simulate arbitrary Boolean circuits with the 6 tiles, at the cost of one change to the model: we allow our assemblies to be disconnected. In this new model, that we call the *Maze-Walking TAM* (in reference to Winfree's *abstract Tile Assembly Model*, *aTAM*, [174]), we also show that even just 4 tiles can have enough power to simulate arbitrary circuits. While building disconnected assemblies could seem unphysical, the intuition that we use and further develop in the wet-lab in Chapter 4, is that all the components of our assemblies are linked to – or more precisely for Chapter 4, operate *within* – an underlying surface or structure, such as a DNA origami scaffold. Hence, while the focus of this chapter is to prove purely theoretical results, it gives relatively realistic (simple, and perhaps implementable) constructions for simulating arbitrary Boolean circuits on a surface, or lattice, with pre-seeded/attached disconnected components, and gives the theoretical motivation for our model and experiments in Chapter 4. As stated, our constructions use a very small number of tile types (only 6 or 4) which can be a useful feature in an experimental context as it requires implementation of only these few possible glue-interactions for the model to reach full Turing-complete power.

3.2 Introduction

We can think of solving a maze as performing computation: the input is a maze, some starting location(s) and an ending location, and the answer to the computation is a yes/no answer signifying whether the exit is reachable from the start, or even an explicit path from start to exit. Figure 3.1 shows how a maze encodes a circuit of OR gates: solving the maze is equivalent to executing the OR circuit with all inputs set to bit 1; and asking about paths in the maze is equivalent to setting some inputs to 1

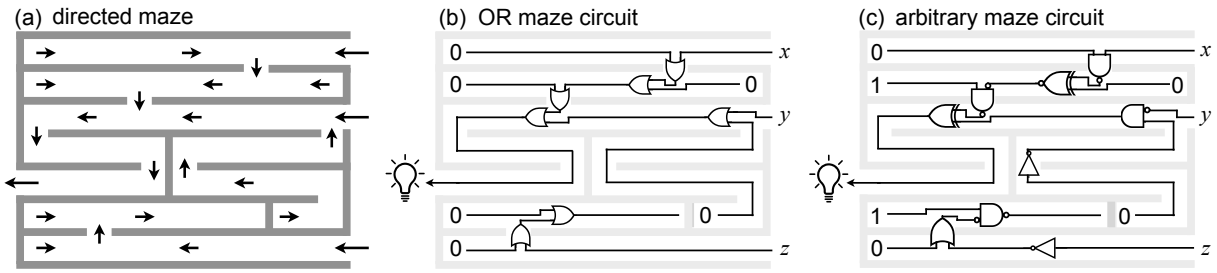


Figure 3.1: Mazes, computation and Boolean circuits. Solving a *directed* maze (a) is formally equivalent to executing an OR circuit (b) if we ask: Are any of the input bits that are set to 1 connected to the output gate? The example in (b) accepts any 3-bit input x, y, z that sets x or y to 1, irrespective of z; equivalently the maze is solvable from the top two inputs only. Even such a simple setup can be used to solve any suitably-encoded problem in the class nondeterministic logspace (NL). (c) We generalise this notion of 'computation via maze solving' in a natural way by having the maze specify arbitrary Boolean gates along the route that need to be evaluated. In the Maze-Walking Tile Assembly Model defined in Section 3.4.1, tiles flow through the maze, building paths from the entrances to the exit, evaluating the circuit as they go.

and seeing which paths have 1 flowing all the way through them. It then becomes meaningful to ask about the computational power of systems capable of solving mazes [119], for example DNA or molecular walker-based systems.

The difficulty of maze solving varies with the complexity of the maze, such as number of dimensions, grid layout versus more general graph, degree of nodes, or whether graph edges are directed or undirected. In computational complexity theory terminology, solving mazes and more general graph reachability problems lie within the class NL, i.e. problems solvable on a nondeterministic Turning machine that uses temporary workspace only logarithmic in input length. At the simplest level, and perhaps counter-intuitively, a system that solves a directed maze consisting of (a number of possibly disconnected) straight line segments has enough computational power to solve any problem in L, the deterministic version of NL.[52]

Here, we suggest two modifications to the problem of solving mazes, which are expressive enough to endow maze solvers with significant computational power (their prediction problem becomes P-complete), yet, we contend, simple enough to be experimentally feasible using DNA engineering and computing principles. The first, and most important, is that we generalise mazes to have paths patterned with logic gates that must be solved in order to pass by them (Figure 3.1(c)). For a maze-walker this would mean it should be able to input one or two bits of information from the site it stands upon, compute, and then output one or two bits to adjacent sites. The second, mainly to keep things simple, is that we assume mazes are directed (meaning a pair of adjacent positions have one directed edge between them that dictates the direction of information flow) and have no cycles. Since we allow for fanout of 0, 1 or 2 per site, one needs to generalise the typical notion of maze-solving somewhat: Are walkers replicating themselves to handle fanout of 2? Or are they leaving little bit-encoding messages for other walkers/themselves to pick up later? Somewhat similarly, how do they handle fanin of 2? These considerations lend themselves to various models, however here we focus on having information-manipulating *tiles* flow through the maze, much like lava flowing from a down a complex volcanic hillside, but clever lava that computes as it moves. Our model is called the Maze-Walking Tile Assembly Model, or Maze-Walking TAM.

The programmer specifies a set of square tiles, with glues on the sides. A problem instance, or maze, is

[52]The PATHREACHABILITY problem is L-complete: given a directed graph whose edges form a set of disconnected line segments (in- and out-degree ≤ 1), and two nodes s and t, is t reachable from s? A deterministic Turing machine can start at s and walk along the graph use only logarithmic workspace (in input length) to keep track of the current node, answering "yes" if it reaches t and "no", if it instead reaches a dead-end. Hence the problem is in L. Conversely, the set of configurations of a deterministic logspace Turing Machine can be encoded as a polynomial-sized instance of PATHREACHABILITY making that problem L-complete [83].

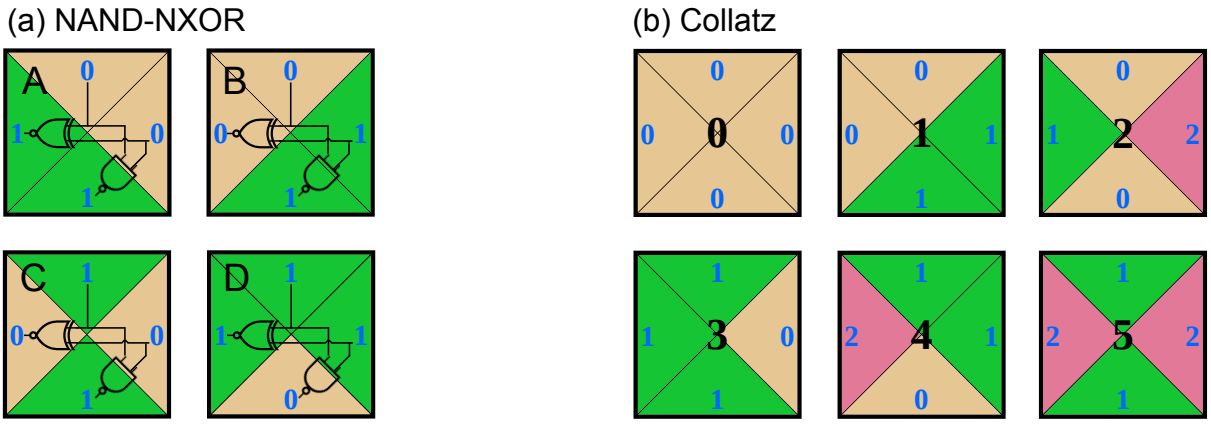


Figure 3.2: Two small tiles sets. (a) NAND-NXOR tile set with 4 tile types. The south side computes the NAND of north and east, and east computes the NXOR of north and east. (b) Collatz tile set with 6 tiles, named for its relationship to the Collatz problem. These are the same tiles that were introduced in Chapter 1, Figure 1.7.

a set of polyominos, painted with information-encoding glues. Starting at special input locations, tiles attach one at a time, asynchronously and in parallel, wherever they match glues on two sides.[53] A typical maze can be thought of as sending a unary ("route finding") signal, whereas our mazes send bits and allow them to meet, interact and be changed.

In this setting, if we allow arbitrary numbers of tiles (or a clever enough walker, or a complex enough asynchronous cellular automaton rule) it is not difficult to see how to simulate arbitrary Boolean circuits. Take a circuit, make it planar by replacing each wire crossing with a crossover gate, then lay the circuit out on a maze-like grid with inputs on the east, and the output on the west. Then simply build a maze with walls tracing out the circuit wiring diagram and painted with arrows (wire directions) and logic gates, and require the output bit(s) to satisfy the circuit logic. The question we ask is: How clever does the maze-solver need to be in this computational setting? More precisely, we ask how many tile types are needed to execute any Boolean circuit in the Maze-Walking TAM?

3.2.1 Main results

Our first main result is for the NAND-NXOR tile set shown in Figure 3.2(a). In the theorem statement, by *simulated* we mean that the function computed by the circuit c is also computed by an instance of the Maze-Walking TAM (see Section 3.4.1).

Theorem 3.1. Any Boolean circuit c is simulated by the 4-tile NAND-NXOR tile set in the Maze-Walking TAM using assemblies containing ≤ 6 tiles per gate and 34 tiles per crossover gate in a planarisation of c.

Our second main result is for the Collatz tile set which has 6 tiles (Figure 3.2(b)) and is so-named because of its ability to embed iterations of the Collatz function (see Chapter 1, Section 1.3.3).

Theorem 3.2. Any Boolean circuit c is simulated by the 6-tile Collatz tile set in the Maze-Walking TAM using assemblies containing ≤ 14 tiles per gate and 33 tiles per crossover gate in a planarisation of c.

We finish this section with a discussion of our two tile sets and some future directions. Section 3.3 sets these results in the context of other theoretical results. and experimental directions. Section 3.4 defines the Maze-Walking TAM. We prove our two main theorems in Sections 3.5 and 3.6. Chapter 1 gives the

[53]The model is equivalent to the abstract Tile Assembly Model [136, 174, 125, 57], with multiple disconnected seed assemblies, and where we have all tile bindings are via two matching attaching to an assembly by matching glues.

background on the Collatz tile set and more specifically its relation to the Collatz problem (Chapter 1, Theorem 1.35).

3.2.2 Discussion: the NAND-NXOR and Collatz tile sets

Theorems 3.1 and 3.2 place focus on the size of assemblies that simulate gates. They omit estimates of the additional tiles (assemblies) required for the circuit wiring diagram, which warrants comment. Our work is partially motivated by a desire to build instances of the Maze-Walking TAM, and in doing so we would highly optimise any implemented circuit wiring diagram. Example circuit implementations, that recognise 3-bit prime numbers, are shown in Figures 3.3(j3) and 3.5(j1), both of which are optimised for short wire length. If we want to have a general wiring procedure for all circuits, and thus not optimised for particular classes of circuits, the overhead incurred will be rather large, typically $O(s^2)$ space for a circuit with s gates [40]. In practice we would not use such overly-bloated constructions.

Then, the NAND-NXOR tile set was found by explicitly trying to find a small tile set: hence its use of a universal gate (NAND) on the south side (output). The NXOR gate (west side) helps with wire routing allowing for even smaller gates than going via NAND-only-based circuit simulation. The Collatz tile set came out of thinking about iterations of the Collatz function in a local digit-by-digit, or tile-by-tile, way. In [156] a cellular automaton-like model is shown to simulate instances of the Collatz function—assemblies of our Collatz tile set show up in iterations (configurations) of that model. The Collatz tile set, along with the non-local rule in [156] (which can be simulated by the addition of two additional tile types (Chapter 1, Figure 1.17 and justification in Remark 1.40), is expressive enough to run Collatz. Here we applied computer search to the Collatz tile set to search for seed structures and assemblies that could be used to compute more generally. We leave as an open question as to what extent such structures, or other computational structures, naturally appear during iterations of the Collatz function.

For running Boolean circuits, if the only metric we cared about was tile set size, the NAND-NXOR tile set wins. However, looking beyond circuits, the Collatz tile set is capable of directly implementing certain arithmetical operations, such as computing powers of 2, powers of 3, and converting from base 3 to base 2 [156] (see Chapter 1, Corollary 1.18). These constructions use much simpler connected seeds than those given in the proof of Theorem 3.2, and lead to more efficient (smaller) assemblies than computing via circuits, for these kinds of arithmetical problems. In this chapter, we used computer search to find that tiles capable of such arithmetical operations are also capable of running circuits, we leave it as future work to discover what other operations they are efficiently capable of.

Theorems 1 and 2 prove that the problem of predicting a tile at distance n from a size n connected seed, is P-hard (and in fact it is P-complete if we assume directed/deterministic growth since a deterministic Turing Machine simulates the entire assembly process in time polynomial in n). It is natural to ask if having maze-like (i.e. disconnected) seeds is necessary for such computational efficiency: we conjecture "yes". That is, for both tile sets, we conjecture that prediction of the tile type that goes at a given position, at distance n from a size n connected seed and assuming directed [149] growth, is in the complexity class NL. In particular this would mean that simulation of arbitrary Boolean circuits in the direct manner shown here is impossible (assuming the widely-believed conjecture that NL $\neq$ P). For the Collatz tile set, and for connected seeds of a certain form, we know that prediction is in NL (and even, NC^1, see Chapter 1, Section 1.4). If one could show that prediction is P-hard, for seeds/inputs that represent natural numbers that occur during iterations of the Collatz function, one could in fact show that the Collatz process embeds rather powerful computational capabilities. Certainly a result of that form would change the perspective on the Collatz conjecture itself.

Our results were developed with the assistance of a simulator: `https://github.com/tcosmo/mawatam`. The reader is invited to experience the results of this chapter through the simulator.

3.2.3 Future work

Experimentally, future work involves implementing instances of the Maze-Walking TAM in the wet-lab, for instance, using a DNA origami as the underlying structure to encode maze seeds [35], building on the systems discussed in Section 3.3.2. One experimentally-relevant criticism of this work could be to ask why we focus on such small tile sets when we know that with DNA it is possible to build systems with hundreds of algorithmic DNA tiles [183]. First, we would say that no algorithmic system of such a high tile complexity, and that runs on the back of a DNA origami, has been engineered to date. Secondly, and of more relevance to this work, is that we are exploring the fundamental boundary and complexity trade-offs between computational power and systems size.

Theoretically, our work leaves open the following questions:

- Can Boolean circuit simulation, or any kind of universal computation, be achieved in the Maze-Walking TAM using tile sets with less than 4 tiles?
- Can interesting behaviour occur in the Maze-Walking TAM with just 1 tile? (At first sight, this question may look odd, however one could imagine encoding a bit by the absence or presence of a tile at a given position in the final assembly, leaving room for expressiveness in the Maze-Walking TAM with 1 tile.)
- Is the Maze-Walking TAM, with ≤ 4 tiles, intrinsically universal [59, 180] for the aTAM?

3.3 Related work: theoretical and experimental

3.3.1 Other routes to finding small universal tile sets

Existing small/simple universal models of computation [182] include the efficiently universal [48, 121] 2-state one-dimensional cellular automaton Rule 110, as well as universal Turing machines with just 22 instructions (5 states & 5 symbols, or 4 states & 6 symbols) [120, 134] or even just with 8 instructions (3 states, 3 symbols, but with the tape input embedded in an infinity repeated pattern) [122].

In the context of the theory of molecular computing, and algorithmic self-assembly in particular, the smallest computationally universal self-assembling tile set to date seems to be a 7-tile system that can be derived from [183].[54] However, that construction leads to large spatial blowup via Rule 110 simulation (Corollary S1.3, SI-A, [183]), $O(s^4 \log^2 s)$ for circuits size s. Another construction uses $O(w^2 d)$ tile types (for a depth d, width w circuit), essentially by hardcoding the routing of the circuit diagram in tile types. Even direct implementation of a small universal Turing machine as a self-assembling tile set, using known methods, although presumably achievable with a few dozen tile types, would require large input encodings [182]. Other methods to obtain a single universal, or intrinsically universal, tile set, or even a single tile, also use indirect and large encoding methods [59, 55, 54, 149].

By allowing for more tile types than our constructions, one could have a maze with glues that explicitly encode gate type (one of 16), as well as glues encoding two bits at a time: that way a single tile attachment event could read two bits and a gate type simultaneously. This idea yields a constant-size tile set with

[54] In Figure S4(b), SI A, [183], gates g and f can be used to simulated Rule 110, and in turn can be simulated by 4 tiles each. These 8 tiles can be further optimised to 7 tiles by sharing one glue type between both half-layers.

perhaps aa few dozen tile types. Although larger than ours, such an approach would have experimental merit. Cantu, Luchsinger, Schweller, and Wylie simulate Boolean circuits with tiles in a covert manner [30].

3.3.2 DNA-based implementations and related models

As future work we plan to give DNA-based designs and implementation for the Maze-Walking TAM. We imagine a 2D information-encoding structure that provides the maze pattern, for example a single flat DNA origami [137], or several DNA origamis tiled together [179, 105, 165, 167], or perhaps even a suitable DNA DX-tile, or single-stranded tile, structure [171, 177, 184, 183]. DNA-based systems for maze-solving have been implemented experimentally: using DNA origami (for the maze) along with hairpin activation [35] or controlled opening of track locations [172] for movement. The phenomenon of DNA condensation was also used for maze exploration [124]. Computation via tile-attachment in the Maze-Walking TAM could be implemented using design principles from algorithmic DNA self-assembly [183, 66], DNA-based molecular walkers that walk on 1D tracks and 2D DNA origami surfaces [187, 147, 141, 123, 74, 164], and other DNA systems that compute on surfaces [23, 25, 151, 34, 36]. Finally, there has been some theoretical and simulation-based analyses of molecular walkers [53, 133, 103, 52] including maze-solving walkers [152], as well as papers that study computation on surfaces [129, 41, 20] using a similar setup to ours but without molecular orientation and using different rule formats. All of these models (and ours) describe sub-classes of asynchronous cellular automata.

3.4 DEFINITIONS

3.4.1 Maze-Walking TAM definition

A *maze* is number of non-intersecting polyominos positioned on $\mathbb{Z}^2$ where each exterior unit-length square-side polyomino edge is labeled with a glue $g = (g', p)$ where $g' \in G$ is from a finite set of *glue types* G, that includes the null glue, and $p \in \{z + 0.5 \mid z \in \mathbb{Z}\}^2$ is a glue position. An *instance* of the Maze-Walking TAM $\mathcal{T} = (T, M)$ has a set of tiles T, where each tile $t \in T$ is a unit-sized square whose four sides labelled with four glue types from G, and a maze M. The process of *self-assembly* proceeds by tiles attaching asynchronously, and one at a time, wherever they match glues on two sides (i.e. two-sided cooperative binding in the abstract Tile Assembly Model [174, 136, 125, 57]). An *assembly* is a maze with tiles attached (thus, assemblies may be connected or disconnected in 2D), and a *terminal assembly* is an assembly such that no tile can be attached.

The tile set T is said to *compute the function* $f : \{0,1\}^n \to \{0,1\}$ in the Maze-Walking TAM if there is a maze M' with n empty (no tile) tile positions $p_0, p_1, \ldots, p_{n-1} \in \mathbb{Z}^2$ and an empty (no glue) output glue position $o \in \{z + 0.5 \mid z \in \mathbb{Z}\}^2$, such adding n *input tiles* at $p_0, p_1, \ldots, p_{n-1}$ to M' gives the maze M_x such that the process of self-assembly yields terminal assemblies that all have the bit $f(x)$ encoded by the glue at position o. (Here, we imagine a many-one encoding function from glue types to bits.)

Maze-Walking TAM systems may be *directed* (one terminal assembly), or *undirected* (several terminal assemblies). In this chapter the systems we study are directed, which is equivalent to saying that, for all sequences of tile additions, at each position $p \in \mathbb{Z}^2$, there is at most one choice for what tile appears at p. Thus, in this chapter, for each $x \in \{0,1\}^n$ there is an associated maze M_x such that $\mathcal{T}_x = (T, M_x)$ has a single terminal assembly is said to compute $f(x)$.

3.4.2 Boolean circuit definition

A Boolean circuit is a directed acyclic graph, where edges are called wires, nodes called *gates* and are labelled. In this chapter, gates have out-degree 1 or 2, except for output gates that have out-degree 0.

Also, a node's label is one of: input (with in-degree 0), output (with in-degree 1, out-degree 0), constant 0 or constant 1 (in-degree 0), fanout gates (in-degree 1, out-degree 2; makes two copies of its input), or is one of the *compute* gates ($\neg$, NOT of in- and out-degree 1, or any of the in-degree 2 out-degree 1 gates that compute functions on bits, e.g. OR, AND, NAND, NXOR,[55] etc.). Also, we define an additional gate called a *crossover gate* (in- and out-degree of 2) swaps its inputs, used to planarise a non-planar circuit (see below). Circuits compute, from the input gates and constant gates to the output gate, by modifying bits according to the functions specified by gate labels.

The *size* of a circuit is its number of gates, and its *depth* is the length of the longest path from any input gate to the output gate. A circuit c_n computes a Boolean (no/yes) function $f : \{0,1\}^n \to 0,1$ on $n \in \mathbb{N}$ of Boolean variables, by its gates computing the bit value at the output in the usual way from the n input bits. A circuit is said to be planar if its graph is planar (can be laid out in the plane without wire crossings).

A *planarisation* of a Boolean circuit c is another Boolean circuit $\hat{c}$ where $\hat{c}$ computes the same function as c, has a planar embedding in $\mathbb{R}^2$, and $\hat{c}$ has exactly the gates of c plus zero or more 2-in 2-out crossover gates (that allow crossing of signals between a pair of wires that would otherwise intersect in the plane). In other words, c is converted to $\hat{c}$ by adding crossover gates so that $\hat{c}$ has a planar embedding. An example is shown in Figure 3.3(j2). A *planar Boolean circuit* c is a Boolean circuit where $\hat{c} = c$, i.e. $\hat{c}$ has zero crossover gates.

3.5 Four tiles: the NAND-NXOR tile set

The NAND-NXOR tile set is depicted in Figure 3.3(a). One of the ideas underlying all of the constructions in this chapter can be understood by the way horizontal wires are implemented with the NAND-NXOR tile set, Figure 3.3(b). A specific $n \times 1$ polyomino seed exposes "1" glues on its south side, which facilitates propagation to the west of any bit coming from the east, following the assembly rules prescribed by the tile set. As described in the proof of Theorem 3.1, the implementation of Boolean circuits using the NAND-NXOR tile set is based on canonical constructions of logic gates exploiting NAND, NOT and NXOR functions as primitive building blocks, Figure 3.3(h1-h5).

Theorem 3.1. Any Boolean circuit c is simulated by the 4-tile NAND-NXOR tile set in the Maze-Walking TAM using assemblies containing ≤ 6 tiles per gate and 34 tiles per crossover gate in a planarisation of c.

Proof. A circuit is simulated by appropriately placing gadgets together to form a maze.

Tiles simulating wires and gates. We will show that the gadgets in Figure 3.3 are building blocks (for a maze) that advertise glues that are designed to force directed growth when given appropriate bit-encoding glue input(s).

Figure 3.3(b,c) details how the NAND-NXOR tile set simulates horizontal and vertical wires. Vertical tile-wires have a parity constraint: in a vertical wire carrying the bit $x \in \{0,1\}$, every second tile correctly advertises x to the south, and every other tile advertises its negation $\sim x$. If the circuit's layout requires a turn from south-to-west, from an *odd length* vertical wire (advertises $\sim x$) then a single horizontal negation gadget (Figure 3.3(h1, right)) is placed at the bottom of the wire to change the signal to x (correct the "error"). With that correction, vertical and horizontal wire segments can be used to send a signal from the origin to any location in the south-west quadrant of $\mathbb{Z}^2$.

[55] In this chapter we use the notation NXOR(x, y) = NOT(XOR(x, y)) (and read "NOT exclusive OR") to denote what is more commonly, but confusingly, written XNOR (read "exclusive NOR").

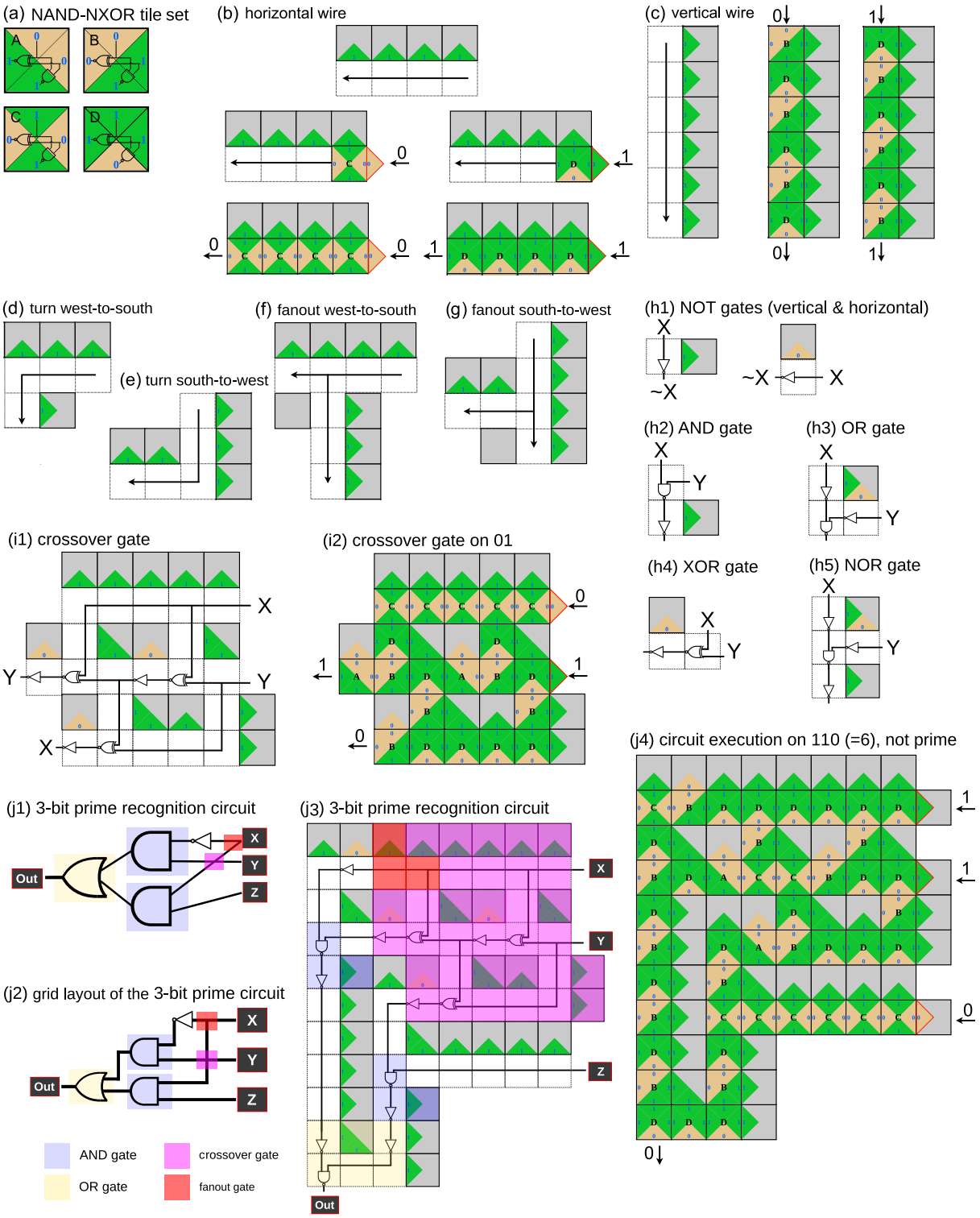


Figure 3.3: Circuit-simulating gadgets for the NAND-NXOR tile set. In all parts of the construction growth proceeds to the west and south (and never north nor east). (a) NAND-NXOR tile set. Seed structures to implement (b) horizontal west-growing and (c) vertical south-growing wires. Examples of communicating of 0 and 1 are shown for each. Vertical wires are of even length; in cases where odd length is required we use a horizontal NOT gates during a turn from south-to-west (see proof of Theorem 3.1). (d) Turn west-to-south, (e) turn south-to-west, (f) fanout west-to-south, and (g) fanout south-to-west. The two isolated unit-size squares in (f,g) are there only to prevent unintended cooperative growth after a fanout. (h1–5) Various logic gates (full set in Figure 3.4). (i1) Crossover gate with an example in (i2) with design based on the 3 XOR gates construction given in [30]. (j1) An example Boolean circuit that decides whether a 3-bit number is prime. (j2) Circuit converted to a grid layout and (j3) implemented using NAND-NXOR tile gadgets. The implementation in (j3) is somewhat optimised for space efficiency. (j4) The terminal assembly (execution) for the circuit example on non-prime input $6_{10} = 110_2$.

Figure 3.3(d–g,h1–h5,i1) shows two turns (south-to-west and west-to-south) and two kinds of fanout-2 gates, as well as a number of compute gates and a crossover gate. In addition NAND, and NXOR, gates are shown in Figure 3.3(a): present inputs x, y at North and East, and read NXOR(x, y) on West and/or NAND(x, y) on South. (For completeness, Figure 3.4 gives direct simulations of all 16 possible gates with 1 or 2 inputs and one output.) No gate is larger than NOR (see Figure 3.3(h5) and Figure 3.4), which uses 6 tiles. The crossover gate is simulated using 34 tiles (intuitively, it uses a well-known idea of implementing crossover with three XOR gates and three fanout gates). This gives the size bounds on tiles per gate and crossovers in the theorem statement.

We claim that each gadget in Figures 3.3(b–g,h1–h5,i1) and Figure 3.4 is directed, meaning that after input glue(s) are given to the gadget, then for each dotted region in the gadget there is exactly one tile type that can be placed. This can be seen by noting that (i) for all gadgets, and all inputs to a gadget, tiles attach using their North and East sides only, and by (ii) the fact that the NAND-NXOR tile set is deterministic on North and East sides.

Laying the circuit out on a grid. For the Boolean circuit c, let $\hat{c}$ be its planarisation as defined in Section 3.4.2; a planarisation always exists—just draw the circuit on the plane replacing each of the $s' \in \mathbb{N}$ wire crossings with a crossover gate (various planarisations may be used to optimise s', or other circuit parameters).

Second, we layer c_n: meaning that we organise gates (including crossover gates) of c_n into consecutive layers with layer 0 containing all input and constant gates, and so that layer i contains gates that take their inputs from the outputs of gates in layers $< i$. The number of layers is equal to the depth d of c_n, with the output gate being the sole gate in layer $d-1$. More precisely, layer i is located at x-coordinate $-i$ (our convention is to draw circuits from right to left).

Third, we increase the height between gates, and width between layers, so that there is enough room to draw all wires so that they are composed of horizontal and vertical segments only (where information flows to the west and to the south, respectively), that meet at right angles (thus wires have south-to-west and west-to-south turns, only). We call the resulting circuit a grid-layout circuit, and an example given in Figure 3.3(j2). Using the gadgets described above, the maze/seed structure traces out the wires and gate locations according to the south-west grid-layout circuit, leaving enough room so that gates and wires to not intersect.

Computation. For any circuit c we have described (at a high level) how to lay out a maze M', in the notation of Section 3.4.1. We next need to encode circuit inputs, as follows. Since input gates are instances of gates, we assume that in M' there are n tile positions that are empty and positioned adjacent to wires (so that their bit values will feed into a layer of gates via horizontal wire gadgets). Let n be the number of inputs to c and let $x = x_0x_1 \cdots x_{n-1} \in \{0,1\}^n$ denote an input to c. To the maze M' we add n more tiles so that the n input glue positions of the maze are of respective types $x_0x_1 \cdots x_{n-1}$, to give an maze M_x that encodes x (the example in Figure 3.3(j4) has 3 encoded input bits).

Assembly proceeds, starting at each of the n input glues in parallel (and at any positions that encode 0/1 constant bits), according to the Maze-Walking TAM definition (Section 3.4.1). Throughout the entire self-assembly process, at each position there is exactly one tile type that can be placed (this is because it is true for individual gadgets as already argued). Also, self-assembly terminates, for the simple reason that no tile can attach outside of the bounding box of the maze M_x. Thus one terminal assembly is eventually produced, that by its definition, encodes an execution of the circuit c with the output bit presented at the glue position that represents the simulated circuit output gate (labeled "out" in the example in Figure 3.3(j3)). □

Example 3.3. Figure 3.3(j1-j4) illustrates the general construction described in Theorem 3.1 in the context of a circuit that recognises prime numbers on 3 bits, i.e. the circuit will output 1 if and only if $xyz \in \{010, 011, 101, 111\}$ which are the binary encodings of numbers $\{2, 3, 5, 7\}$. The circuit implements the formula: (((NOT x) AND y) OR (x AND z)) and uses one crossover as well as one fanout gate, Figure 3.3(j1). To facilitate the final Maze-Walking TAM implementation, the circuit is laid out on a grid using only south-to-west and west-to-south turns, Figure 3.3(j2). Then, the circuit is implemented with tiles, Figure 3.3(j2), using the gadgets of Figure 3.3 and finally, the circuit executes on input $110_2 = 6$ and outputs 0 as 6 is not prime, Figure 3.3(j4). Note two details: (1) The implementation of the crossover gate, Figure 3.3(i1), contains three embedded XOR gadgets and three embedded fanout gadgets—using tiles to implement a known construction to simulate crossover with XORs. (2) The way the OR gate is implemented in Figure 3.3(j3) (yellow overlay) is slightly different than Figure 3.3(h3) as the negation of the east-coming input is performed vertically instead of horizontally; this is an optimisation that exploits the difference in length parity of the two vertical wires coming in to the gate.

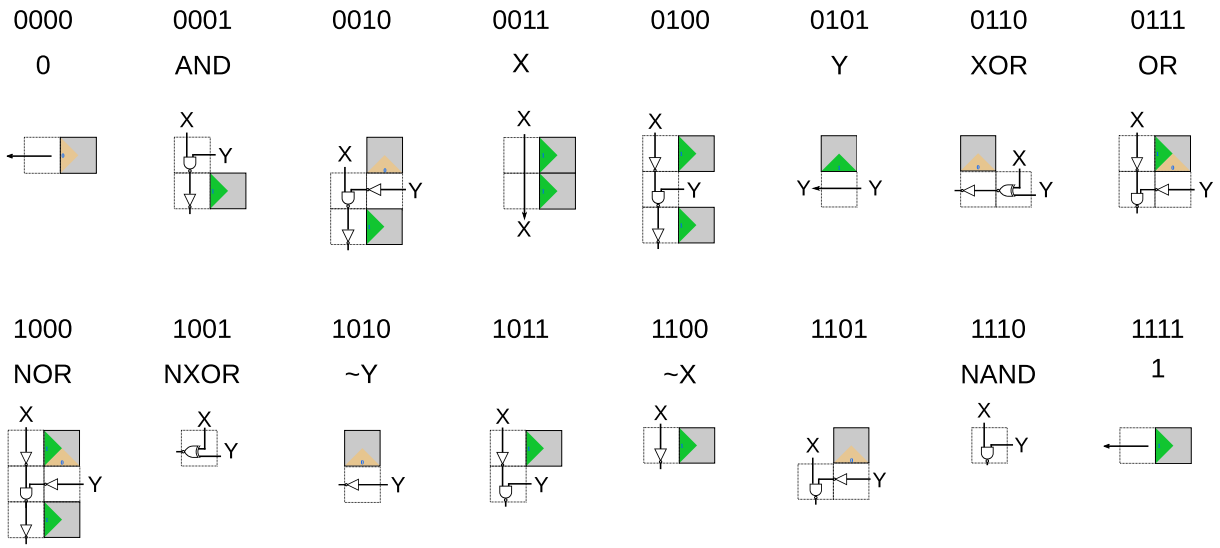


Figure 3.4: Implementation of all 2-input 1-output Boolean gates using gadgets over the NAND-NXOR tile set in the Maze-Walking TAM. The gadgets are ordered with respect to their truth table, which refers to the 4-bit output of the 4 respective inputs 00, 01, 10, 11; i.e. the canonical truth-table definition of a 2-in 1-out gate (we use the same notation for gates with one (NOT, identity) or zero inputs (constants)). For instance, the truth table 1101 encodes gate g such that $g(00) = 1$, $g(01) = 1$, $g(10) = 0$ and $g(11) = 1$. The common English name of the gate is also given when there is one. The constant gadgets (0000 and 1111) are used to simulate constant gates (0/1) and circuit input gates $x_i \in \{0, 1\}$, and require the presence of an additional glue (not shown) to trigger growth, e.g. by being placed next to a wire gadget as shown in Figure 3.3(j4).

3.6 SIX TILES: THE COLLATZ TILE SET

In this section, we illustrate efficient Boolean circuit simulation in the Maze-Walking TAM with the Collatz tile set which consists of of 6 tile types and 3 glues and is shown in Figure 3.2(b).

On the one hand, the NAND-NXOR tile set was explicitly designed to compute, via the placement of a single tile, the universal NAND function. From there it was augmented (with bits on the west sides) that facilitate simulation of circuit wiring, and efficient simulation (few tiles) of non-NAND gates. On the other hand, the Collatz tile set came about from studies on the Collatz problem, see Chapter 1. Specifically, glue patterns in some tiled regions (e.g. rectangles) relate to notoriously hard mathematical problems such as the Collatz conjecture [156] or an open problem of Erdős' [65, 98]: Is it the case that for all $n > 8$ there

is at least one 2 in the ternary representation of 2^n? See Chapters 1 and 2. We noticed that this pattern complexity could be leveraged, with the aid of computer search[56], to build gadgets for computation in the Maze-Walking TAM (Figure 3.5).

Theorem 3.2. Any Boolean circuit c is simulated by the 6-tile Collatz tile set in the Maze-Walking TAM using assemblies containing ≤ 14 tiles per gate and 33 tiles per crossover gate in a planarisation of c.

Proof. **Tiles simulating wires and gates.** We will show that the gadgets in Figure 3.5 can used to build mazes that simulate arbitrary Boolean circuits and that the growth triggered by the placement of input tiles is directed, which in turn implies that the correct bit is output by the simulation of c on some binary input word x.

Figure 3.5(b,c) details how the Collatz tile set simulates horizontal and vertical wires. Horizontal tile-wires have a parity constraint: in a horizontal wire carrying the bit $x \in \{0, 1\}$, every second tile correctly advertises x to the west, and every other tile advertises its negation $\sim x$. To handle this, there are two west-to-south turns, one for turning from even length, and one for turning from odd length, horizontal wires Figure 3.5(d1). Only west-to-south fanout is used in the constructions with this tileset, Figure 3.5(e). This fanout gate comes in two variants whether it is applied at an even or an odd horizontal wire position. If the gadget is applied at an odd wire position, it has the particularity of negating the output west-going signal.

Negating a signal (either to correct a horizontal parity effect, or to simulate a NOT gate) can be achieved in several ways. If the signal ever turns south, this can easily be done thanks to Figure 3.5(d2) which implements both a turn and a negation at the same time. If the signal never turns south, the programmer can use an odd-length horizontal wire which implements a negation. If using an odd-length horizontal wire is not possible given the constraints on circuit layout, the programmer can use the horizontal buffer gadget Figure 3.5(h) which has the effect of copying the incoming signal to the next immediate column to the west which inverts the parity constraint of the horizontal wire and allows it to reproduce the behavior of an odd-length horizontal wire. This method is used in Figure 3.5(j1), for instance on the horizontal wire which connects the input Z to its target AND gate.

Glue labelled polyominos, or seed structures, for south-to-west turns is shown in Figure 3.5(i). Notably, a growth stopper (1×1 polyomino, with four null glues) is used to prevent spurious growth that would happen in the north-west direction otherwise.

A crossover gadget seed structure is given in Figure 3.5(f), it was the smallest found by computer search and it costs 33 tiles. The gate preserves the horizontal alignment of the incoming northern bit: it exits at the south of the gate at the same x-position that it entered. However, the incoming eastern bit is deviated three units to the south.

Seed (polyomino) structures that simulate Boolean (compute) gates are rectangular and were found by computer search using the input convention that signals come from the east and, if there are two of them the inputs should be one vertical block apart[57]. Figure 3.5(g1,g2) gives the seed structure of an OR gate and an AND gate. For completeness, Figure 3.6 gives the implementation of all Boolean gates, the biggest of them is NOR with a cost of 14 tiles. This gives the tiles bounds per gate and crossover in the theorem statement. Remarkably, seed structures for AND, OR, NAND, NOR are very similar in the sense that they differ by at most 2 glues.

[56] Computer search was performed through the Maze-Walking TAM simulator: `https://github.com/tcosmo/mawatam`

[57] Using computer search, we were able to find rectangular seed structures of Boolean gates corresponding to all the input conventions that we experimented with. This leads us to believe that the ability of the Collatz tileset to simulate Boolean gates is not tied to a particular input convention.

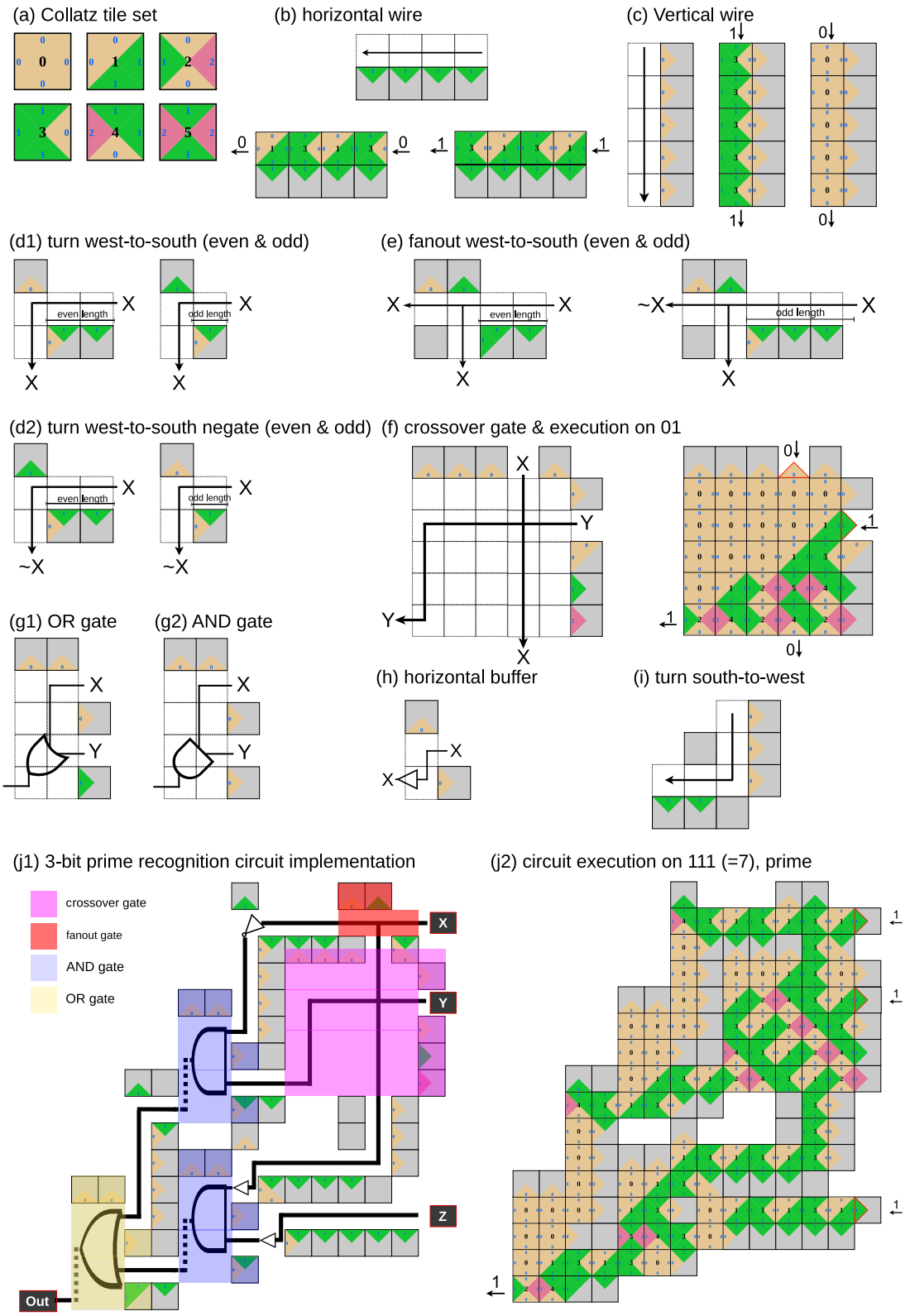


Figure 3.5: Circuit-simulating gadgets for the Collatz tile set. Growth proceeds to the west and south exclusively. (a) the Collatz tile set. Seed structures to implement (b) horizontal west-growing and (c) vertical south-growing wires. Horizontal wires are of even length. When turning to the south the appropriate turn can be used to transmit the signal (d1) or its negation (d2). (e) Fanout gadgets depending on the parity of the incoming horizontal wire, if the length is odd, the gadget also negates the west-going signal. (f) The smallest crossover gate found by computer search. (g) Common Boolean gates, also found by computer search. (h) The buffer gadget is used to change the parity of an horizontal wire. (i) Turn south-to-west. (j1) Collatz-tileset implementation of the 3-bit prime recognition circuit and (j2) execution of the circuit on $7_{10} = 111_2$ which is prime.

We claim that each gadget in Figure 3.5 and Figure 3.6 is directed, meaning that after input glues are supplied to the gadget then for each dotted region in the gadget there is exactly one tile type that can be placed. This can be seen by noting that (i) all gadgets use either North and East sides to attach or South and East sides to attach (South and East attachments are only used for horizontal wires and turn south-to-west gadgets, Figure 3.5(b,i)), (ii) North and East attachments cannot compete with South and East attachments because all signals travel in the south-west direction and South and East constraints are never given directly by the seed but occur after tiles attach, and (iii) the Collatz tile set is deterministic on North and East sides and South and East sides.

Laying the circuit out on a grid. We use the same circuit layout technique given in the the proof of Theorem 3.1.

Computation. Similarly to the proof of Theorem 3.1, throughout the entire assembly process, because of the directedness of all the gadgets that we use, at each position there is exactly one tile type that can be placed. Thus one final assembly is produced, that encodes an execution of the circuit, and in particular outputs the same bit as the n-bit circuit c on any input word $x \in \{0,1\}^n$. □

Example 3.4. The 3-bit prime recognition circuit (Figure 3.3(j1,j2)) was implemented using the Collatz tile set, Figure 3.5(j1,j2).

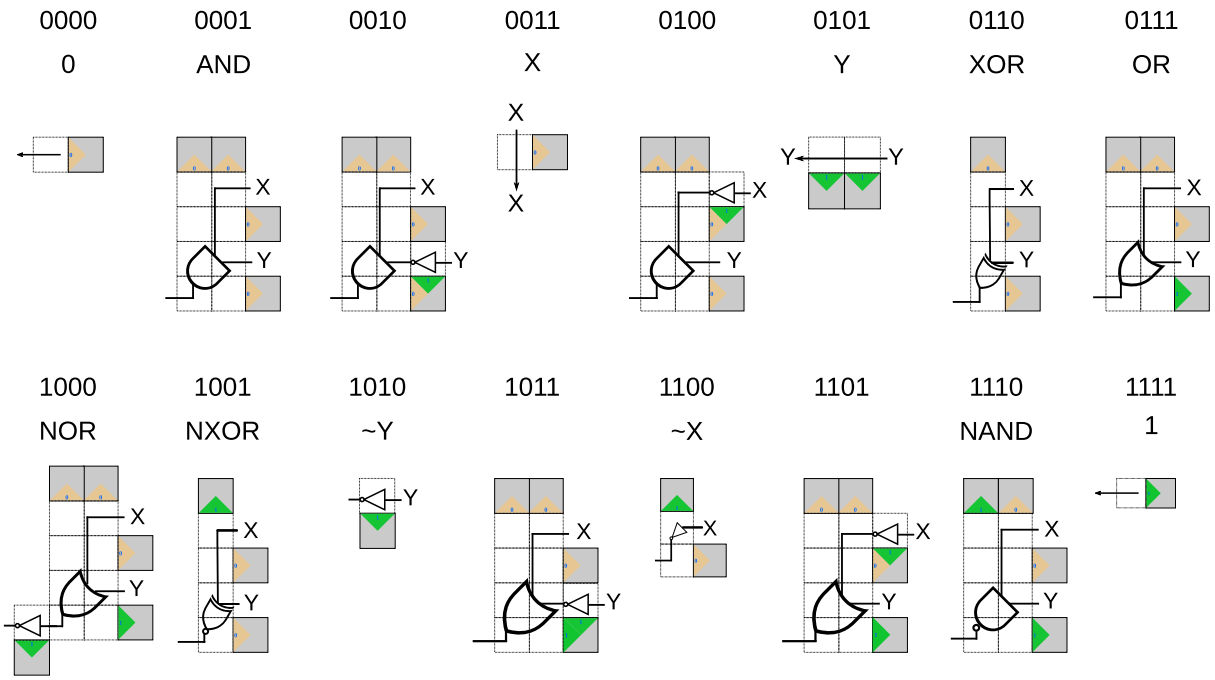


Figure 3.6: Implementation of all 2-input 1-output Boolean gates using gadgets over the Collatz tile set in the Maze-Walking TAM. The gadgets are ordered with respect to their truth table which refers to the 4-bit output of the 4 respective inputs 00, 01, 10, 11; i.e. the canonical truth-table definition of a 2-in 1-out gate (we use the same notation for gates with one (NOT, identity) or zero inputs (constants)). For instance, the truth table 1101 encodes gate g such that $g(00) = 1$, $g(01) = 1$, $g(10) = 0$ and $g(11) = 1$. The common English name of the gate is also given when there is one. The constant gadgets (0000 and 1111) are used to simulate constant gates (0/1) and circuit input gates $x_i \in \{0,1\}$, and require the presence of an additional glue (not shown) to trigger growth, e.g. by being placed next to a wire gadget as shown in Figure 3.5(j2).

Chapter 4

Algorithmic DNA origami: Scaffolded DNA computation

Acknowledgement. I would like to acknowledge and sincerely thank Abeer Eshra who participated to the design of this project and who realised almost all of the circa 100 wet lab experiments presented in this chapter. This work would not have come through without her involvement. Similarly, I would like to thank Damien Woods who accompanied us along all the steps of this project and provided key insights.

4.1 Introduction

In this chapter we introduce a variation of the Maze-Walking TAM (Chapter 3) that we call the Scaffolded DNA Computer. The models are close enough that the proof of Turing-completeness for the Maze-Walking TAM carries over to the Scaffolded DNA Computer, we omit the details. Similarly small tile sets such as the Collatz tile set are enough to simulate arbitrary Boolean circuits in the Scaffolded DNA Computer. We begin by observing that the one dimensional (1D) restriction of the Scaffolded DNA Computer is capable of simulating finite state machines (FSMs) and hence is an interesting model in its own right. We **experimentally implement six programs in the 1D model**. We go on to **implement two 1D programs within a 2D DNA origami**. This work can be considered as a significant first step in the direction of implementing the full 2D model.

At its core, our idea consists in performing computation *within* DNA origami (see below for context on DNA origami) during structure formation: at annealing, staples of the origami are selected algorithmically depending on an input and a function to compute. Our primary design goals are (a) to inherit key properties of DNA origami (such as the thermodynamical favorability of the target structure) and (b) introducing a kinetic pathway for error correction, based on strand displacement[58].

DNA Origami

Introduced in 2006 by Paul W. K. Rothemund [137], DNA origami is arguably the most high-yield and robust technique known for the self-assembly of DNA-based nanostructures. The technique is conceptually

[58] In this work we do not provide data to support the claim that we have a kinetic pathway for error correction, but we hypothesize that the proposed mechanism plays an important role, see Section 4.3.

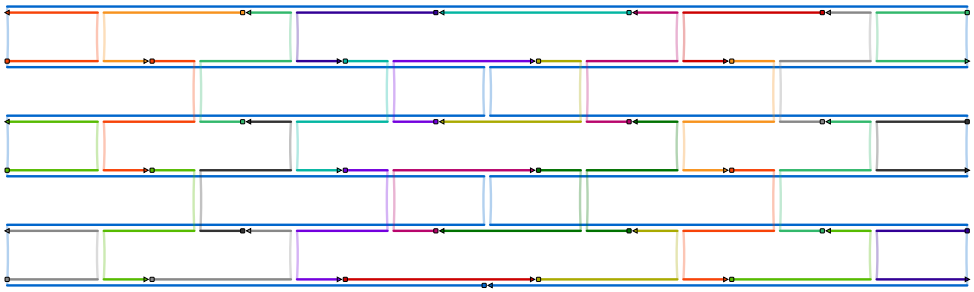


Figure 4.1: Blueprint of a rectangular DNA origami [137] made with the software `scadnano` [58]. DNA origami consists in having a long *scaffold* strand being folded by short *staples* strands. Here, the scaffold is the long, circular, strand in blue and the staples are the short, colored strands. In practice, the scaffold strand is extracted from a virus, M13mp18 [137] (M13 in short), and is put at low concentration (typically between 1nM and 10nM) while staples are chemically synthetised and are put at high concentration (typically 10 times more than the scaffold). To make an origami, mix your scaffold strand with your staples and anneal in a PCR machine – for instance, drop the temperature from 90C to 20C in 2 hours.

simple[59]: a long, low-concentration, strand of DNA, called a *scaffold*, is folded into a specific structure using short, high-concentration, strands, called *staples*. In practice, the scaffold is often circa 7,000 bases long and is extracted from a virus (M13mp18 [137], M13 in short) while the staples are circa 40 bases long, are chemically synthetised and are designed to "grab" intended parts of the scaffold, bring them close together, and thus fold the structure into its target design.

As to the robustness of the technique, one can significantly change many experimental variables, such as concentration, temperature (annealing protocol) or strand purity, without impacting, in most cases, the correct formation of the target structure at high yield (i.e. for 2D shapes, close to 100% of structures should match the target structure when viewed by atomic force microscope (AFM)). The technique has seen many developments throughout the years with, for instance, the assembly of 3D shapes [7, 61], the assembly of *big* shapes [166], *in vivo* usage [190, 104] or use as a fundamental building block for molecular robots [163, 36] or as a seed structure for DNA computing systems [183].

A key property of DNA origami is the inherent thermodynamic favourability of the target structures which is provided by the beautiful design principles of having a low-concentration scaffold that seeds/touches exactly one copy of each high-concentration staple species.

DNA Origami does not compute

However, from the molecular computing point of view, DNA origami (i.e. scaffold plus staple strands) simply **does not compute!**[60] In particular, each pixel in a target shape can be uniquely assigned a short DNA sequence, that specifies that pixel and nothing else—there is no form of function computation nor any computationally complex notion of input-to-output mapping other than uniquely addressed "hardcoding". This type of hardcoding stands in contrast to known molecular computing methods such as algorithmic self-assembly where each pixel in a target structure could have one of several possible associated sequences/strands, the choice of which is determined by (a) a short information-encoding input

[59] We recommend the following animation made by Shawn Douglas which helps to understand DNA origami, visually: `https://www.youtube.com/watch?v=p4C_aFlyhfI`. We also highly recommend this talk by Paul K. Rothemund (DNA origami's inventor): `https://www.youtube.com/watch?v=WhGG__boRxU`.

[60] Many publications in DNA/RNA/protein nanotechnology and engineering use the word "programming" to refer to the *design* capabilities enabled by the use of information-based polymers. However, in this work, when we talk about computing with DNA, we mean something stronger: the ability to encode an *algorithm* in the DNA molecules. We do not wish to debate what is or is not an algorithm, but for here it suffices to say that when claiming something as computing we believe the claim should be backed up by reference or characterisation in terms of a well-defined model of computation, such as finite state machines, Boolean circuits, or other mathematically-defined models of computation that capture what it means to compute a function or some such formal object (whether already existing in the literature or perhaps completely novel).

(e.g. a short bit sequence) and (b) a sequence of information-processing steps (e.g. via Boolean logic gates) leading to the assembly of that pixel [183].[61]

Although published DNA origami designs do not perform computation, origami structures have been used extensively as a structural support tool in various studies on DNA computation, examples include:

1. DNA origami as an information-encoding structural-support *seed* for algorithmic self-assembly systems; including systems where the origami seed encodes inputs with 2 to 6 bits [183, 66, 145, 10].
2. DNA origami being used as a (flat) substrate upon which Boolean DNA circuits are assembled [23, 25, 151, 34, 36], and (somewhat relatedly) upon which DNA walkers trek [187, 147, 141, 123, 74, 164].

Algorithmic DNA origami

We introduce the Scaffolded DNA Computer which performs computation during the assembly of a scaffolded structure, such as a DNA origami. That way, we inherit the beautiful properties of DNA origami and additionally make it compute: we are introducing *Algorithmic DNA Origami*.

The key ideas of our design is (a) to add information-bearing extensions, called *toeholds*, on the ends of the structures' staples (b) have several staples with different toeholds compete for the same scaffold domain. Thermodynamically, the system should eventually select the staples that have matching toeholds to already-present neighbouring staples; some *input* staples are initially present with a unique toehold in order to drive the system towards a unique equilibrium.

Our proposal stands in contrast to prior work (see Section 4.3.2) in that computation occurs in the process of **forming** the origami.

4.1.1 Summary of contributions and chapter structure

We give four main contributions:

1. Theory and design: The Scaffolded DNA Computer, a new theoretical model of computation, is defined in Section 4.2, and some basic properties are stated including its relationship with thermodynamics and relationship with existing models of computation.
2. Our strand-level and domain-level design is described in Section 4.3. DNA sequence design principles (in the presence of non-designed scaffold) are given in Figure 4.6. We compare our system to other DNA-based computation methods in Section 4.3.2.
3. **1D results.** Implementation of 6 Scaffolded DNA Computer programs in 1D (Section 4.4): BIT-COPY, PARITY, ADDITION, MULTIPLYBY3, 3-state NFA, DIVIDEBY2.

   For example the ADDITION program takes two 4-bit numbers as input and outputs their sum in binary. We run these programs on several inputs each, leading to a total of 77 reported experiments (excluding controls). We estimate an overall yield of 91 % for deterministic programs in the 1D Scaffolded DNA Computer, using a bulk fluorescence measurement technique.
4. **2D results.** Implementation of two 2D Scaffolded DNA Computer programs (Section 4.5): BIT-COPY and ADDITION, computations that are run up to 14 helices and output is read using single-molecule AFM readout. The 2D ADDITION program takes two 7-bit numbers as input and outputs their sum in binary.

[61] For brevity, we omit a more formal and nuanced analysis of what is and is not computing in this context.

By 1D/2D we mean that the underlying scaffold that we use is either 1D or 2D. In 1D we use a linear, synthetic 120-base scaffold while in 2D we use a full DNA origami rectangular M13 scaffold. Our computations operate horizontally in 1D and vertically in 2D. Additional supporting material, background context and discussion, for these contributions are given in Appendix E, specifically:

- Appendix E.1 provides a few future work theory questions.
- Appendix E.2 contains design, and experimental controls, for a novel sequence independent reporting mechanism developed for our 1D Scaffolded DNA Computer.
- Appendix E.3 contains DNA sequences, and a few additional results in Appendix E.4.

Finally, although we do not report the designs and data here, the systems presented here build upon two simpler 1D prototypes for BIT-COPY implemented over distance 1, and then over distance 3.

4.2 MODEL OF COMPUTATION: SCAFFOLDED DNA COMPUTER

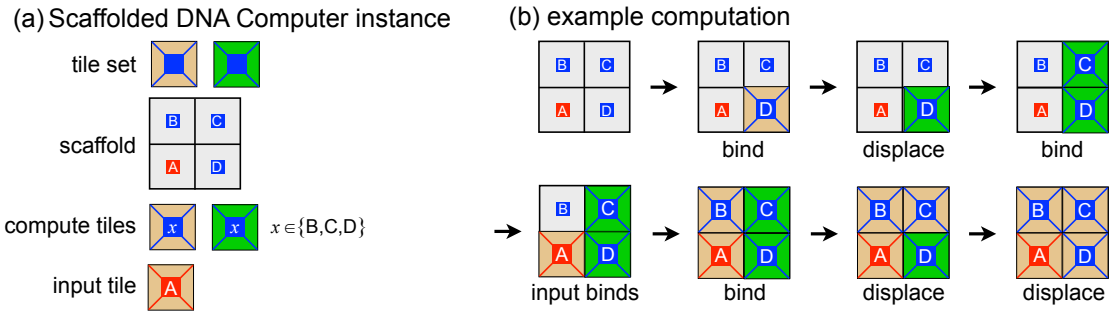


Figure 4.2: A simple 2D bit-copying instance of a Scaffolded DNA Computer. (a) The programmer specifies a tile set, a scaffold design (here, a small 2×2 grid), an input position on the scaffold (red; only one tile is permitted to bind at the input position), and which (compute) tiles may go at other positions. The tiles have only two colours (beige is 0, green is 1) on their binding sides. (b) An example execution. Tiles bind, unbind (if not adjacent to other matching-colour tiles), and can be displaced by other tiles with an equal number, or more, matching colours. Eventually no more moves happen. In this example, the final configuration is unique: all executions lead to the same result, i.e. the input colours are copied throughout the grid.

4.2.1 Model of computation

Our proposed model generalises the Maze-Walking Tile Assembly Model [49] described in Chapter 3 (and shares some of its computational abilities).

Scaffolded DNA Computer definition A *Scaffolded DNA Computer* (Figure 4.2) consists of a finite set of square non-rotatable[62] *tiles* $T = \{t_1, t_2, \ldots, t_k\}$, with a *colour* on each side, as well as a finite connected *scaffold* of locations where tiles can be placed (formally, the scaffold is a connected subset of the 2D grid indexed by pairs of integers $(x, y) \in \mathbb{Z}^2$). Each position $p \in \mathbb{Z}^2$ has an associated set of *p-permitted tiles* $T_p \subset T$ that may be placed at that position p. One or more scaffold locations positions

[62]Although we forbid rotation of tiles, it is of course a physically reasonable property, however (a) any tiles set that allows for rotation can be simulated by a slightly larger tile set (more colours) that does not. Unlike the abstract Tile Assembly Model [174], here there is no notion of differing glue/colour strength nor temperature–although we do consider the number of matching colour (bonds) as an important property.

are called *input positions* $q \in \mathbb{Z}^2$, and in this work[63] they are restricted to allow just one tile type $T_q \in T$. Instead of a typical colour (which can form a bond), a tile side may be the *null* colour, (grey in the figures) meaning it does not bond with any other tile side.

The system begins with no tiles on the scaffold, and then one by one tiles may attach or detach according to the following rule: **the tile that attaches at position $p \in \mathbb{Z}^2$ is chosen nondeterministically from those p-permitted tiles that preserve or increase the number of matching colours**.

Final configuration(s). At each step, the tiles positioned on the scaffold is called a *configuration*. When no more attachments can happen, we have a final configuration; the computation has finished. The final configuration may be unique or not (a good programmer typically endevours to program the system to ensure uniqueness).

Thermodynamically optimal final configurations. Among final configurations, some of them maximise the number of matching colours on adjacent tile sides. We call these configurations *thermodynamically optimal* final configurations. Moreover if there is only one thermodynamically optimal final configuration, we say it is *unique*. In this work, we only design systems with unique thermodynamically optimal final configuration. Examples 4.1, 4.3, 4.4, have a unique final configuration, but one could craft an example that does not.

4.2.2 Examples and computational power of the Scaffolded DNA Computer

Like other 2D tile based models [57, 125, 180], and thermodynamics based models [60, 33] we expect the Scaffolded DNA Computer to have strong computational abilities; indeed it is quite close to, and inspired by, the Maze-Walking Tile Assembly Model, see Chapter 3, which is known to be capable of simulating Boolean circuits (even with as few as four tile types!), a result that carries over to the Scaffolded DNA Computer.

Here, we mainly focus on experimental implementation of the Scaffolded DNA Computer, leaving a full theoretical analysis of the model's computational capabilities to future work. However, we need some basic theoretical results in order to frame our results in terms of known computational models, and to give a systematic way to describe the computations that we implement.

2-sided Scaffolded DNA Computer. In this work we will implement a restricted version of the Scaffolded DNA Computer that we call "2-sided": where the tiles have at most 2 of their sides coloured (the other 2 sides are set to a "null colour" that does not form a bond). We implement this 2-sided restriction of the model, both on 1D (Section 4.4) and 2D scaffolds (Section 4.5).

Example 4.1 (BIT-COPY)**.**

Figure 4.3 gives a minimal 1D example of computation with the 2-sided Scaffolded DNA Computer: copying a bit. Bit 0 is represented by color beige and bit 1 by color green. Putting a beige (resp. green) tile at position A triggers the assembly of an entirely beige (resp. green) chain: we say that bit 0 (resp. 1) has been copied (over 4 scaffold positions on Figure 4.3). Although it is a very primitive type of computation this example depicts the main idea that we exploit in this work: having several tiles (which will become *DNA strands*) **compete** for the same scaffold position and be selected according to their

[63] It is possible to add additional nondeterminism to the model, by permitting one of several tiles at an input position–this would enable a form of randomised algorithm where the randomness comes from the choice of input at some position. We leave exploration of that aspect of the model to future work.

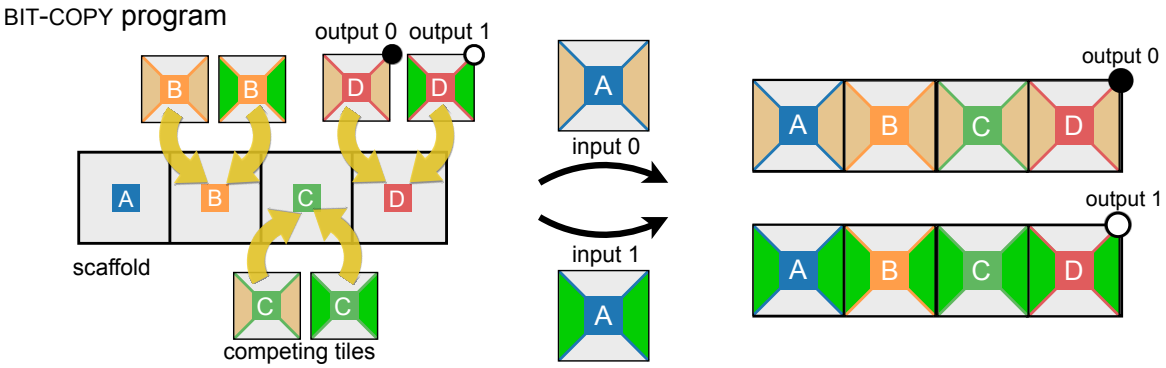


Figure 4.3: BIT-COPY example: Copying a bit with a 1D instance of the Scaffolded DNA Computer (2-sided). The user provides a single input tile, inputting 0 or 1, that attaches to position A. At the other scaffold positions, two tiles compete: one representing bit 0 (beige) and the other bit 1 (green). Putting a beige (resp. green) tile at position A triggers the assembly of an entirely beige (resp. green) chain: bit 0 (resp. 1) has been copied over 4 positions. Tiles competing for position D have an output *decoration* indicating that the output is 0 or 1.

matching colors (which will become matching *toeholds*). We implemented bit copy both in 1D and 2D, cf. Section 4.4.2 and Section 4.5.2.

Example 4.2 (4-bit parity-checker). See Figure 4.10(b–d).

Example 4.3 (4-bit addition). See Figure 4.11(b–d).

Example 4.4 (hardcoded square-grid DNA origami, no computation). An origami is an example of our model where there are as many tile types as scaffold positions (ignoring certain details about DNA origami, such as lattice arrangement of staples, and origami scaffold routing).

Finite state machines. Finite state machines are a well-studied model of computation which allow us to represent familiar computations such as addition, multiplication by a constant, PARITY[64] and many others [118]. A deterministic finite state machine starts in one of a finite set of states, then reads the first input symbol, and moves to a new state (possibly outputting an output symbol). It then reads a new symbol, and so on, until all input symbols are read. The output is either the name of the final state (for example: 'yes' or 'no'), or else an output word produced by the machine. For brevity, we do not give a fully formal set of definitions, but the notions used can be easily formalised [118]. We show that the 2-sided Scaffolded DNA Computer is available to *simulate* any finite state machine, Theorem 4.5. The core idea of the construction is depicted in Figure 4.4: at each scaffold position (apart from the first), a number of tiles compete and that number is equal to the number of states in the machine. At each step a tile is selected, with that selection ultimately depending on the tile at the first position which corresponds to the initial state of the machine (and the first symbol of the input word). Thus the input tile directs/determines the (deterministic) computation. See Appendix E.1.2 and E.1.1 for additional comments and open questions.

Theorem 4.5. Any finite state machines M, on length-$n \in \mathbb{N}$ input, is simulated by a 2-sided Scaffolded DNA Computer S (where *simulate* means that: (a) S's scaffold is a line of n tile positions, with the input position being the leftmost; (b) the final configuration is unique and its rightmost tile-side colour contains the final state of M).

[64] PARITY consists in computing the parity of the number of 1s in a binary string.

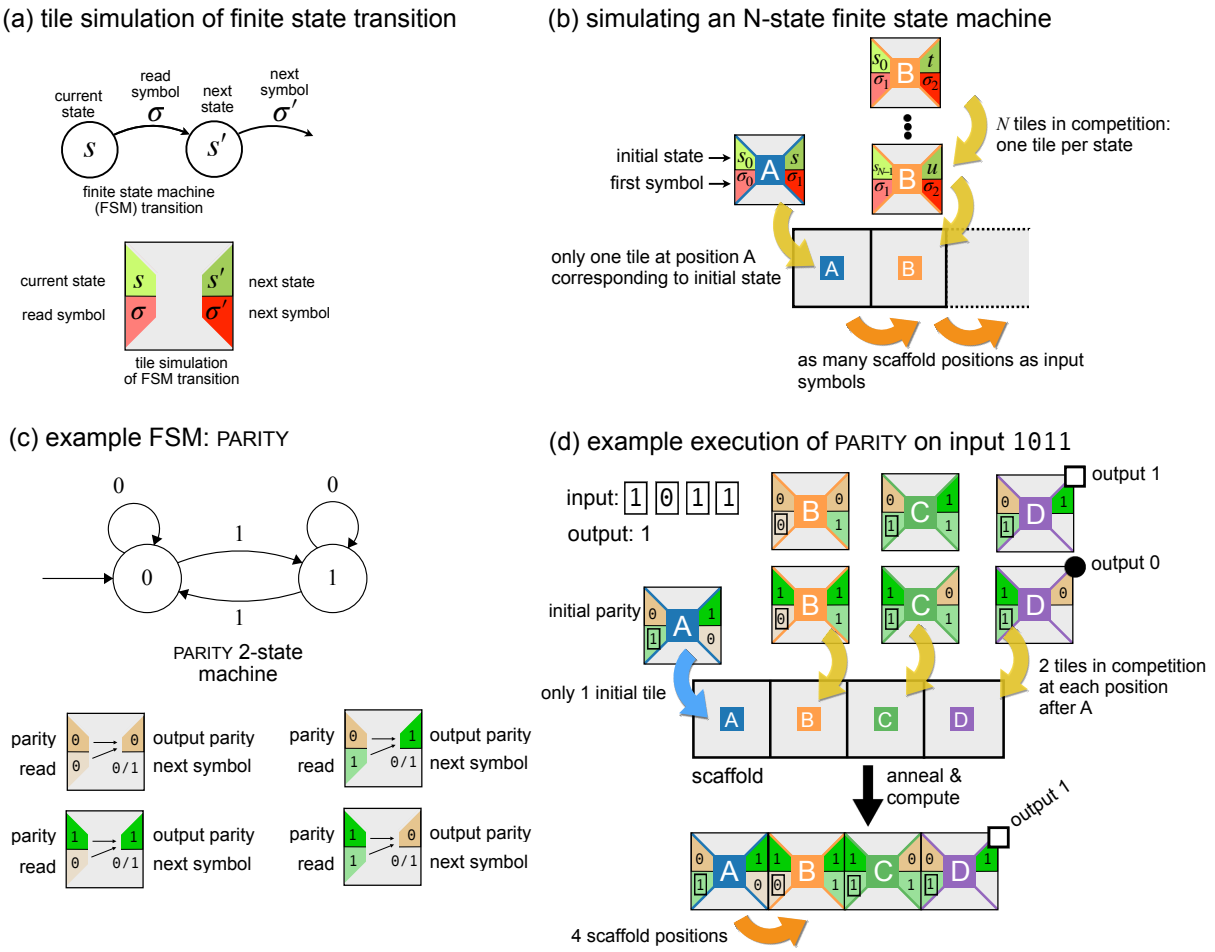


Figure 4.4: Simulating finite state machines with the Scaffolded DNA Computer (2-sided). (a) transitions of the finite state machine are transformed into tiles, colors encode both the input symbols and state transition information (b) assuming the machine has N states then N tile will compete at each scaffold position apart from position A where only one tile can be place – corresponding to the initial state of the machine and first input symbol (c) converting the PARITY 2-state machines into tiles. The machine computes the parity of the number of 1s in a binary string. (d) executing PARITY on input `1011` using the Scaffolded DNA Computer. We expect the output to be 1 (the output is the state in which the machine is after scanning input `1011`) because there is an odd number of 1s in `1011`. The input `1011` is encoded as the bottom-left bit of each tile. Two tiles compete at each scaffold position (they correspond to both parity state 0 and 1) apart from position A where the initial parity is 0.

Proof (sketch). Figure 4.4(a) and (b) give the construction: a tile encodes current state and current read symbol on its left side and next state and next read symbol on its right side. At each scaffold position apart from the first one, 1 tile per machine state is in competition: the winner is the only tile that matches the color on the right side of the tile at the previous position. At the first position the tile (or tiles in the non-deterministic case), are defined by the initial state of the machine and the first read symbol. □

4.3 Fundamental strand-level mechanism

In this work, we present a DNA-based implementation of the 2-sided restriction of the Scaffolded DNA Computer. By 2-sided we mean that the square tiles we implement use only 2 of their 4 sides. Our implementation is based on a simple mechanism, which can be applied both to 1D scaffolds (Figure 4.5a, and see Section 4.4) and 2D scaffolds (Figure 4.5(b), and see Section 4.5). The implementation is based on the following principles:

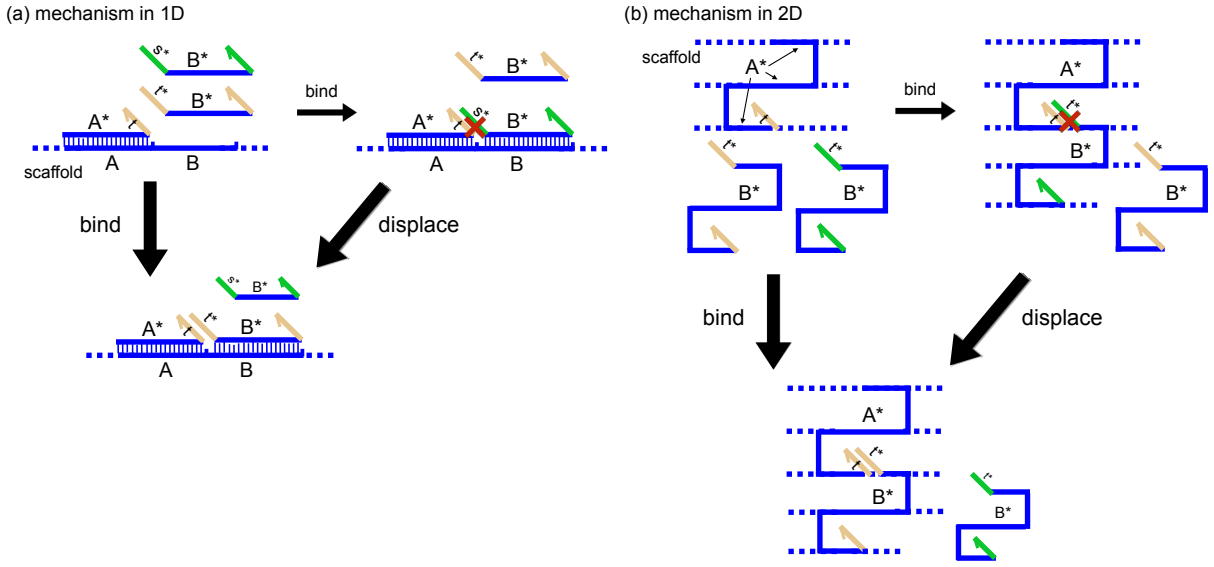


Figure 4.5: Fundamental strand-level mechanism for implementing the 2-sided Scaffolded DNA Computer on 1D scaffolds and 2D scaffolds. Diagram shows two possible pathways of our mechanism in 1D (a) and 2D (b). Two strands (or more in practice) compete for scaffold position B. The thermodynamically favored state is when toehold t is bound to t*, which eventually decides the winner of the competition. In the case of an incorrect t-s* attachement, we rely on strand displacement to correct this error and bring the system to the favored t-t* bond. The mechanism is essentially the same in 1D and 2D apart from the fact that, in 2D, the displacement has to occur through several parallel scaffold helices which is intuitively harder than the 1D case.

1. The scaffold corresponds to a long DNA strand divided into a number of binding domains. Each binding domain is associated to a position in the 2D grid.

2. Tiles are implemented by short single-stranded DNA strands consisting of:

   - A domain that exactly complements one of the binding domains of the scaffold.
   - One toehold at each end of the strand (or only at one end) of which sequence corresponds to a colour of the tile set.

3. **Competition.** The set of permitted tiles at a scaffold position correspond to strands in a 1-to-1 mapping, with the same binding domain but different toeholds that **compete** for the scaffold position. If a strand with incorrect toehold (i.e. tile colour) binds at a position, there is always a **strand displacement** pathway to discard the incorrect strand, see Figure 4.5. Displacement is intuitively harder in 2D than in 1D as, in 2D it has to occur through several parallel scaffold helices.

4.3.1 Toehold design

We designed 8 information-encoding toeholds, each with 12 bases,[65] that allow us to simulate eight different colors in the Scaffolded DNA Computer model, Figure 4.6. We refer to these toeholds by their symbolic names `000`, `001` …`111`. The toeholds were designed with relatively few criteria in mind, Figure 4.6(b). Broadly speaking the only constraint that we designed for was orthogonality: toehold t and toehold s^*

[65] Here we say 8 *information-encoding* toeholds, to delineate from the DNA sequence level of abstraction where we actually designed 16 toeholds: one set of 8 toeholds for even scaffold positions and the other 8 for odd scaffold positions. This was done in order to avoid unintended hairpin formation: if we naively had used only 8 toeholds, a strand having the same logical (symbolic) toehold on both ends would result in one end being the reverse complement of the other, giving an unwanted hairpin. See Appendix E.3 for details.

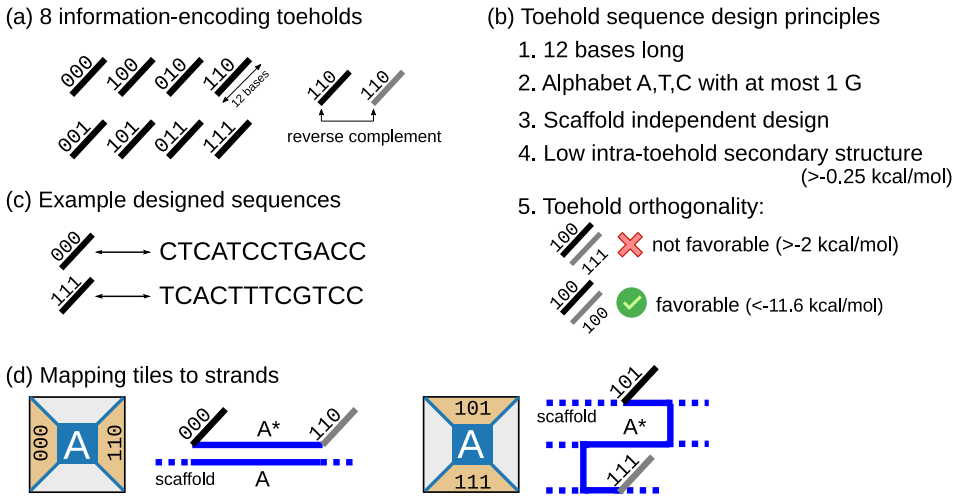


Figure 4.6: (a) symbolic representation of the eight 12-base toeholds that were designed in this work (in fact, two of each toehold were designed to prevent intra-strand binding see Appendix E.3). Light gray represents the reverse Watson-Crick complement of a toehold. b) Main principles for toeholds sequence design. Energies were computed using `NUPACK4` pfunc [69] at 53°C. (c) The designed sequences of toeholds `000` and `111`. (d) Mapping between a 2-sided tile and its DNA-based strand implementation.

should interact favorably if and only if $t = s$. All interactions were measured at 53 °C, [Na] = 1 M and [Mg++] = 0 using `NUPACK4` pfunc [69]. We designed the sequences using Nuad[66].

Note that we intentionally (as a design principle) designed the toeholds independently of the scaffold sequences that we used them with (synthetic 120 base linear scaffold in 1D, whose sequence comes from two pieces of M13 sequence, and a standard 7,249 base M13 scaffold in 2D). This choice was motivated as follows:

1. Designing against spurious interactions with long strands (7,249 bases for M13 scaffold) is a challenging problem. One issue is that it tends to over-constrain the sequence design task because of the huge number of possible interactions.

2. With high formation temperature (> 50 °C is typical for DNA origami), even long toeholds of > 12 bases would be likely not to have too many interactions with the scaffold.

3. In the 2D origami case, the M13 scaffold is mainly entirely bound by perfect staples meaning that it is very unlikely that toeholds have the unintended behaviour of binding the scaffold.

In the case of our 1D system where we use a 120 bases synthetic scaffold we conducted significant post-design analysis on ten sets of designed toeholds before choosing the one that performed best over a set of criteria that included examining some of the many possible unintended interactions. We re-used the same toeholds with no additional analysis in our 2D origami system (see Section 4.5). All the sequences that we designed and used are reported in Appendix E.3.

4.3.2 Comparison with other DNA-based computation methods

From its inception [174] algorithmic self-assembly has been a driving force in the field of DNA computing. Experimental algorithmic self-assembly systems have seen significant success [183, 66, 145, 10], with recent work demonstrating 21 6-bit Boolean circuits on well over 100 distinct computations [183]. There remains much to do in this exciting research direction.

[66] `https://github.com/UC-Davis-molecular-computing/nuad`, project led by David Doty, UC Davis.

Our proposed Scaffolded DNA Computer implementation takes a rather different approach, and by doing so avoids some (not all) of the challenges for algorithmic tile systems. Primarily, spurious (unwanted) nucleation of structures is avoided in our thermodynamically favoured design (but is a major challenge and open research theme for tile-based systems). Just like regular origami, by depending on a low concentration scaffold to tether a completed computation together, it should be impossible for large unwanted structures to spuriously form (assuming we hold above the stable toehold binding temperature), even for computing systems spanning a full M13 scaffold.

For our implementation, the kinds of algorithmic errors we could expect could be different, and perhaps in principle occur at a lower rate than for algorithmic tile self-assembly. As Figure 4.5 shows, for each computing element the correct logical operation (computing step) is favoured, even when an algorithmically-erroneous (mismatching toehold) binding step first occurs—this *displacement of errors* is one of the key design features that we feel is exciting about the approach. Of course, in a large system we might have to anneal very slowly, or wait a long time, but no more than a few hours or a day sounds feasible, this remains to be explored.

DNA strand displacement circuits are another popular form of DNA computing [189, 188, 71, 185, 28]. Many implementations do not have a thermodynamically favoured target (output) configuration, and suffer from 'leak' (error) pathways. Recent work on enforcing kinetic [38, 150] and thermodynamic [168, 169] barriers to leak pathways hold great promise. Here, our proposal is to aim for a purely thermodynamically favoured output.

It is important to make a clear distinction between our proposed Scaffolded DNA Computer implementation, and the decade-long history of DNA computation, and DNA walkers, that use DNA origami as a support. As with previously cited systems, those systems have challenges of self-assembly problems, some needing complex preparation in order to prepare the system in a high-energy state and/or prevent pre-triggering, for example [163]. Our proposed implementation would not have such concerns, it is one-pot and the correct computation is favoured by design, at least theoretical tile-level and strand-level abstractions, and with good design principles also at the DNA implementation level.

Kinetic traps. Of course, like in all programmed nucleic acid systems (whether thermodynamically favoured or not) our implementations of the Scaffolded DNA Computer could have unintended kinetic traps, due to DNA sequence impurities, or inadequate DNA sequence design, or our incomplete biophysical, energetic and mechanical understanding of DNA in the complex setting of DNA origami (helices, crossovers, adversarial scaffold sequence). These present interesting and novel challenges for this approach to DNA computing.

4.4 1D SCAFFOLDED DNA COMPUTER

4.4.1 Experimental setup

The DNA-based mechanism (Figure 4.5) with which we implement the Scaffolded DNA Computer is, to the best of our knowledge, novel and untested. Hence we decided to first experiment with it in a simple 1D environment using a 120-base synthetic scaffold.[67] We use a 120-base sequence extracted from M13 p7249, see Appendix E.3. We divide the scaffold in five 24-base domains A,B,C,D,E. Our 1D Scaffolded DNA Computer operates on domains A,B,C,D while domain E is used for reporting purposes only. We attach our 8 toeholds to both ends of domains A,B,C,D in all possible ways. In total, our system (including

[67] In fact, we also prototyped the model on two earlier smaller systems, not reported here.

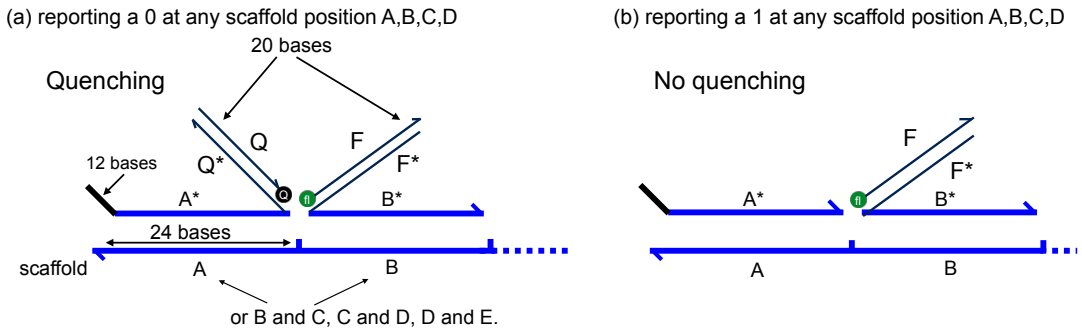


Figure 4.7: Sequence independent reporting mechanism. For reporting at position A, the strand at position B is equipped with domain F* complementary to the ATTO-590-labelled strand F present in solution. In order to report a 0, the strand at position A is equipped with domain Q* complementary to a quencher-labelled strand Q present in solution. However, to report a 1 *no reporting domain* appears on the strand at position A. That way, reporting 0 triggers quenching (a) and reporting 1 does not (b). We use the exact same principle to report at positions B, C and D with the same domains Q* and F* each time: our reporting mechanism is independent of domains A,B,C,D. In a computation, the quencher and fluorophore are always in solution and two or more strands compete at the reporting position, some with domain Q* and some without, that way we read the output of our computations.

reporting) consists of 254 strands, see Appendix E.3. Strands were ordered from IDT, unpurified apart from the 120-base scaffold (PAGE) and quencher/fluorophore labelled strands (HPLC).

Sequence independent reporting mechanism. We use fluorescent labelling to report the results of our computations. We use the fluorophore/quencher pair ATTO-590/Iowa Black FQ. We report a binary signal with the convention that quenching is 0 and absence of quenching is 1. We wish to report outputs at all sites A,B,C,D of our system in a way that is independent of these domains' sequences. We use the mechanism depicted in Figure 4.7 in order to do so. The mechanism uses the fixed domains Q* and F* which are added to adjacent domains on the scaffold in order to report, see Appendix E.2 for sequence design principles of domains Q and F. Strands compete at the reporting position with and without domain Q* depending on their respective output. Consequently, we observe quenching or lack of quenching as an output of our programs. Note that with this system we can report at most one output at a time and that for instance, reporting at site B prevents us to use following sites C and D in our computation. Our strategy is to perform one experiment per output bit: for example, to report a full 4-bit addition we run 4 experiments, one for each output bit (Section 4.4.4). Finally we manage to slightly modify this binary reporting mechanism in order to get a third output value thanks to what we call *late quenching*, Section 4.4.7.

Baseline 0 and baseline 1. We tested our reporting mechanism in 15 different control configurations for reporting bit 0, and 15 others for reporting bit 1, varying the reporting position (A,B,C,D), the strands present at non-reporting positions, and the total amount of DNA present in solution, see Appendix E.2. By control we mean that there was at most one strand present per scaffold position (no competition, no computation) in these experiments. We call *baseline 0* and *baseline 1* the average values (after normalisation, see below) that we got from these bit 0 and bit 1 controls.

Data normalisation. The only normalisation that we perform is dividing each curve by their first data point (the measured fluorescence value after 5min at 80 °C).

Experimental protocol. To conduct a 1D Scaffolded DNA Computer experiment, we mix together the strands corresponding to all of the tiles of the computation (at 1 μM per strand type) together with our

synthetic scaffold (at 100 nM) and fluorescent-reporting related strands (example mix in Appendix E.3)[68]. Then we anneal the mix while reporting fluorescence in a qPCR machine (QuantStudio 5) always using the same ∼3h protocol: Hold at 80 °C for 5 minutes, drop from 80 °C to 70 °C gradually within 20 minutes by -0.5 °C/min, drop from 69 °C to 20 °C over the course of 155 minutes by -0.3 °C/min. Each mix is divided into two qPCR wells and we report the averaged normalised signal from these two wells.

Programs. As presented in Figure 4.4, we intend to simulate Finite State Machines with our 1D Scaffolded DNA Computer. With 8 logical toeholds (16 toehold sequences in practice) we are limited to machines where states and symbols can be encoded on three bits (since $2^3 = 8$) altogether[69]. For instance, we can have two bits of symbols (i.e. ≤4 symbols) and one bit of state (i.e. ≤2 states) or vice-versa. The more states in a machine, the more strands compete for the same scaffold position, Figure 4.4. We report up to four strands competing for the same scaffold position in our results. We implement 6 Scaffolded DNA Computer programs: BIT-COPY, PARITY, ADDITION, MULTIPLYBY3, 3-state NFA, DIVIDEBY2 that we experimentally ran on a total of 35 inputs. Full details are given in Sections 4.4.2 to 4.4.7.

4-bit inputs/outputs. We use the 4 scaffold positions A, B, C, D to encode 4-bit input-words for our programs. Longer scaffold would allow for longer input bitstrings[70]. For most programs, such as our 4-bit adder, Section 4.4.4, we also report 4-bit outputs. This requires us to run 4 experiments: one to read each output bit. This brings the number of experiments that we report to a total of 77 (excluding controls).

Yield estimation. The yield of a Scaffolded DNA Computer program can be defined as the proportion of annealed polymers reporting the correct output. In order to estimate this yield we compute the average distance between the final values of our curves and their corresponding baseline (baseline 0 when the output is supposed to be 0 and baseline 1 when it is supposed to be 1). This estimation formula slightly differs for 3-STATE NFA, see Section 4.4.6. We report a global 91 % yield over our deterministic programs (89 % if we include the more challenging 3-STATE NFA). This 91 % global estimate is almost certainly a pessimistic estimate on the actual yield since we (harshly) compare our signals to a 'no computation' baseline as we currently lack a way to force our systems to full completion [130]. See Section 4.6 for further discussion.

4.4.2 Program 1: BIT-COPY

BIT-COPY **design.** BIT-COPY consists in propagating an input bit 0 or 1 along a chain of tiles. Figure 4.8(b) presents the program at tile-abstraction level: two tiles compete for each scaffold position B,C,D and they are selected according to which tile 0 or 1 is placed at position A. The corresponding strand diagram is given in Figure 4.8(d).

BIT-COPY **results.** Figure 4.8(c) presents the output of the bit-copying at position D (i.e. maximal distance from the input). We observe than when a 0 (resp. 1) is copied the output signal is close to the baseline 0 (resp. 1). Figure 4.9 reports the bit-copy outputs at additional intermediary positions A,B and C (no competition at position A). We observe (i) a sudden drop of the copy 1 signal at position D compared to positions A,B,C that are very close to baseline 1 and (ii) final values of the copy 0 signals are increasingly away from baseline 0 with their distance to A suggesting that the efficiency of our program

[68] In principle 10x or 100x smaller concentrations could be used for out system, but we found signal to be weaker, i.e. worse signal-to-noise, at lower concentrations.

[69] These constraints still allow for many computations: there are at least $8^8 \simeq 16,000,000$ such machines.

[70] With current 24-base domain design, using full M13 p7249 would allow to process 302-bit inputs. Attempting such a massive scale-up introduces new challenges that are left to future work.

program name	competitive complexity	number of inputs tested	corresponding number of experiments	yield estimation
BIT-COPY	2	2	8	90 %
PARITY	2	5	5	89 %
ADDITION	2	4	16	88 %
MULTIPLYBY3	3	7	28	93 %
3-STATE NFA	4	16	16	79 %
DIVIDEBY2 (ternary)	2	1	4	97 %
		total: 35	total: 77	average: 91 % deterministic 89 % including NFA

Table 4.1: Summary of 1D Scaffolded DNA Computer results. The "competitive complexity" of a Scaffolded DNA Computer program is the maximum number of strands in competition at a given scaffold position – it generally corresponds to the number of states in the simulated finite state machine. We tested several inputs for each program. We often have to read 4-bit outputs which requires to run 4 experiments per input: one per output bit. The "yield estimation" is computed by measuring the average distance of the last values of the curves outputting a 0 to baseline 0 and the curves outputting a 1 to baseline 1 (slightly different rule for 3-STATE NFA, see Section 4.4.6).

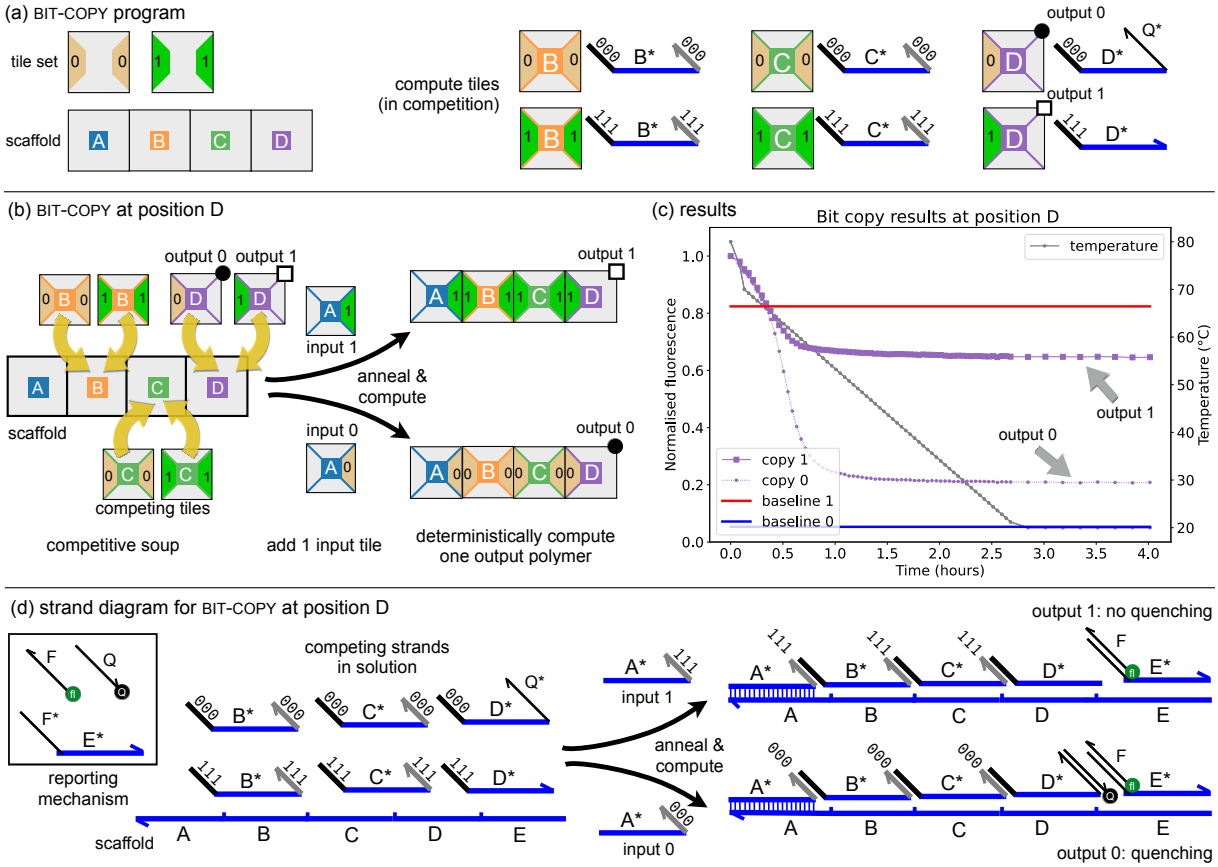


Figure 4.8: Distance 4 BIT-COPY, i.e. reporting at position D. (a) BIT-COPY program: tile set, scaffold and compute tiles. (b) Tile-level abstraction of the computation on inputs 0 and 1. Two tiles compete at each of positions B, C, and D; but only one input tile (0 or 1) is provided at position A. Annealing a mix of DNA strands that encode the program and input performs the BIT-COPY computation. (c) Normalised fluorescence BIT-COPY results at position D. We observe more quenching when copying 0 (bottom purple curve) than when copying 1 (top purple curve). The values baseline 0 and baseline 1 are each the mean completion level of 15 non-computing/non-competitive systems that make polymers of distance 1–4 (Figure E.1). (d) Strand-level abstraction of the computation. Tiles map to strands in a straightforward 1-to-1 manner (Figure 4.6(d)) and a sequence-independent reporting mechanism is used (Figure 4.7).

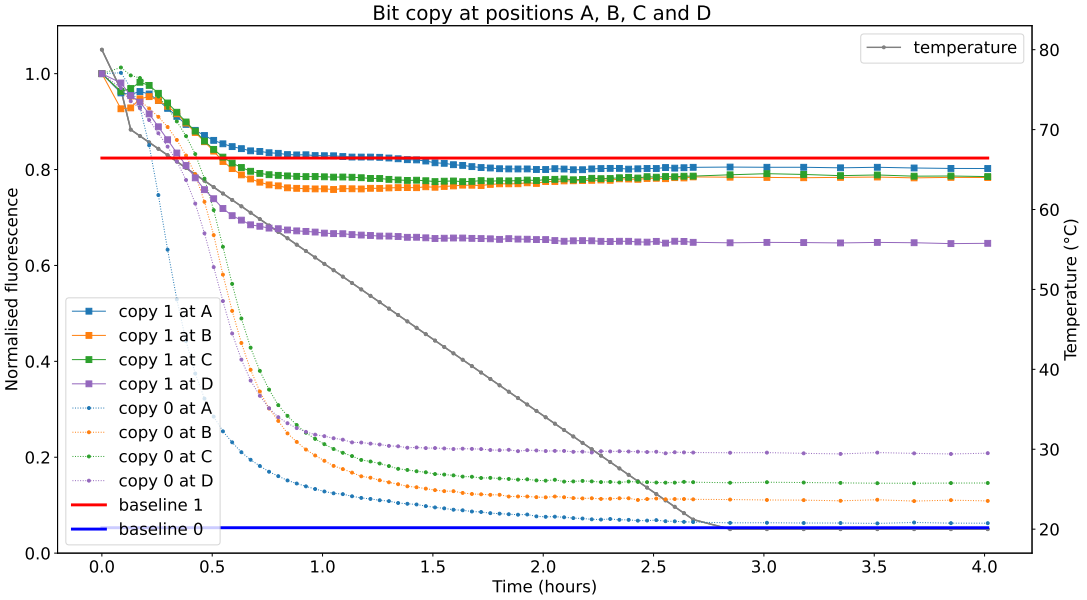


Figure 4.9: BIT-COPY results at positions A,B,C and D. Similarly to Figure 4.8 which focuses on BIT-COPY at position D (also shown here) one can run the BIT-COPY program up to positions A, B and C. Note that there is no competition between tiles at position A because it is the input position hence copy 0/1 at A are effectively controls and should match the baseline.

decreases with the distance from the input, which is what we expect. More quantitatively, we estimate a 90 % yield on BIT-COPY results (yield definition in Section 4.4.1).

4.4.3 Program 2: PARITY

PARITY **design.** PARITY consists in deciding whether the number of ones in a binary string is odd or not. This task is solved by the 2-state machine featured in Figure 4.10: strings with an even number of ones end up in state 0 and strings with an odd number of ones end up in state 1. For instance, the output of PARITY on `1011` is 1 (odd number of ones), see Figure 4.10(c); the output on `1111` is 0 (even number of ones). We run PARITY on 4-bit inputs, 1 bit per scaffold position and we report the final state of the computation, 0 or 1, at D. Two tiles (i.e. strands) compete at positions B,C,D (because there are two states in the PARITY machine). The tile at position A is uniquely determined by the first bit of the input and the fact that the initial parity is 0 (initial machine state).

PARITY **results.** Figure 4.10(e) and (f) report executions of PARITY on 5 inputs. The executions are successful because the final signal of each curve reporting a 0 (resp. 1) is close to baseline 0 (resp. 1). Quantitatively, we measure a 89 % estimated yield on PARITY results (yield definition in Section 4.4.1).

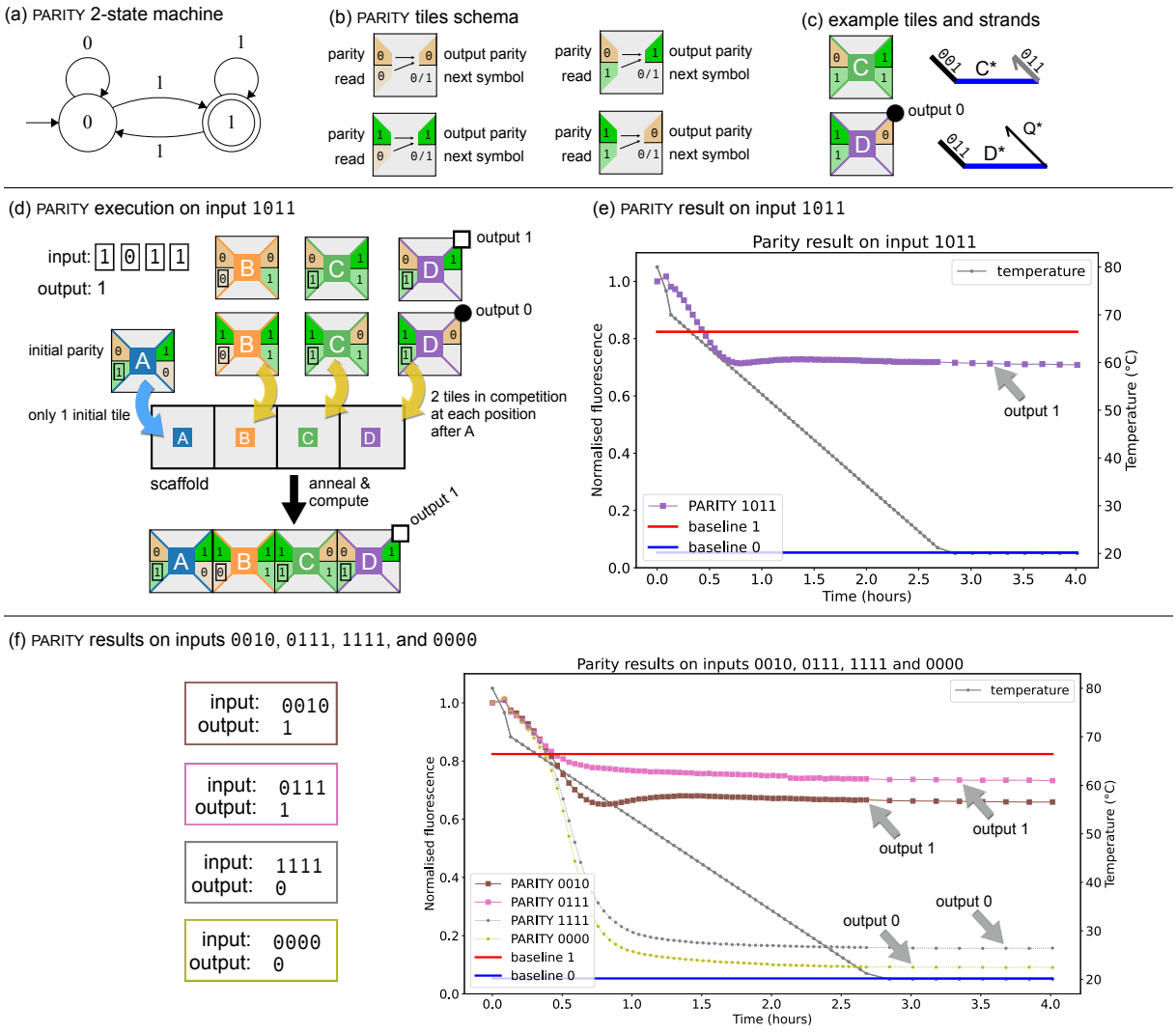


Figure 4.10: PARITY results. (a) 2-state finite state machines that computes PARITY. Input words that end up in the doubly-circled state 1 contain an odd number of ones. For instance, input words 0010 or 0111 contain an odd number of ones and their final state is the doubly-circled state 1. (b) Conversion of the state machine into tiles following the method of Figure 4.4. (c) Example instantiations of the tiles at different scaffold positions and corresponding strands. (d) Example PARITY computation on input 1011: two tiles (i.e. strands) compete at scaffold positions B,C,D. The tile at position A is given by the initial state of the machine (parity 0) and the first input bit. (e) Normalised fluorescence results of PARITY on input 1011. (f) Normalised fluorescence results of PARITY on inputs 0010, 0111, 1111, 0000.

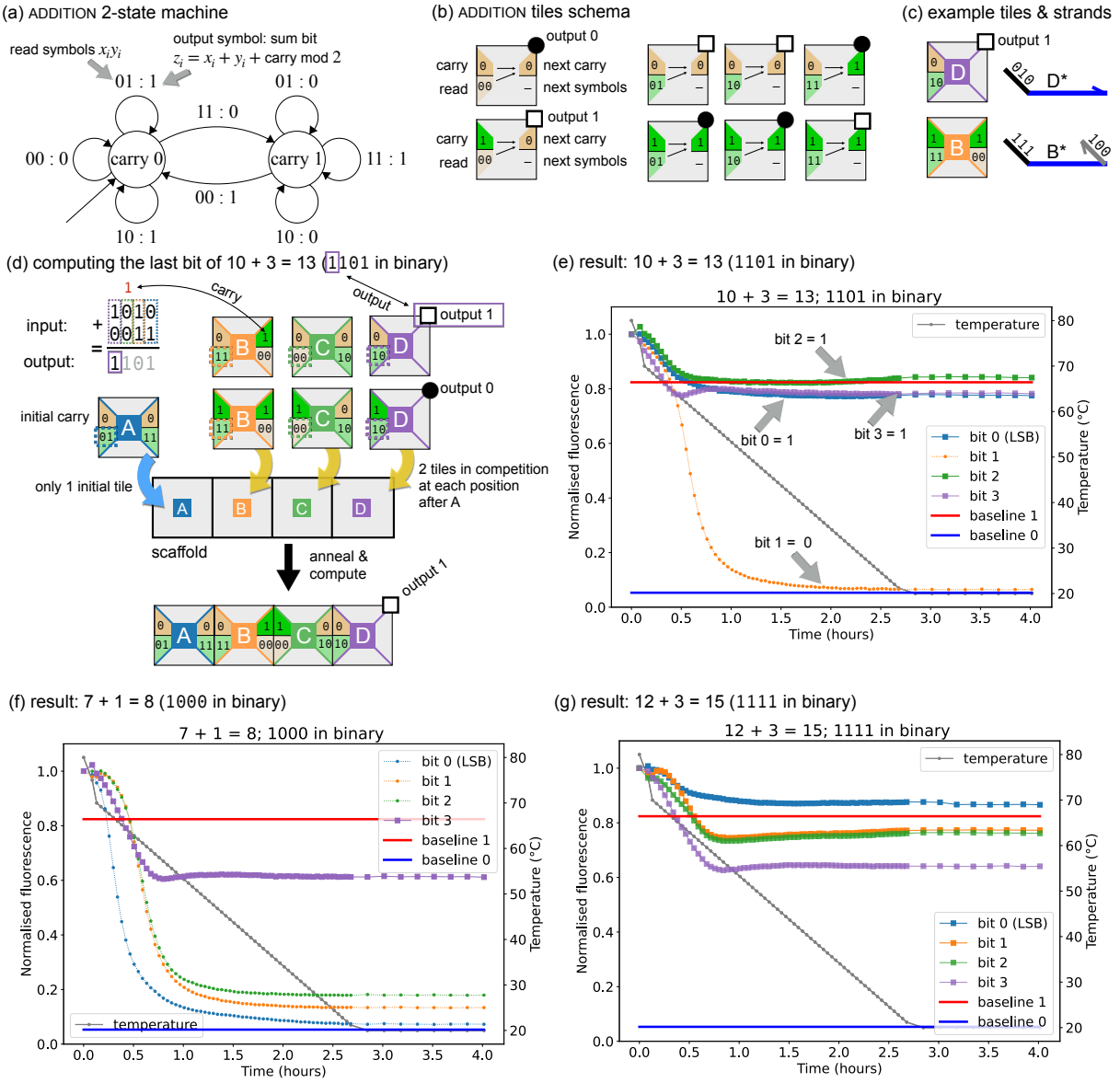


Figure 4.11: ADDITION results. (a) 2-state finite state machine that computes ADDITION. The machine reads two bits a time (one from each input number that we are adding) and outputs the sum of the two bits with the carry, given by the state, at each steps. ADDITION processes inputs from their least-significant bit. (b) Conversion of the state machine into tiles following the method of Figure 4.4. (c) Example instantiations of the tiles at different scaffold positions and corresponding strands. (d) Example computation of the last (most-signficant) bit of $10 + 3 = 13$ (1101 in binary), which is binary digit 1. Two tiles (i.e. strands) compete at each scaffold position B, C and D. The tile at position A is given by the initial state of the machine (carry 0) and the first input bits. (e) To compute the 4 output bits of $10 + 3 = 13$ we perform 4 experiments, one per output bit, reporting successively at positions A, B, C and D. Here we read the output 1101 as each bit's curve finishes close to the corresponding baseline: e.g. bit 0 is the least significant bit of 1101 (underlined) and the blue curve (partially hidden by purple) is closest to baseline 1. (f) and (g) ADDITION results on $7 + 1 = 8$ and $12 + 3 = 15$.

4.4.4 Program 3: ADDITION *(4-bit adder)*

ADDITION design. ADDITION is performed in binary by the finite state machine depicted in Figure 4.11(a). In hindsight the machine is a formal way to express what we know since primary school: the only thing to remember while processing an addition is the current carry. In binary there are only two possible carry values: 0 or 1, hence the two states in the machine. The machine reads two input bits at a time (one from

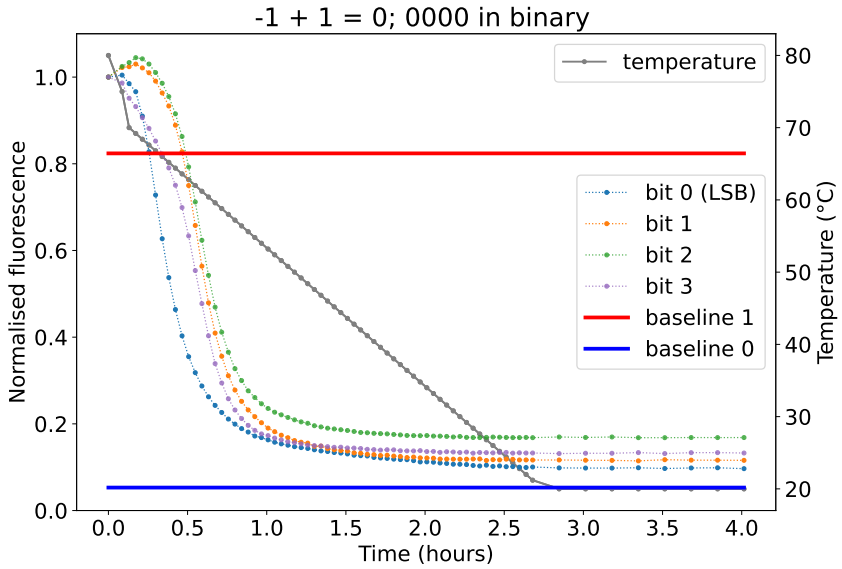


Figure 4.12: Additional ADDITION result: signed addition $-1 = 1 = 0$. The number -1 is represented by binary `1111` which exploits the overflow that `1111` + `0001` = `0000` on 4 bits.

each number we are adding), starting from the least significant side, and outputs the sum of these two bits with the state-encoded carry at each step. Our scaffold has 4 domains hence we can compute the addition of any two 4-bit numbers (discarding potential overflows). With our reporting mechanism we can only output one bit per experiment so we perform four experiments per addition: one per output bit. Figure 4.11(d) gives the tile-level abstraction of computing the last (most-significant) bit of $10 + 3 = 13$ (`1101` in binary) which is the binary digit 1. Two tiles (i.e. strands) compete at scaffold positions B, C, D and the tile at position A is determined by the initial state of the machine (carry 0) and the first input bits. Converting the tile abstraction into strands is a simple conversion and some examples are given in Figure 4.11(c).

ADDITION **results.** Figure 4.11(e) reports the computed 4 bits of $10 + 3 = 13$ (`1101` in binary). Each curve corresponds to an output bit, the blue curve (partially hidden by purple) corresponds to the least-significant bit of `1101` (underlined), computed at scaffold position A. We successfully read `1101` as each curve is closest to the correct baseline. Similarly we correctly read the results of $7 + 1 = 8$ (`1000` in binary) and $12 + 3 = 8$ (`1111` in binary) in Figure 4.11(f) and (g). Figure 4.12 give an additional ADDITION result with the signed addition $-1 + 1 = 0$: the number -1 is represented by binary `1111` which exploits the overflow that `1111` + `0001` = `0000` on 4 bits. Quantitatively, we measure a 88 % estimated yield on PARITY results (definition in Section 4.4.1).

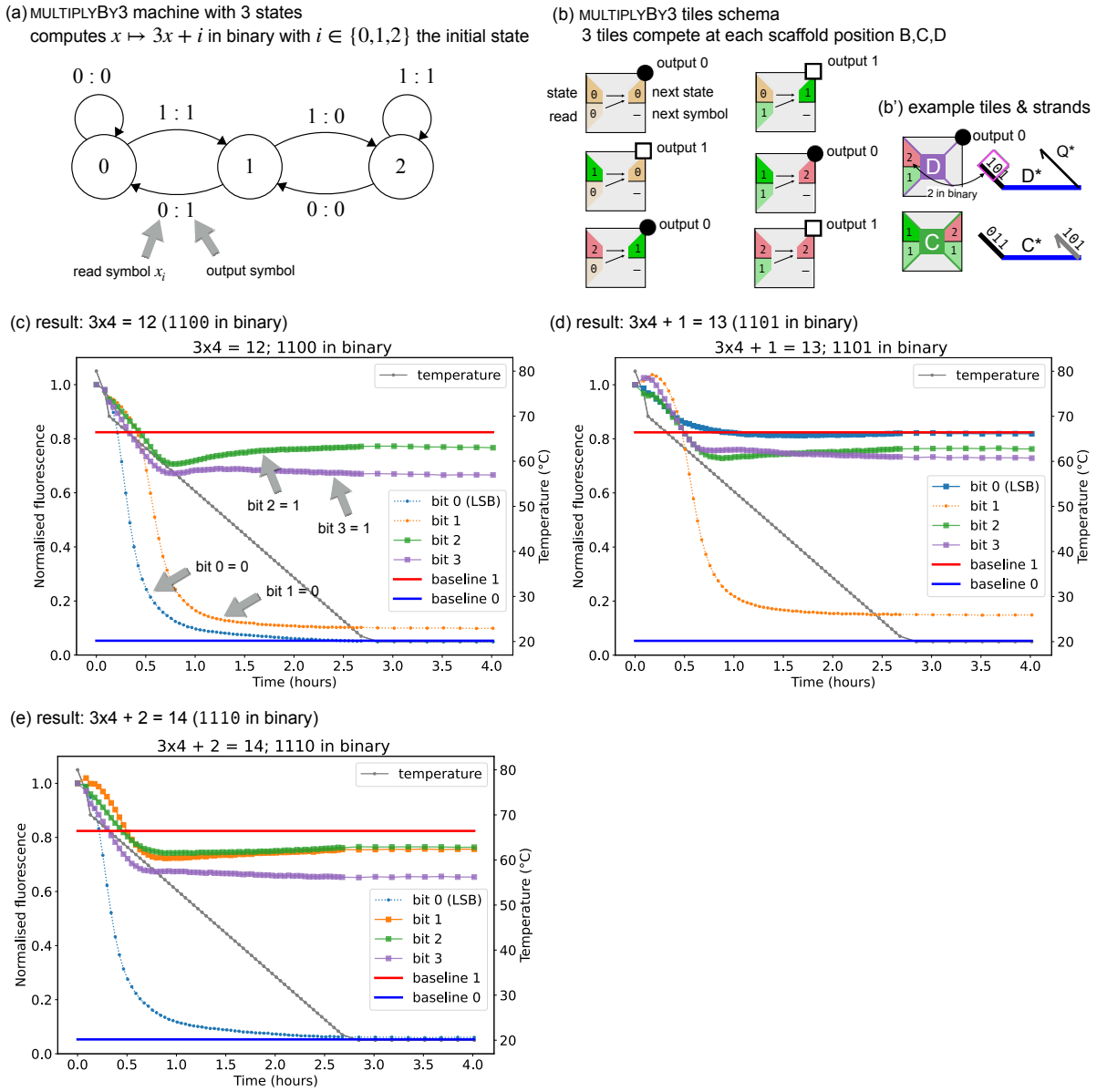


Figure 4.13: MULTIPLYBY3 results. (a) 3-state machine that computes the operations $x \mapsto 3x + i$ ($i \in \{0, 1, 2\}$ is the initial state) in binary. Three tiles compete at each scaffold position B,C,D (because 3 states). Binary inputs are read from their least-significant side. For instance, processing 0100 (4 in binary, least-significant bit is underlined) from state 2 outputs 1110 which is $3 \times 4 + 2 = 14$ in binary. (b) Conversion of the state machine into tiles following the method of Figure 4.4. (b') Example instantiations of the tiles at different scaffold positions and corresponding strands. (c) MULTIPLYBY3 results on $3 \times 4 = 12$, 1100 in binary (least-significant bit 0 is underlined). Bit 0 is read with the blue curve reaching baseline 0. We correctly read output 1100 as each bit's curve is closest to the correct baseline. (d) and (e) MULTIPLYBY3 results on $3 \times 4 + 1 = 13$ (1101 in binary) and $3 \times 4 + 2 = 14$ (1110 in binary).

4.4.5 *Program 4:* MULTIPLYBY3

MULTIPLYBY3 **design.** Figure 4.13 gives the 3-state machine that computes the operations $x \mapsto 3x + i$ ($i \in \{0, 1, 2\}$ is the initial state) in binary, which we first met in Chapter 1 Figure 1.2. Binary inputs are read from their least-significant side. For instance, processing 0100 (4 in binary, least-significant bit is underlined) from state 2 outputs 1110 which is $3 \times 4 + 2 = 14$ in binary [156]. Figure 4.13 (b) and (b') give the conversion from the machine to tiles and examples of how they are turned into strands. Three tiles (e.g strands) compete at each scaffold position B,C,D (because 3 states).

MULTIPLYBY3 **results.** Figure 4.13 (c) shows the results of $3 \times 4 = 12$ (1100 in binary least-significant bit 0 is underlined). Bit 0 is read with the blue curve reaching baseline 0. We correctly read output 1100. Similarly, we read the results of $3 \times 4 + 1 = 13$ (1101 in binary) and $3 \times 4 + 2 = 14$ (1110 in binary), Figure 4.13 (d) and (e). Figure E.3 gives 4 more MULTIPLYBY3 results: $3 \times 5 = 15$, $3 \times 3 + 1 = 10$, $3 \times 1 + 2 = 5$ and $3 \times 2 + 2 = 8$. We measure a 93 % estimated yield (yield definition in Section 4.4.1).

4.4.6 Program 5: 3-STATE NFA

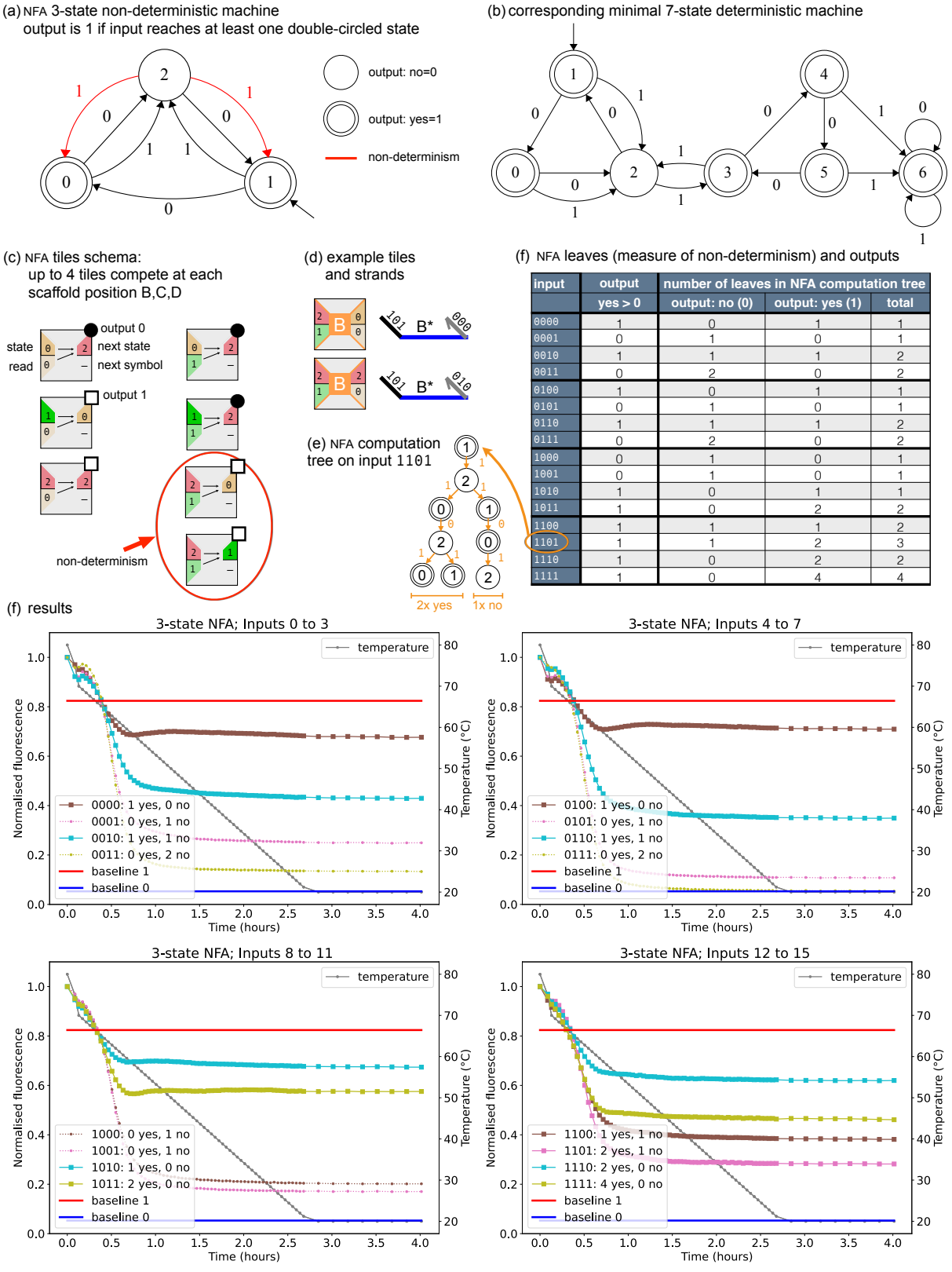
3-STATE NFA **design.** For the sake of simplicity, Figure 4.4 and Theorem 4.5 only covered the simulation of *deterministic* finite sate machines, that is, machines where there is always only one choice for the next state to visit. In fact, our framework can also handle *non-deterministic* machines: an example that we implement is depicted in Figure 4.14(a). The non-determinism comes here from the fact that there are two possible choices for next state when reading symbol 1 in state 2 (red arrows). A deterministic machines outputs a 1 (or yes) from an input if it is able to reach at least one doubly-circled state. Figure 4.14(e) shows that input 1101 is able to reach two "yes" states hence the corresponding output is yes. Figure 4.14(f) gives the output and detailed yes/no counts for all 4-bit input. Any non-deterministic machine can be converted into a deterministic machines that will output yes for the exact same inputs, at the cost of having more states: Figure 4.14(b) shows the smallest deterministic machine corresponding to 3-STATE NFA, it has 7 states.

Practically, as we are toehold-limited, simulating non-deterministic machines is powerful because it allows us to run bigger machines machines than we possibly could deterministically. Here, we don't have enough toeholds to simulate the deterministic 7-state machine: it would require 3 bits for state and 1 bit for symbol and we only have 3 bits in total with our 8 toeholds. However, we can simulate the smaller non-deterministic machine (2 bits for state and 1 bit for symbol), which computes the same results.

In the non-deterministic case, our DNA-based system, instead of assembling one polymer, will assemble as many polymers as leaves in the non-deterministic computation tree of the input, e.g. 3 polymers for input 1011, Figure 4.14(e). Up to four tiles (e.g strands) compete at each scaffold position B,C,D (because up to 4 transitions to consider). Interpreting results becomes more challenging because deducing whether the output is yes or no requires a fine-grained quantification of our quenching signals. Take the case of the two top cyan curves in Figure 4.14(f) corresponding to inputs 0010 and 0110. They roughly reach the half-mark between baseline 1 and baseline 0 which is consistent with the fact that their computation contains 1 yes and 1 no: half of the polymers quench and the global output is yes. More generally, as soon as there is 1 no, signals drop significantly because of the corresponding quenching polymers population.

3-STATE NFA **results.** We are able to read the correct outputs in almost all the cases. The most challenging cases are inputs 12 to 15. For instance, the pink curve corresponding to input 1101 reaches a signal almost as low as the pink curve corresponding to input 0001 (top-left plot) but in the first case the output is yes (2 yes, 1 no) and no in the second case (0 yes, 1 no). Hence it is difficult to clearly differentiate yes/no answers in this case.

Estimated non-deterministic yield. We use a similar idea as for deterministic machines in order to estimate the yield of 3-STATE NFA results. We set the following targets depending on yes/no counts: (a) "1 yes 0 no" curves should reach baseline 1 (b) "0 yes 1 no" and "0 yes 2 no" curves should reach baseline 0 (c) "1 yes 1 no" curves shoud reach halfway between baseline 0 and baseline 1 (d) the "2 yes 1 no" curve should reach 2/3 of the way between baseline 0 and baseline 1. We measure the average distance of the final values of each curve to their respective target. We get a 79 % estimated yield which is lower than deterministic computations but which is expected as non-determinism is a harder task.



input	output	number of leaves in NFA computation tree		
	yes > 0	output: no (0)	output: yes (1)	total
0000	1	0	1	1
0001	0	1	0	1
0010	1	1	1	2
0011	0	2	0	2
0100	1	0	1	1
0101	0	1	0	1
0110	1	1	1	2
0111	0	2	0	2
1000	0	1	0	1
1001	0	1	0	1
1010	1	0	1	1
1011	1	0	2	2
1100	1	1	1	2
1101	1	1	2	3
1110	1	0	2	2
1111	1	0	4	4

Figure 4.14: 3-STATE NFA results. (a) non-deterministic 3-state machine: when reading a 1 in state 2 there are two choices for next state.An input word is accepted (output yes=1) if it can reach at least one doubly-circled state (state 0 and 1). (b) any non-deterministic machine can be simulated by a determinitic one, at the cost of more states, here the minimal number of states in the deterministic case is 7. (c) Translating the machine into tiles, the non-determinism appears as there are two tiles with West color state-2 symbol-1 with different East color (circled in red). (d) Translation to strands. (e) Example execution on input 1101: the computation is a tree, 2 yes and 1 no are reached hence this output is 1 (at least 1 yes). (f) Detail of the machine output and yes/no counts for all 16 4-bit inputs. (f) Results of all 16 4-bit executions.

4.4.7 Program 6: DIVIDEBY2 *(ternary reporting)*

DIVIDEBY2 **design.** We implement division by 2 in base 3 which is performed by the 2-state machine[71] depicted in Figure 4.15(a), which we first met in Chapter 1, Section 1.5. **Important:** contrarily to ADDITION or MULTIPLYBY3, this machine processes inputs starting at their most-significant base-3 digit (trit). For instance, the operation $76/2 = 38$ is computed by processing $\underline{2}211$ (76 written in ternary, most significant trit underlined) from state 0 which outputs `1102` (38 written in ternary).

Reporting output 2 with *late quenching*. The reporting mechanism that we use was originally designed to only report binary outputs: 0 or 1, Figure 4.7. By a happy mistake mistake, we found a way to use it which allows us to report a third output, 2. We call this phenomenon *late quenching*. Indeed, if instead of using 20-base domain Q* at a reporting position (Figure 4.7 (a)), we use one of our 12-base computing toeholds (in particular toehold 5, sequence CACTACCAGTCC), Figure 4.15(d), we obtain a quite strong secondary structure with our quencher-labelled strand Q (-10.61 kcal/mol at 37C, Figure 4.15(e)). This means that the quencher-labelled strand Q will spuriously bind to toehold 5 at temperatures lower than the Q-Q* interaction hence we should see quenching occuring *later* than when reporting a 0. We tend to see normal Q-Q* quenching in the 65-75C range (green curve in Figure 4.15(f)) whereas this *late quenching* phenomenon occurs in the 30-50C range: cyan in Figure 4.15(f) is a control for the mechanism (no competition between strands).

DIVIDEBY2 **results.** Using late-quenching we can report computations that use a ternary alphabet. Figure 4.15(f) reports successful results for $76/2 = 38$: we read `1102` as curves for the first two `1`s reach baseline 1 (blue and orange curves), the green curve for `0` reaches baseline 0 and the purple curve for `2` features *late quenching* which correspond to reading a 2. We measure a 93 % estimated yield (definition in Section 4.4.1, ignoring ternary value 2).

[71] This machine is very close in structure to the binary PARITY machine (Figures 4.4 and 4.10) and this is related to the mathematical fact that deciding if a ternary number is divisible by 2 amounts to computing the parity of its number of 1s, which is exactly what PARITY does [156].

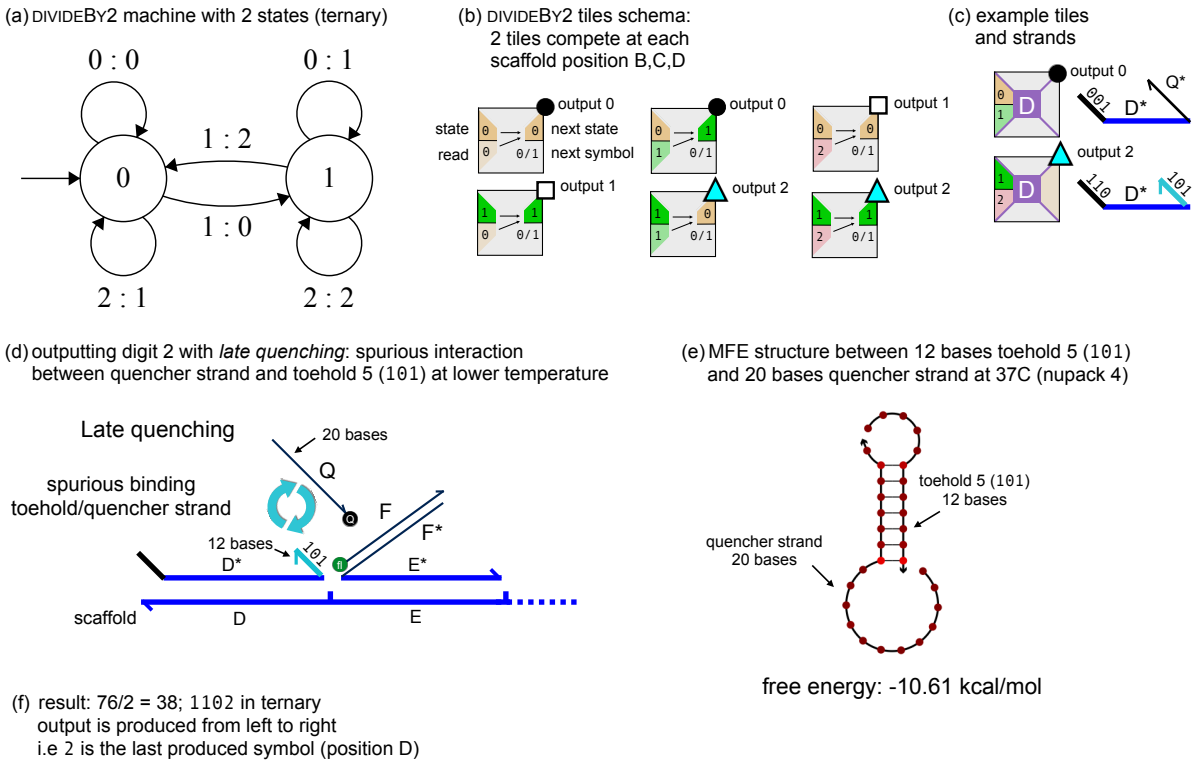


Figure 4.15: DIVIDEBY2 results. (a) 2-state machine that computes $x \mapsto x/2$ in base 3 (ternary). **Important:** this machine processes inputs starting at their most-significant base-3 digit (trit). For instance, the operation $76/2 = 38$ is computed by processing $\underline{2}211$ (76 in ternary, most significant trit underlined) from state 0 which outputs 1102 (38 in ternary). (b) and (c) Correponding tiles and example strands. (d) and (e) In order to report the base-3 digit 2 we introduce a mechanism that we call *late quenching*. The quencher-labelled domain Q has a strong secondary structure with toehold 5 (101 in binary) shown in (e) of -10.61 kcal/mol free energy. This means that at lower temperature than the Q-Q* interaction we expect to see Q spuriously bind to toehold 5. (f) $76/2 = 38$ results. The cyan curve is a control for late quenching (no competition) and we see that the quenching is triggered later than usual Q-Q* quenching (which happens in the green curve). We successfully read output 1102 where least significant digit 2 is reported at scaffold position D using late quenching (purple curve).

4.5 2D Scaffolded DNA Computer

4.5.1 Scaffolded DNA Computer instance on a 2D scaffold

For implementation of the Scaffolded DNA Computer on a 2D scaffold, we chose a rectangular DNA origami chassis [44] shown in Figure 4.16. The origami design uses a relatively standard crossover pattern. We added a square of twelve 5' biotin staple modifications (bottom left), used as a control for streptavidin labelling efficiency (bottom left).

Primarily, the design includes a computing region (shown as tiles on right hand side, Figure 4.16) that houses a 7-bit Scaffolded DNA Computer instance. We used our "universal" set of 16 toeholds (design in Section 4.3), appending them to staples according to the schema Figure 4.6. This gave a total of 392 designed computing staples (8 at the input location A, and 64 each at the other 6 locations B–G), although only 126 were ordered for the computations reported here.

Some of the computing strands/staples have 3' and/or 5' biotins; typically those toeholds that encode bit 1 are labelled, and bit-0 toeholds are unlabelled. Staple computing strands, were ordered unpurified in plates, as were biotin-modified staple computing strands, both from IDT. M13 scaffold was ordered from Tillibit.

Experimental Protocol AFM imaging was carried out on a Bruker Fastscan, with Fastscan-D probes. Streptavidin protein was purchased from Rockland. Samples were annealed as follows: Hold at 90C for 5 minutes; drop to 60C by -1 °C per minute; drop to 50 °C by -0.1 °C per minute; drop to 45 °C by -0.1 °C per minute; drop to 20 °C and hold.

For AFM imaging, typically 1.5 μl of sample was deposited into 80 μl of 12.5 mM Mg++, 1xTAE on a freshly cleaved mica surface. After imaging to check for quantity and quality of origami, 1 μl of 5 μM, streptavidin was added using a protocol similar to that from previous work [183]. Streptavidin imaging of the kind used here (single streptavidin molecules on the end of 12bp duplexes) is challenging: during some imaging sessions, streptavidin didn't show brightly. In that scenario, we added 0.5 μl more to the droplet on mica. Imaging was not consistent in terms of quality during different sessions. We repeated imaging of the samples to get the best images we could.

As noted, read-out of our computations is by AFM imaging of biotin-streptavidin labelled strands on DNA origami. Although similar techniques have been used with success in the past [24, 66], it is known to be technically challenging to read-out single molecule labels in some contexts [183], and our system is perhaps even more challenging to image due to the streptavidin proteins being on the end of a 12-base duplex extending from the origami surface.

4.5.2 Program 1: BIT-COPY

See Figure 4.17 for the 7-bit 2D Scaffolded DNA Computer BIT-COPY program. Section 4.4.2 gives the details of the tile set, and Figure 4.8 shows the strand level of the computation for 4-bit copy (in the 1D setting). If input A is 1, then all subsequent positions should compute 1. and if A is 0, then all positions should encode 0. We labeled the 1's with biotin to be visible when imaging with streptavidin, and 0's were regular unmodified strands. We see in the diagram in Figure 4.17(left) copy 0 shows no orange dots (labels) and copy 1 shows 7 dots.

Hardcoded controls. For BIT-COPY computations we had additional control experiments where only single-bit strands are added (meaning no competition, and thus no computation). We call this a "hardcoded" control. *Hardcoded-0* means we only add the strands {A0, B0, C0, D0, E0, F0, G0} and

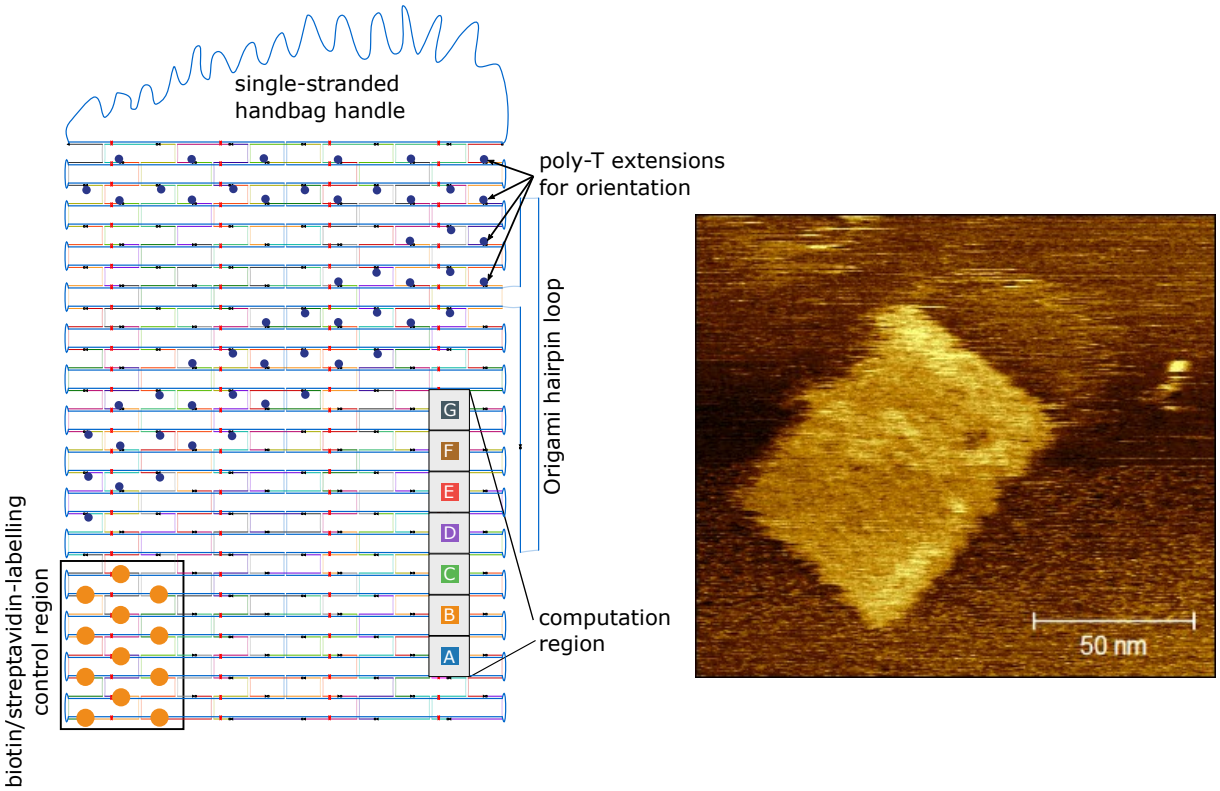


Figure 4.16: 2D Scaffolded DNA Computer chasis, taken from [44]. Left: scadnano design [58] (based on an original origami design by David Doty for use in another project; the origami has some features not needed for our work here including the single stranded region on top and the poly-T extensions in the shape of a number 7). M13 scaffold is in blue and staples are various colours. Small blue dots indicate poly-T extensions, in the shape of the number '7'. Twelve biotin modifications (orange dots, bottom left) were added to staples to control for streptavidin imaging. Along the right had side there is a vertical *computation region*, labelled A–G, where seven staples positions house staples modified with our set of 16 toeholds (Appendix E.3). Right: AFM of single origami. The poly-T extensions (shaped like a '7'), and the computation region (a line segment of six 12bp duplexs), are seen as lighter in colour (taller, in z direction) than the rest of the origami. Also, the single stranded part of the handle is more faintly visible above and to the right of the origami. The biotin control square is *not* visbile because this AFM image is taken before adding streptavidin.

hardcoded-1 means we only add strands {A1, B1, C1, D1, E1, F1, G1}. Since each staple position has exactly one staple that fits at that position, and since origami is known to form with relatively high yield of staples bound, we hope to have as high as possible yield of imaged streptavidins at those positions.

Of course experiments with computing strands should have, if anything, lower yield of streptavidin labelling than these hardcoded controls. Section 4.5.3 gives an analysis of the controls (to be used in the following paragraph).

Analysis of COPY 0 and COPY 1 experiments. Biotin/streptavidin labelling challenges and limitations are discussed below in Section 4.5.3. COPY -0 wide scan in Figure 4.17(g) shows 4 structures out of 10 with what might have 1 streptavidin label, i.e. a putative error in the computation. The rate of erroneous structures is 40%, giving a success rate of 60% which fairs reasonably with our our 74% HARDCODED-1: 2-LABELLING rate (see below), but of course suffers from small numbers in the statistics. COPY 1 wide scan in Figure 4.17(i) shows 15 structures with correct answer out of 23 total with 68% success rate which is good considering the HARDCODED-1: 2-LABELLING rate.

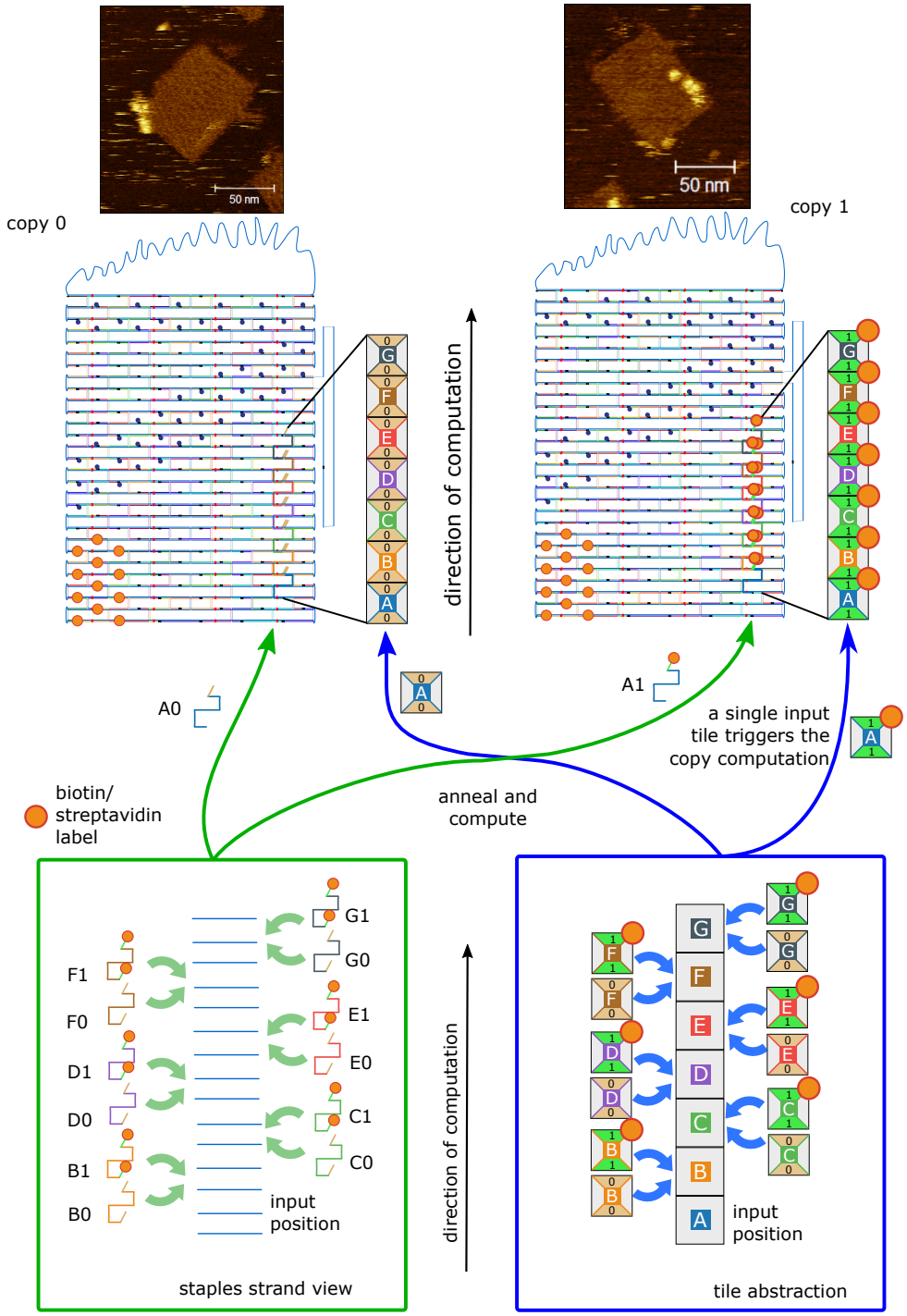


Figure 4.17: BIT-COPY within origami results, depth 7. At the bottom, we have two views of the computing circuit. On the left, in the green box, it is a strand view where staples and M13 scaffold are represented. On the right, in the blue box, is the tile view (Scaffolded DNA Computer). The computation is visualised for both as competing tiles/strands at each scaffold position. Based on the input (either 0 or 1), and the competitive process, the sequence of computing strands is chosen. The middle of the figure is showing a representation of expected result depending on the input. If input is 1, then 1 is to be copied by selecting the strands modified with biotin (right). If input is 0, then 0 bit is to be copied where selected strands are not labeled with biotin (left). On the top of the figure are the AFM results where when copying 0 there is no streptavidin label and when copying 1 streptavidin is showing bright.

4.5.3 Control data: hardcoded-1 and hardcoded-0 labelling reference rates

The following data is reported in Figure 4.19.

Hardcoded-1: 2-labelling rate. In the hardcoded-1 control, Figure4.19(h), if we consider every structure that has at least 2 putative streptavidins (i.e. two visible dots/blobs at valid computing region positions) to be "reporting", we get a count of 23 reporting structures out of 31, i.e. 74 %. In some cases, we found that most of the structures were upside-down (flipped). If we eliminate those structures, we will get a total of 24 structures where 19 out of them are reporting (83 %).

Hardcoded-1: 3-labelling rate. In the hardcoded-1 control, Figure4.19(h), if we consider only the structures that show 3 dots, then there are 17 reporting structure out of 31 (55 %).

Hardcoded-0. Hardcoded-0 gave exactly what is expected, which is a biotin/streptavidin control square showing good labelling, but not streptavidins visible on the computing region, see Figure 4.19(f) where we have a small sample count of four structures. However, we counted other images of the same sample (data not shown), and as expected they all showed no trace of streptavidin (except in the control square region).

On the other hand, hardcoded-1 showed biotin/streptavidin column with varying number of streptavidin labels, Figure 4.19(h), but always ≤ 4. We note that, assuming perfect DNA strand synthesis and perfect staple binding there should be 13 biotins present (at the 3' and 5' ends of 6 duplexes). However, each streptavidin has a valency of 4 (binds up to 4 biotins), so the number of streptavidins imaged does not precisely map to the number of biotins present (which is what we really care about). Finally streptavidin binding is not always favoured [183], and of course the AFM tip may not "see" a streptavidin hanging at the end of a wobbly 12bp duplex stem.

We saw 4 dots on very few structures. Seeing less than 3 dots (on hardcoded-1) denotes that streptavidin binding, or streptavidin imaging, or strand purity (no biotin!), is certainly affecting our labelling/imaging yield. One of our future work priorities is to refine the streptavidin imaging protocol, but more likely we will focus on changing the design to make readout easier and have less variability.

As a streptavidin-labelling control, we will consider the hardcoded-1 control to be our guide as to the maximum achievable labelling rate, even for a perfect computation (of say copy-1).

4.5.4 Program 2: ADDITION *(7-bit adder)*

See Figure 4.18 for the 7-bit 2D Scaffolded DNA Computer ADDITION program.

Analysis of AFM images for 7-bit ADDITION

To analyse our data, given in Figure 4.19, we will follow the argument from Section 4.5.2 that the maximum expected success rate for streptavidin labelling is 74 %. In Figure 4.19:

(a) shows a wide scan of the $1 + 1 = 2$ computation that shows a total of 35 structures. In this case we saw that more than 70 % of the structures were upside down (face down on the mica), and we note that we found it quite tricky to see streptavidin on upside-down structures. We counted 9 out of 35, or 26 %, with the correct result. But eliminating all upside-down structures, whether they answer correctly or not gives 10 structures, with 5 of them reporting the correct answer (50 %; but smaller statistics).

(b) shows a wide scan of $1 + 7 = 8$ computation that shows total of 14 structures. We see 9 structures reporting the correct answer, i.e. 64 %. Eliminating all upside-down structures, whether they answer correctly or not gives 6 structures, with 4 of them reporting the correct answer (67 %).

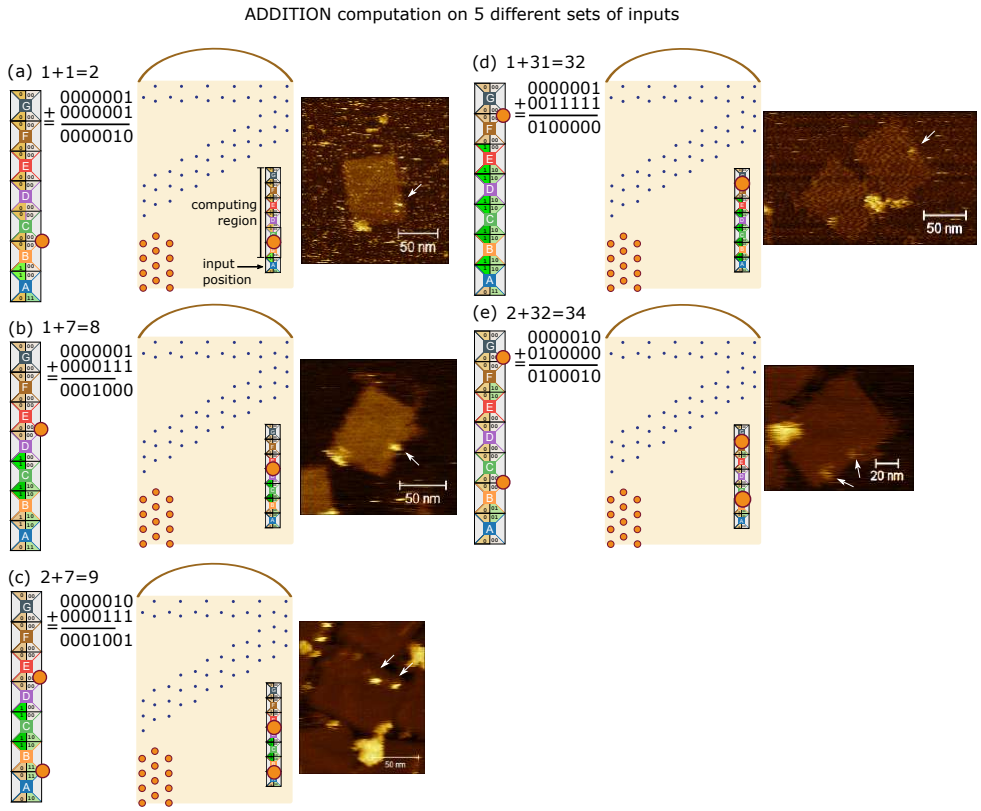


Figure 4.18: Results for Scaffolded DNA Computer 7-bit ADDITION program on 2D origami. For each of (a–e) the 7-tile output of the abstract tile-level program is shown on the left hand side, read from bottom to top (LSB on bottom). An orange dot is used to denote the bit 1, which corresponds in turn to a biotin-streptavidin label on the DNA origami. In AFM images, white arrows point at the biotin/streptavidin conjugation that reports bit 1. (a) ADDITION in binary of: 1+1=2, giving 0000010_2 in binary. (b) ADDITION of binary 1+7=8 (0001000_2). (c) ADDITION of binary 2+7=9 (0001001_2). (d) ADDITION of binary 1+31=32 (0100000_2). (e) ADDITION of binary 2+32=34 (0100010_2).

(c) shows a wide scan of $2+7=9$ computation that shows total of 28 structures. 14 structure report the correct answer, i.e. 50 %. Eliminating all upside-down structures, whether they answer correctly or not gives 9 structures, with 7 of them reporting the correct answer (76 %; which is more than the hardcoded-1: 2-labelling rate.).

(d) shows a wide scan of $1+31=32$ computation that shows total of 19 structures. We see 7 structures reporting the correct answer, i.e. 37%. If we eliminate all of the upside-down structures either giving correct answer or not, we should have total of 8 structures where 4 of them are giving the correct answer (50 %).

(e) shows a wide scan of $2+32=34$ computation that shows total of 11 structures. We see only 2 structures reporting the correct answer with ratio of 18%. If we eliminate all of the upside-down structures either giving correct answer or not, there are 6 structures with 2 of them are giving the correct answer (33 %).

Discussion. Due to streptavidin labelling problems, as already discussed, it could well be the case that the algorithmic error rate for our system is significantly lower (better) than the streptavidin labelling error rate we measure by AFM, which ranges widely from 18 % to 78 %; depending on the computation and on the metric used (flipped or not), although we know from the hardcoded-1 controls that we should not really expect more than 74 %, in the very best non-competitive/'no computation' case.

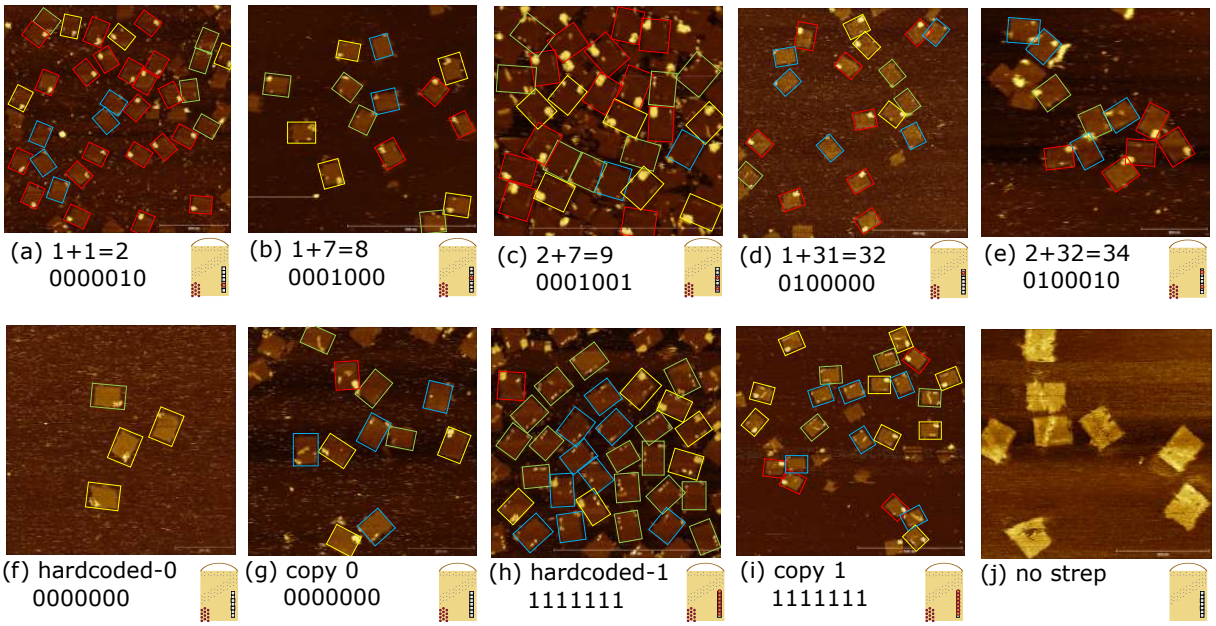


Figure 4.19: Wide-scan AFM images. (a) ADDITION in binary of 1+1=2, giving 0000010_2 in binary. (b) ADDITION of binary 1+7=8 (0001000_2). (c) ADDITION of binary 2+7=9 (0001001_2). (d) ADDITION of binary 1+31=32 (0100000_2). (e) ADDITION of binary 2+32=34 (0100010_2). (f) DARDCODED-0 control (0000000_2). (g) COPY 0 (0000000_2). (h) HARDCODED-1 control (1111111_2). (i) COPY 1 (1111111_2). (j) Origami without addition of streptavidin. DNA Origamis are annotated with coloured rectangles depending on whether orientation and answer. Green boxes annotate structures with correct answers that fell in the correct orientation. Blue boxes show structures that fell in the correct orientation but either not showing the answer or showing an incomplete one. Yellow boxes annotate structures that fell upside-down and showing correct answer. Red boxes annotate upside-down structures with no answer.

The main issues are: (a) One needs good imaging, and not too large of scans, in order to see the control square clearly, and only then can one choose that image for data analysis. This takes time, skill and patience. (b) Even with such an image, there is a lot of variability when imaging streptavidin by AFM (in particular with our setup, as already discussed) – even on three subsequent scans of the same region a streptavidin may appear, disappear and then appear again (this is explored in [183], SI-A). Hence, we plan to improve our imaging protocol to get better statistics, and to investigate new reporting mechanisms that, for example, encode more redundancy (so that multiple, e.g. 20, streptavidins are used to report a single 1, much like in [183]), or use super-resolution fluorescence microscopy. See Section 4.6 for further discussion.

4.6 DISCUSSION

We report a new method for computing with DNA which features a strand displacement mechanism (Figure 4.5) that is used in combination with a self-assembly process that occurs during a DNA origami annealing protocol. We introduce a tile-based model, the Scaffolded DNA Computer, which allows us to reason about and program these systems. The model is expressive, allowing for implementation of arbitrary Boolean circuits, see Chapter 3. With the proposed model, the structure corresponding to the correct execution of a computation is thermodynamically favoured. We leave in-depth theoretical analysis of this model to future work.

We implement the 2-sided restriction (tiles use at most 2 sides) of the Scaffolded DNA Computer using a synthetic 120-base scaffold (derived from M13) in 1D and a full DNA origami 2D. We can run any finite state machine within this restricted model, see Figure 4.4 and Theorem 4.5. We implement 6 programs

in 1D (testing 35 inputs for a total of 77 experiments excluding controls), including a functional 4-bit adder, and 2 programs in 2D (7 executions), including a functional 7-bit adder. We estimate 91 % yield for deterministic programs in the 1D Scaffolded DNA Computer.

We observe an overall success of our implementations which validates the soundness of the mechanism of (Figure 4.5) and its computational applications. We report some limitations, in particular with respect to precise results quantification, which open avenues for future improvements to the systems presented here. In 1D:

1. We tend to observe signals farther from the baseline when the length of the computation increases, suggesting a drop in efficiency of our mechanism with longer lengths. This observation does not always hold, we sometimes observe similar performances at all lengths, Figure 4.11(e) is an example. More experiments and theory about our computations are needed to understand this point better.

2. We are not able to precisely quantify the proportion of correctly executed computations from our fluorescence curves as we lack a mechanism to "force our systems to completion" [130] and therefore deduce the "true" signals for 0/1 outputs in each sample. Precisely evaluating the completion levels of our samples is an important next step in this project.

3. Based on non-reported data, we also suspect that our system is energetically biased in favor of quenching configurations, Figure 4.7(a), meaning that fluorescence curves are reporting signals lower than they would be without this bias. In future work, we plan on evaluating this bias and potentially compensate it (using longer toeholds for instance).

In 2D:

1. We currently rely on identifying the presence of individual streptavidin markers at precise positions to report success. This has proven to be particularly challenging to image at the AFM, leaving us in uncertainty when we do not see a streptavidin: (i) is the computation wrongly executed? (ii) did no streptavidin bind yet? (iii) are the AFM parameters, and/or the long 12bp duplex attachment levers, preventing us to seeing the streptavidin? We are actively working on reporting mechanisms that are experimentally less challenging.

2. For an unknown reason, an outstanding proportion the origamis that we observed were flipped (on separate imaging sessions), i.e. the origami is facing the mica (we can deduce this thanks to the bottom-left biotin square control region). We suspect that this affects our ability to image the streptavidins as we are imaging the origamis "from behind". We want to read the outputs of our future systems reliably independently of this flip.

Looking more broadly to the future, we contend that our proposal open a number of doors for many extensions and generalisation of the work presented here. We intend to work on the following:

- in 1D, use a full-length M13 scaffold (e.g. 7,249 bases, instead of our 120-base M13-derived sequence) which will allow us to run our programs on up to 302-bit inputs instead of the current 4-bit system. This massive (potential) scale-up would no doubt present numerous challenges, but could also give exciting new insights to scaffolded computation.

- Make full use of the 2D scaffold. We would remove the 2-sided constraint (only 2 side per tile, or only 2 toeholds per staple) by implementing a mechanism to have more than 2 toeholds per staples (e.g. with toehold-loopouts).

- Computations where the result is a computed shape is a particularly exciting direction.
- Triggering a computation, *after* an origami has been annealed presents another form of computation, perhaps more kinetically/slowly driven than the thermodynamically ("anneal it slowly") approach given here. In particular it could allow for our next point for shape-configuration: anneal one shape, add a few strands to trigger a computation that gives a new shape!

As these ideas should attest, we hope the Scaffolded DNA Computer has an exciting future ahead of it.

Bibliography

[1] Scott Aaronson. "The Busy Beaver Frontier". In: *SIGACT News* 51.3 (Sept. 2020). Preprint: https://www.scottaaronson.com/papers/bb.pdf, pp. 32–54. ISSN: 0163-5700. DOI: 10.1145/3427361.3427369. URL: https://doi.org/10.1145/3427361.3427369.

[2] Boris Adamczewski and Colin Faverjon. *Mahler's method in several variables II: Applications to base change problems and finite automata*. Preprint: https://arxiv.org/abs/1809.04826. Sept. 2018.

[3] Cesar O. Aguilar. *An Introduction to Real Analysis*. 2022.

[4] Shigeki Akiyama. "Mahler's Z-number and 3/2 number systems". In: *Unif. Distrib. Theory* 3 (2009), pp. 91–99.

[5] Shigeki Akiyama, Christiane Frougny, and Jacques Sakarovitch. "Powers of rationals modulo 1 and rational base number systems". In: *Israel Journal of Mathematics* 168 (Jan. 2007), pp. 53–91. DOI: 10.1007/s11856-008-1056-4.

[6] Paul Andaloro. "On total stopping times under $3x + 1$ iteration". In: *The Fibonacci Quarterly* 38 (Feb. 2000).

[7] Ebbe S. Andersen, Mingdong Dong, Morten M. Nielsen, Kasper Jahn, Ramesh Subramani, Wael Mamdouh, Monika M. Golas, Bjoern Sander, Holger Stark, Cristiano L. P. Oliveira, Jan Skov Pedersen, Victoria Birkedal, Flemming Besenbacher, Kurt V. Gothelf, and Jørgen Kjems. "Self-assembly of a nanoscale DNA box with a controllable lid". In: *Nature* 459.7243 (May 2009), pp. 73–76. DOI: 10.1038/nature07971. URL: https://doi.org/10.1038/nature07971.

[8] David Barina. "Convergence verification of the Collatz problem". In: *The Journal of Supercomputing* (July 2020). https://doi.org/10.1007/s11227-020-03368-x.

[9] Robert D Barish, Paul WK Rothemund, and Erik Winfree. "Two computational primitives for algorithmic self-assembly: Copying and counting". In: *Nano letters* 5.12 (2005), pp. 2586–2592.

[10] Robert D Barish, Rebecca Schulman, Paul WK Rothemund, and Erik Winfree. "An information-bearing seed for nucleating algorithmic self-assembly". In: *Proceedings of the National Academy of Sciences* 106.15 (2009), pp. 6054–6059.

[11] Elaine Barker. *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*. en. 2000. URL: https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=151231.

[12] David A. Mix Barrington, Neil Immerman, and Howard Straubing. "On Uniformity within NC^1". In: *Journal of Computer and System Sciences* 41.3 (1990), pp. 274–306.

[13] Lawrence Bassham, Andrew Rukhin, Juan Soto, James Nechvatal, Miles Smid, Stefan Leigh, M Levenson, M Vangel, Nathanael Heckert, and D Banks. *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*. en. 2010. URL: https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=906762.

[14] R. Berger. *The Undecidability of the Domino Problem*. Memoirs ; No 1/66. American Mathematical Society, 1966. ISBN: 9780821812662. URL: https://books.google.ie/books?id=Hq0YuwEACAAJ.

[15] Daniel J. Bernstein. "A Non-Iterative 2-Adic Statement of the 3N + 1 Conjecture". In: *Proceedings of the American Mathematical Society* 121.2 (1994), pp. 405–408. ISSN: 00029939, 10886826. URL: http://www.jstor.org/stable/2160415 (visited on 09/03/2022).

[16] Jacek Blazewicz and Alberto Pettorossi. "Some properties of binary sequences useful for proving Collatz's conjecture". In: *Foundations of Control Engineering* 8 (Jan. 1983).

[17] Corrado Böhm and Giovanna Sontacchi. "On the existence of cycles of given length in integer sequences like $x\_n+1 = x\_n/2$ if x_n even, and $x\_n+1 = 3x\_n+1$". en. In: *Atti della Accademia Nazionale dei Lincei. Classe di Scienze Fisiche, Matematiche e Naturali. Rendiconti* 64 (1978), pp. 260–264. URL: http://dml.mathdoc.fr/item/RLINA_1978_8_64_3_260_0.

[18] A. H. Brady. "The Busy Beaver Game and the Meaning of Life". In: *A Half-Century Survey on The Universal Turing Machine*. Berlin, Germany: Oxford University Press, Inc., 1988, pp. 259–277. ISBN: 0198537417.

[19] Allen H. Brady. "The Determination of the Value of Rado's Noncomputable Function $|sum(k)$ for Four-State Turing Machines". In: *Mathematics of Computation* 40.162 (1983), pp. 647–665. ISSN: 00255718, 10886842. URL: http://www.jstor.org/stable/2007539 (visited on 10/25/2022).

[20] Tatiana Brailovskaya, Gokul Gowri, Sean Yu, and Erik Winfree. "Reversible Computation Using Swap Reactions on a Surface". In: *DNA Computing and Molecular Programming*. Ed. by Chris Thachuk and Yan Liu. Cham: Springer International Publishing, 2019, pp. 174–196. ISBN: 978-3-030-26807-7.

[21] G. Brassard. "A note on the complexity of cryptography (Corresp.)" In: *IEEE Transactions on Information Theory* 25.2 (1979), pp. 232–233. DOI: 10.1109/TIT.1979.1056010.

[22] M. Bruschi. *Two Cellular Automata for the 3x+1 Map*. 2005. eprint: arXiv:nlin/0502061.

[23] Hieu Bui, Vincent Miao, Sudhanshu Garg, Reem Mokhtar, Tianqi Song, and John Reif. "Design and Analysis of Localized DNA Hybridization Chain Reactions". In: *Small* 13.12 (2017), p. 1602983.

[24] Hieu Bui, Craig Onodera, Carson Kidwell, YerPeng Tan, Elton Graugnard, Wan Kuang, Jeunghoon Lee, William B Knowlton, Bernard Yurke, and William L Hughes. "Programmable periodicity of quantum dot arrays with DNA origami nanotubes". In: *Nano letters* 10.9 (2010), pp. 3367–3372.

[25] Hieu Bui, Shalin Shah, Reem Mokhtar, Tianqi Song, Sudhanshu Garg, and John Reif. "Localized DNA Hybridization Chain Reactions on DNA Origami". In: *ACS Nano* 12.2 (2018), pp. 1146–1155.

[26] S Buss, S Cook, Arvind Gupta, and Vijaya Ramachandran. "An optimal parallel algorithm for formula evaluation". In: *SIAM Journal on Computing* 21.4 (1992), pp. 755–780.

[27] Samuel R Buss. "The Boolean formula value problem is in ALOGTIME". In: *STOC: Proceedings of the nineteenth annual ACM symposium on Theory of computing*. 1987, pp. 123–131.

[28] Javier Cabello-Garcia, Wooli Bae, Guy-Bart V Stan, and Thomas E Ouldridge. "Handhold-mediated strand displacement: a nucleic acid based mechanism for generating far-from-equilibrium assemblies through templated reactions". In: *ACS nano* 15.2 (2021), pp. 3272–3283.

[29] Jin-Yi Cai. *Lecture 13: Circuit Complexity*. CS 810: Introduction to Complexity Theory. [Online; accessed 13-September-2022]. Apr. 2003. URL: https://pages.cs.wisc.edu/~jyc/02-810notes/lecture13.pdf.

[30] Angel A Cantu, Austin Luchsinger, Robert Schweller, and Tim Wylie. "Covert computation in self-assembled circuits". In: *Algorithmica* 83.2 (2021). arXiv preprint: arXiv:1908.06068, pp. 531–552.

[31] Jose Capco. *Odd Collatz Sequence and Binary Representations*. RISC Report Series. Altenberger Straße 69, 4040 Linz, Austria: Research Institute for Symbolic Computation (RISC), Johannes Kepler University Linz, 2015.

[32] Xavier Caruso. *Computations with p-adic numbers*. 2017. DOI: 10.48550/ARXIV.1701.06794. URL: https://arxiv.org/abs/1701.06794.

[33] Cameron T. Chalk, Jacob Hendricks, Matthew J. Patitz, and Michael Sharp. "Thermodynamically Favorable Computation via Tile Self-assembly". In: *Unconventional Computation and Natural Computation - 17th International Conference, UCNC 2018, Fontainebleau, France, June 25-29, 2018, Proceedings*. Ed. by Susan Stepney and Sergey Verlan. Vol. 10867. Lecture Notes in Computer Science. Springer, 2018, pp. 16–31. DOI: 10.1007/978-3-319-92435-9_2. URL: https://doi.org/10.1007/978-3-319-92435-9_2.

[34] A. R. Chandrasekaran, O. Levchenko, D. S. Patel, M. MacIsaac, and K. Halvorsen. "Addressable configurations of DNA nanostructures for rewritable memory". In: *Nucleic Acids Res* 45.19 (2017), pp. 11459–11465.

[35] Jie Chao, Jianbang Wang, Fei Wang, Xiangyuan Ouyang, Enzo Kopperger, Huajie Liu, Qian Li, Jiye Shi, Lihua Wang, Jun Hu, Lianhui Wang, Wei Huang, Friedrich C. Simmel, and Chunhai Fan. "Solving mazes with single-molecule DNA navigators". In: *Nature Materials* 18.3 (2019), pp. 273–279.

[36] Gourab Chatterjee, Neil Dalchau, Richard A. Muscat, Andrew Phillips, and Georg Seelig. "A spatially localized architecture for fast and modular DNA computing". In: *Nature Nanotechnology* 12.9 (2017), pp. 920–927.

[37] Ho-Lin Chen, Rebecca Schulman, Ashish Goel, and Erik Winfree. "Reducing facet nucleation during algorithmic self-assembly". In: *Nano Letters* 7.9 (2007), pp. 2913–2919.

[38] Xi Chen. "Expanding the rule set of DNA circuitry with associative toehold activation". In: *Journal of the American Chemical Society* 134.1 (2012), pp. 263–271.

[39] Andrew Chiu, George Davida, and Bruce Litow. "Division in logspace-uniform NC^1". en. In: *RAIRO - Theoretical Informatics and Applications - Informatique Théorique et Applications* 35.3 (2001), pp. 259–275. URL: http://www.numdam.org/item/ITA_2001__35_3_259_0/.

[40] Marek Chrobak and Thomas H Payne. "A linear-time algorithm for drawing a planar graph on a grid". In: *Information Processing Letters* 54.4 (1995), pp. 241–246.

[41] Samuel Clamons, Lulu Qian, and Erik Winfree. "Programming and simulating chemical reaction networks on a surface". In: *Journal of The Royal Society Interface* 17.166 (2020), p. 20190790.

[42] Livio Colussi. "The convergence classes of Collatz function". In: *Theor. Comput. Sci.* 412 (2011), pp. 5409–5419. URL: https://doi.org/10.1016/j.tcs.2011.05.056.

[43] Keith Conrad. "The p-adic expansion of rational numbers". In: (). https://kconrad.math.uconn.edu/blurbs/gradnumthy/rationalsinQp.pdf.

[44] Damien Woods Constantine Evans David Doty. "Growth dynamics: Precisely controlled self-assembly order of DNA tile nanostructures". In: (). In submission to DNA28, Track B.

[45] J. H. Conway. "FRACTRAN: A Simple Universal Programming Language for Arithmetic". In: *Open Problems in Communication and Computation*. Ed. by Thomas M. Cover and B. Gopinath. New York, NY: Springer New York, 1987, pp. 4–26. ISBN: 978-1-4612-4808-8. DOI: 10.1007/978-1-4612-4808-8_2. URL: https://doi.org/10.1007/978-1-4612-4808-8_2.

[46] J.H Conway. "Unpredictable iterations". In: *Number Theory Conference* (1972).

[47] Matthew Cook. "Universality in Elementary Cellular Automata". In: *Complex Systems* 15 (2004).

[48] Matthew Cook. "Universality in Elementary Cellular Automata". In: *Complex Systems* 15 (2004).

[49] Matthew Cook, Tristan Stérin, and Damien Woods. "Small Tile Sets That Compute While Solving Mazes". In: *27th International Conference on DNA Computing and Molecular Programming, DNA 27, September 13-16, 2021, Oxford, UK (Virtual Conference)*. Ed. by Matthew R. Lakin and Petr Sulc. Vol. 205. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 8:1–8:20. DOI: 10.4230/LIPIcs.DNA.27.8. URL: https://doi.org/10.4230/LIPIcs.DNA.27.8.

[50] H. S. M. Coxeter. *Cyclic sequences and frieze patterns: The Fourth Felix Behrend Memorial Lecture, Vinculum* **8**. 1971.

[51] Karel Culik. "An aperiodic set of 13 Wang tiles". In: *Discrete Mathematics* 160.1 (1996), pp. 245–251. ISSN: 0012-365X. DOI: https://doi.org/10.1016/S0012-365X(96)00118-5. URL: https://www.sciencedirect.com/science/article/pii/S0012365X96001185.

[52] Neil Dalchau, Harish Chandran, Nikhil Gopalkrishnan, Andrew Phillips, and John Reif. "Probabilistic analysis of localized DNA hybridization circuits". In: *ACS synthetic biology* 4.8 (2015), pp. 898–913.

[53] Frits Dannenberg, Marta Kwiatkowska, Chris Thachuk, and Andrew J. Turberfield. "DNA Walker Circuits: Computational Potential, Design, and Verification". In: *DNA Computing and Molecular Programming*. Ed. by David Soloveichik and Bernard Yurke. Cham: Springer International Publishing, 2013, pp. 31–45. ISBN: 978-3-319-01928-4.

[54] Erik D. Demaine, Martin L. Demaine, Sándor P. Fekete, Matthew J. Patitz, Robert T. Schweller, Andrew Winslow, and Damien Woods. "One Tile to Rule Them All: Simulating Any Tile Assembly System with a Single Universal Tile". In: *ICALP: Proceedings of the 41st International Colloquium on Automata, Languages, and Programming*. Vol. 8572. LNCS. Arxiv preprint: arXiv:1212.4756. Springer, 2014, pp. 368–379.

[55] Erik D. Demaine, Matthew J. Patitz, Trent A. Rogers, Robert T. Schweller, Scott M. Summers, and Damien Woods. "The two-handed tile assembly model is not intrinsically universal". In: *ICALP: Proceedings of the 40th International Colloquium on Automata, Languages, and Programming*. Vol. 7965. LNCS. Arxiv preprint: arXiv:1306.6710. Springer, July 2013, pp. 400–412.

[56] Vassil S. Dimitrov and Everett W. Howe. *Powers of 3 with few nonzero bits and a conjecture of Erdős*. https://arxiv.org/abs/2105.06440. 2021. arXiv: 2105.06440 [math.NT].

[57] David Doty. "Theory of Algorithmic Self-Assembly". In: *Communications of the ACM* 55.12 (2012), pp. 78–88.

[58] David Doty, Benjamin L Lee, and Tristan Stérin. "scadnano: A browser-based, scriptable tool for designing DNA nanostructures". In: *DNA 2020: Proceedings of the 26th International Meeting on DNA Computing and Molecular Programming*. Ed. by Cody Geary and Matthew J. Patitz. Vol. 174. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020, 9:1–9:17. ISBN: 978-3-95977-163-4. DOI: 10.4230/LIPIcs.DNA.2020.9. URL: https://drops.dagstuhl.de/opus/volltexte/2020/12962.

[59] David Doty, Jack H. Lutz, Matthew J. Patitz, Robert T. Schweller, Scott M. Summers, and Damien Woods. "The tile assembly model is intrinsically universal". In: *FOCS: Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science*. Arxiv preprint: arXiv:1111.3097. New Brunswick, New Jersey: IEEE, Oct. 2012, pp. 439–446.

[60] David Doty, Trent A Rogers, David Soloveichik, Chris Thachuk, and Damien Woods. "Thermodynamic binding networks". In: *DNA23: The 23rd International Conference on DNA Computing and Molecular Programming*. Vol. 10467. LNCS. Springer. 2017, pp. 249–266.

[61] Shawn M. Douglas, Hendrik Dietz, Tim Liedl, Björn Högberg, Franziska Graf, and William M. Shih. "Self-assembly of DNA into nanoscale three-dimensional shapes". In: *Nature* 459.7245 (May 2009), pp. 414–418. DOI: 10.1038/nature08016. URL: https://doi.org/10.1038/nature08016.

[62] Artūras Dubickas. "On the powers of 3/2 and other rational numbers". In: *Mathematische Nachrichten* 281.7 (2008), pp. 951–958. DOI: https://doi.org/10.1002/mana.200510651. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/mana.200510651. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/mana.200510651.

[63] Taylor Dupuy and David E. Weirich. "Bits of 3^n in binary, Wieferich primes and a conjecture of Erdős". In: *Journal of Number Theory* 158 (2016), pp. 268–280. ISSN: 0022-314X. DOI: https://doi.org/10.1016/j.jnt.2015.05.022. URL: https://www.sciencedirect.com/science/article/pii/S0022314X15002139.

[64] Shalom Eliahou. "The 3x+1 problem: new lower bounds on nontrivial cycle lengths". In: *Discrete Mathematics* 118.1 (1993), pp. 45–56. ISSN: 0012-365X. DOI: https://doi.org/10.1016/0012-365X(93)90052-U. URL: https://www.sciencedirect.com/science/article/pii/0012365X9390052U.

[65] Paul Erdös. "Some Unconventional Problems in Number Theory". In: *Mathematics Magazine* 52.2 (1979), pp. 67–70. URL: https://doi.org/10.1080/0025570X.1979.11976756.

[66] Constantine Evans. "Crystals that count! Physical principles and experimental investigations of DNA tile self-assembly". PhD thesis. Caltech, 2014.

[67] C.J Everett. "Iteration of the number-theoretic function f(2n) = n, f(2n + 1) = 3n + 2". In: *Advances in Mathematics* 25.1 (1977), pp. 42–45. ISSN: 0001-8708. DOI: https://doi.org/10.1016/0001-8708(77)90087-1. URL: https://www.sciencedirect.com/science/article/pii/0001870877900871.

[68] Leopold Flatto, Jeffrey Lagarias, Murray Hill, Andrew Pollington, and Ut Provo. "On the range of fractional parts (p/q) n". In: *Acta Arithmetica* 2 (Jan. 1995). DOI: 10.4064/aa-70-2-125-147.

[69] Mark E Fornace, Nicholas J Porubsky, and Niles A Pierce. "A unified dynamic programming framework for the analysis of interacting nucleic acid strands: Enhanced models, scalability, and speed". In: *ACS Synthetic Biology* 9.10 (2020), pp. 2665–2678.

[70] Kenichi Fujibayashi, Rizal Hariadi, Sung Ha Park, Erik Winfree, and Satoshi Murata. "Toward Reliable Algorithmic Self-Assembly of DNA Tiles: A Fixed-Width Cellular Automaton Pattern". In: *Nano Letters* 8.7 (2008), pp. 1791–1797.

[71] Anthony J Genot, Teruo Fujii, and Yannick Rondelez. "Scaling down DNA circuits with competitive neural networks". In: *Journal of The Royal Society Interface* 10.85 (2013), p. 20130212.

[72] Jeffrey R. Goodwin. *The* $3x + 1$ *Problem and Integer Representations*. 2015. eprint: arXiv:1504.03040.

[73] Raymond Greenlaw, H James Hoover, and Walter L Ruzzo. *Limits to parallel computation: P-completeness theory*. Oxford University Press, 1995.

[74] Hongzhou Gu, Jie Chao, Shou-Jun Xiao, and Nadrian C Seeman. "A proximity-based programmable DNA nanoscale assembly line". In: *Nature* 465.7295 (2010), pp. 202–205.

[75] Aurore Guillevic and François Morain. "Discrete Logarithms". In: *Guide to pairing-based cryptography*. Ed. by Nadia El Mrabet and Marc Joye. 2nd version: fix some font bugs and typos (minor modifications). CRC Press - Taylor and Francis Group, Dec. 2016, p. 42. URL: https://hal.inria.fr/hal-01420485.

[76] Richard K. Guy. "Don't Try to Solve These Problems!" In: *The American Mathematical Monthly* 90.1 (1983), pp. 35–41. ISSN: 00029890, 19300972. URL: http://www.jstor.org/stable/2975688 (visited on 07/24/2022).

[77] William Hesse, Eric Allender, and David A Mix Barrington. "Uniform constant-depth threshold circuits for division and iterated multiplication". In: *Journal of Computer and System Sciences* 65.4 (2002), pp. 695–716.

[78] Patrick Chisan Hew. "Working in binary protects the repetends of $1/3^{\mathrm{h}}$: Comment on Colussi's 'The convergence classes of Collatz function'". In: *Theor. Comput. Sci.* 618 (2016), pp. 135–141. URL: https://doi.org/10.1016/j.tcs.2015.12.033.

[79] Lubin (https://math.stackexchange.com/users/17760/lubin). *Is* $\mathbb{Q}_r$ *algebraically isomorphic to* $\mathbb{Q}_s$ *while* r *and* s *denote different primes?* Mathematics Stack Exchange. URL:https://math.stackexchange.com/q/95128 (version: 2011-12-30). eprint: https://math.stackexchange.com/q/95128. URL: https://math.stackexchange.com/q/95128.

[80] roydiptajit (https://math.stackexchange.com/users/780430/roydiptajit). *Can we have* n*-adic integers?* Mathematics Stack Exchange. URL:https://math.stackexchange.com/q/3946639 (version: 2020-12-13). eprint: https://math.stackexchange.com/q/3946639. URL: https://math.stackexchange.com/q/3946639.

[81] EuYu (https://math.stackexchange.com/users/9246/euyu). *2 is a primitive root mod* 3^h *for any positive integer* h. Mathematics Stack Exchange. URL:https://math.stackexchange.com/q/594800 (version: 2017-04-13). eprint: https://math.stackexchange.com/q/594800. URL: https://math.stackexchange.com/q/594800.

[82] J Håstad. "Almost Optimal Lower Bounds for Small Depth Circuits". In: *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*. STOC '86. Berkeley, California, USA: Association for Computing Machinery, 1986, pp. 6–20. ISBN: 0897911938. DOI: 10.1145/12130.12132. URL: https://doi.org/10.1145/12130.12132.

[83] Neil Immerman. *Descriptive Complexity*. Ed. by David Gries and Fres B. Schneider. Springer, 1999.

[84] Emmanuel Jeandel and Michaël Rao. "An aperiodic set of 11 Wang tiles". In: *Advances in Combinatorics* (2021). DOI: 10.19086/aic.18614. URL: https://doi.org/10.19086%2Faic.18614.

[85] Jarkko Kari. "A small aperiodic set of Wang tiles". In: *Discrete Mathematics* 160.1 (1996), pp. 259–264. ISSN: 0012-365X. DOI: https://doi.org/10.1016/0012-365X(95)00120-L. URL: https://www.sciencedirect.com/science/article/pii/0012365X9500120L.

[86] Jarkko Kari. "Cellular Automata, the Collatz Conjecture and Powers of 3/2". In: *Developments in Language Theory*. Ed. by Hsu-Chun Yen and Oscar H. Ibarra. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 40–49. ISBN: 978-3-642-31653-1.

[87] Jarkko Kari and Johan Kopra. "Cellular automata and powers of p/q". In: *RAIRO Theor. Informatics Appl.* 51.4 (2017), pp. 191–204. DOI: 10.1051/ita/2017014. URL: https://doi.org/10.1051/ita/2017014.

[88] Neeraj Kayal. *E0 309: Topics in Complexity Theory: Lecture 5*. [Online; accessed 21-September-2022]. Feb. 2015. URL: https://www.csa.iisc.ac.in/~chandan/courses/arithmetic_circuits/notes/lec5.pdf.

[89] František Kaščák. "Small universal one-state linear operator algorithm". In: *Mathematical Foundations of Computer Science 1992*. Ed. by Ivan M. Havel and Václav Koubek. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 327–335. ISBN: 978-3-540-47291-9.

[90] Pascal Koiran and Cristopher Moore. "Closed-form analytic maps in one and two dimensions can simulate universal Turing machines". In: *Theoretical Computer Science* 210.1 (Jan. 1999), pp. 217–223. DOI: 10.1016/s0304-3975(98)00117-0. URL: https://doi.org/10.1016/s0304-3975(98)00117-0.

[91] Johan Kopra. "Cellular automata with complicated dynamics". PhD thesis. Univeristy of Turku, 2019.

[92] Johan Kopra. *Multiplication cubes and multiplication automata*. 2022. eprint: arXiv:2211.15293.

[93] Ivan Korec. "The $3x + 1$ problem, generalized Pascal triangles and cellular automata". eng. In: *Mathematica Slovaca* 42.5 (1992). http://eudml.org/doc/32424, pp. 547–563.

[94] Ivan Korec and Stefan Znám. "A Note on the 3x + 1 Problem". In: *American Mathematical Monthly* 94 (1987), pp. 771–772.

[95] Stuart A. Kurtz and Janos Simon. "The Undecidability of the Generalized Collatz Problem". In: *TAMC 2007*. 2007, pp. 542–553. URL: https://doi.org/10.1007/978-3-540-72504-6_49.

[96] J.C. Lagarias. *The Ultimate Challenge: The 3x+1 Problem*. Monograph Bks. American Mathematical Society, 2010. ISBN: 9780821849408.

[97] Jeffrey C. Lagarias. "Ternary expansions of powers of 2". In: *J. Lond. Math. Soc. (2)* 79.3 (2009), pp. 562–588. ISSN: 0024-6107. DOI: 10.1112/jlms/jdn080. URL: https://doi.org/10.1112/jlms/jdn080.

[98] Jeffrey C. Lagarias. "Ternary expansions of powers of 2". In: *Journal of the London Mathematical Society* 79.3 (2009), pp. 562–588. DOI: https://doi.org/10.1112/jlms/jdn080.

[99] Jeffrey C. Lagarias. "The $3x + 1$ Problem and Its Generalizations". In: *The American Mathematical Monthly* 92.1 (1985), pp. 3–23. ISSN: 00029890, 19300972. URL: http://www.jstor.org/stable/2322189.

[100] Jeffrey C. Lagarias. *The $3x + 1$ problem: An annotated bibliography (1963–1999) (sorted by author)*. 2003. eprint: arXiv:math/0309224.

[101] Jeffrey C. Lagarias. *The $3x + 1$ Problem: An Annotated Bibliography, II (2000-2009)*. 2006. eprint: arXiv:math/0608208.

[102] Jeffrey C. Lagarias. "The 3x+1 Problem: An Overview". In: (2021). DOI: 10.48550/ARXIV.2111.02635. URL: https://arxiv.org/abs/2111.02635.

[103] Matthew R Lakin, Rasmus Petersen, Kathryn E Gray, and Andrew Phillips. "Abstract modelling of tethered DNA circuits". In: *International Workshop on DNA-Based Computers*. Springer. 2014, pp. 132–147.

[104] Suping Li, Qiao Jiang, Shaoli Liu, Yinlong Zhang, Yanhua Tian, Chen Song, Jing Wang, Yiguo Zou, Gregory J Anderson, Jing-Yan Han, Yung Chang, Yan Liu, Chen Zhang, Liang Chen, Guangbiao Zhou, Guangjun Nie, Hao Yan, Baoquan Ding, and Yuliang Zhao. "A DNA nanorobot functions as a cancer therapeutic in response to a molecular trigger in vivo". In: *Nature Biotechnology* 36.3 (Feb. 2018), pp. 258–264. DOI: 10.1038/nbt.4071. URL: https://doi.org/10.1038/nbt.4071.

[105] Wenyan Liu, Hong Zhong, Risheng Wang, and Nadrian C Seeman. "Crystalline two-dimensional DNA-origami arrays". In: *Angewandte Chemie International Edition* 50.1 (2011), pp. 264–267.

[106] Josefina López and Peter Stoll. "The 2-Adic, Binary and Decimal Periods of 1/3k Approach Full Complexity for Increasing k". In: *Integers [electronic only]* 5 (Jan. 2012). DOI: 10.1515/integers-2012-0013.

[107] K. Mahler. "An unsolved problem on the powers of 3/2". In: *Journal of the Australian Mathematical Society* 8.2 (1968), pp. 313–321. DOI: 10.1017/S1446788700005371.

[108] Chengde Mao, Thomas H. LaBean, John H. Reif, and Nadrian C. Seeman. "Logical computation using algorithmic self-assembly of DNA triple-crossover molecules". In: *Nature* 407.6803 (2000), pp. 493–496.

[109] H. Marxen and Jürgen Buntrock. "Attacking the Busy Beaver 5". In: *Bull. EATCS* 40 (1990), pp. 247–251.

[110] Pascal Michel. *Simulation of the Collatz 3x+1 function by Turing machines*. Tech. rep. https://arxiv.org/abs/1409.7322. 2014. arXiv: 1409.7322 [math.LO].

[111] Pascal Michel. *The Busy Beaver Competition: a historical survey*. Tech. rep. https://arxiv.org/abs/0906.3749v6. Sept. 2019. arXiv: 0906.3749 [math.LO].

[112] Pascal Michel. *The Busy Beaver Competitions*. Tech. rep. https://webusers.imj-prg.fr/~pascal.michel/bbc.html.

[113] Pascal Michel and Pavel Kropitz. *New bound on BB(6,2)*. 2022. URL: https://groups.google.com/g/busy-beaver-discuss/c/-zjeW6y8ER4/m/Qv2qfJ5-AAAJ.

[114] Pascal Michel and Maurice Margenstern. "Generalised 3x+1 functions and the theory of computation". In: *The Ultimate Challenge: the 3x+1 problem*. American Mathematical Society, 2011, pp. 105–125.

[115] Marvin L. Minsky. *Computation: Finite and Infinite Machines*. USA: Prentice-Hall, Inc., 1967. ISBN: 0131655639.

[116] David A Mix Barrington and Alexis Maciel. *Lecture 5: Boolean Formulas, NC^1, and M-Programs*. IAS/PCMI Summer Session 2000 Clay Mathematics Undergraduate Program Advanced Course on Computational Complexity. [Online; accessed 13-September-2022]. July 2000. URL: https://lin-web.clarkson.edu/~alexis/PCMI/Notes/lectureA05.pdf.

[117] Kenneth Monks. "The sufficiency of arithmetic progressions for the $3x+1$ conjecture". In: *Proceedings of the American Mathematical Society* 134 (Oct. 2006). DOI: 10.2307/4098142.

[118] Cristopher Moore and Stephan Mertens. *The nature of computation*. Oxford University Press, 2011.

[119] Niall Murphy and Damien Woods. "AND and/or OR: Uniform polynomial-size circuits". In: *MCU 2013: Machines, Computations and Universality. Electronic Proceedings in Theoretical Computer Science (EPTCS)*. Vol. 128. arXiv:1212.3282v2. Zürich, Switzerland, 2012, pp. 150–166.

[120] Turlough Neary and Damien Woods. “Four small universal Turing machines”. In: *Fundamenta Informaticae* 91.1 (2009), pp. 123–144.

[121] Turlough Neary and Damien Woods. “P-completeness of cellular automaton Rule 110”. In: *ICALP: International Colloquium on Automata, Languages, and Programming*. Vol. 4051, part 1. LNCS. Springer. 2006, pp. 132–143.

[122] Turlough Neary and Damien Woods. “Small weakly universal Turing machines”. In: *International Symposium on Fundamentals of Computation Theory*. Springer. 2009, pp. 262–273.

[123] Tosan Omabegho, Ruojie Sha, and Nadrian C Seeman. “A bipedal DNA Brownian motor with coordinated legs”. In: *Science* 324.5923 (2009), pp. 67–71.

[124] Günther Pardatscher, Dan Bracha, Shirley S Daube, Ohad Vonshak, Friedrich C Simmel, and Roy H Bar-Ziv. “DNA condensation in one dimension”. In: *Nature nanotechnology* 11.12 (2016), pp. 1076–1081.

[125] Matthew J. Patitz. “An introduction to tile-based self-assembly and a survey of recent results”. In: *Natural Computing* 13.2 (2014), pp. 195–224. URL: https://doi.org/10.1007/s11047-013-9379-4.

[126] Yvette Perrin. “A journey throughout the history of p-adic numbers”. In: Jan. 2018, pp. 261–272. ISBN: 9781470434915. DOI: 10.1090/conm/704/14171.

[127] S. Pohlig and M. Hellman. “An improved algorithm for computing logarithms overGF(p)and its cryptographic significance (Corresp.)” In: *IEEE Transactions on Information Theory* 24.1 (1978), pp. 106–110. DOI: 10.1109/TIT.1978.1055817.

[128] Lulu Qian and Erik Winfree. “A simple DNA gate motif for synthesizing large-scale circuits”. en. In: *J R Soc Interface* 8.62 (Feb. 2011), pp. 1281–1297.

[129] Lulu Qian and Erik Winfree. “Parallel and Scalable Computation and Spatial Dynamics with DNA-Based Chemical Reaction Networks on a Surface”. In: *DNA Computing and Molecular Programming*. Ed. by Satoshi Murata and Satoshi Kobayashi. Cham: Springer International Publishing, 2014, pp. 114–131. ISBN: 978-3-319-11295-4.

[130] Lulu Qian and Erik Winfree. “Scaling up digital circuit computation with DNA strand displacement cascades”. In: *science* 332.6034 (2011), pp. 1196–1201.

[131] Tibor Radó. “On non-computable functions”. In: *Bell System Technical Journal* 41.3 (1962). https://archive.org/details/bstj41-3-877/mode/2up, pp. 877–884.

[132] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. “Zero-Shot Text-to-Image Generation”. In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, 2021, pp. 8821–8831. URL: https://proceedings.mlr.press/v139/ramesh21a.html.

[133] John H. Reif and Sudheer Sahu. “Autonomous programmable DNA nanorobotic devices using DNAzymes”. In: *Theoretical Computer Science* 410.15 (2009). Aspects of Molecular Self-Assembly, pp. 1428–1439. ISSN: 0304-3975.

[134] Yuri Rogozhin. “Small universal Turing machines”. In: *Theoretical Computer Science* 168.2 (1996), pp. 215–240.

[135] Paul W K Rothemund, Nick Papadakis, and Erik Winfree. “Algorithmic self-assembly of DNA Sierpinski triangles”. In: *PLoS Biology* 2.12 (2004), e424.

[136] Paul W K Rothemund and Erik Winfree. "The program-size complexity of self-assembled squares". In: *STOC: Proceedings of the thirty-second annual ACM symposium on Theory of computing*. ACM. 2000, pp. 459–468.

[137] Paul WK Rothemund. "Folding DNA to create nanoscale shapes and patterns". In: *Nature* 440.7082 (2006), pp. 297–302.

[138] Olivier Rozier. "Démonstration de l'absence de cycles d'une certaine forme pour le problème de Syracuse". In: *Singularité* 1 (1990), pp. 9–12.

[139] Olivier Rozier. "Parity Sequences of the 3x+1 Map on the 2-adic Integers and Euclidean Embedding". In: *Integers* 19 (2019), A8.

[140] Olivier Rozier. *Upper bound on the elements of a Collatz cycle*. Mathematics Stack Exchange. `https://math.stackexchange.com/q/4526766` (version: 2022-09-07).

[141] Sudheer Sahu, Thomas H LaBean, and John H Reif. "A DNA nanotransport device powered by polymerase phi29". In: *Nano letters* 8.11 (2008), pp. 3870—3878. ISSN: 1530-6984.

[142] John SantaLucia Jr and Donald Hicks. "The thermodynamics of DNA structural motifs". In: *Annu. Rev. Biophys. Biomol. Struct.* 33 (2004), pp. 415–440.

[143] Rebecca Schulman and Erik Winfree. "Programmable control of nucleation for algorithmic self-assembly". In: *SIAM Journal on Computing* 39.4 (2009), pp. 1581–1616.

[144] Rebecca Schulman and Erik Winfree. "Synthesis of crystals with a programmable kinetic barrier to nucleation". In: *Proceedings of the National Academy of Sciences* 104.39 (2007), pp. 15236–15241.

[145] Rebecca Schulman, Bernard Yurke, and Erik Winfree. "Robust self-replication of combinatorial information via crystal growth and scission". In: *Proceedings of the National Academy of Sciences* 109.17 (2012), pp. 6405–6410.

[146] Jeffrey Shallit and David A. Wilson. "The "3x + 1" Problem and Finite Automata". In: *Bulletin of the EATCS* 46 (1992), pp. 182–185.

[147] William B Sherman and Nadrian C Seeman. "A precisely controlled DNA biped walking device". In: *Nano letters* 4.7 (2004), pp. 1203–1207.

[148] Peter W. Shor. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer". In: *SIAM Journal on Computing* 26.5 (1997), pp. 1484–1509. URL: `https://doi.org/10.1137/S0097539795293172`.

[149] David Soloveichik and Erik Winfree. "Complexity of Self-Assembled Shapes". In: *SIAM Journal on Computing* 36.6 (2007), pp. 1544–1569.

[150] Tianqi Song, Nikhil Gopalkrishnan, Abeer Eshra, Sudhanshu Garg, Reem Mokhtar, Hieu Bui, Harish Chandran, and John Reif. "Improving the performance of DNA strand displacement circuits by shadow cancellation". In: *ACS nano* 12.11 (2018), pp. 11689–11697.

[151] Tianqi Song, Shalin Shah, Hieu Bui, Sudhanshu Garg, Abeer Eshra, Daniel Fu, Ming Yang, Reem Mokhtar, and John Reif. "Programming DNA-Based Biomolecular Reaction Networks on Cancer Cell Membranes". In: *Journal of the American Chemical Society* 141.42 (2019), pp. 16539–16543.

[152] Darko Stefanovic. "Maze Exploration with Molecular-Scale Walkers". In: *Theory and Practice of Natural Computing*. Ed. by Adrian-Horia Dediu, Carlos Martín-Vide, and Bianca Truthe. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 216–226. ISBN: 978-3-642-33860-1.

[153] Ray P. Steiner. "A Theorem on the Syracuse Problem". In: *Proc. 7-th Manitoba Conference on Numerical Mathematics and Computing*. (MR 80g:10003). Congressus Numerantium XX, Utilitas Math.: Winnipeg, Manitoba, 1978, pp. 553–559.

[154] Tristan Stérin. "Binary expression of ancestors in the Collatz graph". In: *Reachability Problems*. Ed. by Sylvain Schmitz and Igor Potapov. Springer International Publishing, 2020, pp. 115–130.

[155] Tristan Stérin and Damien Woods. *The Collatz process embeds a base conversion algorithm*. arXiv version 4: arXiv:2007.06979v4 [cs.DM]. 2020.

[156] Tristan Stérin and Damien Woods. "The Collatz Process Embeds a Base Conversion Algorithm". In: *RP2020: 14th International Conference on Reachability Problems*. Ed. by Sylvain Schmitz and Igor Potapov. Vol. 12448. LNCS. Springer, 2020, pp. 131–147. DOI: 10.1007/978-3-030-61739-4_9.

[157] Douglas R. Stinson. *Cryptography: Theory and Practice*. USA: CRC Press, Inc., 1995. ISBN: 0849385210.

[158] Tristan Stérin and Damien Woods. *On the hardness of knowing busy beaver values BB(15) and BB(5,4)*. 2021. DOI: 10.48550/ARXIV.2107.12475. URL: https://arxiv.org/abs/2107.12475.

[159] Terence Tao. "Almost all orbits of the Collatz map attain almost bounded values". In: *Forum of Mathematics, Pi* 10 (2022), e12. DOI: 10.1017/fmp.2022.8.

[160] Terence Tao. *The Collatz conjecture, Littlewood-Offord theory, and powers of 2 and 3*. Last accessed June 30th 2021. https://terrytao.wordpress.com/2011/08/25/the-collatz-conjecture-littlewood-offord-theory-and-powers-of-2-and-3/. URL: https://terrytao.wordpress.com/2011/08/25/the-collatz-conjecture-littlewood-offord-theory-and-powers-of-2-and-3/.

[161] Riho Terras. "A stopping time problem on the positive integers". eng. In: *Acta Arithmetica* 30.3 (1976), pp. 241–252. URL: http://eudml.org/doc/205476.

[162] Riho Terras. "On the existence of a density". eng. In: *Acta Arithmetica* 35.1 (1979), pp. 101–102. URL: http://eudml.org/doc/205619.

[163] Anupama J. Thubagere, Wei Li, Robert F. Johnson, Zibo Chen, Shayan Doroudi, Yae Lim Lee, Gregory Izatt, Sarah Wittman, Niranjan Srinivas, Damien Woods, Erik Winfree, and Lulu Qian. "A cargo-sorting DNA robot". In: *Science* 357.6356 (2017). ISSN: 0036-8075. DOI: 10.1126/science.aan6558. eprint: https://science.sciencemag.org/content/357/6356/eaan6558.full.pdf. URL: https://science.sciencemag.org/content/357/6356/eaan6558.

[164] Anupama J. Thubagere, Wei Li, Robert F. Johnson, Zibo Chen, Shayan Doroudi, Yae Lim Lee, Gregory Izatt, Sarah Wittman, Niranjan Srinivas, Damien Woods, Erik Winfree, and Lulu Qian. "A cargo-sorting DNA robot". In: *Science* 357.6356 (2017).

[165] Grigory Tikhomirov, Philip Petersen, and Lulu Qian. "Fractal assembly of micrometre-scale DNA origami arrays with arbitrary patterns". In: *Nature* 552.7683 (2017), pp. 67–71.

[166] Grigory Tikhomirov, Philip Petersen, and Lulu Qian. "Fractal assembly of micrometre-scale DNA origami arrays with arbitrary patterns". In: *Nature* 552.7683 (Dec. 2017), pp. 67–71. DOI: 10.1038/nature24655. URL: https://doi.org/10.1038/nature24655.

[167] Grigory Tikhomirov, Philip Petersen, and Lulu Qian. "Programmable disorder in random DNA tilings". In: *Nature nanotechnology* 12.3 (2017), p. 251.

[168] Boya Wang, Chris Thachuk, Andrew D Ellington, Erik Winfree, and David Soloveichik. "Effective design principles for leakless strand displacement systems". In: *Proceedings of the National Academy of Sciences (PNAS)* 115.52 (2018), E12182–E12191.

[169] Boya Wang, Chris Thachuk, and David Soloveichik. "Speed and correctness guarantees for programmable enthalpy-neutral DNA reactions". In: *bioRxiv* (2022). DOI: 10.1101/2022.04.13.488226. eprint: https://www.biorxiv.org/content/early/2022/05/16/2022.04.13.488226.full.pdf. URL: https://www.biorxiv.org/content/early/2022/05/16/2022.04.13.488226.

[170] Hao Wang. "Proving Theorems by Pattern Recognition — II". In: *Bell System Technical Journal* 40.1 (1961), pp. 1–41. DOI: https://doi.org/10.1002/j.1538-7305.1961.tb03975.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/j.1538-7305.1961.tb03975.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/j.1538-7305.1961.tb03975.x.

[171] Bryan Wei, Mingjie Dai, and Peng Yin. "Complex shapes self-assembled from single-stranded DNA tiles". In: *Nature* 485.7400 (2012), pp. 623–626.

[172] Shelley F. J. Wickham, Jonathan Bath, Yousuke Katsuda, Masayuki Endo, Kumi Hidaka, Hiroshi Sugiyama, and Andrew J. Turberfield. "A DNA-based molecular motor that can navigate a network of tracks". In: *Nature Nanotechnology* 7.3 (2012), pp. 169–173. DOI: 10.1038/nnano.2011.253.

[173] Wikipedia contributors. *Pohlig-Hellman algorithm — Wikipedia, The Free Encyclopedia*. [Online; accessed 10-September-2022]. 2022. URL: https://en.wikipedia.org/w/index.php?title=Pohlig%E2%80%93Hellman_algorithm&oldid=1085124194.

[174] Erik Winfree. "Algorithmic Self-Assembly of DNA". PhD thesis. California Institute of Technology, 1998.

[175] Erik Winfree. *Simulations of computing by self-assembly*. CaltechCSTR:1998.22, Pasadena, CA. 1998.

[176] Erik Winfree, Furong Liu, Lisa A Wenzler, and Nadrian C Seeman. "Design and self-assembly of two-dimensional DNA crystals". In: *Nature* 394.6693 (1998).

[177] Erik Winfree, Furong Liu, Lisa A Wenzler, and Nadrian C Seeman. "Design and self-assembly of two-dimensional DNA crystals". In: *Nature* 394.6693 (1998), pp. 539–544.

[178] Günther Wirsching. "On the combinatorial structure of 3N + 1 predecessor sets". In: *Discrete Mathematics* 148.1-3 (Jan. 1996), pp. 265–286. DOI: 10.1016/0012-365x(94)00243-c.

[179] Sungwook Woo and Paul WK Rothemund. "Programmable molecular recognition based on the geometry of DNA nanostructures". In: *Nature chemistry* 3.8 (2011), p. 620.

[180] Damien Woods. "Intrinsic universality and the computational power of self-assembly". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 373.2046 (2015), p. 20140214. DOI: https://doi.org/10.1098/rsta.2014.0214.

[181] Damien Woods and Turlough Neary. "On the time complexity of 2-tag systems and small universal Turing machines." In: *In 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. Berkeley, California: IEEE, Oct. 2006, pp. 132–143.

[182] Damien Woods and Turlough Neary. "The complexity of small universal Turing machines: A survey". In: *Theoretical Computer Science* 410.4-5 (2009), pp. 443–450.

[183] Damien Woods†, David Doty†, Cameron Myhrvold, Joy Hui, Felix Zhou, Peng Yin, and Erik Winfree. "Diverse and robust molecular algorithms using reprogrammable DNA self-assembly". In: *Nature* 567.7748 (2019). †Joint first authors, pp. 366–372.

[184] Hao Yan, Thomas H. LaBean, Liping Feng, and John H. Reif. "Directed nucleation assembly of DNA tile complexes for barcode-patterned lattices". In: *Proceedings of the National Academy of Sciences* 100.14 (2003), pp. 8103–8108. ISSN: 0027-8424.

[185] Xiaolong Yang, Yanan Tang, Sarah M Traynor, and Feng Li. "Regulation of DNA strand displacement using an allosteric DNA toehold". In: *Journal of the American Chemical Society* 138.42 (2016), pp. 14076–14082.

[186] Adam Yedidia and Scott Aaronson. “A Relatively Small Turing Machine Whose Behavior Is Independent of Set Theory”. In: *Complex Systems* 25.4 (Dec. 2016), pp. 297–328. DOI: 10.25088/complexsystems.25.4.297. URL: https://doi.org/10.25088/complexsystems.25.4.297.

[187] P. Yin, H. Yan, X. G. Daniell, A. J. Turberfield, and J. H. Reif. “A unidirectional DNA walker that moves autonomously along a track”. In: *Angewandte Chemie* 43.37 (2004), pp. 4906–4911.

[188] David Yu Zhang and Georg Seelig. “Dynamic DNA nanotechnology using strand-displacement reactions”. In: *Nature chemistry* 3.2 (2011), pp. 103–113.

[189] David Yu Zhang and Erik Winfree. “Control of DNA strand displacement kinetics using toehold exchange”. In: *Journal of the American Chemical Society* 131.47 (2009), pp. 17303–17314.

[190] Qian Zhang, Qiao Jiang, Na Li, Luru Dai, Qing Liu, Linlin Song, Jinye Wang, Yaqian Li, Jie Tian, Baoquan Ding, and Yang Du. “DNA Origami as an In Vivo Drug Delivery Vehicle for Cancer Therapy”. In: *ACS Nano* 8.7 (June 2014), pp. 6633–6643. DOI: 10.1021/nn502058j. URL: https://doi.org/10.1021/nn502058j.

Appendix A

A survival guide to p-adic integers

Chapter 1 heavily relies on the notion of p-adic integers. The set of p-adic integers, denoted $\mathbb{Z}_p$, is the set of p-ary strings that are infinite on their most significant side (so, infinite to the **left** following the convention used in this thesis). In this appendix, while we will not go through the historical reasons why these *numbers* were introduced at the end of the 19th century [126, 32], we will try to summarize some of their brillant properties. We will intend to convince the reader that they provide a *natural* generalisation of the arithmetics that we have been taught in elementary school, i.e. base-10 addition and multiplication on finite representations. We will focus on $\mathbb{Z}_2$ as it is particularly relevant to Collatz due to Lagarias' periodicity conjecture [99] (see Chapter 1, Section 1.1) but we will make clear what generalises to $\mathbb{Z}_p$ with p prime (or not). The ring $(\mathbb{Z}_2, +, \times)$ can be defined symbolically:

Definition-Lemma A.1 (The ring $\mathbb{Z}_2$)**.** The set of binary strings that are infinite on their most significant side is called Z_2, the set of **2-adic integers**. For instance, $\ldots 0101011101 = (01)^{\infty}1101$ is an element of $\mathbb{Z}_2$. Define $+$ and $\times$ as the same elementary-school algorithms that we learnt on finite representations but extended to these semi-infinite ones. Then, $(\mathbb{Z}_2, +, \times)$ forms a ring.

There is a lot to Definition-Lemma A.1. Let's first explicit with examples what "extending the elementary-school algorithms" for addition and multiplication means. Extending addition is the simplest, as it just requires to continue adding and propagating carries for ever to the left (most significant side), with no room for potential complications, see Figure A.1(a). However, for multiplication, at first sight, it is not obvious that the elementary-school algorithm is well-defined for infinite representations. In fact, it turns out that it is: the trailing zeros that are added at each addition step of the elementary-school algorithm for multiplication (in bold in Figure A.1(b)) finitely limit the number of input bits that contribute to each output bit. Let's see all of this in the below examples:

We now have a good intuitive idea of what the operations $+$ and $\times$ in $\mathbb{Z}_2$ are. We won't prove the claimed fact of Definition-Lemma A.1 that $(\mathbb{Z}_2, +, \times)$ is a ring and will take it for granted. We refer the interested reader to [32] for more, an excellent course on the topic which is very well suited for computer scientists[72]. Definition-Lemma A.1 can seamlessly be generalised to $\mathbb{Z}_n$ for any $n \in \mathbb{N}$ (no need to be prime for $\mathbb{Z}_n$ and its operations to be defined).

[72]This course also features two other ways to construct the p-adic integers: (a) analytically, in a similar fashion to $\mathbb{R}$, by "completing" $\mathbb{Q}$ w.r.t to "p-adic" norm, which gives the slightly bigger set (and field) $\mathbb{Q}_p$ and (b) algebraically, as the "projective limit" of sets $\mathbb{Z}/p^k\mathbb{Z}$, which corresponds to adding more and more base-p digits to create a full 2-adic integer.

	...	$\overline{0}$	$\overline{1}$	$\overline{0}$	1	1	1	0	1
+	...	1	1	1	1	0	0	1	0
=	...	0	1	0	0	1	1	1	1

(a) The last 8 bits of addition $(01)^\infty 1101 + (1)^\infty 0010$ in $\mathbb{Z}_2$. Carries are represented by $\overline{0}$ and $\overline{1}$.

	...	0	1	0	1	1	1	0	1
×	...	1	1	1	1	0	0	1	0
	...	$\overline{0}$	$\overline{0}$	$\overline{0}$	0	0	0	0	0
	...	1	0	1	1	1	0	1	**0**
	...	0	0	0	0	0	0	**0**	**0**
	...	0	0	0	0	0	**0**	**0**	**0**
	...	1	1	0	1	**0**	**0**	**0**	**0**
	...	1	0	1	**0**	**0**	**0**	**0**	**0**
	...	0	1	**0**	**0**	**0**	**0**	**0**	**0**
	...	1	**0**	**0**	**0**	**0**	**0**	**0**	**0**
+	...	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
=	...	1	1	1	0	1	0	1	0

(b) The last 8 bits of multiplication $(01)^\infty 1101 \times (1)^\infty 0010$ in $\mathbb{Z}_2$. Each trailing 0s that is added at each addition step of the elementary-school multiplication algorithm is shown in bold. These 0s finitely limits the number of input bits that contributes to an output bit.

Figure A.1: Computing the addition and the multiplication of 2-adic integers $(01)^\infty 1101$ and $(1)^\infty 0010$.

We give some fundamental facts about $\mathbb{Z}_2$:

1. $\mathbb{Z}_2$ is not countable, it has the same cardinality as $\mathbb{R}$. However, $\mathbb{Z}_2$ and $\mathbb{R}$ are not isomorphic[73]. A simple argument is that the number $1/2$ cannot be represented in $\mathbb{Z}_2$. Indeed, $2 \times x$ with $x \in \mathbb{Z}_2$ necessarily ends with a 0 whereas $2 \times 1/2 = 1$, which ends with a 1. Other fundamental differences between $\mathbb{Z}_2$ and $\mathbb{R}$ are that there is no total order on $\mathbb{Z}_2$ that preserves operations[74], and that the sign of a 2-adic integer is not always defined. This point also holds for $\mathbb{Z}_n$ with arbitrary $n \in \mathbb{N}$: in $\mathbb{Z}_n$, the number $1/p_i$ cannot be represented, with p_i prime factor of n, given a similar argument that $1/2 \notin \mathbb{Z}_2$.

2. $\mathbb{N} = \{0, 1, \dots\}$ is isomorphic to the subset of $\mathbb{Z}_2$ of binary representations that start with an infinite amount of leading 0s. For instance, 3 is the 2-adic integer $(0)^\infty 11$. Same in $\mathbb{Z}_n$ with arbitrary $n \in \mathbb{N}$.

3. $\mathbb{Z} \setminus \mathbb{N}$ is isomorphic to the subset of $\mathbb{Z}_2$ of binary representations that start with an infinite amount of leading 1s. For instance, -1 is represented by $(1)^\infty$. This makes sense because, with our extended addition algorithm we get that $(1)^\infty + 1 = 0$ as a carry propagates creating output bits 0s for ever[75]. We also get that the number $x = (1)^\infty 0010$ used in Figure A.1 is -14 since $x + (0)^\infty 1110 = 0$. Same in $\mathbb{Z}_n$ with arbitrary $n \in \mathbb{N}$, but the representation of negative integers starts with prefix $(n-1)^\infty$ where $n-1$ is the biggest base-n digit.

4. $\mathbb{Q}$ – without fractions with even denominator – is isomorphic to the subset of $\mathbb{Z}_2$ with eventually periodic representations. See a proof of this result in [43]. For instance, $x = (01)^\infty$ represents the number $-1/3$. Indeed, like two combs with teeth and holes aligned, we have that $2x + x = (1)^\infty = -1$, hence $3x = -1$ and $x = -1/3$. We also get that the number $x = (01)^\infty 1101$ used in Figure A.1 is $\frac{23}{3}$ since $x + 2x = (0)^\infty 10111$, meaning $3x = 23$. Same in $\mathbb{Z}_n$ with arbitrary $n \in \mathbb{N}$, but considering fractions whose denominator is co-prime with n (if n is prime this means denominators that are not multiples of n).

[73] Here, isomorphic means that there exists a *ring isomorphism* between the two rings, which is a bijection that preserves operations $+$ and $\times$. Intuitively, $\mathbb{Z}_2$ and $\mathbb{R}$ not being isomorphic means that $\mathbb{Z}_2$ is not $\mathbb{R}$ under disguise.

[74] Meaning, a total order $<$ on $\mathbb{Z}_2$ such that $a + c < b + c$ for all c if $a < b$ and $0 < ab$ if $0 < a$ and $0 < b$.

[75] We get a link here with the usual 2-complement representation of fixed-size signed integers in computer architectures. The number -1 is represented by $(1)^k$ with k the size of the encoding, and the operation $-1 + 1 = 0$ works by overflow. More generally, this practical representation is a finite approximation of $\mathbb{Z}_2$.

5. $\mathbb{Z}_2$ is a ring but **not a field**, indeed, as seen in Point 1, the number 2 has no inverse in $\mathbb{Z}_2$. This can be remediated easily by allowing for a decimal point and a **finite** number of decimals[76]. This construction leads to $\mathbb{Q}_2$, the set of 2-adic *numbers* which is a field – i.e. all numbers have an inverse. The inverse of 2 in $\mathbb{Q}_2$ is 0.1. The field $\mathbb{Q}_p$ can be constructed in the same way if p is prime.

6. The field $\mathbb{Q}_2$ (see Point 5) is still not isomorphic to $\mathbb{R}$, indeed $\mathbb{Z}_2 \subset \mathbb{Q}_2$ contains esoteric roots[77] that are not real, such as $\sqrt{-7}$, i.e. a number x such that $x \times x = -7 = (1)^\infty 001$. Although, as in $\mathbb{R}$, there are at most two possible roots x_1 and x_2 and they satisfy $x_1 = -x_2$ – however, we can't say that one is "positive" or "negative". The last twelve digits of one of the 2-adic roots of -7 are `...100010110101`. Because $\sqrt{-7}$ is not a rational number, its 2-adic expression is not eventually periodic. In $\mathbb{Q}_p$ with p prime, we could find other examples of negative roots but not necessarily the same as in $\mathbb{Q}_2$ (for instance, $\sqrt{-7} \notin \mathbb{Q}_3$, [79]). In general, it is known that there is no ring isomorphism between $\mathbb{Q}_p$ and $\mathbb{Q}_q$ when $p \neq q$ are primes [79], i.e. $\mathbb{Q}_p$ and $\mathbb{Q}_q$ really are different *kinds* of numbers.

Remark A.2 (p-adic analysis and $\lim_{n\to\infty} 2^n = 0$)**.** Real analysis is fundamentally based on the existence of the the norm $x \mapsto |x|$ and its associated distance $d(x, y) = |x - y|$. One beautiful feature of p-adic numbers ($\mathbb{Q}_p$ with p prime) is that one can do analysis with them pretty much like in $\mathbb{R}$ by considering the p-adic norm, $|x|_p = p^{-\mathrm{val}(x)}$ where $\mathrm{val}(x)$ is the number of trailing 0s of x. This means that x and y in $\mathbb{Q}_p$ are close to one another, i.e. $|x - y|_p$ is close to 0, if they share a same suffix of digits. Thanks to this metric, one can define continuity, differentiability etc... pretty much like in $\mathbb{R}$ and do analysis in a similar way[78] [32]. Under this metric we have $\lim_{n\to\infty} p^n = 0$. In $\mathbb{Q}_2$ (and $\mathbb{Z}_2$ by restriction) we get the odd-looking $\lim_{n\to\infty} 2^n = 0$ (since each power of 2 adds one more digit 0 as suffix, one has to look farther and farther differentiate from 0).

Let's prove the following Lemma that is used in Chapter 1 Section 1.1, when computing the Collatz sequence of rational numbers:

Theorem A.3 (The parity of a 2-adic rational is the parity of its numerator)**.** Let $x \in \mathbb{Q} \cap \mathbb{Z}_2$, which means that $x = a/b$ with b odd (Point 4) and $x = \ldots b_2 b_1 b_0 \in \mathbb{Z}_2$. Then, $b_0 = a \bmod 2$, i.e. the parity of x (its least significant 2-adic bit) is the parity of its numerator.

Proof. Because x is rational, it has an eventually periodic 2-adic representation (Point 4 and [43]). If we write w_0 and w_1 in $\{0, 1\}^*$ such that the 2-adic representation of x is $(w_1)^\infty w_0$, then it is known that $x = [\![w_0]\!]_2 + 2^{|w_0|}[\![w_1]\!]_2 \frac{1}{1-2^{|w_1|}}$ [43, 32]. We want to show that the least significant bit of w_0 is the least significant bit of a in $x = a/b$. We have: $x = \frac{[\![w_0]\!]_2 + 2^{|w_0|}[\![w_1]\!]_2 - 2^{|w_1|}[\![w_0]\!]_2}{1-2^{|w_1|}}$. Note that the denominator is odd, so no power of 2 can be removed from the numerator. Hence $a \equiv [\![w_0]\!]_2 + 2^{|w_0|}[\![w_1]\!]_2 - 2^{|w_1|}[\![w_0]\!]_2 \bmod 2$, giving $a \equiv [\![w_0]\!]_2 \bmod 2$, giving what we want: the least significant bit of a is the least significant bit of w_0 which is the least significant bit of x. □

Remark A.4 (Computing the 2-adic representation of a rational)**.** If $x \in \mathbb{Q}$ has an odd denominator then it has a 2-adic representation (Point 4). To get its representation, we apply the same algorithm than we would use on natural numbers: thanks to Theorem A.3 we know that $b \in \{0, 1\}$, the least significant

[76] Allowing for a finite number of decimals only is crucial in order to preserve the soundness of the definition of $+$ and $\times$.

[77] It is known that integer $k \neq 0$ admits a square root in $\mathbb{Z}_2$ if $k = 4^a(8b + 1)$ for some $a \in \mathbb{N}$, $b \in \mathbb{Z}$. Integer -7 is given by $a = 0$ and $b = -1$. See `https://math.stackexchange.com/questions/473595/characterization-of-integers-which-has-a-2-adic-square-root`. In general, there are known algorithms to compute square roots in $\mathbb{Q}_2$ (when they exist) and interestingly, these methods are very similar to Newton's method for finding the 0s of a real function [32].

[78] One can even run Newton's algorithm for finding the 0s of a function[32]!

bit of x, is given by the parity of the numerator of x; we store b then we do $x := (x - b)/2$ then iterate to get all the 2-adic digits of x. In other words, the 2-adic representation of x is its *parity vector* (see Chapter 1, Section 1.3.3) under the "trivial" Collatz-like map $x \mapsto x/2$ if x is even and $x \mapsto (x-1)/2$ when x is odd – this map is also known as the *shift map*[26] and is denoted $\mathcal{S}$ [139]. Rationals (with odd denominator) are exactly the numbers that reach a cycle under iteration of that map. Whether this is also true for the Collatz map itself is Lagarias' periodicity conjecture (Chapter 1, Section 1.1).

The other result that we have used in Chapter 1 (Remark 1.24) is the following:

Theorem A.5 (Aperiodicity in $\mathbb{Z}_6$)**.** Let $\phi : \mathbb{Z}_2 \times \mathbb{Z}_3 \to \mathbb{Z}_6$ be the ring isomorphism from $\mathbb{Z}_2 \times \mathbb{Z}_3$ to $\mathbb{Z}_6$ (Lemma 1.23). Then, for $x, y \in \mathbb{Q} \cap \mathbb{Z}_2 \cap \mathbb{Z}_3$ we have $\phi(x, y) \in \mathbb{Q}$ iff $x = y$. The later means that $\phi(x, y)$ is eventually periodic if and only if x and y are eventually periodic **and** represent the same rational, see Remark 1.24 for the consequence of this fact in tilings.

Proof. Having $x \in \mathbb{Q} \cap \mathbb{Z}_2 \cap \mathbb{Z}_3$ means that x is a rational with denominator that is not a multiple of 2 nor a multiple of 3 (Point 4). Hence the denominator of x is not a multiple of 6 and hence we know that x has a representation (eventually periodic) in $\mathbb{Z}_6$: $x \in \mathbb{Z}_6$. Assume $x = y$ and $x = a/b$, $b \neq 0$. Then by ring isomorphism, $b\phi(x, x) - a = \phi(bx - a, bx - a) = \phi(0, 0) = 0$. Hence $\phi(x, x) = a/b = x \in \mathbb{Z}_6$ and is eventually periodic. Assume that $\phi(x, y) = a/b \in \mathbb{Q}$ with $b \neq 0$, then, by Point 4, $a/b \in \mathbb{Z}_6 \cap \mathbb{Q}$ implies that b contains no factor 3 or 2, hence $a/b \in \mathbb{Q} \cap \mathbb{Z}_2 \cap \mathbb{Z}_3$. By ring isomorphism we have $\phi(bx - a, by - a) = 0$, hence $bx - a = by - a = 0$ and $x = y = a/b$. □

Appendix B

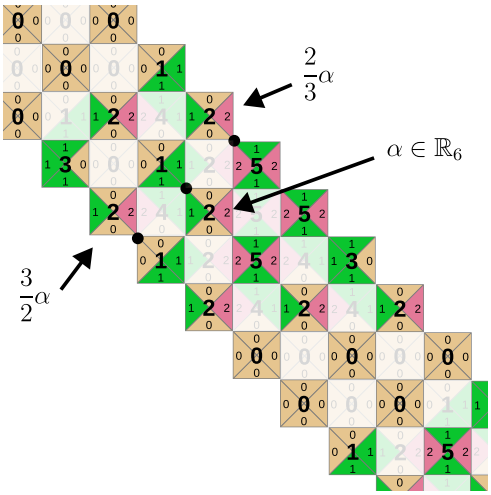
Tiling interpretation of Mahler's 3/2 problem

Mahler's 3/2 problem was stated in 1968 as follows [107]: does there exist $\alpha > 0 \in \mathbb{R}$ such that for all $n \in \mathbb{N}$ the fractional part of $\alpha(3/2)^n$ is less than 1/2? Such α is called a Z-number. By writing real numbers in base 6, the problem asks whether the first fractional digit of $\alpha(3/2)^n$ can remain in $\{0, 1, 2\}$ for all $n \in \mathbb{N}$ or not. Mahler conjectured that there are no Z-numbers. The problem has been studied by many authors but remains open to this day [68, 5, 62, 87, 91]. Here, (a) we reformulate the problem using the 6 Collatz tiles introduced in Chapter 1, (b) we reformulate the link established by Mahler [107] between the 3/2 problem and the Generalised Collatz Map defined in $\mathbb{Z}_2$ by $M(x) = 3x/2$ when x is even and $M(x) = (3x + 1)/2$ when x is odd (it's almost Collatz!) with the result that there is no Z-number between x and $x + 1$ with $x \in \mathbb{N}$ if iterating M starting from x eventually gives two successive odd iterates and (c) we give a visual, tile-based, proof that there are no Z-numbers between 0 and 1. Our tile-based reformulation of Mahler's 3/2 problem is very close in spirit to the cellular-automaton reformulation of Jarkko Kari [86] and Johan Kopra [91, 87].

The results presented here were nurtured at the occasion of a research trip to Finland where I visited Jarkko Kari and Johan Kopra. I thank them for this inspiring trip.

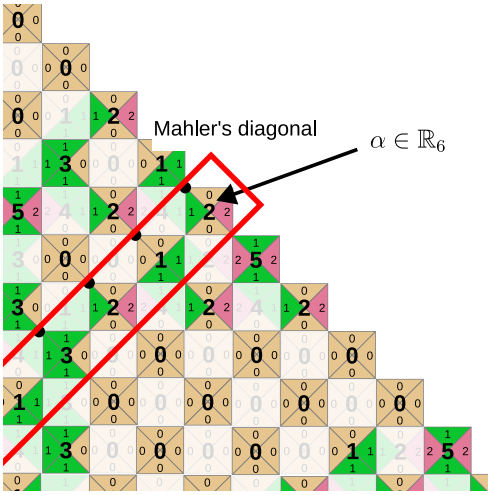
B.1 Tile-based reformulation

The central idea, which comes from [86] is that the tiles can encode a base-6 encoded real on a south-east going diagonals with finitely many non-0 tiles in the north-west direction (i.e. the integer part of a real is of finite size) and that parallel diagonals respectively multiply by 3/2 in the south-west direction and 2/3 in the north-east direction:



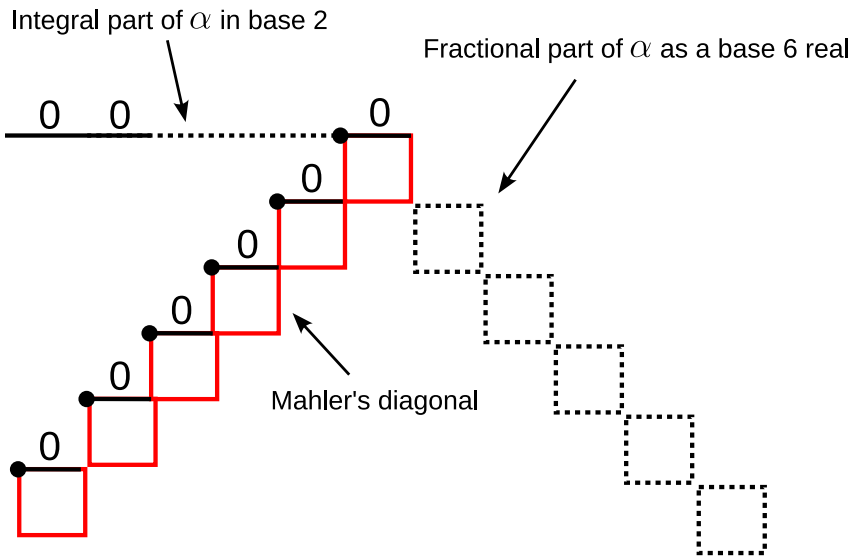
In the above example, an arbitrary $\alpha > 0 \in \mathbb{R}$ is given on the central diagonal as a base-6 represented real (set abusively denoted $\mathbb{R}_6$) $\alpha = 21.252005\ldots$. On the diagonal parallel below it we read $32.12001\ldots$ which is the beginning of the base-6 representation of $\frac{3}{2}\alpha$ and on the diagonal parallel above it we read $12.55320\ldots$ which is the beginning of the base-6 representation of $\frac{2}{3}\alpha$. In low opacity we have represented intermediate tiles involved in the computation of these $\frac{3}{2}\alpha$ and $\frac{2}{3}\alpha$ diagonals. The placement of the decimal point is arbitrary as long as there are only finitely non-0 tiles to the north-west of it (i.e. if we changed the position of the decimal point in the above example the result would still hold for the new numbers that we would read). Note that Chapter 1 does not give this result since it focuses mainly on p-adic integers instead of reals, the result is given by [86] (and exploited in [91, 87]) which interprets the successive base-6 reals as configurations of a cellular automaton.

Then, Mahler's problem is all about the (anti) diagonal outlined in red, that we call "Mahler's diagonal" below:



Mahler's question reformulates into whether or not there exists an $\alpha > 0 \in \mathbb{R}$ (called Z-number) such that all the tiles on Mahler's diagonal are in $\{0, 1, 2\}$ i.e. the north color of the tiles on Mahler's diagonal is always 0. Here we see that $\alpha = 21.252005\ldots$ is **not** a Z-number since there is a tile 3 on Mahler's diagonal.

Schematically, the following represents the constraints satisfied by a Z-number α, note that using Corollary 1.18 we have chosen to represent the integral part of α in binary as it removes part of the tiling that is "only" doing base-6 to base-2 conversion:



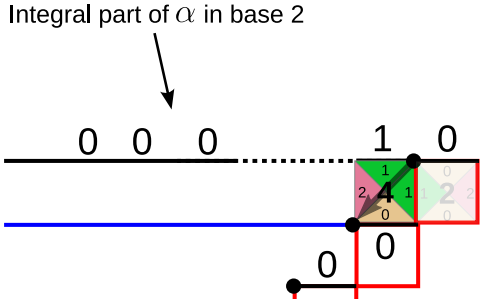
There are two constraints for α to be a Z-number (solid traits): (a) its integral part must be a natural number, hence starting with prefix 0^∞, or equivalently, have finitely many 1s, and (b) the north color of all tiles on Mahler's diagonal (solid red) must be 0 (this is equivalent to asking that these tiles are in $\{0, 1, 2\}$). Note that Mahler's diagonal is entirely determined by the choice of α. The converse is not true: while setting Mahler's diagonal will uniquely determine the integral part of α, it does not uniquely map to a fractional part because it involves using the north-west corner of the tiles that is **nondeterministic** (see Figure 1.7), hence no corresponding fractional part may exist at all or potentially, several could.

With this schema we get a sense of the hardness of Mahler's problem: a Z-number is *over-constrained*. Indeed, if you lift constraint (a) of the integral part being a natural number (i.e. starting with prefix 0^∞) then, for *any* choice of fractional part, a unique valid tiling is determined and reconstructs an "integral part" in $\mathbb{Z}_2$ such that Mahler's diagonal is valid (all its tiles are in $\{0, 1, 2\}$). However, such numbers with infinite integral part and infinite decimal part do not form a ring so we can hardly call them *numbers*.

B.2 Link with Mahler's Collatz-like map

Here we want to convince you of the fact proven by Mahler in [107] that the sequence of integral parts of $\alpha(3/2)^n$ with α a Z-number is given by iterating the Collatz-like map $M(x) = 3x/2$ if x is even and $M(x) = (3x+1)/2$ if x odd, on the integral part of α, i.e. $\lfloor \alpha \rfloor$. Like the Collatz map, this map is defined not only on $\mathbb{N}$ but on $\mathbb{Z}_2$ entirely, see Chapter 1 and Appendix A.

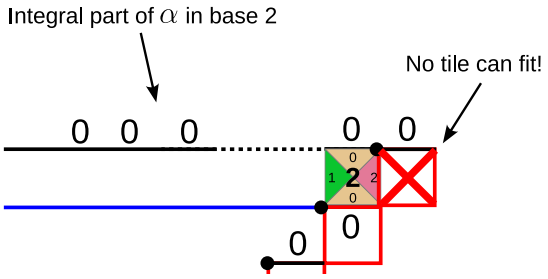
If α is a Z-number with odd integral part we are in the following situation:



The tile below the last bit of the integral part of α is constrained to have north color 1 (the integral part is odd) and to have south color 0 (Mahler's diagonal constraint). Only one tile can fit: tile 4 (see

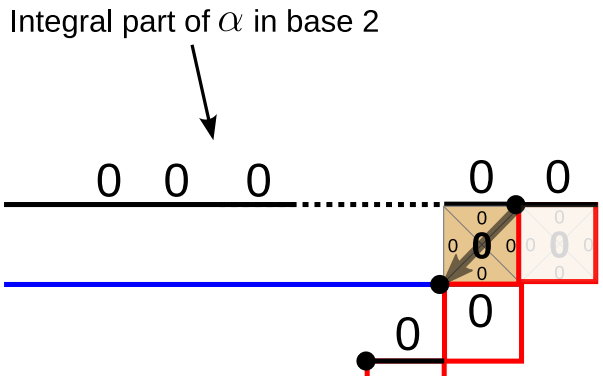
Figure 1.7) – we can also deduce that the tile directly to its right is tile 2 (low opacity above). Hence, by Theorem 1.25, the next integral part (in blue above) is given by $f(\lfloor\alpha\rfloor)$ with f the function computed by $(\swarrow, 4)$ which is $x \mapsto (3x+1)/2$, see Chapter 1, Section 1.3.1 for more details.

If α is a Z-number with even integral part, the constraints applied to the tile just below the last bit of the integral part are 0 on the north (even integral part) and 0 on the south (Mahler's diagonal constraint), which would theoretically allow for two tiles to fit, tile 0 and tile 2 (Figure 1.7). Let's try tile 2:



The issue if using tile 2 is that the position directly to the right of it, which holds the first digit on Mahler's diagonal, cannot admit any tile because it has color 2 on the west and color 0 on the north and no tiles of the Collatz tile set satisfy this constraint (Figure 1.7)! Hence we can't use tile 2 as it violates the hypothesis that α was a Z-number, thus with defined first digit on Mahler's diagonal.

Hence we have to pick tile 0 and we are in the following situation – we also deduce that the tile immediately to its right is tile 0 (low opacity):



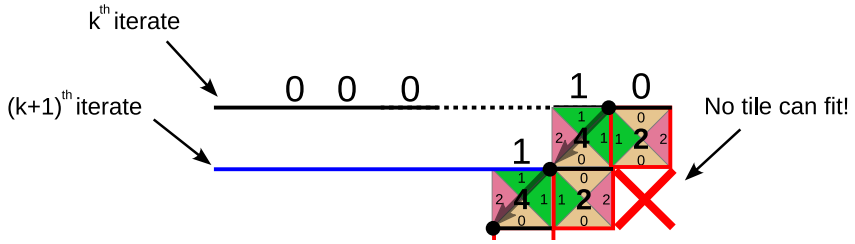
Hence, by Theorem 1.25, the next integral part (in blue) is given by $f(\lfloor\alpha\rfloor)$ with f the function computed by $(\swarrow, 0)$ which is $x \mapsto 3x/2$, see Chapter 1, Section 1.3.1 for more details.

Altogether, by iterating this argument, we get that the successive integral parts of $\alpha(3/2)^n$ of α a Z-number are given by iterating $M(x) = 3x/2$ if x is even and $M(x) = (3x+1)/2$ if x is odd, on $\lfloor\alpha\rfloor$ (the integral part of α).

Mahler proved that if the iterates of $M(x)$, with $x \in \mathbb{N}$, eventually produce two successive odd numbers then there are no Z-numbers between x and $x+1$. We prove this fact with the tiles:

Theorem B.1 ((16) in [107])**.** Let $x \in \mathbb{N}$, if there exists $k \in \mathbb{N}$ such that the iterates $M^k(x)$ and $M^{k+1}(x)$ are both odd then there is no z-number between x and $x+1$.

Proof. The proof is visual. Given the points that we have made in this section, we know that two successive odd iterates of M corresponds to the following situation:



There are two successive tiles 2 on Mahler's diagonal, which create the constraint of having west color 2 and north color 0 on the position marked with a red cross. There are no tiles that fit this constraint hence there is no possible reconstruction of a fractional part for any potential Z-number between x and $x+1$. Hence, there are no Z-number between x and $x+1$. □

Iterating Mahler's map starting from any $x \in \mathbb{N}^+$ seems to always lead to having 2 successive odd iterates, for instance, here are the iterates of 237:
237, 356, 534, 801, 1202, **1803**, **2705**, 4058, 6087, 9131, 13697, 20546, ...
We reach two successive odd iterates, 1803 and 2705 after 5 iterations: there are no Z-number between 237 and 238.

This observation leads to the following Collatz-flavored conjecture which because of Theorem B.1 (and Section B.3 for the case $x = 0$), implies the non-existence of Mahler Z-numbers:

Conjecture B.2. For all $x \in \mathbb{N}^+$, there exists $k \in \mathbb{N}$ such that the iterates $M^k(x)$ and $M^{k+1}(x)$ are both odd, with $M(x) = 3x/2$ when x is even, and $M(x) = (3x+1)/2$ when x is odd.

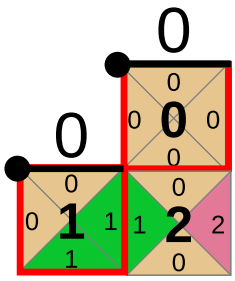
Remark B.3 (Tile-based interpretation of (11) in [107]?)**.** Mahler also gives the following result: there is at most one Z-number between x and $x+1$ for $x \in \mathbb{N}$ ((11) in [107]). We currently fail to interpret this result with the tiles as it seems to involve working with the nondeterministic north-west corner of the tiles and seems to indicate that there is at most one possible reconstruction in the nondeterministic direction, which is surprising to us. We leave to future work to understand this point.

B.3 No Z-number between 0 and 1

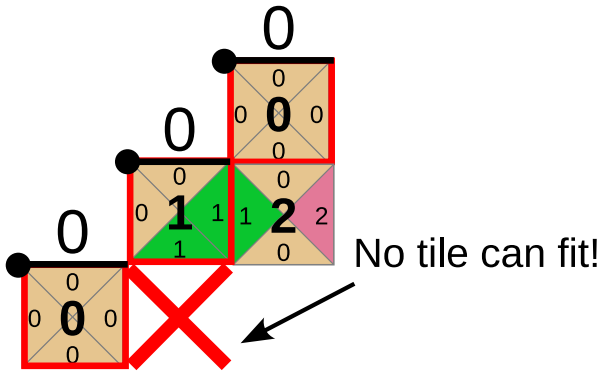
Theorem B.1 cannot be used in the case $x = 0$ since iterating Mahler's map M from 0 will produce only 0s, which are all... even. Note that 0, if it was not excluded of the definition of Z-numbers then it would be a perfectly correct Z-number, and probably the result referred to in Remark B.3 that there is at most one Z-number between x and $x+1$ for $x \in \mathbb{N}$ would still apply in this edge case giving that there can be no other Z-number than 0 in $[0, 1]$, but one would have to check the original paper carefully [107].

We take another route by showing that there are no Z-number between 0 and 1 with the tiles. Because the integral part of α is 0 and because of Mahler's constraint we know that all the tiles north-west of Mahler's diagonal (not including the diagonal) are tiles 0s. This implies that there cannot be any 2 on Mahler's diagonal because the west color of tile 2 is 1 and it would contradict that the east color of the tile directly to its left must be a 0 since this tile is tile 0.

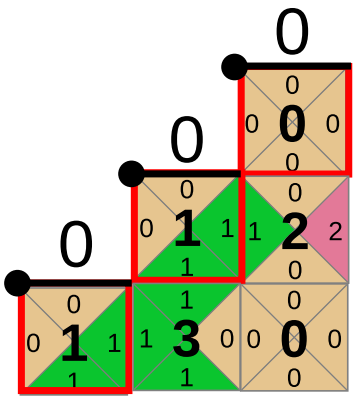
Hence, we can only have tile 0 or 1 on Mahler's diagonal. Take the first time a tile 1 is place on Mahler's diagonal, locally, we are in the following situation:



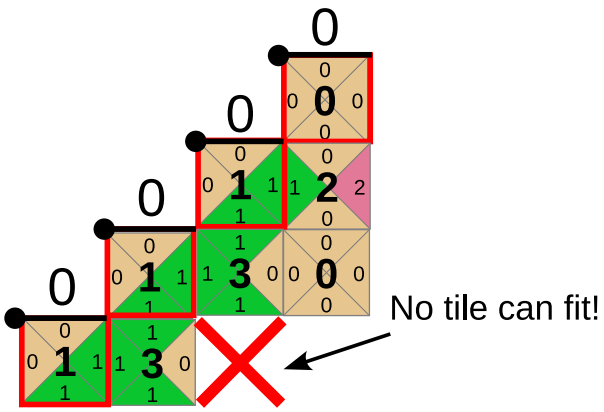
The next tile cannot be tile 0 because it creates the constraint of having west color 0 and north color 1 for the tile directly to its right, which doesn't correspond to any of the Collatz tiles:



Hence, it must be tile 1:



The tile after that cannot be tile 0 for the same reason as above (a tile 0 cannot come right after a tile 1 on Mahler's diagonal) and it also cannot be another tile 1:



In all cases, we reached the impossibility of reconstructing a fractional part, hence there are no Z-number between 0 and 1.

Remark B.4 (Forbidden patterns on Mahler's diagonal)**.** More generally we have shown that several patterns are forbidden on Mahler's diagonal if we are to have a Z-number: Theorem B.1 relies on the fact that we can't have two successive tiles 2 on Mahler's diagonal, and above we saw that we cant have tile 0 directly after tile 1 or three successive tile 1s in a row on Mahler's diagonal.

Appendix C

`coreli`: the Collatz research library

In this Appendix, we give a tutorial on `coreli v0.0.3`, the Collatz research library[79], which is a Python library containing tools useful for researching on Collatz, including visualisation of the tilings presented in Chapter 1[80].

This tutorial is relevant for version `v0.0.3` and might unfortunately become obsolete for later versions as the library evolves.

Installation

`coreli` is hosted on Github: `https://github.com/tcosmo/coreli` (`v0.0.4` at the time of this writing). You can install it (for instance, in a python3 virtual environment) by doing: `pip install coreli`. If you have any trouble installing coreli, please open an issue on `https://github.com/tcosmo/coreli/issues`. `coreli` has a documentation that is hosted at `https://dna.hamilton.ie/tsterin/coreli/docs/`.

We will now illustrate `coreli`'s features through code snippets.

Base conversion

When numbers are represented as strings, the least significant digit is to the right but when represented as list we put the least significant digit at index 0.

```
>>> from coreli import int_to_base
>>> int_to_base(13,2)
'1101'
>>> int_to_base(13,2,9)
'000001101'
>>> int_to_base(13,2,to_str=False)
[1, 0, 1, 1]
>>> int_to_base(313,5)
'2223'
>>> int_to_base(313,5,to_str=False)
[3, 2, 2, 2]
```

[79] Any similarity with Arcangelo Corelli is purely coincidental: `https://www.youtube.com/watch?v=5BPhkY6xIP8`

[80] This library was initially released in the context of our first work on the Collatz process, [154].

p-adic integers

Our p-adic integers (p is arbitrary, not necessarily prime) implementation consists in specifying a `digit_function` that returns the value of the n^{th} digit of the p-adic integer. That function can be inferred automatically if the p-adic integer is constructed from an int or rational (assuming the rational can be represented in $\mathbb{Z}_p$).

```
>>> from coreli import Padic
>>> from sympy import Rational
>>> Z2 = Padic(2)
>>> Z2.from_int(25).to_str()
'...0000011001'
>>> Z2.from_rational(Rational(-1,23)).to_str(20)
'...00101100100001011001'
>>> x = Z2(digit_function = lambda n: n\%2)
>>> x.to_str()
'...1010101010'
```

`coreli` implements rings operations on p-adic integers, addition:

```
>>> from coreli import Padic
>>> Z2 = Padic(2)
>>> x = Z2.from_int(25)
>>> (47 + x).to_str()
'...0001001000'
>>> z = Z2(digit_function = lambda n: n\%2)
>>> (z + z + x).to_str(20)
'...01010101010101101101'
```

and multiplication:

```
>>> from coreli import Padic
>>> Z2 = Padic(2)
>>> x = Z2.from_int(3)
>>> (2*x).to_str()
'...0000000110'
>>> (5*x).to_str()
'...0000001111'
>>> (27*x).to_str()
'...0001010001'
>>> (x*x).to_str()
'...0000001001'
>>> y = Z2(digit_function=lambda x: (x+1)%2)
>>> (3*y + 1).to_str()
'...0000000000'
```

`coreli` also implements left and right shifts operations:

```
>>> from coreli import Padic
>>> Z2 = Padic(2)
>>> x = Z2.from_int(3)
>>> x.to_str()
'...0000000011'
>>> (x << 3).to_str()
'...0000011000'
>>> (x >> 1).to_str()
'...0000000001'
>>> (x >> 2).to_str()
'...0000000000'
```

Rational p-adic integers

Rational p-adic integers are eventually periodic (see Appendix A). `coreli` gives the eventually periodic decomposition of a p-adic rational, i.e. its period and initial segment:

```
>>> from coreli import Padic
>>> Z2 = Padic(2)
>>> x = Z2.from_int(0)
>>> x.rational_periodic_representation()
'(0)*'

>>> x = Z2.from_int(-17)
>>> x.rational_periodic_representation()
'(1)* 01111'

>>> x = Z2.from_rational(Rational(1778,53))
>>> x.rational_periodic_representation()
'(0010000111001111101100101011011110001100000100110101)* 101010'

>>> x = Z2.from_rational(Rational(-1,23))
>>> x.rational_periodic_representation()
'(00001011001)*'

>>> Z3 = Padic(3)
>>> x = Z3.from_rational(Rational(346,86))
>>> x.rational_periodic_representation()
'(10222101001020220111012000121221202002111 2)* 22'
```

Collatz maps

`coreli` implements Collatz maps C and T on integers, rationals (with odd denominator), and 2-adic integers.

Examples with T:

```
>>> from coreli import T, Padic
>>> from sympy import Rational
>>> T(2)
1
>>> T(3)
5

>>> T(Rational(-2,23))
-1/23
>>> T(Rational(-1,23))
10/23

>>> Z2 = Padic(2)
>>> minus_one_third = Z2(digit_function = lambda n: (n+1)%2)
>>> T(minus_one_third).to_str()
'...0000000000'

>>> from sympy.ntheory.primetest import is_square
>>> some_2_adic = Z2(digit_function = lambda n: int(is_square(n)))
>>> some_2_adic.to_str(40)
'...0001000000000010000000010000001000010011'
>>> T(some_2_adic).to_str(40)
'...0001100000000011000000011000001100011101'
```

You can iterate the Collatz maps thanks to the `iterate` and `iterates` utilities:

```
>>> from coreli import T, iterate, iterates
>>> iterate(T,0,23)
23
>>> iterate(T,7,23)
5
>>> iterates(T,7,23)
[23, 35, 53, 80, 40, 20, 10, 5]
```

Parity vectors

Collatz parity vectors (see Chapter 1, Section 1.3.3) can be represented easily with `coreli` either from a given parity sequence or computed from the Collatz sequence of a number (here 23):

```
>>> from coreli import ParityVector
>>> ParityVector([0,1,1,0,1])
[0, 1, 1, 0, 1]
>>> ParityVector.from_Collatz(23,7)
[1, 1, 1, 0, 0, 0, 0, 1]
```

From a parity vector, we can compute its corresponding α and β (see Chapter 1, Corollary 1.36) which are the smallest integers in $\mathbb{N}$ such that $\beta = T^n(\alpha)$ with n the length of the parity vector and such that the iterates follow the parities given by the parity vector:

```
>>> from coreli import ParityVector, T, iterates
>>> pv = ParityVector([0,1,1,0,1])
>>> pv.first_occurrence()
(22, 20)
>>> pv.first_occurrence(symbolic=True)
('10110', '202')
>>> iterates(T, len(pv), 22)
[22, 11, 17, 26, 13, 20]
```

In the above example $(\alpha, \beta) = (22, 20)$ for parity vector $p = [0, 1, 1, 0, 1]$ of length $n = 5$ since $20 = T^n(22)$ and iterates $[22, 11, 17, 26, 13]$ have the parities given by p. Note that because of fences and poles `iterates(T, len(pv), 22)` is of size $n + 1$. If `symbolic` is set to `True` then α is given in base 2 on n bits and β in base 3 on k bits with k the number of 1s in p.

Rational cycles

By Chapter 1, Theorem 1.58, we know that to each parity vector is associated exactly one rational that cycles under Collatz iterations while following the parities given by the parity vector. Given a parity vector, `coreli` computes this rational:

```
>>> from coreli import ParityVector
>>> pv = ParityVector([0,1,1,0,1])
>>> pv.cyclic_rational()
46/5
```

Indeed, 46/5 cycles under Collatz (we can apply Collatz on odd-denominator rationals as they are 2-adic integers, Chapter 1, Section 1.1 and Appendix A) following $[0, 1, 1, 0, 1]$ parities:

```
>>> from coreli import T, iterates
>>> from sympy import Rational
>>> iterates(T, len(pv), Rational(46,5))
[46/5, 23/5, 37/5, 58/5, 29/5, 46/5]
```

Most complex tile

Chapter 1, Section 1.5 defines the "most complex tile" (MCT) of a parity vector, Definition 1.64. We can compute MCT with `coreli`, using an example of Figure 1.21:

```
>>> from coreli import ParityVector
>>> ParityVector([1,1,0,1]*2).most_complex_tile()
2

>>> ParityVector([1,1,0,1]*2).rotate()
[1, 0, 1, 1, 1, 0, 1, 1]
>>> ParityVector([1,1,0,1]*2).rotate().most_complex_tile()
3

>>> ParityVector([1,1,0,1]*2).rotate(-1)
[1, 1, 1, 0, 1, 1, 1, 0]
>>> ParityVector([1,1,0,1]*2).rotate(-1).most_complex_tile()
5
```

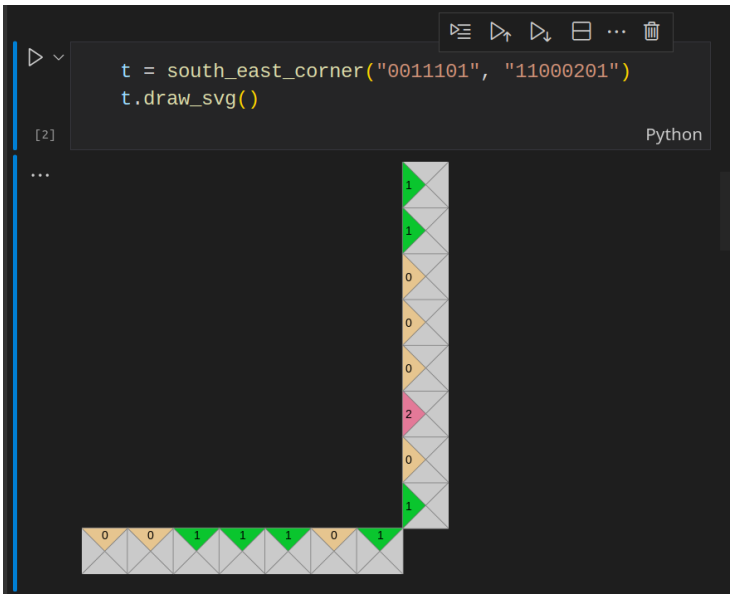
Tilings

For this feature, we highly recommend using `Jupyter Lab` (it can be used directly in `vscode`) which we allow for direct visualisation and even interactivity.

`coreli` provides helpers to construct some seed tilings: `south_east_corner`, `south_west_corner`, `north_east_corner`, `base6_diagonal`, `base32_diagonal`.

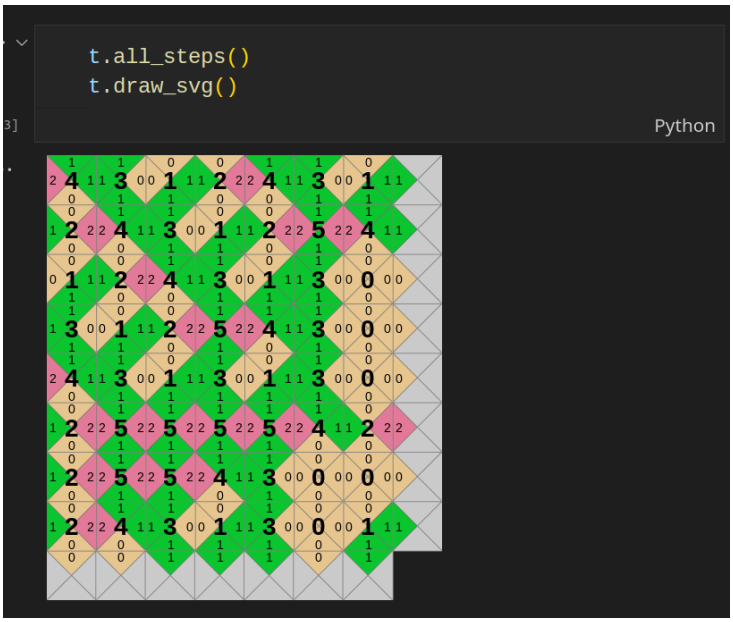
Example on `south_east_corner` (please see the doc[81] for description of the other listed helpers):

```
>>> from coreli.Collatz_tilings import south_east_corner
>>> t = south_east_corner("0011101", "11000201")
>>> t.draw_svg()
```




```
>>> t.all_steps()
>>> t.draw_svg()
```

[81] https://dna.hamilton.ie/tsterin/coreli/docs/

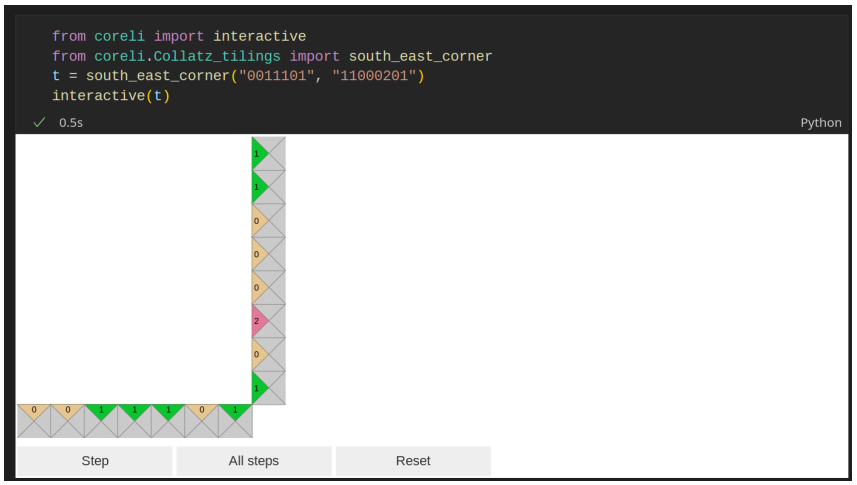


Function `t.all_steps` performs tiling steps until no more are possible. The function `t.step` performs just one step. The function `t.draw_svg` returns a svg that can be further saved to a svg file or png:

```
>>> t.draw_svg().saveSvg("tiling.svg")
>>> t.draw_svg().savePng("tiling.png")
```

In a Jupyter notebook a simulation can be made **interactive** easily:

```
>>> from coreli import interactive
>>> from coreli.Collatz_tilings import south_east_corner
>>> t = south_east_corner("0011101", "11000201")
>>> interactive(t)
```

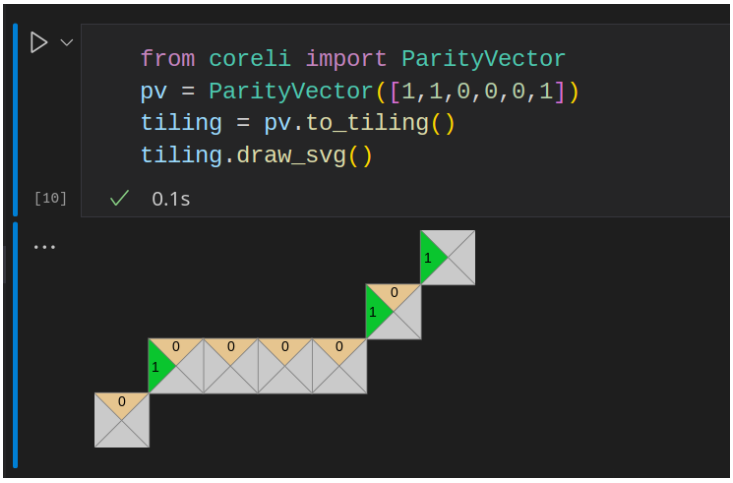


Pressing the button **Step** will update the frame with one more tile, **All steps** will update the frame with all tiles and **Reset** will reset the simulation.

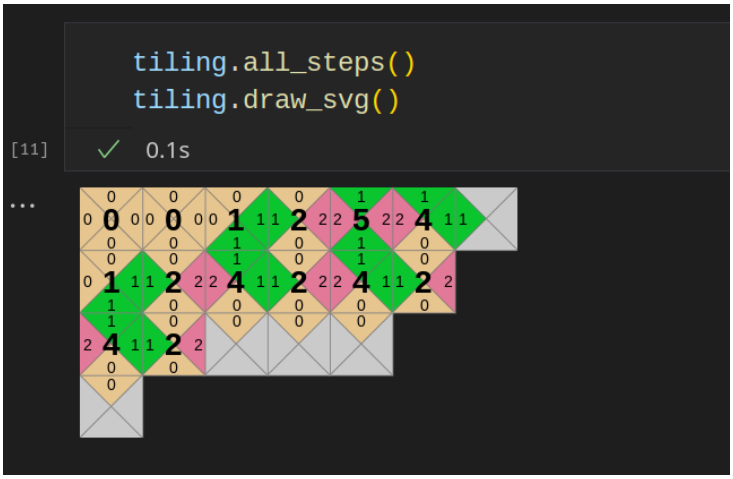
Tilings from parity vectors

A parity vector can be converted into the associated tiling (see Chapter 1, Section 1.3.3):

```
>>> from coreli import ParityVector
>>> pv = ParityVector([1,1,0,0,0,1])
>>> tiling = pv.to_tiling()
>>> tiling.draw_svg()
```




```
>>> tiling.all_steps()
>>> tiling.draw_svg()
```

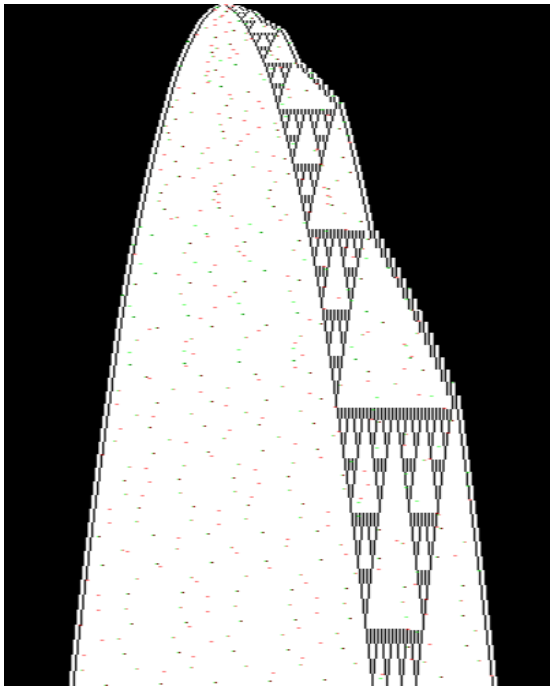


Collatz ancestors regular expressions

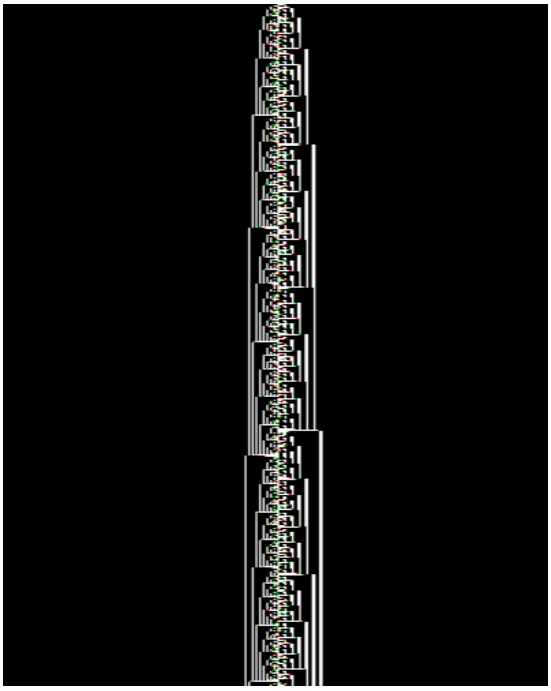
Present in earlier versions of `coreli`, this feature, which explicitly constructs regular expressions for odd predecessor sets (see Chapter 1, Section 1.6 and [154]) is currently being re-implemented and should be available soon.

Appendix D

bbchallenge.org



(a) Space-time diagram of bbchallenge machine #9,005,190 which prints Sierpiński triangles. `https://bbchallenge.org/9005190&s=30000&ox=0.4`. Found by bbchallenge collaborator @IijiI.



(b) Space-time diagram of bbchallenge machine #14,263,231 which simultaneously counts in base 2 (left) and 3 (right). `https://bbchallenge.org/14263231`.

Figure D.1: Two out of the 88,664,064 five-state Turing machines of `http://bbchallenge.org`.

After having worked on busy beaver value BB(15) and having realised that it is very unlikely that we'd ever know it as that would imply to solve a hard number-theoretical conjecture, see Chapter 2, we were motivated to take the problem "by the other end" and to study the smallest unknown busy beaver value: BB(5). It is conjectured that BB(5) = 47,176,870 because the current champion, `https://bbchallenge.org/1RB1LC_1RC1RB_1RD0LE_1LA1LD_1RZ0LA&status=halt`, has been undefeated since the 1990s and halts after 47,176,870 steps [109, 1]. To settle the conjecture, based on the method given in

[109], one has to study a set of roughly 88M Turing machines and decide for each of them whether it halts or not (from blank tape). We believe that this task, due to its magnitude, is inherently collaborative and that its only hope of success resides in the joint efforts of many passionate people coming up with verifiable ways to automatically decide these machines. This is why we created the platform `https://bbchallenge.org` which makes the 88M machines publically available (`https://bbchallenge.org/method#download`) and keeps track of these ongoing efforts. For the results of these efforts to be accepted and trusted by the wider scientfic community our project distinguishes itself by having a rather strict "reproducibility and verifiability statement" which is imposed on any piece of software developped for the project.

For more information, please refer to:

- `https://bbchallenge.org/story`
- `https://bbchallenge.org/method`
- `https://bbchallenge.org/contribute`

Appendix E

Appendix to Algorithmic DNA Origami

E.1 Background and additional theory

E.1.1 A few future work theory questions

Using the conventions of Section 4.2:

1. Are final configurations (when they exist) always reachable?
2. Are thermodynamically optimal configurations always reachable?
3. Are thermodynamically optimal configurations always final?
4. We know from Chapter 3 that it is possible to simulate Boolean circuits in the Scaffolded DNA Computer. However, is it possible to implement any Boolean circuit in a 'kinetic sense' where there is only one reachable final configuration that is thermodynamically optimal?

E.1.2 Our tile set encoding does not cheat

Our simulation of finite state machines (FSMs) with the Scaffolded DNA Computer, Figure 4.4, is unusual for a few reasons. First, a single tile encodes both the current input symbol(s), and next input symbols(s), whereas in 2D aTAM [125, 57] systems, these are generally de-coupled and encoded by distinct tiles. Second, here, a single tile, and even a single tile side's colour (glue), encodes both input (read) information, and finite state machine state name. Third, one could imagine our 1D computations are simply a bunch of translators [168]!

To address these questions we note that when defining a new model of computation, it is important to very clearly set out the permitted computational power of the *encoding function*: a mathematical function that maps inputs (here, bit/trit strings) and programs (FSMs) to instances of the model (a tile set). We need the encoding to be so simple, it provably does not clandestinely perform the computation 'under the hood'. Here, we note that mapping a FSM and its bit-string input to a tile set is a simple task, specifically a task that is simpler than the computation itself (except for the trivial case of bit-copy, which is barely a computation). For brevity, we omit a full mathematical formalisation.[82]

[82]We would formalise this intuition by using the mathematical notion of *uniformity* from computational complexity theory [118, 77]. Specifically, all of our input encoding schemes are AC0-uniform, and many of our computations, e.g parity, are not. Even stronger, *all* of our tile sets are encodable by the more restrictive notion of uniformity called DLOGTIME-

E.2 SEQUENCE INDEPENDENT REPORTING MECHANISM

Mechanism. The reporting mechanism used to read results of the 1D Scaffolded DNA Computer is depicted in Figure 4.7. It consists in a Iowa Black FQ quencher attached to 20-base domain Q and ATTO-590 fluorophore attached to 20-base domain F. In order to report, for instance at position A, the strand at position B is prefixed with domain F* to allow the fluorophore to bind. Then, strands reporting a 0 at position A will be suffixed with domain Q* to allow the quencher to bind and trigger quenching of the fluorophore. In order to report a 1, no domain is suffixed and quenching is not triggered. In practice, several strands compete at the reporting position, some with domain Q* and some without, depending on the output that they encode.

Sequence design. The following principles were used to design domains F and Q of the reporting mechanism (also called `SIRM.RQ` and `SIRM.RF`):

- Hard constraints (i.e. non-compliant sequences are rejected):
  1. 20 bases.
  2. Alphabet A,T,C and at most 1 G.
  3. No patterns CCCC, GGGG, AAAAA, TTTTT.
  4. Watson-Crick energy (according to nearest-neighbour model [142]) be between -21.7 and -21.95 kcal/mol (-21.7 is the average Watson-Crick energy computed over 10,000 20-base strands).
  5. 4 bases next to fluorophore should not be G.
- Soft constraints (i.e. sequences may not comply but are optimised to do their best):
  1. Intra-domain secondary structure $\geq$ -0.1 kcal/mol (using pfunc from NUPACK4 [69])
  2. Binding energy[83] between SIRM.RF and SIRM.RQ $\geq$ -3 kcal/mol (in all possible 4 ways in terms of reverse complementation)

We used sequence designer DSD[84], which in turn makes extensive use of NUPACK4 [69].

Sequences. The designed sequences are:

- `SIRM.RQ`: `CCCACCTCTCCACACTACCC/3IABkFQ/`
- `SIRM.RF`: `/5ATTO590N/ACCATCCCTTCGCATCCCAA`

Controls. We would hope that our reporting mechanism gives similar signals for reporting a bit 0 (resp. bit 1) at all positions A/B/C/D. In order to control for this we ran 15 samples reporting bit 0 and 15 samples reporting bit 1 varying the position of reporting (A,B,C,D), the strands present at non-reporting positions and the total amount of DNA in solution. Normalised fluorescence results are reported in Figure E.1. More precisely:

- 15 bit-0 samples (Figure E.1a):

uniform, and except for the trivial case of BIT-COPY, our computations are not solvable in DLOGTIME. Thus parity, and the other non-trivial problems, are not solvable by the encoding function. Formalising all of this requires encoding tile sets as strings, and is straightforward by known methods, if a bit tedious.

[83] We use the same notion of binding energy as in [183]: `binding(A,B) = pfunc(A,B)-pfunc(A)-pfunc(B)`.

[84] `https://github.com/UC-Davis-molecular-computing/nuad`, project led by David Doty, UC Davis.

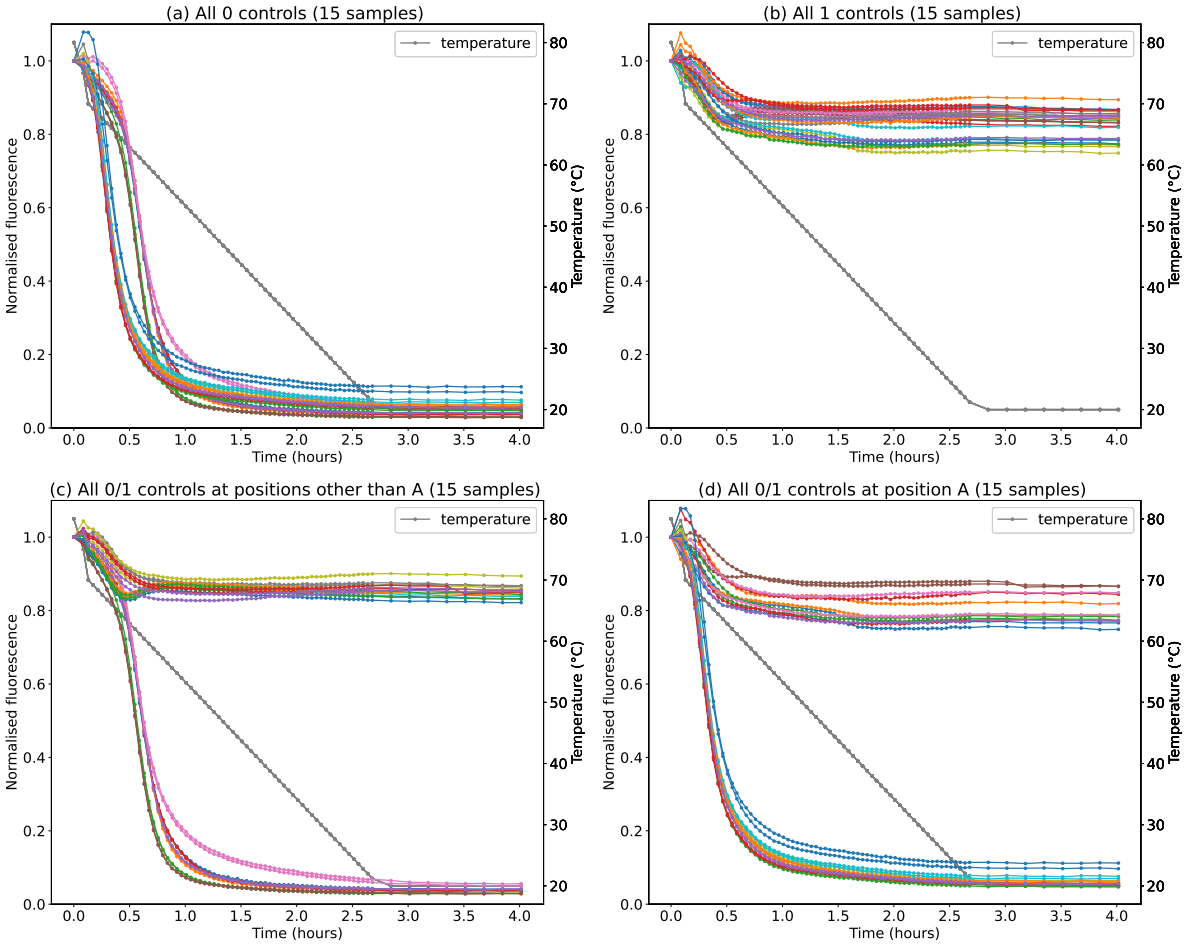


Figure E.1: 0/1 controls for the sequence independent reporting mechanism. Each control sample was divided into two qPCR wells: there are two curves per sample (same color). In (a) and (b) all the controls for reporting 0/1 are displayed. What varies between control samples can be: the reporting position, the strands present at non-reporting positions or the total amount of DNA in solution. We observe **bi-modality** in the controls reporting a 1: we can distinguish two subsets of curves, one centered around 0.8 and the other centered around 0.9. In (b) and (c), we explore the bi-modality by splitting all the controls in two: (c) controls placed that at positions other than A (i.e. B,C,D) and (d) controls placed solely at position A. We happen to have as many controls at position A as are at positions B,C,D. In (c,d) we see that the bi-modality in controls for 1 is only present among those at position A.

  - 8 samples at position A, no variation (i.e. 8 samples with exact same mix content)
  - 3 samples at position B, varying non-reporting strands and total quantity of DNA in solution (1x and 2x)
  - 2 samples at position C, varying non-reporting strands
  - 2 samples at position D, varying non-reporting strands

- 15 bit-1 samples (Figure E.1b):

  - 7 samples at position A, no variation (i.e. 7 samples with exact same mix content)
  - 2 samples at position B, varying non-reporting strands
  - 2 samples at position C, varying non-reporting strands
  - 4 samples at position D, varying non-reporting strands and total quantity of DNA in solution (1x,2x,4x)

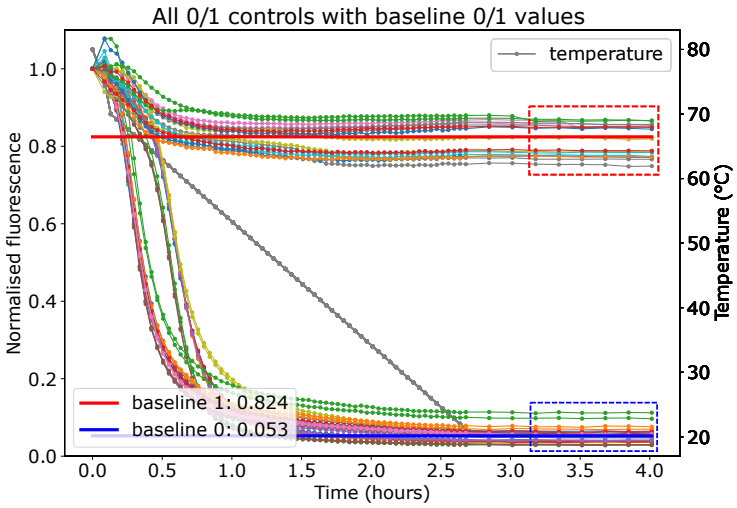


Figure E.2: baseline 0 and 1 values. Dashed boxes outline the data points averaged to compute the baseline values.

Our control dataset is biased at position A in the following ways:

- We have as many controls for position A alone (Figure E.1d) as for all other positions B,C,D (Figure E.1c).
- All the bit-1 (resp. bit-0) controls at position A follow the exact same recipe, there is no variation in non-reporting strands (there are none of them) and the total quanity of DNA in solution is the same (up to experimental error and precision).

Although we have performed the exact same 0/1 reporting mixes at position A, but on different days, we observe bi-modality in the distribution of the bit-1 samples, Figure E.1(d): we can distinguish two subsets of curves, one centered around 0.8 and the other centered around 0.9. This variance indicates sensitivity of the reporting mechanism to experimental conditions such as pipetting volume, strand concentration, strand synthesis purity, etc.

Baseline 0/baseline 1. We average the 6 consecutive data points at 20 °C after the third one (to let the signals settle) of each bit-0 (resp. bit-1) curve in Figure E.1(a) (resp. Figure E.1(b)) in order to define the baseline value of bit-0 (resp. bit-1). We find the values: baseline 0 = 0.053 and baseline 1 = 0.824, Figure E.2.

E.3 DNA SEQUENCES

Full length M13 scaffolds were purchased from Tilibit, all other DNA strands were purchased from IDT. The synthetic 120-base scaffold was PAGE purified.

E.3.1 1D Scaffolded DNA Computer

For the 1D Scaffolded DNA Computer, a total of 254 strands were designed, plus one scaffold strand whose sequence comes from two segments of M13 including one scaffold strand. Here we give some details on sequencing design, beyond those already given in Section 4.3.

Synthetic 120-base scaffold. We use a 120-base synthetic scaffold whose sequence is from two contiguous regions of M13 p7249. We divide the scaffold into 5 consecutive 24-base domains called A,B,C,D,E where E is on the 5' side of the scaffold:

- Domain E: `ATTTTTGATTTATGGTCATTCTCG`
- Domain D: `TTTTCTGAACTGTTTAAAGCATTT`
- Domain C: `GAGGGGGATTCAATGAATATTTAT`
- Domain B: `GACGATTCCGCAGTATTGGACGCT`
- Domain A: `CTGGCAAATTAGGCTCTGGAAAGA`
- Full scaffold sequence: `ATTTTTGATTTATGGTCATTCTCG TTTTCTGAACTGTTTAAAGCATTTG GAGGGGGATTCAATGAATATTTAT GACGATTCCGCAGTATTGGACGCT CTGGCAAATTAGGCTCTGGAAAGA`

Toeholds. As described in Section 4.3, Figure 4.5, we sought to design 8 information encoding toeholds (each 12-bases long) that can be attached to domains A,B,C,D in order to simulate tile colors. If we directly used such sequences at the 5' end, and their reverse complements at the 3'-end of strands we would permit formation of unwanted stable hairpins on those strands that advertising the same (tile) colour at both sides. Hence we instead designed 16 toeholds, i.e. two sets of 8 toeholds: one set at even scaffold positions (A, C, etc.) and the other at odd scaffold positions (B, D, etc.). This principle avoids unintentional hairpin formation when using these 16 strands. Here are the sequences of these 16 toeholds:

1. Toehold 0: `CTCATCCTGACC` (even), `CCTCTTCTCAGC` (odd)
2. Toehold 1: `TCAACTCCGTTC` (even), `CATCTCCGATCC` (odd)
3. Toehold 2: `AATGCCACCATT` (even), `TCTTTCCAAGCC` (odd)
4. Toehold 3: `ACAACCCTTGTC` (even), `TCAATCCTTGCC` (odd)
5. Toehold 4: `CTGTTCCCAACA` (even), `CACATCCCTGTT` (odd)
6. Toehold 5: `CACTACCAGTCC` (even), `CCATGTCCCATT` (odd)
7. Toehold 6: `ACACACACTGTC` (even), `CAACCAACGTTC` (odd)
8. Toehold 7: `TCACTTTCGTCC` (even), `TCACACTTCGTC` (odd)

Strands. We use the following naming conventions for naming strands: `3B4*` refers to the strand at scaffold position B, having toehold 3 on the 5' side and toehold 4 on the 3' side. B is considered an odd scaffold position (A is even) hence toehold 3 will be chosen to be odd: `TCAATCCTTGCC`. Toehold 4 will be even and reverse complemented: `TGTTGGGAACAG`. Finally, strand `3B4*` uses the reverse complement of scaffold domain B so it can bind to B. The complete sequence of strand `3B4*` is: `TCAATCCTTGCC AGCGTCCAATACTGCGGAATCGTC TGTTGGGAACAG`. In total there are 64 possible strands at each position B, C, D. Position A has only 8 strands since no strand ever preceeds it, and thus no left hand side toehold is needed. This gives a total of $8 + 64 * 3 = 200$ strands that are equipped with 5' and 3' toeholds. We also use strands specific to reporting: strand `3B` has no 3' toehold and is used to report a 1; strand `3BQ*` is equipped with 20-base domain Q* to report a 0; strand `F*C` is prefixed with 20-base domain F* in order to enable the reporting mechanism for reporting at position B (see Appendix E.2). This adds an extra $2 + 3 * 8 * 2 + 4 = 54$ strands for a total of 254 strands in use in the 1D Scaffolded DNA Computer.

E.3.2 2D Scaffolded DNA Computer

The 2D Scaffolded DNA Computer implementation used the same toeholds as the 1D system, as described in Section 4.5.

E.4 ADDITIONAL RESULTS

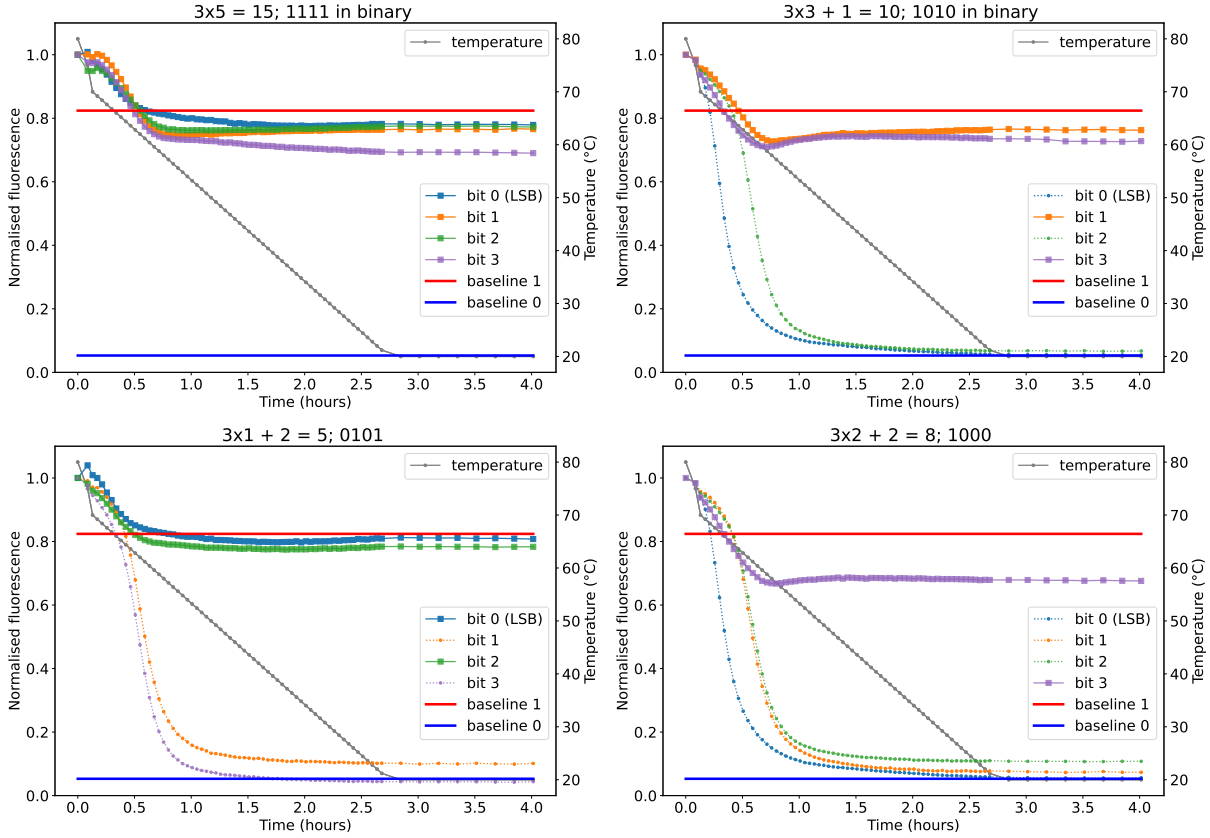


Figure E.3: Additional MULTIPLYBY3 results. (a) $3 \times 5 = 15$ (1111 in binary) (b) $3 \times 3 + 1 = 10$ (1010 in binary) (c) $3 \times 1 + 2 = 5$ (0101 in binary) (d) $3 \times 2 + 2 = 8$ (1000 in binary).