On the performance and programming of reversible molecular computers

'Engines of Parsimony'

Amelie Hannah Earley

March 2021

This thesis is submitted for the degree of
Doctor of Philosophy,

and was supervised by
Gos Micklem

in the
Department of Applied Mathematics and Theoretical Physics,

and
Trinity Hall,

of the
UNIVERSITY OF CAMBRIDGE.

This thesis was also advised by
Stephen Eglen,

and was examined internally by
Sebastian Ahnert

and externally by
Michael P. Frank

on the
30th of April, 2021.

to Ann,

reunited with Dennis,

and to Jim,

who we and Patsy yet miss

Acknowledgements

Gos, without whom none of this would be possible;
your guidance, identical sleep schedule, and tangents
have got me through this path ig-noble.
P.S. I forgive you for trying to kill me with peanuts.

Mum and Dad, you gave me life,
and helped me through my toil and strife,
not to mention my sisters three,
whose love and kinship supported me.

My office-mate Pete(r),
I miss our late-night pizza
and basketball sessions;
may MIT sustain your passions.

Swaraj, though you moved to Oxford,
your dependability never wavered;
your friendship is immeasuarable,
and willingness to read my drafts inexplicable.

Acknowledgements

Georgeos, in but a year of your acquaintance,
fast friends we have become;
though I have exploited it for articular comments,
fast friends I hope we go on.

Marianne, who accompanied me through much of my journey;
even though our relationship may not have survived the PhD,
I may not have survived the PhD if not for you,
for which you have my gratitude.

My office-mate subsequent Jiaee,
who cyclically accompanied me to Ely:
about the distance, you and Eleanor lied to me,
nonetheless, around your upbeat attitude is fun to be.

Finally, these acknowledgements would be incomplete,
wihout mentioning the bodies whose funding replete
upon me bestowed; DAMTP, Trinity Hall
and EPSRC, I thank you all.

Declaration

This thesis is the result of my own work and includes nothing which is the outcome of work done in collaboration except as declared in the Preface and specified in the text. It is not substantially the same as any that I have submitted, or, is being concurrently submitted for a degree or diploma or other qualification at the University of Cambridge or any other University or similar institution except as declared in the Preface and specified in the text. I further state that no substantial part of my thesis has already been submitted, or, is being concurrently submitted for any such degree, diploma or other qualification at the University of Cambridge or any other University or similar institution except as declared in the Preface and specified in the text. It does not exceed the prescribed word limit for the relevant Degree Committee.

Variants of the core chapters of this thesis, Chapters II to V, have been posted as preprints [1, 2, 3, 4]. This thesis is my own work, though I will exercise nosism throughout. This work was supported by the Engineering and Physical Sciences Research Council, project reference 1781682 [5].

Hannah Earley[1]
December 2020

[1] h.earley@damtp.cam.ac.uk, orcid.org/0000-0002-6628-2130

Abstract

If the 20$^{\text{th}}$ century was known for the computational revolution, what will the 21$^{\text{st}}$ be known for? Perhaps the recent strides in the nascent fields of molecular programming and biological computation will help bring about the 'Coming Era of Nanotechnology' promised in Drexler's *Engines of Creation* [6]. Though there is still far to go, there is much reason for optimism. This thesis examines the underlying principles needed to realise the computational aspects of such 'engines' in a performant way. Its main body focusses on the ways in which thermodynamics constrains the operation and design of such systems, and it ends with the proposal of a model of computation appropriate for exploiting these constraints.

These thermodynamic constraints are approached from three different directions. The first considers the maximum possible aggregate performance of a system of computers of given volume, V, with a given supply of free energy. From this perspective, reversible computing is imperative in order to circumvent the Landauer limit [7, 8]. A result of Frank [9] is refined and strengthened, showing that the adiabatic regime reversible computer performance is the best possible for any computer—quantum or classical. This therefore shows a universal scaling law governing the performance of compact computers of $\sim V^{5/6}$, compared to $\sim V^{2/3}$ for conventional computers. For the case of molecular computers, it is shown how to attain this bound. The second direction extends this performance analysis to the case where individual computational particles or sub-units can interact with one another. The third extends it to interactions with shared, non-computational parts of the system. It is found that accommodating these interactions in molecular computers imposes a performance penalty that undermines the earlier scaling result. Nonetheless, scaling superior to that of irreversible computers can be preserved, and appropriate mitigations and considerations are discussed. These analyses are framed in a context of molecular computation, but where possible more general computational systems are considered.

The proposed model, the $\aleph$-calculus, is appropriate for programming reversible molecular computers taking into account these constraints. A variety of examples and mathematical analyses accompany it. Moreover, abstract sketches of potential molecular implementations are provided. Developing these into viable schemes suitable for experimental validation will be a focus of future work.

Contents

List of Figures

List of Listings

I. Introduction

In recent years, unconventional forms of computing ranging from molecular computers made out of DNA to quantum computers have started to be realised. Not only that, but they are becoming increasingly sophisticated and have a lot of potential to influence the future of computing. In this work, we will look at the intersection of molecular computing and another interesting class of unconventional computer: that of reversible computers.

Molecular Computation The field of Molecular Computation began in 1994 with Len Adleman's proof-of-concept use of DNA [10] in the solution of the classic Hamiltonian path problem, as illustrated in Figure I.1. DNA and other nucleic acids proved an apt choice for building molecular/chemical computers, due to its information bearing capacity and the strong preference that single strands show for binding to complementary sequences, not to mention the low cost with which it can be synthesised and analysed as well as the large body of knowledge available from decades of biochemical research. Inspired by Adleman's paper, a community of researchers designing and building DNA computers arose in concert with the DNAX (and shortly thereafter, the complementary FNANOX) series of conferences[1].

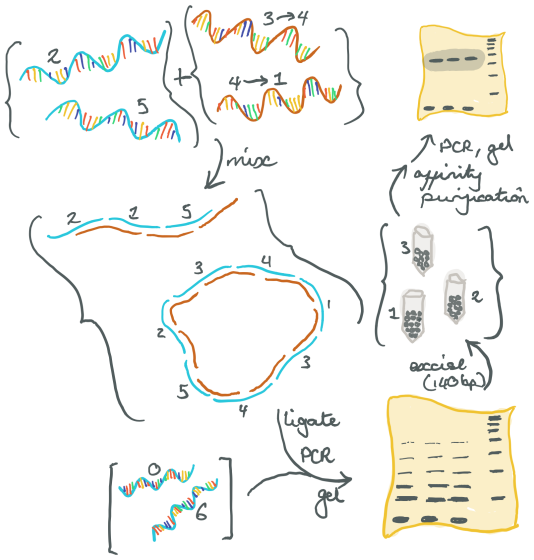


Figure I.1: An illustration of Adleman's DNA scheme for solving the Hamiltonian path problem.

DNA, or deoxyribonucleic acid, is a family of polymeric molecules that is employed by all known organisms to store their genomic information, amongst other uses. Each molecule can be characterised by its sequence: a string of the monomers adenine (A), cytosine (C), guanine (G) and thymine (T). These monomers are each bound to a deoxyribose sugar molecule, which are joined together by phosphodiester bonds. Whilst this ability to store information through the arbitrary combination of these monomers is advantageous, the key to its success is the

[1] https://isnsce.org/conferences/

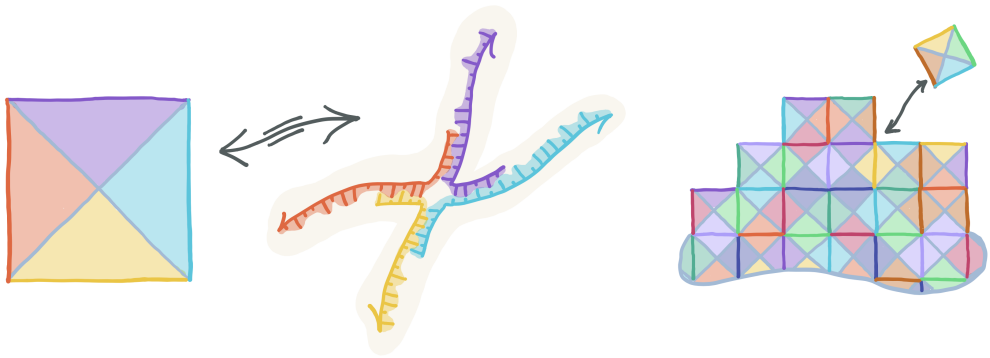
special hydrogen-bonding properties that these monomers possess. When two DNA strands are aligned adjacently, if adenine and thymine are opposite each other they will be perfectly positioned to form two favourable hydrogen bonds; similarly, if cytosine and guanine are opposite each other they will be perfectly positioned to form three favourable hydrogen bonds. Whilst other combinations may also form hydrogen bonds, the positions are suboptimal and thermodynamically unfavourable when compared to the pairs `A-T` and `C-G`. In nature, DNA is found almost exclusively in double-stranded form, whereby each strand has a sequence complementary to the other: if one strand has sequence `GGCACAACGT` then the other will have sequence `CCGTGTTGCA`. These complementary strands then *hybridise* to one another. In fact, this is an oversimplification: in addition to their information content, DNA molecules have an intrinsic orientation, conventionally notated via the free 5′ phosphoryl and 3′ hydroxy groups on the terminal ribose sugars. DNA strands bind anti-parallel, and so the sequences should be written `5'-GGCACAACGT-3'` and `5'-ACGTTGTGCC-3'`. Schematically, this looks like

```
5' — G — G — C — A — C — A — A — C — G — T — 3'
     |   |   |   |   |   |   |   |   |   |
3' — C — C — G — T — G — T — T — G — C — A — 5'
```

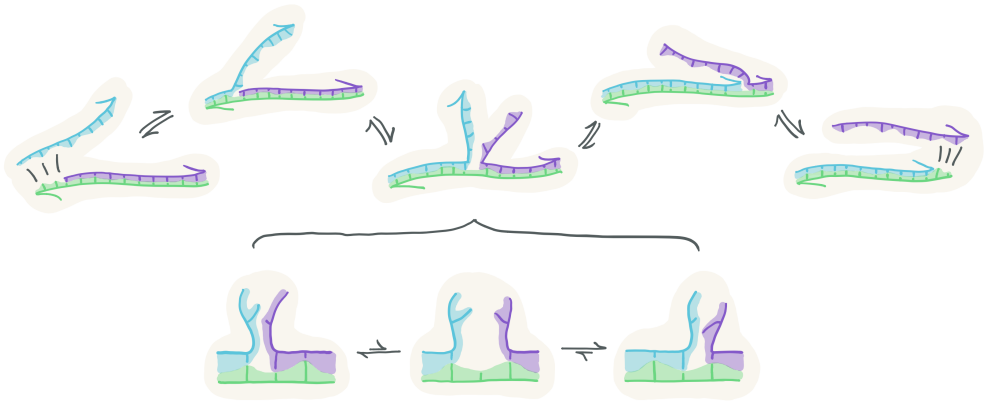
but in reality the strands twist together into a double helix, the structure of which was found by Rosalind Franklin, James Watson and Francis Crick [11]. Eponymously, the characteristic complementary 'base pairing' is known as Watson-Crick base pairing. The DNA computing motifs designed by Adleman and others make use of this Watson-Crick base pairing, but exploit partial hybridisation and the dynamics and interactions of single-stranded DNA.

Whilst Adleman's proof of concept required manual actuation of each step of the computation, and was limited to NP-complete problems, schemes demonstrating universal computation and autonomous operation were soon developed. An example from the first of the DNAX series is the proposed DNA and restriction enzyme implementation of a Turing Machine of Rothemund [16]. Whilst universal in principle, it was still not autonomous and has never been experimentally realised due to its complexity. Nevertheless, successful approaches and motifs employing both DNA and enzymes have been developed, such as the PEN toolbox of Yannick Rondelez's group [17] and the PER reaction of Jocelyn Kishi [18]. Arguably the most popular approaches in the field are enzyme-free approaches. Two famous examples are given by the Tile-Assembly Model (TAM), introduced by Erik Winfree [13], and DNA-strand displacement (DSD) schemes, first popularised by Georg Seelig et al [19]. These are illustrated in Figure I.2.

Some impressive feats have been achieved, from neural networks [20, 21] to cargo sorting robots [22] to reprogrammable iterative binary logic circuits [23]. Nevertheless, DNA computing is not without its challenges. For example, DSD systems often exhibit spurious 'leak' reactions, and optimisation and tricks are required to mitigate this [24]. Moreover, traditional DSD systems occupy the entire reaction volume, which can therefore only perform a single computation. Strictly speaking, one could perform multiple computations by using orthogonal DNA sequences such that a set of DNA



(a) The canonical embodiment of the Tile Assembly Model: in the abstract, square 'tiles' are designed with four sticky edges. The edges are assigned colours such that only edges of the same colour will stick to one another. By choosing the tile set, different patterns can be assembled, both deterministic and non-deterministic. In fact, it was shown by Hao Wang in the '60s [12] that appropriately chosen tile sets can simulate a Turing Machine, and so arbitrary computation is possible. During his PhD, Erik Winfree showed how to realise these tilesets using DNA [13], per the illustration above. Although in practice the tiles adopt a non-square shape, the same general properties of Wang Tiles are observed. In fact, not only is the TAM Turing complete, but there even exists a universal tile set that can simulate any other [14].



(b) DNA-strand displacement schemes rely on the primitive illustrated here of branch migration, in which an incumbent strand B (purple) is displaced by an incident strand A (blue). We start with B hybridised to a gate strand (green), except that the gate strand is slightly longer and has an exposed single-stranded 'toehold' domain. This toehold is complementary to part of A, whereby A is reversibly recruited to the gate strand. Moreover, the remainder of A is complementary to much of the gate strand that B is bound to, and so can compete with A for hybridisation. In a thermal context, then, B may transiently 'fray' away from the gate strand at its ends, providing an opportunity for A to partially hybridise in its place. A random walk ensues, in which the junction between A and B may migrate in either direction with roughly equal chance. If the branch fully migrates to the other end, then B is left bound by only a short region: another toehold. As with A, B can then reversibly attach or detach from the gate strand. If A was slightly longer and bound the entirety of the gate strand, then the reaction could be (and routinely *is*) made effectively irreversible. Sophisticated cascades can then be prepared whereby B may interact with downstream gates, or where a gate may bind multiple strands. Other sophisticated behaviours can be realised, for example, by the use of common or orthogonal toeholds and other sequence domains. The culmination of this is that, up to a tunable probability of error, DSD supports universal computation [15].

Figure I.2: Illustrations of the principles underlying two common schemes of DNA computation.

species performing one computation doesn't interact with another performing a different computation; however, toeholds typically can't be much longer than 6 nucleotides in length to ensure reversibility of association, limiting the set of distinct toeholds to $4^6 = 4096$. Worse still, it transpires that DNA's sequence specificity is not absolute, and so only a fraction of this design space is useful: in fact, as a single mismatch or two does not prevent hybridisation but does affect the rate of hybridisation, it is often exploited for tuning kinetics [25, 26, 27, 28]. We have not even mentioned the influence of palindromic sequences and other motifs manifesting non-trivial secondary structures. This seriously limits the complexity that can be realised in these systems, because the CRNs that are compiled into DSD schemes (using, e.g., `VisualDSD` [29]) do not make use of complexation of CRN species, and therefore each sub-module in a program requires a fresh set of CRN species with their own unique DNA sequences—even if the sub-module is identical to another, just used in a different place in the program. Additionally, these DNA computers are analogue which significantly limits their computational complexity if one wants to be certain of the result [30], although if one is comfortable with slightly less certainty then better complexity is possible [15]. They are also very slow, taking on the order of several hours or even days to progress. The *tour-de-force* examples of neural networks [20] and (4-bit) square rooting [31], for example, took on the order of *10 hours* to converge to an answer.

If instead one makes use of localisation, as with the TAM, then much of these issues with DSD can be avoided [32]: localising DSD species to a surface allows 'circuits' to be constructed which are fast, modular, and can reuse sequence domains. The problem is that constructing these is substantially more difficult than free-solution DSD, and that gate strands are 'consumed' during computation making these one-shot reactions. Free-solution DSD *also* consumes its gate strands, but at least in this case it is possible (if non-trivial) to continually supply fresh gate and fuel species and remove waste complexes. The TAM does not get off completely unscathed either, and methods to suppress errors to mitigate its share of problems have also been developed [33].

This is not to say that computing with molecules such as DNA is intrinsically flawed, but the difficulties in overcoming these issues has left many in the field pessimistic about initial beliefs that DNA could be used to perform arbitrary computation in an effective manner. Instead, the general consensus is that DNA computing may be best used to perform small logical calculations as part of a synthetic biological system. My view is more optimistic, in that I believe that an as-yet undiscovered DNA computing motif exists that will solve all these problems simultaneously. Whilst perhaps the existence of such a panacea is wishful thinking, I believe that the abstract molecular schemes discussed in Sections IV.4 and V.2 lend evidence to this view, and developing these further will be the subject of future work.

Reversible Computation As mentioned, the other class of unconventional computer we shall consider is that of reversible computers. Reversible computers impose the requirement that all state transitions be invertible, and therefore conserve information.

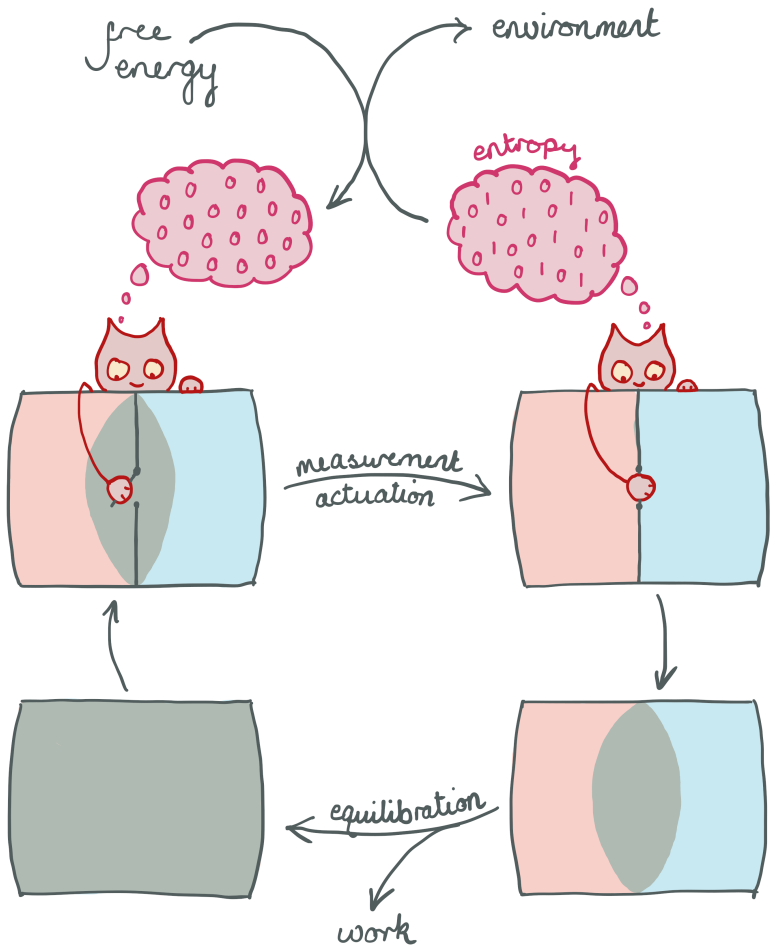
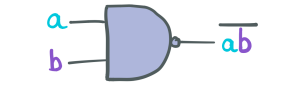


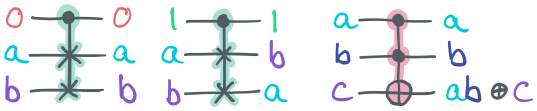
Figure I.3: Maxwell's Dæmon (1867) is a thought experiment which, if valid, would offer a method to violate the second law of thermodynamics. Consider a box divided in two; to the left of the divider is a red gas, and to the right a blue gas. If the divider is removed, then mechanical work can be extracted as the gases mix and equilibrate. Now suppose the divider is re-inserted, and endowed with a small window whose hinge is frictionless and can therefore be opened and closed without energetic cost. Imagine that a dæmon or other such entity, with particularly keen eyesight and nimble fingers, is positioned at the window. If she observes a red particle in the right compartment moving towards the window, or a blue particle in the left compartment moving towards the window, she briefly opens it to allow the particle's passage; otherwise she leaves the window closed. Over time, this process will bring the box back to its initial state, thereby apparently reducing its entropy.

The problem is that this process requires the dæmon to learn information about the state of the gas particles. In effect, the entropy has simply moved from the state of the particles to the state of the dæmon's memory. No entropy reduction has occurred. Moreover, eventually her memory will be filled up, and she will likely want to 'erase' it to continue her task. Forgetting all this information will correspond to moving the entropy into the state of another system. If this entropy now manifests as heat, then Landauer showed that discarding a quantity of information ΔI requires the production of heat $\Delta Q \geq kT\Delta I$ where k is Boltzmann's constant, T the temperature, and with equality only in the limit of a thermodynamically reversible process (i.e. taking infinite time).

The reason for this restriction is that the laws of physics, as far as is currently known[2], are fundamentally reversible. In contrast, conventional computers are irreversible in the sense that they do not conserve information and that it is not generally possible to deterministically wind back the state of computation. This has surprising consequences: from a thermodynamic perspective, it transpires that this loss of information manifests as a commensurate increase in entropy, whilst from a quantum perspective, the loss of information of some quantum state necessarily results in decoherence of any quantum state entangled with said state. This quantum mechanical behaviour is not of great concern in classical computers, but is untenable for quantum computers and hence they must necessarily be programmed reversibly.



(a) NAND and FANOUT gates



(b) Fredkin and Toffoli gates

(c) Deutsch Gate

Figure I.4: Three sets of universal gates. (a) Two gates which can be used to synthesise any Boolean function $\mathbb{B}^m \to \mathbb{B}^n$. (b) Two reversible logic gates, *each* of which can simulate NAND and FANOUT gates given suitable ancillary bits (inputs fixed to either 0 or 1). (c) An example gate that is universal for quantum computation: that is, any quantum circuit can be constructed from a series of Deutsch gates (parameterised by the angle θ).

Expanding more on the entropic manifestation of information loss, simulating the ability to discard information in a reversible setting touches deeply on the connection between thermodynamics and information theory, and was formalised by Szilard [8] and Landauer [7] in the twentieth century in order to resolve Maxwell's eponymous thought experiment, the Maxwell Dæmon (Figure I.3). For every bit of information discarded, a commensurate entropy increase of at least $k \log 2$ is produced elsewhere, where k is Boltzmann's constant. Physically, the information is encoded into environmental degrees of freedom, and this information promptly decorrelates with the original computational system. This decorrelation is effectively irreversible, and hence the information becomes entropic. More generally, whenever a quantity of information I bits is discarded, an entropy increase of $\Delta S \geq kI \log 2$ manifests, with equality only in the limit of thermodynamic reversibility in which the discarding process takes infinitely long. If this entropy were allowed to remain in the computational system, it would eventually accumulate to such an extent that it would interfere with the well ordered operation of the computational mechanism. As a visceral example, should the entropy take the form of heat, $\Delta Q = T\Delta S$

[2]This includes quantum mechanics; the Schrödinger equation specifies that the time evolution operator is unitary and hence invertible. Furthermore, it seems that even the supposedly irreversible process of wavefunction collapse is in fact a reversible one of progressive decoherence, as espoused by the Many-Worlds Interpretation [34].

where T is the system temperature, the system will eventually transmute into a form incompatible with its function—such as melting or turning into a plasma—as the temperature becomes too great. Even if the system is physically heat-resistant, the increased entropy will result in errors so frequent that almost all the computational capacity is directed to error correction, slowing productive computation to a crawl.

At room temperature, this heat generation will be on the order of $\sim 1 \times 10^{-21}\,\mathrm{J} \sim 1 \times 10^{-2}\,\mathrm{eV}$ per bit discarded. Conventional computers typically expend a factor of around $1{\times}10^{8}$ times this much for reasons relating to reliability, speed, and the relatively large size of their transistors in comparison to the atomic scale. It is somewhat challenging to precisely quantify this overhead as the information processing capacity of a modern consumer processor is difficult to ascertain, due to the extensive use of pipelining, vectorisation and multiple-instruction dispatch. Nevertheless, we can obtain a reasonable estimate. arm's processors are well known for emphasising power efficiency, and are used extensively in mobile electronics as well as some laptops and desktops (notably, three of Apple's new device lineup). Considering arm's A-78 processor micro-architecture [35], we find numbers of around 1 W/core, with each core running at a clock speed of 3 GHz and executing up to 6 (macro)instructions per clock cycle with a width of up to 128 bit. Taken together, these give an energy dissipation of roughly $1.5{\times}10^{-13}\,\mathrm{J\,bit^{-1}}$. In contrast, at a temperature of around 300 K, Landauer's bound gives just $2.9{\times}10^{-21}\,\mathrm{J\,bit^{-1}}$, a $5{\times}10^{7}$ overhead for the arm processor. Intel's chips have thermal design powers typically exceeding 100 W, but have compensatingly greater vectorisation widths of some instructions, such that a similar Landauer overhead can be achieved when fully exploiting the processor's capabilities.

The origin of irreversibility in conventional computers can be subtle, but it is in fact ubiquitous. This may be considered to ultimately originate in early mathematical models of computations such as the Turing Machine and Lambda Calculus in which overwriting and discarding, respectively, are implicit to the primitive operations of the models. At the lowest levels of abstraction of modern computers, semiconductor logic gates and buses are operated in an irreversible manner: somewhat oversimplifying, traditional CMOS-style logic dissipates energy every time the transistor states switch, and during each clock cycle an equilibration process occurs in which the nodes of the circuit are connected to bus lanes at various reference voltages and any transient voltage disparity is eliminated. Both of these processes are capable of discarding information in the system, although exactly where in these processes the information erasure should be attributed is a non-trivial matter. In fact it is possible to drive CMOS-style logic gates *adiabatically* [36, 37, 38], achieving asymptotically zero dissipation in the limit of infinitely slow switching. This is assuming that the instructions are logically invertible, perhaps subject to certain pre- and post-conditions in which case it is sometimes referred to as *conditionally reversible*. Reintroducing irreversibility is possible, but dissipation of information should be performed separately to maintain adiabaticity in the operation of the logic gates. The idea of using reversible logic gates in irreversible computation may seem counter-productive, but it is a useful potential way to make substantial gains in energy efficiency and to get closer to the Landauer bound.

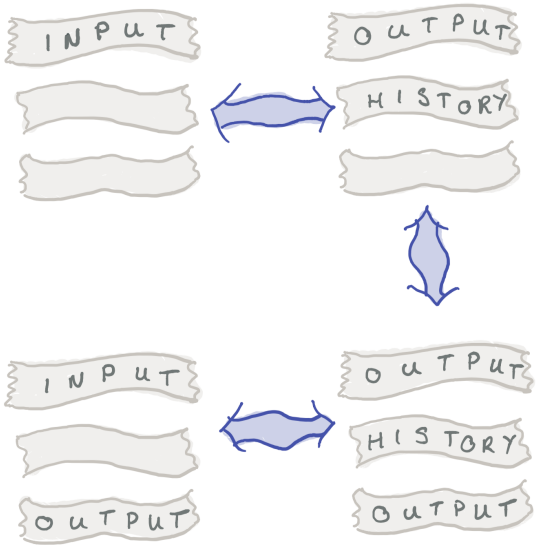
At higher levels of abstraction potential sources of information erasure include the obvious, such as overwriting/mutating a variable, to the less obvious such as variables going out of scope or ignoring a function's return value. Even more subtly, any iteration or recursion which does not maintain a record of its initiation condition as well as its termination condition is in fact irreversible, as shall later become clear.

With such ubiquitous application of information-destructive primitives, it is hard to see how algorithms can be rewritten reversibly. Indeed, whilst Landauer could envisage what logically reversible logic gates might look like, even describing [7] what would eventually become known as the Toffoli or `CCNOT` gate (see Figure I.4), he came to the conclusion that one would end up having to manually keep track of the lost information thus rapidly filling the computer's memory with useless 'garbage' or 'history' data. 12 years later, Charles Bennett—often considered the 'Father of Reversible Computing'—developed a remarkably simple algorithm [39] (Figure I.5a) for embedding any irreversible function $x \mapsto f(x)$ as the reversible injection $x \leftrightarrow (x, f(x))$ without producing any net garbage. Later, he extended this algorithm [40] to reduce the amount of interim garbage data produced via a recursive application of the original algorithm, the general principle of which is illustrated in Figure I.5b.

With the viability of reversible computing proven, the stage was then set for more faithful programming of reversible computers to be demonstrated beyond simple injective embedding of irreversible programs. Whilst Bennett did introduce the idea of a Reversible Turing Machine, arguably the first description of a reversible computer that might be physically realisable was given by Fredkin and Toffoli [42]. Fredkin and Toffoli introduced conservative logic, in which the number of 1 bits (and optionally also the number of 0 bits) is conserved; they proceeded to show that this logic could be implemented by a system of ideal elastic 'billiard balls', projected across an ideal frictionless surface decorated with ideally rigid walls. The motivation for this construction was that it is precisely the sort of system considered by classical mechanics. If one could build such a billiard-ball computer, then aside from the initial kinetic energy supplied to the system, there would be no need to supply additional energy nor any dissipation; moreover, one could recover the kinetic energy at the end for amortised-free computation. To illustrate the principles of billiard-ball computation, Andrew Ressler's implementation of a Fredkin gate is shown in Figure I.6. Going further, Ressler designed an entire reversible computer processor—complete with arithmetic logic unit—within the billiard-ball scheme for his Master's thesis [41], demonstrating that not only were reversible computers theoretically as capable as irreversible ones from a Turing universality perspective, but that they were also practically as capable.

Unfortunately, Bennett's analysis of the billiard-ball model showed it to be impracticable as a fully dissipationless model of reversible computation:

> "Even if classical balls could be shot with perfect accuracy into a perfect apparatus, fluctuating tidal forces from turbulence in the atmospheres of nearby stars would be enough to randomise their motion within a few hundred collisions. Needless to say, the trajectory would be spoiled much sooner if stronger nearby noise sources (e.g., thermal radiation and conduction) were not eliminated." — Bennett [43]



(a)

TABLE 2
Reversible simulation in time $O(T^{\log 3/\log 2})$ and space $O(S \cdot \log T)$.

Stage	Action	Checkpoints in storage (0 = initial ID, checkpoint $j = (jm)$th step ID)
0	Start	0
1	Do segment 1	0 1
2	Do segment 2	0 1 2
3	Undo segment 1	0 2
4	Do segment 3	0 2 3
5	Do segment 4	0 2 3 4
6	Undo segment 3	0 2 4
7	Do segment 1	0 1 2 4
8	Undo segment 2	0 1 4
9	Undo segment 1	0 4
10	Do segment 5	0 4 5
11	Do segment 6	0 4 5 6
12	Undo segment 5	0 4 6
13	Do segment 7	0 4 6 7
14	Do segment 8	0 4 6 7 8
15	Undo segment 7	0 4 6 8
16	Do segment 5	0 4 5 6 8
17	Undo segment 6	0 4 5 8
18	Undo segment 5	0 4 8
19	Do segment 1	0 1 4 8
20	Do segment 2	0 1 2 4 8
21	Undo segment 1	0 2 4 8
22	Do segment 3	0 2 3 4 8
23	Undo segment 4	0 2 3 8
24	Undo segment 3	0 2 8
25	Do segment 1	0 1 2 8
26	Undo segment 2	0 1 8
27	Undo segment 1	0 8

(b)

Figure I.5: Two of Bennett's algorithms for reversibly simulating irreversible programs.

(a) Bennett's initial algorithm [39] for reversibly simulating an irreversible program with no net garbage data, except for a retained copy of the original input. In the same paper, Bennett described a reversibilised analogue of the Turing Machine (TM), the *Reversible Turing Machine* (RTM), whose every operation is logically invertible and which could simulate any Turing Machine by placing the garbage data on a separate tape instead of erasing it. Bennett's algorithm makes use of the reversibility of this process: after first running the simulated TM forwards, we peek at the output and make a copy onto a third tape; we then forget about the third tape and reverse the simulation on the first two tapes, consuming the garbage and output and yielding the original input. Copying can be performed reversibly by, for example, using the output as a *one-time pad* to encrypt the contents of the third tape (if the RTM alphabet is binary, this is just the `XOR` operation). If the third tape is empty, then this will result in a copy; if it is not, then the result will generally be garbled.

(b) A reproduced figure of Bennett's 'pebbling' algorithm [40], illustrating the general principle. The pebbling algorithm runs the algorithm of Figure I.5a partially forwards and backwards, making and un-making partial copies of the intermediate states of the computation. This yields a spatial overhead logarithmic in the total computation time, instead of linear as with the original algorithm. The overhead in the time complexity is more significant however, but Bennett presented even more sophisticated algorithms to render the exponent arbitrarily small.

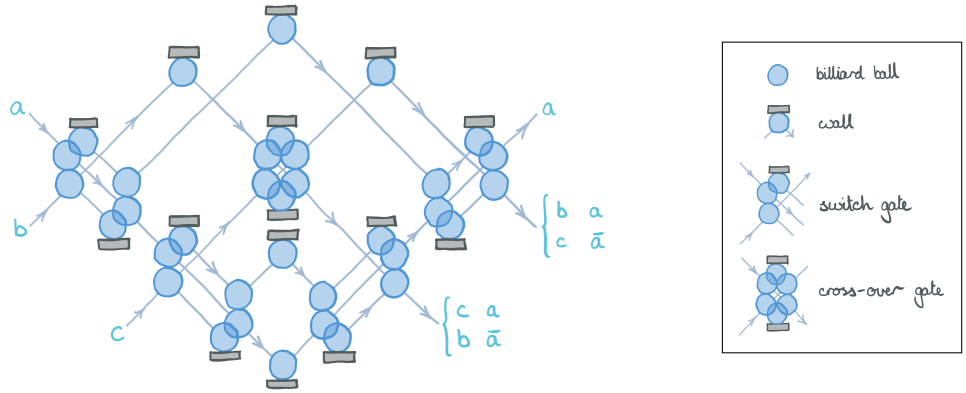
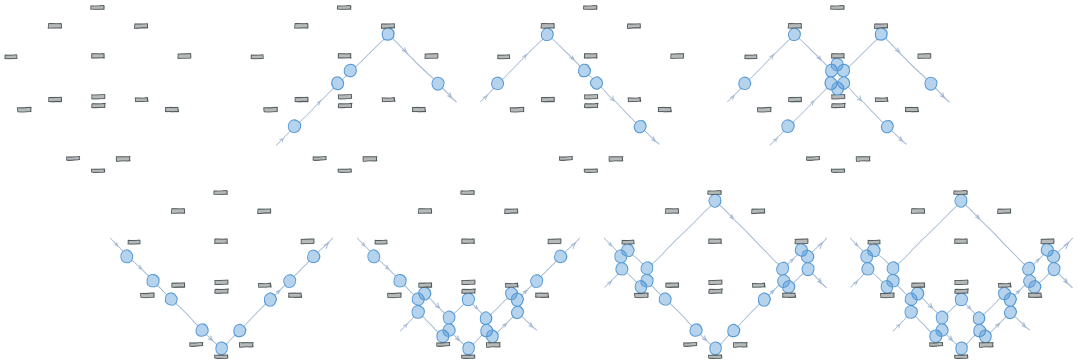


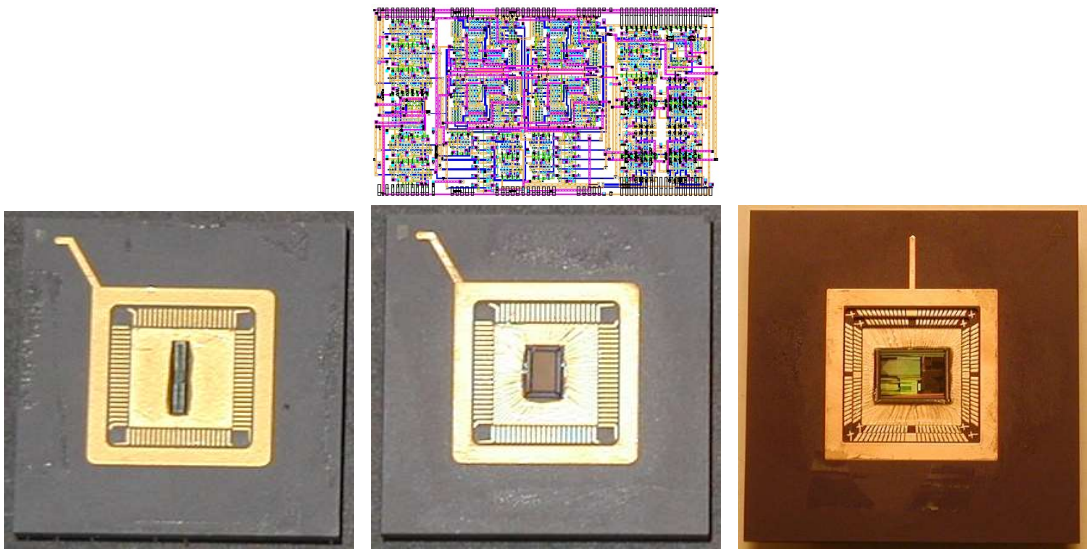
Figure I.6: A billiard-ball implementation of a Fredkin gate due to Ressler [41]. As primitives, it makes use of eight 'switch' gates, one 'cross-over' gate (middle), and five lone walls for simple redirection. The implementation differs slightly from the canonical Fredkin gate in that the second and third outputs are swapped. Note that not all the ball positions depicted necessarily occur for each possible combination of inputs; an input is given by the presence or absence of a ball for each of a, b and c, and these balls are synchronised in that at any given time they are aligned vertically, as shown in the below gate instantiations:



We shall have more to say about this in Chapter II, but since then a number of adiabatic designs of classical reversible computers have been developed and experimentally verified, not to mention the recent progress in building quantum computers. We highlight the *Pendulum Instruction-Set Architecture* (PISA), first introduced by Vieri [44] and later refined by Frank [9] and Vieri [45]. This architecture was successfully realised adiabatically with semiconductors by employing the 'Split-Level Charge Recovery Logic' (SCRL) motif [37], as shown in Figure I.8. Adiabatic reversible computers achieve far lower dissipation than irreversible computers, but still dissipate energy at a rate inversely proportional to the time per operation. Questions of whether or not lower energy dissipation, or even the 'holy grail' of fully dissipationless computation, are achievable have continued to be asked[3], and we believe we have answered these in the negative in

[3]Such as at the recent 2020 *Physics and Engineering Issues in Adiabatic/Reversible Classical Computing Visioning Workshop*.

Chapter II.

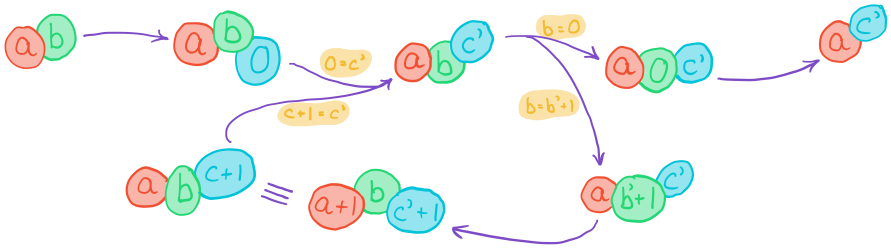


(a) TICK (b) FLATTOP and its unit cell (c) PENDULUM

Figure I.8: Three iterations of experimentally realised reversible processor between 1996 and 1999 by Frank, Ammer, Love, Rixner and Vieri. TICK (1996) was an 8-bit subset of PISA, but was non-adiabatic. FLATTOP (1996) was an adiabatic programmable gate array implementing the *Billiard Ball-Model Cellular Automaton* (BBMCA) of Margolus [46]. PENDULUM (1999 [45]) was the culmination of Vieri's work for his graduate thesis, and a 12-bit implementation of PISA.

Programming Reversibly With such ubiquitous application of information-destructive primitives, it is hard to see how algorithms and programs can be rewritten reversibly. In Chapter V we shall introduce a model of reversible computation appropriate for Brownian computers such as the molecular computers discussed earlier, in addition to many examples illustrating the principles of programming reversibly. Learning to do so requires a paradigm shift in one's thought process, in much the same way as an imperative/procedural programmer undergoes when learning a functional programming language for the first time. Reversibility as a concept is orthogonal to other language paradigms such as imperative/procedural, functional, and declarative, and indeed examples of reversible languages exist for each of these (we believe the ℵ-calculus and `alethe`, introduced in Chapter V, constitute the first example of a reversible declarative language).

To program reversibly takes some care. At a high level, one must ensure that each block (whether a function, a subroutine, a loop, a conditional branch, a statement, etc) is information preserving: that is, there must be a bijective map between the set of *legal* inputs and *legal* outputs. We say legal, because it is possible for some inputs to not map to outputs and vice-versa, instead producing a logical error during execution or entering an infinite loop. For example, consider Bennett's embedding protocol applied



(a) The logical state transition diagram for the reversible addition. This is a reversible embedding of the Peano-axiomatic definition of natural number addition, and therefore assumes as primitives that we can increment/decrement and check whether or not a number is zero.

$$(3,4) \leftrightarrow (3,4,0) \overset{\substack{c=0=0\\ b=4\neq 0}}{\longleftrightarrow} (4,3,1) \overset{\substack{c=1\neq 0\\ b=3\neq 0}}{\longleftrightarrow} (5,2,2) \overset{\substack{c=2\neq 0\\ b=2\neq 0}}{\longleftrightarrow} (6,1,3) \overset{\substack{c=3\neq 0\\ b=1\neq 0}}{\longleftrightarrow} (7,0,4) \overset{\substack{c=4\neq 0\\ b=0=0}}{\longleftrightarrow} (7,0,4) \leftrightarrow (7,4)$$

(b) An example execution path of the program shown in (a). Seen in the forward direction, we are adding 3 and 4 to get 7; seen in the reverse, we are subtracting 4 from 7 to get 3.

Figure I.10: An example of the reversible control flow for a possible reversible implementation of addition of two natural numbers, a and b. In order to be reversible, there must be a bijection between legal inputs and legal outputs and one such way is to retain one of the inputs—here b. This differs from the Bennett-style embedding, which would retain both inputs i.e. $(a,b) \leftrightarrow (a,b,a+b)$, and by being more parsimonious in our use of information we obtain the converse operation—subtraction—for free.

to the factorial function. This would produce a reversible program performing the map $(n) \leftrightarrow (n,n!)$, e.g. $(4) \leftrightarrow (4,24)$. If, however, one attempted to feed in the output $(4,30)$ and run this program backwards then an error[4] would occur because there is no input mapping to this output. Once one has ensured that a block is information preserving, the next step is to try to decompose it into smaller information-preserving blocks, just as with regular programming. Expertise with this comes with practice, and my experience with reversible programming leads me to believe that it is possible to become just as proficient with reversible programming as with irreversible. Whilst programming reversibly is to be preferred where practicable, a useful 'trick' for implementing a difficult bijection f can be achieved by first writing irreversible programs for $f : x \mapsto y$ and $f^{-1} : y \mapsto x$, embedding these using Bennett's algorithm as $f' : x \leftrightarrow (x,y)$ and $(f^{-1})' : y \leftrightarrow (x,y)$, and then constructing the composition $((f^{-1})')^{-1} \circ f' : x \leftrightarrow y$.

Arguably the most challenging concepts to learn are those of reversible control flow: conditional branching and loops. Provided the information used to discriminate the different cases in a branch is available before and after, then this branching is clearly reversible. The problem comes when merging these branches later: in irreversible languages this merging of control flow is implicit, but in a reversible language we must

[4] If we assume this is implemented on an RTM, then the error occurs when reversing the copy step: tapes 1 and 3 are initialised to 4 and 30 respectively, then the irreversible factorial TM is run forwards on tapes 1 and 2 to obtain 24 and some entropic garbage; the RTM then attempts to consume the copy of 24 on tape 3, but $30 \neq 24$ and so the machine will either stall or its state may become corrupt and meaningless.

ensure that we are able to tell from which branch we came immediately after merging and so merging must be made explicit. In fact, reversibility provides an elegantly symmetric solution to this without introducing a new primitive for merging. The idea is that merging is simply branching in reverse, and so one reuses the branching primitive and supplies a set of orthogonal conditions that must be satisfied by each branch of the computation before control is merged. If the information used to perform the original branch is preserved, then the original conditions can be reused; it is not necessary that the final conditions be the same as the initial, however, and this can often be used for good effect when data is transformed to a different—but provably isomorphic—representation. A loop is similar to a two-branch conditional, except that instead of the execution direction on both sides being in the same direction, they are in opposite directions in order to produce a loop. The consequence is that effectively the entry to a loop is a merge of control flow: one must know whether a loop iteration is being entered for the first time, or as a continuation, so that on reversal it is possible to 'un-enter' the loop. The termination condition remains the same as in the irreversible case: at the end of each iteration, it is checked whether we should continue looping or exit the loop. These principles are illustrated abstractly in Figure I.9, more concretely in Figure I.10, and more extensively in Chapter V.

We conclude with a brief review[5] of some (classical) reversible programming languages.

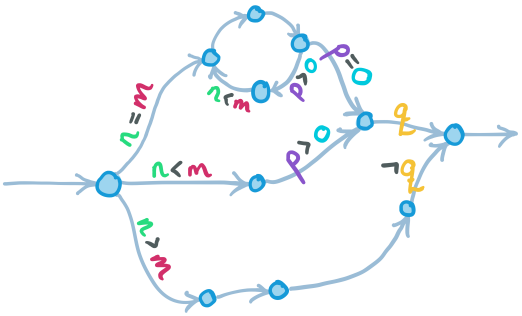


Figure I.9: An illustration of the principles of reversible control flow in the form an abstract state transition diagram. The meanings of the variables are not important. Arrow direction specifies the 'forwards' direction of the program, and can be reversed to obtain the reversed program.

(Janus) To our knowledge, the first such programming language—Janus—was designed by Lutz and Derby [47] in 1982 for a Caltech class project, and later formalised by Yokoyama and Glück [48]. It is an imperative procedural language, and is named after the Roman god associated with duality and time. Listing I.11 illustrates most of its syntax through three increasingly sophisticated examples, or see Definition I.1 for a more detailed exposition.

The first example (due to Yokoyama and Glück [48]), *fib*, is a recursive procedure which computes the n^{th} and $n+1^{\text{th}}$ *Fibonacci* numbers into x_0 and x_1 respectively, consuming n in the process. The second (due to us), *fac*, is an iterative procedure that replaces n with $n!$ (providing $n > 0$), using k as a 'scratch' variables for implementing $m \leftrightarrow m \cdot n$ and m to hold the burgeoning factorial as n is consumed. The third (due to Lutz and Derby [47]), *sort*, performs a bubble sort on the list ℓ of length $n < 100$, and sets

[5] In some code snippets, notation of some arithmetic expressions and operators has been slightly modified where it is thought that the medium of mathematical typesetting provides greater clarity than the `ASCII` source.

```
n x[2]
procedure fib
    if n = 0
    then x_0 += 1
         x_1 += 1
    else n -= 1
         call fib
         x_0 += x_1
         x_0 <=> x_1
    fi x_0 = x_1
```

```
n m k
procedure fac
    m += 1
    from m = 1
    loop m <=> k
         from m = 0
         loop m += n
              k -= 1
         until k = 0
         n -= 1
    until n = 1
    n -= 1
    m <=> n
```

```
n ℓ[99] σ[99] i j
procedure sort
    from i = 0
    loop j += n - 2
         from j = n - 2
         loop if ℓ_j > ℓ_{j+1}
              then ℓ_j <=> ℓ_{j+1}
                   σ_j <=> σ_{j+1}
              fi σ_j > σ_{j+1}
              j -= 1
         until j = i - 1
         j -= i - 1
         i += 1
    until i = n - 1
    i -= n - 1
```

Listing I.11: Three select examples of `Janus` programs.

σ to the permutation mapping between the original and sorted lists (as a precondition, σ must be set to the identity permutation $(1\,2\,\cdots\,99)$). The absence of dynamic-length arrays or memory allocation (and hence use of 99 as a fixed buffer size in the above) shows a limitation of `Janus`, though the augmentation of `Janus` to support such features would be a straightforward extension.

Definition I.1 (Syntax and semantics of `Janus`)**.** A program consists of a series of (global) variable declarations and procedure declarations. Variables are either machine-precision integers, initialised to 0, or fixed-length arrays thereof as indicated by the square brackets. Variables are modified by in-place updates, or can be swapped with $\Leftrightarrow$. There are three possible in-place operators, $+=$, $-=$ and $\oplus=$ (in-place bit-wise `XOR`), and the right-hand side of a statement can be any compound expression making use of the unary operators $-$ (arithmetic negation), $\neg$ (bit-wise negation); the arithmetical binary operators $+$, $\times$, $-$, $/$ (quotient), $\%$ (remainder); the bit-wise binary operators $\oplus$, $\wedge$, $\vee$; or the Boolean binary operators $<$, $\leq$, $>$, $\geq$, $=$, $\neq$ where FALSE is represented by 0 and TRUE by -1. These expressions need not be information-preserving, as the semantics make use of a Bennett embedding: computing in the forward direction the value, performing the modification, and then uncomputing the value. As a result, it is illegal for the variable being modified to appear in the expression modifying it.

Procedures can be called in their forward direction by the aptly-named **call** statement, and in their reverse direction by **uncall**. `Janus` also supports conditionals and looping, per the aforementioned principles. Conditionals are represented by **if** p [**then** σ] [**else** σ'] **fi** q. They consist of a branch and unbranch condition, respectively p and q, and optional statements to be executed in the TRUE (**then**) and FALSE (**else**) cases. If p evaluates to

TRUE before the branch, then q *must* evaluate to TRUE afterwards, and similarly for the converse, or else a run-time error occurs. Loops are similarly defined, represented by **from** p [**do** σ] [**loop** σ'] **until** q. Upon reaching a loop, p must evaluate to TRUE. The statement(s) σ, if any, are then executed. Then q is tested; if it evaluates to FALSE then the statement(s) σ' are executed, else the loop exits. Otherwise, p is evaluated once again (and must this time evaluate to FALSE), and execution continues from σ. Not described here are the rudimentary I/O operations and `Janus` run-time system.

(Ψ-Lisp) A little while later, in 1992, Henry Baker introduced two `Lisp` variants: `Linear Lisp` [49] and `Ψ-Lisp` [50]. The former employs *linear logic* [51], in which every value must be consumed precisely once (usage optional), in order to do away with expensive garbage collection in functional languages. This idea is seeing a resurgence, with support in recent updates to languages like `Haskell` [52]. Whilst linear logic shares some properties with reversible computation, indeed quantum logic is often contextualised as a subset of linear logic, it is not restrictive enough to guarantee reversibility. Baker's etymologically-palindromic `Ψ-Lisp`, however, does guarantee injectivity of its primitives and is therefore reversible. It is itself a variant of `Linear Lisp` appropriately modified. The most conspicuous modification is that of the **if**-expression. Baker introduces the merge condition in an asymmetrical way, as the return value is not necessarily named: the first two sub-expressions are, respectively, a Boolean *expression* indicating which branch to take, and a Boolean *predicate* to be applied to the result. These predicates are handled specially, in comparison to the rest of the language, in that they are permitted to be non-invertible. As with expressions in `Janus`, these predicates are run forwards to select a branch, and then backwards to erase the garbage. Unfortunately the paper is somewhat lacking in precise detail of the language design, and so it is not clear how one defines new predicates (other than the mention of an implicitly instantiated history stack created when these predicates are evaluated), nor is it clear what happens if a predicate is called outside of such a scope.

```
(defun fact (n)
  (assert (and (integerp n) (> n 0)))
  (if (onep n) #'onep
      n
      (× n (fact (1− n)))))
```

Listing I.12: The (erroneous) reference implementation of the factorial function in `Ψ-Lisp`

To showcase the syntax and semantics of `Ψ-Lisp`, Baker [50] provides a reference implementation of the factorial function (reproduced in Listing I.12). Unfortunately, this definition is *not* reversible as it is not possible, given output 24, to mechanically invert the expression $(\times\ n\ (\mathit{fact}\ (1-\ n)))$ to obtain $n = 4$. In fact this is not a failure of the reversibility of `Ψ-Lisp`, but rather a failure to adhere to the principles of linearity: n is

used twice in this expression! Moreover, the presumed definition of $\times$ is not invertible or possible. This demonstrates the ease with which one can introduce a bug when programming reversibly, although were an interpreter/compiler for Ψ-Lisp available it would surely not accept the proposed program.

```
(defun first-onep (car · cdr) (discard cdr) (onep car))
(defun fac (n d)
  (assert (and (integerp n) (> n 0)))
  (if (onep n) #'first-onep
      (list n d)
      (let ((n' d') (fac (1- n) (1+ d))
            (n'' d'') (× n' d'))
           (list n'' (1- d'')))))
(defun fact (n) (let ((m 1) (fac n 1)) m))
```

Listing I.13: A corrected implementation of the factorial function in Ψ-Lisp

In Listing I.13, we attempt to provide a corrected Ψ-Lisp implementation of the factorial. This implementation is an iterative loop disguised in recursive form, as it does not appear that tail-recursion is possible in a reversible context due to the loss of information pertaining to recursion depth. The original paper is somewhat light on details, and so this definition is somewhat speculative, particularly with regard to user-defined predicates: here we define an auxiliary function *first-onep* which returns whether its first argument is 1 or not, and assume the existence of a **discard** primitive to access the implicit history stack. We also (reasonably) assume an invertible implementation of $\times$ in which $(\times\ x\ y)$ evaluates to $(\text{list}\ xy\ y)$, although the linearity restrictions in Ψ-Lisp mean we must refer to the output y by a different name than the input y. The parameter d tracks recursion depth, increasing from an initial value of 1 to a maximum of n_0, whilst n is simultaneously decremented from n_0 to 1. As we retrace our steps, we multiply n by the depth counter d, and then decrement d, such that when we finally return from *fac* the depth counter has been restored to its initial value (e.g. 1). The wrapper function *fact* handles supplying and consuming this depth counter, and assumes that Ψ-Lisp permits pattern matching on constants or some other way to consume a known constant.

(ЯR) Another language with Lisp-like syntax, ЯR ("yar"), was introduced in 1997 by Frank [53]. However, unlike Ψ-Lisp, the language is procedural and C-like—using the Lisp-like syntax for convenience of implementation (which is written in Common Lisp). To the best of our knowledge, this is the first, and perhaps only, (classical) language for which a compiler targeting a reversible processor exists; namely, the aforementioned *Pendulum Instruction-Set Architecture* (PISA). Indeed, the example program in Listing I.14 was compiled to PISA assembly and executed on an emulator (though, to our knowledge,

not on a physical PISA implementation such as PENDULUM).

```
(defsub pfunc  (ψ′ ψ i αś ε)
  ((ψ′ _ i) += ((αś _ i) ×/ (ψ _ i)))
  ((ψ′ _ i) −= (ε ×/ (ψ _ ((i + 1) ∧ 127))))
  ((ψ′ _ i) −= (ε ×/ (ψ _ ((i − 1) ∧ 127)))))

(defsub schstep  (ψR ψI αś ε)
  (for i = 0 to 127 (call pfunc ψR ψI i αś ε))
  (for i = 0 to 127 (rcall pfunc ψI ψR i αś ε)))
```

Listing I.14: A ЯR implementation of the discretised one-dimensional Schrödinger wave equation

Semantically, ЯR is not dissimilar to `Janus`, but it does support local variable scoping which makes it more powerful as a structured programming language. On the other hand, it does not (as of 1999) possess the ability to have different conditions on entry and exit for conditionals and loops, nor does it appear to support unbounded **for** loops. Nonetheless, such features can be simulated and so this is not a fundamental limitation of the language.

The example in Listing I.14 begs some further discussion. The Schrödinger equation was first discretised in a reversible way by Fredkin [54]. It has the remarkable property that, although its update rule is in some sense inexact, it nevertheless conserves amplitude (observed by Fredkin and proved by Feynman). To understand the example, we first briefly sketch how the Schrödinger equation, $i\hbar\frac{\partial}{\partial t}\psi = \hat{H}\psi$ which describes the time evolution of the wavefunction ψ under the action of the Hamiltonian $\hat{H}$, may be discretised. This equation has the formal solution $\psi(t) = \exp(-i\hat{H}t/\hbar)\psi(0)$; this is formal in the sense that $\hat{H}$ is an operator, which in common formulations takes the form of a Hermitian matrix or a partial differential operator, and so some care must be taken when manipulating it. Supposing we wish to use a timestep δt, we can write $\psi(t+\delta t) = \exp(-i\hat{H}\delta t/\hbar)\psi(t)$ for any value of t. It may be more convenient to supply the program with $\hat{U} \equiv \exp(-i\hat{H}\delta t/\hbar)$, having also pre-discretised ψ such that $\hat{U}$ is a matrix and ψ a vector, in which case the simulation can be performed exactly and reversibly[6]. Here, however, the program receives $\hat{H}$. Specifically, we are considering a one-dimensional cyclic potential well in which $\hat{H} = -\frac{\hbar^2}{2m}\frac{\partial^2}{\partial x^2} + V$, and the value of the potential V is supplied at each discretised point. Expanding $\hat{U}$ for small δt and discretising space, we find

$$\psi(x,t+\delta t) - \psi(x,t) = i\frac{\hbar\delta t}{2m\delta x^2}[\psi(x+\delta x,t) - 2\psi(x,t) + \psi(x-\delta x,t)] - i\frac{V(x)\delta t}{\hbar}\psi(x).$$

[6]It may be observed that matrix multiplication for non-singular $\hat{U}$ (which $\hat{U}$ is, being unitary) may be performed reversibly using Gaussian-elimination. Specifically, applying the row reduction algorithm to $[\hat{U}^{-1}|\mathbb{1}|\vec{\psi}]$ yields $[\mathbb{1}|\hat{U}|\hat{U}\vec{\psi}]$ where $\mathbb{1}$ is the identity matrix. As $\hat{U}$ is unitary, computing its inverse is trivial (the inverse of a unitary matrix is its conjugate transpose).

Here the approaches of Fredkin and Frank differ; Frank identifies that the real part of the wavefunction at the next timestep is only informed by the imaginary part of the wavefunction at the previous and vice-versa, in order to use this directly. In contrast, Fredkin seeks a more symmetric approach, transforming this to a form $\psi(x, t+\delta t) - \psi(x, t-\delta t)$: in this form the reversible implementation is more obvious without needing the observation of real-imaginary orthogonality, but one must retain copies of the wavefunction at two timepoints instead of the one in Frank's version. With the discretised Schrödinger equation thus derived, Frank's program is a simple transcription of this (after performing the routine algebra to discriminate the real and imaginary parts) with the array $\alpha\wp$ corresponding to $V\delta t/\hbar$ and ε to the coefficient $\hbar\delta t/m\delta x^2$ (twice the phase rotation per time-step due to the particle's kinetic energy). It also assumes space has been discretised 128-fold. Finally we comment on the operator _ which indexes into an array, and the operator $\times/$, peculiar to ⍘R, which returns the upper 32 bits of the 64-bit product of two 32-bit signed integers (the purpose being to implement the rounded product of an integer with a fixed point number in $[-1, 1)$).

(Kayak) Perhaps marking the debut of reversible programming into public perception, Ben Rudiak-Gould introduced Kayak in 2002 as an entry [55] for the second instance of the Esoteric Awards ('*Essies*'). The *Essies* are an annual competition in design and use of *esoteric* programming languages. Esoteric programming languages are not targeted for real-world use but instead are experiments in designing languages that may, for example, be intentionally difficult to program in or are very minimal whilst remaining Turing complete. The most well-known esoteric language is Brainfuck (BF), created in 1993 by Müller [56]. As well as being known for being hard to program in (though certainly not the most difficult, the honour of which arguably goes to Malbolge [57]), its syntactic minimalism and semantic simplicity make BF attractive for proving Turing-completeness of new languages and indeed the original distribution of Kayak included a BF-to-Kayak compiler (as does our language, alethe, introduced in Chapter V).

Definition I.2 (Syntax and semantics of Kayak)**.** Kayak has very few primitives. A program is a series of procedures, each consisting of a bipartite name (e.g. *swap*()*paws*), an input and output set of |-delimited variables (these sets should be of equal length), and a body. Procedure bodies can contain one of four 'commands': variable invocation, temporary register negation, conditional execution and procedure invocation. Variables are infinite stacks of bits, and invoking a variable will pop the top-most bit and place it in the 'temporary' register if this register is empty, otherwise it will empty the temporary register onto the top of the named stack. The temporary register is local to procedure bodies and conditional blocks. Variables not named in the inputs/outputs are also local to procedure bodies (but not conditionals), and are initialised to an infinite number of 0s (and should be restored as such upon procedure exit). If the temporary register is occupied, then the | command will negate its bit. The $[\cdots]$ command is a conditional block, which will run the nested commands if and only if the temporary register is occupied and set to 1. Finally, a procedure can be invoked by calling its bipartite name with some variables; for example, *swap*$(x|y)$*paws* will swap the contents of the stacks

x and y. Procedures can be run in reverse by reversing their names; for example, a factorial calculated using the program in Listing I.15 can be un-calculated by calling *lairo*$(n!)$*tcaf*. Of course, if a name is palindromic then its reverse is inaccessible, and so it is recommend that only self-inverse procedures have palindromic names. A fascinating property of this syntax is that inverting a program is as simple as reversing the characters of its file and mirroring the characters {([<>])}.

```
swap(a|b) {} (a|b)paws

add(a|b) {a[ add(a|b)bus z|b ]a} (a|b)bus

less-than(a|b|p) {a
    [b[ less-than(a|b|p)or-equal ]b] | [ p|p ]
|a} (a|b|p)or-equal

remquo(n|d|q) {
   less-than(d|n|p)or-equal
   p[ sub(n|d)dda remquo(n|d|q)ddalum ]q
} (n|d|q)ddalum

fact(n|d) {n[n[
   z|n z|d
   fact(n|d)orial'
   muladd(n|d|z)ouqmer
   swap(n|z)paws
   d|z n|z n|z
]n]n} (n|d)orial'

fact(n) {z|d
   fact(n|d)orial'
d|z} (n)orial
```

Listing I.15: Our `Kayak` implementation of the factorial function

```
s(a|b) {} (a|b).
t(a|b) {a[t(a|b). z|b]a} (a|b).
u(a|b|p) {a[b[u(a|b|p).]b] | [p|p]|a} (a|b|p).
v(n|d|q) {u(d|n|p). p[.(n|d)t v(n|d|q).]q} (n|d|q).
w(n|d) {n[n[z|n z|d w(n|d). .(n|d|z)v s(n|z). d|z n|z n|z]n]n} (n|d).
x(n) {z|d w(n|d). d|z} (n).
```

Listing I.16: Our `Kayak` implementation of the factorial function, 'golfed' and obfuscated

`Kayak` seems inspired by `BF` in terms of its syntactic obscurity, as may be gleaned from Definition I.2. Nevertheless, it is possible to write useful programs in it as we demonstrate in Listing I.15 with an implementation of the factorial function. In fact, the algorithm employed is much the same as that we used in Listing I.13. Moreover, we also provide[7] convenience routines for the reading and writing of base-10 `ASCII` representations of numbers, thus showing that the presence of modular abstractions—even in a language as cryptic as `Kayak`—permits arbitrarily complex programs to be

[7] https://github.com/hannah-earley/kayak/blob/master/arith.kayak

readily written. In keeping with its obscure intention, we also provide a 'golfed' variant in Listing I.16.

('YAG') Whilst Ψ-Lisp went some way towards the construction of a functional reversible programming language, its incomplete definition leaves it unclear whether this was achieved—not to mention that Lisp-esque languages are not deemed to be 'pure' functional languages. In 2012, Yokoyama, Axelsen and Glück [58] introduced a language—hereafter referred to as 'YAG'—which seems to meet the criterion of being a functional language. The example of computing Fibonacci numbers from the paper is reproduced in Listing I.17, and the semantics are summarised in Definition I.3.

$$
\begin{aligned}
&\mathit{plus}\ \langle x, y\rangle \triangleq \mathbf{case}\ y\ \mathbf{of}\\
&\qquad\qquad Z \quad\ \to \lfloor\langle x\rangle\rfloor\\
&\qquad\qquad S(u) \to \mathbf{let}\ \langle x', u'\rangle = \mathit{plus}\ \langle x, u\rangle\ \mathbf{in}\ \langle x', S(u')\rangle\\
&\mathit{fib}\ n \triangleq \mathbf{case}\ n\ \mathbf{of}\\
&\qquad Z \quad\ \to \langle S(Z), S(Z)\rangle\\
&\qquad S(m) \to \mathbf{let}\ \langle x, y\rangle = \mathit{fib}\ n\ \mathbf{in}\\
&\qquad\qquad\quad \mathbf{let}\ z = \mathit{plus}\ \langle y, x\rangle\ \mathbf{in}\ z
\end{aligned}
$$

Listing I.17: A 'YAG' implementation of the Fibonacci numbers

Definition I.3 (Syntax and semantics of 'YAG')**.** Like most functional languages, 'YAG' is expression oriented. A primary difference, however, is that function application may only occur in **let** and **rlet** expressions; this increased indirection helps enforce the reversibility of the semantics. These expressions take the form $(\mathbf{r})\mathbf{let}\ p = f\ q\ \mathbf{in}\ e$ where p and q are what we shall call[8] *patterns*, f is a function name, and e an expression in which to substitute the bindings of p. The **let** form runs f forwards with input q, and the **rlet** runs f backwards with output q. A pattern is either a variable, e.g. x, a constructor applied to some number of sub-patterns, e.g. the Peano numbers would be encoded inductively by the forms $Z \equiv Z()$ and $S(n)$, or the special 'duplication' pattern $\lfloor\langle x\rangle\rfloor$ (where $\langle x\rangle$, $\langle x, y\rangle$, etc are syntactic sugar for the tuple constructors $\mathit{Tuple}_1(x)$, $\mathit{Tuple}_2(x, y)$, etc). The 'duplication' pattern is unique[9] to 'YAG'; for a full explanation, the paper [58] is the best source, but essentially $\lfloor\langle x\rangle\rfloor$ reduces to $\langle x, x\rangle$ whilst $\lfloor\langle x, y\rangle\rfloor$ reduces to $\langle x\rangle$ if $x = y$, else to $\langle x, y\rangle$, and thus by matching on these two forms via

[8] In the original paper, these are called left-expressions by analogy to the common understanding of left-expressions in language syntaxes, however the occurence of *left*-expressions on the *right*-hand side of the equals sign is somewhat confusing.

[9] Duplication and equality testing as concepts are of course not unique to 'YAG', but the particular place this concept holds in the syntax and semantics *is* a seemingly novel feature.

a **case** expression, equality can be tested (and exploited to un-copy in a safe way). The remaining expression form is given by **case** p **of** $\{q_i \to e_i\}_{i=1}^{m}$, where the body defines some number of branches m. Here also 'YAG' takes a slightly different approach from other languages in this space. Whilst most languages require that the potential matching patterns q_i be orthogonal, in 'YAG' this is not the case; instead, the first pattern q_j matching the input p selects the branch taken. Similarly, once the branch returns an output from e_j, this must be unique *only amongst the expressions* $i = 1 \dots j$ in contrast to most other languages where it must be unique amongst *all* the expressions. Finally, a program is a series of function definitions of the form $f\ p \triangleq e$.

$$
\begin{array}{l}
\textbf{type}\ a + b = \mathrm{InL}\ a \mid \mathrm{InR}\ b \\
\textbf{type}\ a \times b = (a, b) \\
\textbf{type}\ \mathbb{N} = \mathrm{Z} \mid \mathrm{S}\ \mathbb{N} \\
\\
\mathit{trace} :: f{:}(a + b \leftrightarrow a + c) \to b \leftrightarrow c \\
\quad \mid \text{-- } \textit{definition omitted} \\
\\
\mathit{addSub} :: \mathbb{N} + \mathbb{N} \leftrightarrow \mathbb{N} + \mathbb{N} \\
\quad \mid \mathrm{InL}\ (\mathrm{S}\ n) \leftrightarrow \mathrm{InL}\ n \\
\quad \mid \mathrm{InL}\ \mathrm{Z} \leftrightarrow \mathrm{InR}\ \mathrm{Z} \\
\quad \mid \mathrm{InR}\ n \leftrightarrow \mathrm{InR}\ (\mathrm{S}\ n) \\
\\
\mathit{add}_1 :: \mathbb{N} \leftrightarrow \mathbb{N} :: \mathit{sub}_1 \\
\quad \mid n \leftrightarrow \mathit{trace}\ \tilde{}f{:}\mathit{addSub}\ n \\
\\
\mathit{add} :: \mathbb{N} \times \mathbb{N} \leftrightarrow \mathbb{N} \times \mathbb{N} :: \mathit{sub} \\
\quad \mid a, b \leftrightarrow \mathit{iter}\ \$\ a, \mathrm{Z}, b \\
\quad \mid \mathit{iter}\ \$\ \mathrm{S}\ a, a', b \leftrightarrow \mathit{iter}\ \$\ a, \mathrm{S}\ a', \mathit{add}_1\ b \\
\quad \mid \mathit{iter}\ \$\ \mathrm{Z}, a, b \leftrightarrow a, b \\
\quad \textbf{where}\ \mathit{iter} :: \mathbb{N} \times \mathbb{N} \times \mathbb{N}
\end{array}
$$

$$
\begin{array}{l}
\mathit{cantorUnpair} :: \mathbb{N} \leftrightarrow \mathbb{N} \times \mathbb{N} :: \mathit{cantorPair} \\
\quad \mid n \leftrightarrow \mathit{iter}\ \$\ n, (\mathrm{Z}, \mathrm{Z}) \\
\quad \mid \mathit{iter}\ \$\ \mathrm{S}\ n, (\mathrm{Z}, b) \leftrightarrow \mathit{iter}\ \$\ n, (\mathrm{S}\ b, \mathrm{Z}) \\
\quad \mid \mathit{iter}\ \$\ \mathrm{S}\ n, (\mathrm{S}\ a, b) \leftrightarrow \mathit{iter}\ \$\ n, (a, \mathrm{S}\ b) \\
\quad \mid \mathit{iter}\ \$\ \mathrm{Z}, (a, b) \leftrightarrow (a, b) \\
\quad \textbf{where}\ \mathit{iter} :: \mathbb{N} \times (\mathbb{N} \times \mathbb{N}) \\
\\
\mathit{fib}' :: (\mathbb{N} \times \mathbb{N}) + \mathbb{N} \leftrightarrow (\mathbb{N} \times \mathbb{N}) + (\mathbb{N} \times \mathbb{N}) \\
\quad \mid \mathrm{InR}\ n \leftrightarrow \mathit{iter}\ \$\ (n, (\mathrm{Z}, \mathrm{Z})) \\
\quad \mid \mathit{iter}\ \$\ (\mathrm{S}\ n, (a, b)) \leftrightarrow \mathit{iter}'\ \$\ (n, \mathit{add}\ (b, a)) \\
\quad \mid \mathit{iter}\ \$\ (\mathrm{Z}, (a, b)) \leftrightarrow \mathrm{InR}\ (\mathit{add}_1\ a, \mathit{add}_1\ b) \\
\quad \mid \mathit{iter}'\ \$\ (n, (a, b)) \leftrightarrow \mathit{iter}\ \$\ (n, (a, \mathrm{S}\ b)) \\
\quad \mid \mathrm{InL}\ (n, (\mathrm{S}\ a)) \leftrightarrow \mathit{iter}\ \$\ (n, (\mathrm{S}\ a, \mathrm{Z})) \\
\quad \mid \mathrm{InL}\ (n, \mathrm{Z}) \leftrightarrow \mathrm{InL}\ (\mathit{cantorUnpair}\ n) \\
\quad \textbf{where}\ \mathit{iter} :: \mathbb{N} \times (\mathbb{N} \times \mathbb{N}) \\
\qquad\quad \mathit{iter}' :: \mathbb{N} \times (\mathbb{N} \times \mathbb{N}) \\
\\
\mathit{fib} :: \mathbb{N} \leftrightarrow \mathbb{N} \times \mathbb{N} \\
\quad \mid n \leftrightarrow \mathit{trace}\ \tilde{}f{:}\mathit{fib}'\ n
\end{array}
$$

Listing I.18: A `Theseus` implementation of the Fibonacci numbers

(`Theseus`) The last language we will review is `Theseus`, whose name is inspired by the legendary *Ship of Theseus* in that its computation '[replaces] values by apparently equal values'. Based on the family of point-free[10] combinator calculi Π introduced by

[10] Point-free languages do not support explicit variable binding, instead modelling computation as 'flow through pipelines'; a point-free approach can often be used to elegant effect, and in languages supporting a mixed approach such as `Haskell` this is encouraged. Honourable mentions in this space are the languages `Inv` and `Fun` of Mu, Hu and Takeichi [59], whose motivation was in developing a language for

Roshan James for his graduate thesis [60], James and Sabry [61] introduced an equivalent higher level language. The motivation for Π is to provide a Category-Theoretic underpinning of reversible programming: specifically, Π^0 corresponds to dagger symmetric traced bimonoidal categories. Category Theory has long been a source of fruitful cross-fertilisation between the realms of mathematics and computer science, with monadic effects in `Haskell` a prime example. The typical direction is to attempt to find a category that models an existing programming language, such as HASK[11] for `Haskell`; `Theseus` goes the other direction, deriving a language from a category. The objective of this is as an aid to reasoning about categorical constructions rather than to produce a useful language as such, nevertheless a reference interpreter exists[12].

Definition I.4 (Syntax and semantics of `Theseus`)**.** `Theseus` is based on type isomorphisms, and has the strongest and most developed type system of those reviewed here. A `Theseus` program is a series of type definitions and isomorphism ('map') definitions, and its syntax largely resembles a subset of `Haskell`'s. A map is defined by a name, a type isomorphism, an optional name for its inverse, and a series of pattern clauses. Unlike in `Haskell`, *either* side of a pattern clause may invoke an isomorphism; if an invocation occurs on a 'target' side then it will be called and its return value substituted, as expected, if it occurs on the 'matching' side, however, then its inverse will be called and the result then matched. There are two further extensions to maps. Firstly, maps may be parameterised in order to simulate higher-order isomorphisms (this is just a simulation, as maps are not first-class values in `Theseus`). Secondly, maps may have typed 'labels', syntactically represented by *label* \$ v where v is its appropriately typed argument. The function of labels is as a sort of 'go-to' operator. Implicitly a map with type $a \leftrightarrow b$ and label types ℓ_1 and ℓ_2 is transformed to a map $(\ell_1 + \ell_2) + a \leftrightarrow (\ell_1 + \ell_2) + b$. Then, the map is called repeatedly ('traced') until it returns a value of type b and can thus be used to implement looping. For example, if such a labelled map f yields a loop of 3 iterations, then we would call the modified map f' four times on $\mathrm{InR}\ x$ and then match on the output, i.e. $\mathrm{InR}\ y = f^4(\mathrm{InR}\ x)$. A core difference between this language and the others reviewed here is that the clauses of a map must not only be orthogonal, but also *exhaustive*. This complicates definitions of even something as simple as adding 1 to a natural number, because the output of add_1 cannot be 0; to circumvent this, explicit divergence of these cases must be employed, i.e. introducing an infinite loop.

The semantics of `Theseus` are summarised in Definition I.4 and illustrated in the example implementation of Fibonacci numbers in Listing I.18. In Listing I.18, the definitions of *trace*, *addSub*, add_1 and *add* are due to James and Sabry [61] whilst the remaining definitions, *cantorUnpair*, *fib'* and *fib*, are due to us[13]. The aforementioned

bi-directional transformations. Bi-directional transformations are another thread in the field of reversible computation, useful for such applications as editors in which there is an isomorphism between the (editable) document description and the resultant document view.

[11] The familiar reader will object that HASK is not, strictly speaking, a category due to the interaction of the primitive *seq* with bottom values.

[12] Mirrored at https://github.com/hannah-earley/theseus

[13] The original paper also supplies a definition of *fib*, but as a Bennett-esque embedding.

exhaustivity requirement makes this substantially more complicated than, e.g., that in Listing I.17 because we must manufacture an exhaustive handling of the error condition. In *fib* this is very prominent because the subset of $\mathbb{N} \times \mathbb{N}$ corresponding to valid Fibonacci pairs has vanishing measure. Additionally, we attempt to reduce the extraneous cases by internally operating on $(F_n - 1, F_{n+1} - 1)$. James and Sabry do suggest at the end of their paper that it may be possible to extend `Theseus` with more conventional error handling, which we believe would be the more optimal choice for a programming language. Explicit divergence has a number of downsides in our opinion: from a programmer perspective, coding up divergence conditions is quite cumbersome[14], and there is no 'correct' choice of how to do so as these cases are meaningless, thus making it more of an art than a science. Furthermore, from a user's perspective this divergence behaviour is sub-optimal as one cannot distinguish a long-running/intentionally non-halting computation from an error condition or bug, not to mention the excessive resultant CPU utilisation. Of course, these objections are besides the point of `Theseus`, but would serve as a small modification with disproportionate benefit for adoption of `Theseus` as a viable typed functional language for reversible computation.

We will revisit the matter of reversible programming languages in Chapter V, but for the next three chapters we shall return to the physics of reversible computation.

[14] We would estimate it occupied 95% of our time in constructing the *fib* example!

In this chapter we will consider the maximum sustained computational performance—in terms of operations per second—that can be extracted from a given region of space, under fairly general assumptions. Namely, we assume the known laws of physics, and that one will need to supply the system with energy over time in order to keep it going. We show, similarly to early work by Frank [9], that for any realisable computer reversible computers are strictly better than irreversible computers at any size. We also derive universal scaling laws describing just how much better a reversible computer could be compared to an irreversible computer, proving that the adiabatic *Time-Proportionally Reversible Architectures* (TPRAs) of Frank are the best possible, and suggest a path to achieving this bound with molecular computers.

To illustrate these results, summarised in Figure II.1, suppose we wish to build the most powerful computer we can within some spherical region of space, of radius R. If the computer is irreversible, such as conventional silicon-based processors, then it is found that essentially only the surface of this sphere can be used for computation and the interior must be empty or inert. This clearly limits the rate of computation proportional to $4\pi R^2$. The reason for this restriction is thermodynamic, arising from constraints on both supply of power and rejection of heat. If R is very big, such that the computer threatens to collapse into a black hole, then it is found that the computational rate can now only scale proportional to R.

For a reversible computer the situation is substantially improved. In principle, a reversible computer can operate without producing an increase in entropy and without requiring any input of energy, and so the rate of computation could scale with the volume. Unfortunately in practice this is not possible, and a more careful consideration of the system's entropy is required. The rate of computation is found to scale proportional to $\frac{4}{\sqrt{3}}\pi R^{5/2}$, geometrically between the area and volume. For large R, general relativistic effects become significant and this falls to the more restrictive bound $R^{3/2}$. For even larger R this eventually falls to proportional to R, coinciding with the irreversible computer. The reason for this additional threshold between scaling laws is that the sub-relativistic reversible computers—where gravitational effects on spacetime are negligible—are not taking full computational advantage of their volume due to thermodynamic constraints; as the system gets larger beyond the collapse threshold (when the geometry transitions to a thick shell rather than a sphere) the thermodynamic constraints gradually relax, allowing a scaling that 'temporarily' exceeds the limit value of R.

Therefore we see that at almost all scales, reversible computers substantially outperform irreversible ones, whilst at extremely small—typically sub-nanometer—and large—order of the visible universe in size—scales they coincide.

II. Limits on Computational Rates in Physical Systems

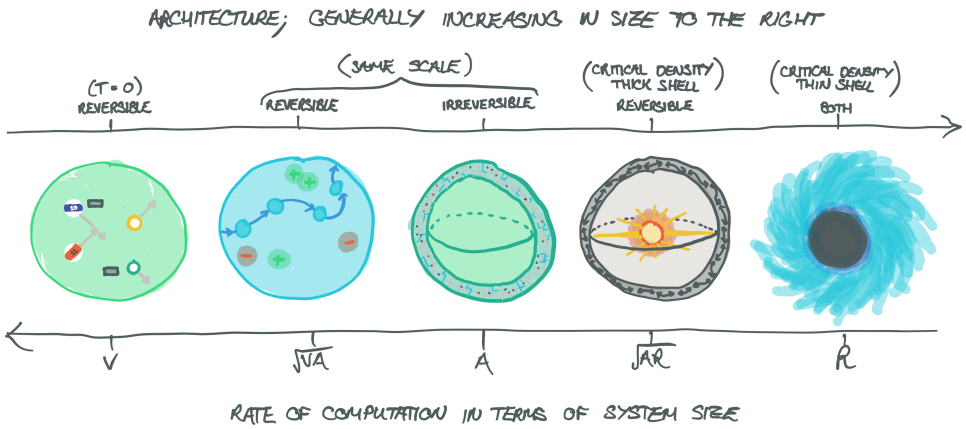
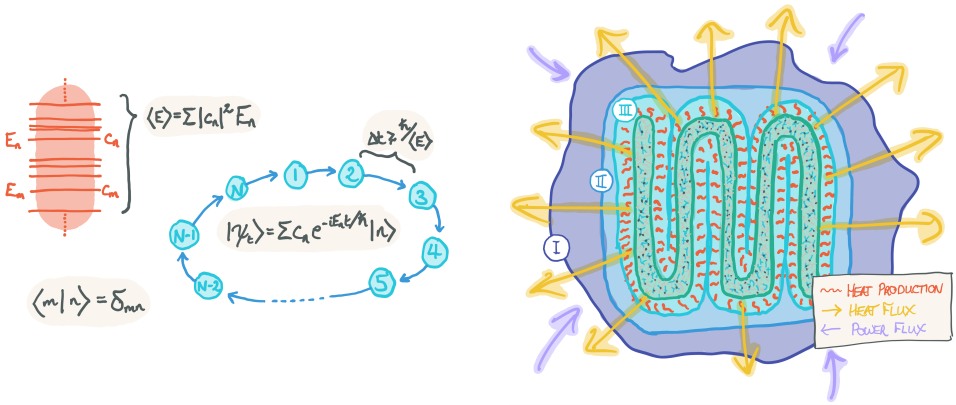


Figure II.1: An illustration of the primary results of this chapter. Consider a spherical region of computational matter with radius R, convex surface area A and enclosed volume V. The expressions indicate how the computational rate of each architecture scales, proportionally. From left to right, these regimes are:

V : reversible computers at absolute zero;
$\sqrt{AV}$: reversible computers at finite temperature;
A : irreversible/canonical computers;
$\sqrt{RA}$: critical density (thick shell on the cusp of gravitational collapse);
R : critical density (thin shell on the cusp of gravitational collapse).

II.1. Introduction

The ubiquity of computer technology—and the increasing demands set upon it by intensive algorithms from fields such as machine learning, physics simulations, and cloud computing, among others—render the question of computer performance of considerable interest. Perhaps the most well known observation on computer performance was published by Moore [63], who noticed the trend of microchip component density doubling every 18 months, later dubbed 'Moore's law' in his honour. This law seemed



(a) (b)

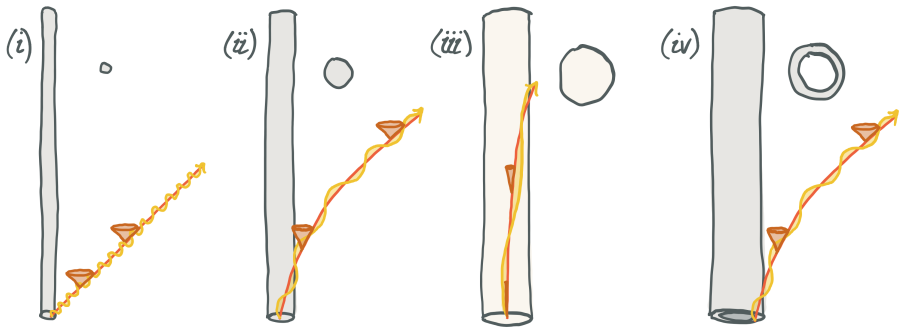
Figure II.2: An illustration of the three dominant physical constraints that apply to computational systems. Constraints (a) and (b) are described in Section II.1 and (c) in Section II.5. (a) The quantum mechanical constraint bounding the minimum time for a state transition for a system of given average mass-energy $\langle E \rangle$ where $E_0 = 0$. This illustration includes example energy levels of the eigenstates and a cyclic sequence of orthogonal states $\{1, \ldots, N\}$ which are each a superposition of the eigenstates, with the same average energy $\langle E \rangle$. (b) A combination geometric-thermodynamic constraint. Thermodynamically, each computational operation generates entropy (generally heat) and this is bounded from below in the case of irreversible computation, but still non-zero in the reversible case. Geometrically, we can only dissipate this entropy at a rate scaling with the convex bounding surface, (II). The surface (III), whilst larger than (II), is not useful as the entropy flux must still pass through surface (II). Surface (I) is also larger, but the flux must first pass through surface (II). The green region is the computational system proper. At steady state there is an amortised balance between the entropy dissipation flux and input power flux. (continued on next page)

to apply to many quantities in computing, including clock speed, *FLOPS*[1] of the top 500 supercomputers combined, and inverse storage cost. A popular debate arose over when, if ever, Moore's law would stagnate; for clock speed, this point has come and gone, with consumer processor clock speeds frozen around 3 GHz to 4 GHz.

Whilst it is not surprising that contemporary technology may pose limits on computational performance, it is pertinent to ask whether the laws of physics impose hard bounds on performance. Indeed, our knowledge of quantum physics, thermodynamics and relativity reveals the answer to be affirmative, as hinted at in the introduction chapter. For a brief overview of some of these constraints see, for example, Lloyd's analysis [64].

In this chapter, we investigate how these bounds vary as a given computational system is scaled up. The results we find apply, with different constants of proportionality, for any given computational architecture. A computer constructed from a mix of architectures can be treated as a linear combination, and therefore these scaling results apply to any computer. This builds upon the work of Frank [9], proving that his notion

[1] *FLOPS*, or FLoating point Operations Per Second, is a common measure of supercomputer performance.



(c)

Figure II.2: (continued) (c) The (general) relativistic constraint which informs the optimal mass distribution of the system. Each example computational system is shown as a space time diagram, with the horizontal axis corresponding to space and the vertical to time. The above-right circles are cross-sections showing the mass distribution. Cones are light-cones showing the local causal structure and indicating spacetime curvature due to the stress-energy tensor. The emanating light waves correspond to the output of the computer. For small computers such as (i), general relativistic effects are negligible. For larger computers such as (ii), space is noticeably curved within the vicinity of the system and so the frequency of emitted photons is red-shifted, meaning that the computer appears slower to a distant observer. For computers on the cusp of gravitational collapse such as (iii)—even if the density is lowered to keep the Schwarzschild radius small, as indicated by the lighter shade of the system—light takes a very long time to escape, diverging to infinity as the system radius approaches $\frac{9}{8}$ of the Schwarzschild radius. The factor $\frac{9}{8}$ comes from the Chandrasekhar limit [62] and is explained in Section II.5. In fact, this is an ideal case; in practice the computational rate will vanish and the system will spontaneously collapse at even larger radii depending on the heat capacity ratio. To avoid this while keeping the same total mass and radius, the system should be reconfigured into a spherical shell of the same density as (i,ii), as depicted in (iv). The geometry can then be optimised to minimise the effect of time dilation to at most a threefold slowdown, thus retaining our qualitative power-law scaling.

of adiabatic *Time-Proportionally Reversible Architectures* (TPRAs) are the best possible, confirming the relevant scaling results in the mesoscopic regime (as well as analysing scaling in very small and very large regimes in more detail), and excluding the possibility of architectures of lower dissipation. Moreover, the detailed calculations herein yield specific constants of proportionality for broad classes of reversible computational architectures.

There are many metrics by which one may wish to measure computational performance. In this chapter, we shall investigate 'speed' as defined by the rate of state transitions the computer is able to execute. As we are more concerned with the general form of scaling law rather than the specific constants for a specific architecture, the exact definition of 'state' and 'transition' are not too important. In general the most 'obvious' definitions can be assumed, for example a conventional silicon-based computer would have as its states each possible value of all its registers, memory and any attached storage device, and as

its transitions a single machine instruction[2]. More rigorous definitions can be found in quantum mechanics, wherein a state would be an eigenstate of the computational basis Hamiltonian, and a transition would correspond to the complete evolution of the state between orthogonal eigenstates.

Other performance metrics of interest might concern synchronisation between distinct computational elements within a parallel system, and interaction with some arbitrary non-equilibrium system such as a supply of additional memory resources. These shall be covered in Chapters III and IV, respectively.

Quantum Constraint on Computer Performance As a first approach to bounding computational performance, we turn to quantum mechanics. Bremermann [65] gave a back of the envelope calculation in 1962, subsequently refined and elaborated upon by Margolus and Levitin [66] and summarised in Figure II.2a, to show that a system with energy E can change state at a maximum rate of $\nu \leq E/h_P$ where h_P is Planck's constant. Assuming this energy is due primarily to rest mass, this gives $\nu \leq 1.36 \times 10^{50}\,\mathrm{Hz\,kg^{-1}}$. Restricting our attention to a specific architecture as defined earlier, we assume a fixed density—or at least that the density is bounded from above—such that $\nu \leq V\rho c^2/h_P$. This immediately implies that the maximum rate of computation scales with volume.

Thermodynamic and Geometric Constraint on Computer Performance This quantum limit assumes a closed computational system, requiring no external power source. As we saw earlier this is not possible for conventional computing architectures, being irreversible and hence subject to the Landauer bound [7, 8]. To recall, the Landauer bound quantifies the increase in entropy that occurs whenever information is irreversibly discarded into the environment (as must occur when trying to overwrite data or otherwise reset some arbitrary state to a known state). Explicitly, whenever a quantity of information I bits is discarded, an entropy increase of $\Delta S \geq kI \log 2$ manifests where k is Boltzmann's constant, with equality only in the limit of thermodynamic reversibility in which the discarding process takes an infinite period of time.

In order to sustain computation then, it is necessary to remove the additional entropy at the same amortised rate as its production. In the case of heat, this is achieved by cooling the system; this process can be generalised to other forms of entropy, such as disorder in a spin bath [67], but we shall restrict out attention to heat for conceptual convenience. We must thus enact a flow of work into the system and heat out in order to sustain computation, but here we encounter a geometric constraint. This energy flow applies at every bounding surface, assuming that the heat is ultimately rejected to infinity. Consider a convex bounding surface of area A, and assume our technology of choice is capable of transporting energy up to a certain flux ϕ, then the maximal power that can be exchanged with the system is given by $P \leq \phi A$. As has been established, however, the rate of heat generation scales with the rate of computational transitions for an irreversible computer, and therefore the rate of computation is ultimately bounded by

[2] To be specific, we refer to the combination of a single op-code and its parameters.

the system's convex surface area, rather than its volume as contended by the quantum mechanical limit. See Figure II.2b for an illustration.

This is far more restrictive than the volumetric upper bound derived earlier, and suggests that for large computers only a subvolume equivalent to the outermost shell of the volume can perform useful computation with the remaining bulk relegated to dormant inactivity[3]. That is, we take the more restrictive of the two bounds, which for small systems is the volumetric bound and for larger (irreversible) systems is this areametric bound.

Ballistic Computation Does this areametric thermodynamic bound render the volumetric quantum bound inaccessible? A trivial exception is found at small scales, where the surface area to volume ratio becomes so large that the volumetric bound is in fact more restrictive than the areametric bound, and thus takes priority. To more robustly improve on the rate of computation, however, we must reduce the entropic cost associated with a computational transition. As was elaborated upon in the introduction chapter, this necessitates conserving information during computation by using the concept of *reversible* computation. Whilst the utility of reversible computation was initially controversial, with Landauer initially [7] believing it to not be useful, it was soon shown by Bennett [39] that reversible computation was viable. Moreover, he showed how any irreversible program could be efficiently simulated on such a reversible computer without excessive memory overhead, thus proving that reversible computing was equipotent with conventional computing. For a brief exposition of reversible programming, refer back to Chapter I.

An idealised reversible computer would produce no entropy in its transitions. Fredkin and Toffoli [42] described such an idealised system, a physical model of reversible computing that could operate without being actively powered. This model consisted of a frictionless table upon which hard elastic billiard balls were projected. The balls would bounce off of each other and some strategically placed walls, with the precise configuration describing the resultant computation. As a testament to its capability, Fredkin's student, Ressler [41], proposed the design of a fully programmable billiard ball computer, complete with arithmetic logic unit. See Figure I.6 for an illustration of the general principles of the billiard ball model. Given that such a computer would be powered solely by its initial kinetic energy, the volumetric quantum bound would be attainable. Unfortunately, Bennett [43] calculated that such a design would ultimately be infeasible as even the most distant and subtle influences would be sufficient to rapidly and completely thermalise the motions of the system. Such influences could in principle be suppressed by error correction mechanisms, but error correction corresponds to the discarding of excess entropy, and would seem to reconstitute the very issues we were trying to avoid.

Our last refuge against unwarranted entropic influences is in quantum ground state

[3] The remaining bulk need not be entirely devoid of purpose. It may, for example, be used for storage and memory. It must however have negligible entropy generation, such as that resulting from structural deterioration due to cosmic rays, spontaneous tautomerisation, etc.

condensates. Phenomena such as superfluidity and superconductivity arise in a sufficiently cooled system, wherein a macroscopic fraction of particles are found to inhabit their ground state, manifesting macroscopic quantum behaviours and a subsystem with vanishing entropy and temperature. The utility of such systems is manifold, but unfortunately it is doubtful that they can be exploited for dissipationless computation; the reason is that, in order to enact transitions to orthogonal quantum states, the Schrödinger equation requires us to prepare a superposition of (distinct) energy states which then rotates under the action of the Hamiltonian. The presence of non-ground eigenstates will necessarily result in a renewed vulnerability to thermodynamic perturbations. Nevertheless, such cold temperatures are not altogether useless, as unwanted fluctuations would still be significantly suppressed.

Brownian Dynamics The fluctuations and dissipation that result from these thermodynamic perturbations lead to diffusion of the distribution in phase space, and hence loss of absolute control over it. Consequently there will be some level of unpredictability to the state of the system and its (generalised) velocities. In the limit of negligible control, the dynamics are (mostly) dominated by the noise source; this regime is referred to variously as Brownian or Langevin dynamics. In Section II.4 it shall be shown how to leverage the minimal extant control over the system to make net progress, and to even robustly surpass irreversible computers in performance.

Summary Whilst fully dissipationless computation is all but impossible, it is still possible in principle to tune the system so as to bring the entropic transition cost as close to zero as desired. In exchange, the rate of computation may itself be diminished. In the chapter that follows, this compromise is evaluated across the range of known physics—from quantum to classical, non-relativistic to relativistic—covering the full spectrum of feasible computational architectures. In so doing, a universal scaling limit is discovered, exceeding that of irreversible computers yet still falling short of the volumetric quantum bound.

II.2. Quantum Ballistic Architectures and the Quantum Zeno Effect

We first consider a quantum architecture. A viable quantum architecture must be ballistic in the sense of following an exact prescribed trajectory in phase space, as to do otherwise would lead to decoherence undermining the computational state. Of course we have established that a ballistic quantum system cannot truly be isolated from entropic effects; in particular, the third law of thermodynamics precludes lowering the temperature of any system to absolute zero, and so we must incorporate a heat bath into our analysis.

Suppose the ballistic Hamiltonian is given by H_0 and the perturbative effect of the heat bath by $V(t)$, such that the true Hamiltonian is $H = H_0 + V$. Further, let the initial density be ρ and introduce the projector P corresponding to the subspace of valid states, such

that $P\rho = \rho P = \rho$. To ensure processive computation, we must therefore periodically correct errors introduced by the perturbation. The cost of such error correction will be bounded from below by the entropy increase due to V, which can be expressed in terms of the probability of an error δp corresponding to the erroneous subspace $P^\perp = 1 - P$.

If corrections are to be made at intervals δt, then this error will be found to be $\delta p(t) = \mathrm{tr}\big[P^\perp \rho(t+\delta t)\big]$. For a time-dependent Hamiltonian, the quantum Liouville equation tells us that $\dot{\rho} = i[\rho, H/\hbar_P]$ where $\hbar_P$ is Planck's constant. We can expand ρ as

$$\begin{aligned}\rho(t+\delta t) &= \rho + \delta t \frac{\partial \rho}{\partial t} + \tfrac{1}{2}\delta t^2 \frac{\partial^2 \rho}{\partial t^2} + \mathcal{O}\big(\delta t^3\big) \\ &= \rho + i[\rho, \varepsilon] + \tfrac{1}{2}(i\delta t[\rho, \dot{\varepsilon}] - (\varepsilon^2\rho - 2\varepsilon\rho\varepsilon + \rho\varepsilon^2)) + \mathcal{O}\big(\delta t^3\big)\end{aligned}$$

where we have omitted explicit time dependence for brevity and written $\varepsilon \equiv H\delta t/\hbar_P$. We then find

$$\begin{aligned}\delta p &= \mathrm{tr}\Big[P^\perp(\rho + i[\rho, \varepsilon + \tfrac{1}{2}\delta t\dot{\varepsilon}] - \tfrac{1}{2}(\varepsilon^2\rho - 2\varepsilon\rho\varepsilon + \rho\varepsilon^2))\Big] + \mathcal{O}\big(\varepsilon^3\big) \\ &= \mathrm{tr}\Big[P^\perp \varepsilon\rho\varepsilon\Big] + \mathcal{O}\big(\varepsilon^3\big) \equiv \mathrm{tr}\Big[\rho\varepsilon P^\perp \varepsilon\Big] + \mathcal{O}\big(\varepsilon^3\big)\end{aligned}$$

where the last line uses the fact that $P^\perp\rho = \rho P^\perp = 0$. Note that if the $\rho\varepsilon P^\perp\varepsilon$ term vanishes, then so do all higher terms, and therefore this term is always the leading order approximant.

In order to evaluate this trace, we first write H_0 and V as block matrices in the basis $(P, P^\perp)$, yielding

$$H_0 = \begin{pmatrix} h_{00} & 0 \\ 0 & h_{11} \end{pmatrix}, \qquad V = \begin{pmatrix} v_{00} & v_{01} \\ v_{10} & v_{11} \end{pmatrix}.$$

Using the idempotence of projectors, i.e. $P^\perp \equiv P^\perp P^\perp$, and the fact $\rho \equiv P\rho$, we can see that $\rho H P^\perp H \equiv \rho V P^\perp V = \rho(V^2 - VPV)$. Therefore, we have

$$\delta p = \left(\frac{\delta t}{\hbar_P}\right)^2 (\langle V^2\rangle - \langle VPV\rangle).$$

The first thing to notice about this expression is that it varies with the square of δt. This is characteristic of the Quantum Zeno Effect (QZE [68])—that is, in the limit of constant measurement (i.e. $\delta t \to 0$), we can freeze dynamical evolution. Here, we are performing a partial measurement by projecting onto the subspace of either valid or erroneous computational states, thus allowing computational evolution to continue. Of course, the Zeno rate itself is subject to the Margolus-Levitin limit [66] and so such constant measurement is not possible.

To proceed in further determining δp, and thereby δh, it is important to elaborate on certain architectural details. Margolus and Levitin's analysis shows that one gets the same E/h_P total rate regardless of whether one considers transitions of the system as a whole ('serial') or the combined transitions of a system partitioned into subsystems

('parallel'). For a physically realistic system, measuring large subsystems as a whole is impractical and so a more fine-grained architecture is likely preferable. Nevertheless, we shall proceed generally, finding the result is independent of this detail.

Let the subsystems be indexed by $i \in \mathcal{I}$. For a fully serial system, $|\mathcal{I}| = 1$. We allow each subsystem to evolve independently, assuming that interaction events are negligibly frequent relative to our Zeno corrections. Specialising our above result for δp, we have

$$\delta p_i = \left(\frac{\delta t_i}{\hbar_P}\right)^2 (\langle V_i^2 \rangle - \langle V_i P_i V_i \rangle)$$

The strength of the perturbations V_i will be controlled by an effective temperature for that subsystem, T_i. Suppose the subsystem has n_i degrees of freedom[4], hereafter referred to as *computational primitives*. By the equipartition theorem, the energies of each primitive will be individually perturbed. For an uncorrelated perturbation, we expect that $\overline{\langle V \rangle} = 0$ where $\overline{x}$ is the time-average value of x. Therefore, $\overline{\langle V^2 \rangle} \equiv \mathrm{var}_{\langle\cdot\rangle} V$. The expected aggregate perturbation over all primitives, then, is $n_i(kT_i)^2$ assuming V is Gaussian and where $k \equiv k_B$ is Boltzmann's constant.

As for the second term, VPV, we expect that on average V will mix (near-)degenerate states indiscriminately. VPV represents the probability that the perturbation stays within the intended computational subspace. The state space can be divided into disjoint computational subspaces, within which the dynamics perform a reversible cycle over computational states. Let the total state space have cardinality Ω_i, and the number of microstates associated with each subspace be ω_{ij} where j indexes the different subspaces such that $\Omega_i = \sum_j \omega_{ij}$. Assuming similar energy distributions between subspaces, the chance of remaining within the original subspace is ω_{ij}/Ω_i. We further approximate this by ω_i/Ω_i where $\omega_i = \langle \omega_{ij} \rangle_j$ is the average subspace cardinality. Equivalently, we can define $\Omega_i/\omega_i \equiv W_i$, the number of distinct programs that the subsystem is capable of executing. Therefore,

$$\overline{\delta p_i} = n_i \left(\delta t_i \frac{kT_i}{\hbar_P}\right)^2 (1 - 1/W_i)$$

In fact, we may wish to correct the second term errors in which we jump within the same subspace, as this may interfere with synchronisation between different subsystems (or it may be difficult to ascertain whether we have jumped within the same subspace). In this case, the expression reduces to

$$\overline{\delta p_i} = n_i \left(\delta t_i \frac{kT_i}{\hbar_P}\right)^2 (1 - 1/\Omega_i)$$

In any case, $1 - 1/W_i$ is always finite and $\in [\frac{1}{2}, 1)$ for any useful subsystem; that is, a useful subsystem should have $W_i \geq 2$. For composite/non-primitive subsystems, we would like the number of useful programs to scale exponentially with the number of primitives, i.e. $W_i = g_i^{n_i}$ for some g_i (typically on the order of unity but greater than 1).

[4] Excluding any frozen out by low temperatures.

For a composite subsystem of even moderate size, this exponential scaling will render the $1 - 1/W_i$ term effectively unity.

We are now able to determine the entropy increase between Zeno cycles. The indiscriminate mixing of the perturbation means that, in the event of an error, the entropy attains its maximum value. This yields the following expression for the information entropy[5],

$$\begin{aligned}\delta h_i &= [-(1-\delta p_i)\log(1-\delta p_i) - \delta p_i \log \delta p_i + (1-\delta p_i)\log\omega_i + \delta p_i \log \Omega_i] - [\log \omega_i] \\ &= \delta p_i(1 + \log \tfrac{\Omega_i}{\omega_i} - \log \delta p_i) - \mathcal{O}\big(\delta p_i^2\big) \\ &\equiv \delta p_i(\log W_i + 1 - \log \delta p_i) - \mathcal{O}\big(\delta p_i^2\big) \\ &\geq n_i \delta p_i \log g_i - \mathcal{O}\big(\delta p_i^2\big)\ .\end{aligned}$$

This expression assumes ignorance of states within the current computational subspace; again, we may wish to correct these errors too, in which case we may simply substitute $\omega_i = 1$. This would effectively lead to taking $W_i = \Omega_i = g_i'^{n_i}$ where $g_i' \geq g_i$, and so the inequality remains valid.

Putting these together, we get

$$\dot{h}_i \geq \frac{n_i^2}{r_{Z,i}} \underbrace{\zeta_i \Big(\frac{kT_i}{\hbar_P}\Big)^2 \log g_i}_{\equiv \gamma_i}$$

where $r_{Z,i} \equiv 1/\delta t_i$ is the Zeno measurement rate for the subsystem and $\zeta_i = 1 - 1/W_i \in [\frac{1}{2}, 1)$. The aggregate rate of entropy generation is given by $\dot{H} = \sum_i \dot{h}_i$ and is subject to the constraint $k\hat{T}\dot{H} \leq P$ where $\hat{T}$ is the system temperature that governs the Landauer bound and $P = \phi A$ is the heat dissipation power, proportional to the system's surface area. It may seem surprising that $\dot{h}_i \propto n_i^2$. In fact, this only applies for sufficiently small δp_i; when δp_i becomes significantly large, $\dot{h}_i$ approaches its maximum of $n_i r_{Z,i} \log g_i$.

We wish to maximise the computational rate, subject to this constrained entropy production. Per Margolus and Levitin, this rate depends on the combined energy of the computational primitives. If the average energy per primitive in subsystem i is ε_i, then the energy available for computation is $E_C = \sum_i \varepsilon_i n_i$ and the computational rate is subject to $R_C \leq E_C/h_P$.

Introducing the Lagrangian multiplier α, we wish to maximise Λ with respect to n_i;

$$\Lambda = \sum_i \varepsilon_i n_i - \frac{1}{2\alpha}\sum_i \frac{n_i^2}{r_{Z,i}}\gamma_i \qquad \Longrightarrow \qquad 0 = \varepsilon_i - \frac{1}{\alpha}\frac{n_i}{r_{Z,i}}\gamma_i\,.$$

This gives $\dot{h}_i \geq \alpha \varepsilon_i n_i$ and so

$$\dot{H} \geq \alpha E_C\,;$$

[5] We use H to refer to the information theoretical entropy, connected to the thermodynamical entropy as $S = k_B H$. Furthermore, we use the lowercase quantity h to refer to the entropy of a subsystem or particle.

solving for α and summing over i,

$$\alpha r_{Z,i} = \frac{n_i \gamma_i}{\varepsilon_i} \equiv \frac{n_i \varepsilon_i \gamma_i}{E_C \varepsilon_i^2} E_C$$
$$\alpha R_Z = E_C \left\langle \frac{\gamma_i}{\varepsilon_i^2} \right\rangle_{E_{C,i}} .$$

where the average is taken over the computational energy distribution and R_Z gives the net Zeno rate of the system as a whole. Substituting into the $\dot{H}$ constraint and making use of the fact that $R_Z \leq E_Z / h_P$,

$$\frac{P}{k\hat{T}} \geq \dot{H} \geq \frac{E_C^2}{R_Z} \left\langle \frac{\gamma}{\varepsilon^2} \right\rangle$$
$$\frac{P}{k\hat{T}} \frac{E_Z}{h_P} \left\langle \frac{\gamma}{\varepsilon^2} \right\rangle^{-1} \geq E_C^2$$

where $\hat{T}$ can be called the 'Szilard' temperature: the temperature of the system in which entropy is generated and must be subsequently erased. Finally, we obtain an expression for the net computational rate R_C,

$$R_C \leq \sqrt{\frac{P}{k\hat{T}} \frac{E_Z}{h_P} \left\langle \zeta_i \left(\frac{2\pi k T}{\varepsilon} \right)^2 \log g \right\rangle^{-1}} = \frac{1}{2\pi} \left\langle \frac{\zeta_i \log g}{\beta^2 \varepsilon^2} \right\rangle^{-1/2} \sqrt{\frac{P}{k\hat{T}} \frac{E_Z}{h_P}} .$$

Now, as the upper bound of P is proportional to the bounding surface area, and the upper bound of E_Z is proportional to the system's volume, we get the scaling law

$$R_C \lesssim \sqrt{AV} \sim V^{5/6} ,$$

consistent with the scaling law for adiabatic architectures found by Frank [9] and showing it to be an upper bound.

We note that Levitin and Toffoli [69] derive a related expression for the rate of energy dissipation/heat production within a quantum harmonic oscillator (QHO),

$$P = \frac{\varepsilon h_P R_C^2}{(1-\varepsilon) N} ,$$

where ε is the probability of error per state transition and N is the number of QHO energy levels. Given that their analysis assumes maximal R_C, however, this simplifies to

$$P = \frac{\varepsilon \Delta E}{1-\varepsilon} R_C$$

where ΔE is the separation of energy levels. Given that ε is assumed constant in the analysis, this then yields an areametric bound on R_C. Our analysis permits ε to vary, and its optimum value as a function of system parameters is obtained. We also remain general in the systems studied, and thus confirm the conjecture that for a fixed

error probability the energy-dissipation rate increases quadratically with the rate of computation.

We also note the recent results[6] of Pidaparthi and Lent [70] on the Landau-Zener effect (LZE) [71, 72]. In a closed quantum system, with no thermal coupling to the external environment, the Landau-Zener effect predicts that the energy dissipation decays *exponentially* with the switching time of the system (the inverse of the computation rate). That this is non-zero even in the absence of an external environment shows the difficulty in achieving ballistic computation even in 'idealised' conditions, but it certainly points towards a route to achieving near-ballistic computation in regimes in which thermal coupling is negligible. As expected, once thermal coupling is re-introduced to the system Pidaparthi and Lent show how the adiabatic behaviour is recovered. Moreover, their numerical results indicate that the LZE does not appear to be a practicable approach to achieving super-adiabaticity, except perhaps in a very narrow region of system conditions and size. Moreover, the results of Pidaparthi and Lent show the occurrence of a local minimum of dissipation with respect to switching time that would lead to engineering difficulties in increasing computer size for this parameter range. In any case, our asymptotic results stand, although it would be instructive to characterise the parameter range in which the LZE is exploitable and in which the aforementioned engineering difficulties arise.

II.3. Classical Architectures I: A General Lagrangian Approach

If use of the QZE is not possible, perhaps because of high temperatures or an excessive Zeno rate, then Fermi's golden rule applies, giving $\delta p \propto \delta t$. Consequently $\dot{h}$—and thus $\dot{H}$—is subject to a finite lower bound. We find $\dot{h} \geq n\dot{p}\log g$, yielding $\dot{H} \geq E_C\langle \dot{p}\log g/\varepsilon\rangle$. Combining with our constraint $\dot{H} \leq \hat{\beta}P$, the inequality in R_C becomes

$$R_C \leq \left\langle \frac{\dot{p}\log g}{\varepsilon/h_P} \right\rangle^{-1} \frac{P}{k\hat{T}}$$

where h_P here is Planck's constant, and so the scaling law falls to $R_C \lesssim A \sim V^{2/3}$. That is, any ballistic system not making use of the QZE will be subject to the same areametric limit as irreversible computers.

The breakdown of the QZE corresponds to an increase in thermal coupling. By abandoning the desire to maintain well defined quantum states, we instead enter the incoherent regime of the classical realm wherein quantum effects are significantly suppressed. Proceeding generally, we adopt a Lagrangian formalism in order to remain noncommittal in our choice of coordinates $\vec{q}$.

We introduce a reasonably general Lagrangian, $\mathcal{L} = T - V$,

$$\mathcal{L} = [\tfrac{1}{2}c_{ij}\dot{q}_i\dot{q}_j] - [V + w_i\dot{q}_i + \mathcal{O}(\dot{\vec{q}}^3)] \equiv \tfrac{1}{2}\dot{q}^\top C\dot{q} - V - W\dot{q} + \mathcal{O}(\dot{q}^3)$$

[6] The author gratefully acknowledges Mike Frank for pointing them towards this body of work.

where the coefficients c, V, w... may depend continuously on the coordinates $\vec{q}$, and Einstein notation is used for repeated indices. We have rewritten the Lagrangian in matrix form for concision, where C is a matrix, V a scalar, W a row vector, and q a column vector. We assume the system is (effectively) closed, self-contained, or otherwise independent of its environment, and thus $\mathcal{L}$ will not formally depend on the time parameter. In addition, this property implies invariance under global translation of any of its coordinates and thus conservation of all associated momenta. We obtain the Hamiltonian $\mathcal{H}$ and generalised momenta thus

$$\begin{aligned}\mathcal{H} &= \frac{\partial \mathcal{L}}{\partial \dot{q}_i}\dot{q}_i - \mathcal{L} = \tfrac{1}{2}\dot{q}^\top C\dot{q} + V + \mathcal{O}(\dot{q}^3)\\ p_i &= \frac{\partial \mathcal{L}}{\partial \dot{q}_i} = C\dot{q} - W + \mathcal{O}(\dot{q}^3)\end{aligned}$$

Notice that the terms of our potential linearly dependent on the velocities drop out of the Hamiltonian. Those dependent on the cube or higher remain, but will turn out to be negligible at the velocities optimal for computation (and are typically not physically relevant in any case).

For brief collisions, the change in coordinates is negligible and so the spatiotemporal variance in coefficients can be neglected. In this case, we should conserve the Hamiltonian and momenta. We introduce superscripts to the coordinates, $q_i^{(n)}$, to represent different particles. We also introduce notation for the velocities before, u, after, v, and their change, $\Delta u = v - u$. For a collision between particles m and n,

$$\begin{aligned}\Delta\mathcal{H} &= [\tfrac{1}{2}v^{(m)\top}C^{(m)}v^{(m)} - \tfrac{1}{2}u^{(m)\top}C^{(m)}u^{(m)}] + [\tfrac{1}{2}v^{(n)\top}C^{(n)}v^{(n)} - \tfrac{1}{2}u^{(n)\top}C^{(n)}u^{(n)}]\\ &= \tfrac{1}{2}(u^{(m)} + v^{(m)})^\top C^{(m)}\Delta u^{(m)} + \tfrac{1}{2}(u^{(n)} + v^{(n)})^\top C^{(n)}\Delta u^{(n)} = 0\\ \Delta p &= C^{(m)}\Delta u^{(m)} + C^{(n)}\Delta u^{(n)} = 0\,,\end{aligned}$$

where we use the fact that C is symmetric. Substituting Δp into $\Delta\mathcal{H}$, we find,

$$\begin{aligned}2\Delta\mathcal{H}^{(m,n)} &= [(u^{(m)} + v^{(m)}) - (u^{(n)} + v^{(n)})]^\top C^{(m)}\Delta u^{(m)}\\ 0 &= [(2u^{(m)} + \Delta u^{(m)}) - (2u^{(n)} + \Delta u^{(n)})]^\top C^{(m)}\Delta u^{(m)}\\ &= [2(u^{(m)} - u^{(n)}) + (1 + C^{(n)-1}C^{(m)})\Delta u^{(m)}]^\top C^{(m)}\Delta u^{(m)}\\ &= [2(u^{(m)} - u^{(n)}) + (C^{(m)-1} + C^{(n)-1})C^{(m)}\Delta u^{(m)}]^\top C^{(m)}\Delta u^{(m)}\\ &\equiv [2(u^{(m)} - u^{(n)}) + \mu^{(m,n)-1}C^{(m)}\Delta u^{(m)}]^\top C^{(m)}\Delta u^{(m)}\\ &= [2\underbrace{\mu^{(m,n)+1/2}(u^{(m)} - u^{(n)})}_{-y} + \underbrace{\mu^{(m,n)-1/2}C^{(m)}\Delta u^{(m)}}_{x}]^\top \mu^{(m,n)-1/2}C^{(m)}\Delta u^{(m)}\,,\end{aligned}$$

where μ is the generalised reduced mass of the (m, n) system. We can solve for x, and thereby $\Delta u^{(m)}$, by rewriting thus,

$$|x - y|^2 = |y|^2\,.$$

This form reflects the fact that we lack sufficient constraints to unambiguously solve for the post-collision picture. In Cartesian coordinates for point particles, this manifests as an unknown direction of motion afterwards. This is usually resolved by introducing the constraint that momenta perpendicular to the normal vector between the colliding bodies are unchanged. In our generalised coordinate system, the appropriate constraint is unspecified. The general solution is given by

$$x = y + |y|\hat{n} = |y|(\hat{y} + \hat{n})$$

where $\hat{n}$ is a unit vector of unknown direction. We now find an expression for the change in kinetic energy of our particle of interest, (m),

$$\begin{aligned}\Delta\mathcal{H}^{(m)} &= (u^{(m)} + \tfrac{1}{2}\Delta u^{(m)})^\top C^{(m)} \Delta u^{(m)} \\ &= u^{(m)\top} \mu^{(m,n)1/2} |y|(\hat{y} + \hat{n}) + \mathcal{O}(\Delta u^{(m)2}) \\ &= \eta |u^{(m)\top} \mu^{(m,n)} (u^{(m)} - u^{(n)})| + \mathcal{O}(\Delta u^{(m)2})\end{aligned}$$

where we have introduced a factor $\eta \in [-2, 2]$ to take into account our uncertainty in the final direction, and where we have assumed $|\Delta u| \ll u$. Being more careful, we can determine $\Delta u^{(m)}$ thus,

$$\begin{aligned}\Delta u^{(m)} &= C^{(m)-1} \mu^{(m,n)+1/2} |\mu^{(m,n)+1/2} (u^{(n)} - u^{(m)})| (\hat{y} + \hat{n}) \\ |\Delta u^{(m)}| &= \eta' |C^{(m)-1} \mu^{(m,n)} (u^{(n)} - u^{(m)})| \\ &= \eta' |(C^{(m)} + C^{(n)})^{-1} (C^{(n)} u^{(n)} + C^{(m)} u^{(m)}) - u^{(m)}| \\ &= \eta' |\tilde{u}^{(m,n)} - u^{(m)}|\end{aligned}$$

where $\tilde{u}^{(m,n)}$ is the average of prior velocities, weighted by their generalised masses $C^{(\cdot)}$, and $\eta' \in [0, 2]$ is another uncertainty factor related to η. When $u^{(m)}$ is large compared to $u^{(n)}$, $(u^{(m)} + \Delta u^{(m)}) \approx u^{(m)}$. When it is small, $(u^{(m)} + \Delta u^{(m)}) \approx u^{(n)}$. Thus a better approximation to $\Delta\mathcal{H}$ is given by

$$\Delta\mathcal{H}^{(m)} \approx \eta |\tilde{u}^{(m,n)\top} \mu^{(m,n)} (u^{(m)} - u^{(n)})| \,.$$

$\Delta\mathcal{H}$ gives the energy lost from the computational particle (m) to some environmental particle (n), and thus is the amount of energy that must be supplied to the computational system and dissipated from the thermal system. In order to proceed we must determine the rate of this energy transfer; if we assume that the kinetic dynamics of the system are significantly faster than the computational transitions, as suggested by our inability to sufficiently control the quantum state, then we can assume negligible extant correlations between the positions and velocities of different particles. Thus we can employ a kinetic theory approach to determine this rate.

We first determine the mean free path ℓ, the average distance a particle travels between collisions. Given the vanishing correlations, we are able to treat collisions between different species of particle separately. For spherically symmetric particles, we find $\pi(r^{(\alpha)} + r^{(\beta)})^2 \ell^{(\alpha;\beta)} = 1/n^{(\beta)}$ for the mean free path of α particles between collisions

with β particles, where $n^{(\beta)} = N^{(\beta)}/V^{(\beta)}$ is the number density of β particles. More generally, we can replace $\pi(r^{(\alpha)} + r^{(\beta)})2$ with $A^{(\alpha,\beta)}$, the effective collision cross-section for arbitrarily shaped particles averaged over impact orientation. We also need to determine the average relative speed between α and β particles,

$$\begin{aligned}
\bar{v}_{\text{rel}}^{(\alpha,\beta)2} &= \langle (v^{(\alpha)} - v^{(\beta)})^2 \rangle \\
&= \langle v^{(\alpha)2} \rangle + \langle v^{(\beta)2} \rangle - 2\langle v^{(\alpha)} \cdot v^{(\beta)} \rangle \,, \\
\bar{v}_{\text{rel}}^{(\alpha,\beta)} &= \sqrt{\bar{v}^{(\alpha)2} + \bar{v}^{(\beta)2}} \,, \\
\nu^{(\alpha;\beta)} &= \bar{v}_{\text{rel}}^{(\alpha,\beta)} / \ell^{(\alpha;\beta)} \\
&= A^{(\alpha,\beta)} n^{(\beta)} \sqrt{\bar{v}^{(\alpha)2} + \bar{v}^{(\beta)2}} \,,
\end{aligned}$$

where we have used the fact that the velocities are uncorrelated to cancel the cross term. Summing over these rates for each class, we can find the rate of loss of kinetic energy for α particles,

$$\dot{\mathcal{H}}^{(\alpha)} = \sum_{\beta} \eta^{(\alpha;\beta)} |\tilde{u}^{(\alpha,\beta)\top} \mu^{(\alpha,\beta)} (u^{(\beta)} - u^{(\alpha)})| A^{(\alpha,\beta)} n^{(\beta)} \sqrt{\bar{u}^{(\alpha)2} + \bar{u}^{(\beta)2}} \,.$$

When α particles are heavy and fast, this reduces to

$$\begin{aligned}
\dot{\mathcal{H}}^{(\alpha)} &= \sum_{\beta} \eta^{(\alpha;\beta)} |u^{(\alpha)\top} C^{(\beta)} u^{(\alpha)}| A^{(\alpha)} n^{(\beta)} \bar{u}^{(\alpha)} \\
&= \sum_{\beta} \eta^{(\alpha;\beta)} \rho^{(\beta)} A^{(\alpha)} \bar{u}^{(\alpha)3}
\end{aligned}$$

where $\rho = n|C|$ is the generalised density. Notice that this takes the same form as the equation for hydrodynamic drag, taking the drag coefficient to be 2η. Assuming there exists some correspondence between the generalised velocity $u^{(\alpha)}$ and the rate of computation, we find

$$P \geq \sum_{\alpha} N^{(\alpha)} A^{(\alpha)} \left(\ell^{(\alpha)} \frac{R^{(\alpha)}}{N^{(\alpha)}} \right)^3 \sum_{\beta} \eta^{(\alpha;\beta)} \rho^{(\beta)}$$

where α ranges over computational particles and ℓ is the characteristic generalised displacement corresponding to a single computational transition. To maximise the net computational rate $R = \sum_{\alpha} R^{(\alpha)}$, we find that we need to let $\bar{u}^{(\alpha)} \to 0$, but this violates our assumption that α particles are fast. For finite velocity computational particles, we thus find that our net computational rate scales as $R \lesssim A \sim V^{2/3}$.

In the limit of slow α particles, $|u^{(\alpha)}| \ll |u^{(\beta)}|$, $\dot{\mathcal{H}}$ instead reduces to

$$\dot{\mathcal{H}}^{(\alpha)} = \sum_{\beta} \eta^{(\alpha;\beta)} |\tilde{u}^{(\alpha,\beta)\top} C^{(\beta)} u^{(\alpha)}| A^{(\alpha)} n^{(\beta)} \bar{u}^{(\beta)}$$

where we have taken $\langle u^{(\beta)} - u^{(\alpha)} \rangle = \bar{u}^{(\alpha)}$. If the generalised mass of the α particles is not significantly heavier than the β particles, then $\tilde{u}^{(a,b)}$ will have a strong dependence on $u^{(\beta)}$ and this will lead to the same scaling limit of $R \lesssim A$. In order to improve on this, we must let the α particles be significantly heavier. In this limit, $\tilde{u}^{(\alpha,\beta)} \approx u^{(\alpha)}$ and we get

$$\begin{aligned}
\dot{\mathcal{H}}^{(\alpha)} &= \sum_{\beta} \eta^{(\alpha;\beta)} |u^{(\alpha)\top} C^{(\beta)} u^{(\alpha)}| A^{(\alpha)} n^{(\beta)} \bar{u}^{(\beta)} \,, \\
P &\geq \sum_{\alpha} N^{(\alpha)} A^{(\alpha)} \left(\ell^{(\alpha)} \frac{R^{(\alpha)}}{N^{(\alpha)}} \right)^2 \sum_{\beta} \eta^{(\alpha;\beta)} \rho^{(\beta)} \bar{u}^{(\beta)} \\
&\geq \sum_{\alpha} \frac{R^{(\alpha)2}}{N^{(\alpha)}} \underbrace{A^{(\alpha)} \ell^{(\alpha)2} \sum_{\beta} \eta^{(\alpha;\beta)} \rho^{(\beta)} \bar{u}^{(\beta)}}_{\gamma^{(\alpha)}} \,.
\end{aligned}$$

To proceed, we maximise the rate subject to this power constraint,

$$\Lambda = \sum_{\alpha} R^{(\alpha)} - \frac{1}{2\lambda} \sum_{\alpha} \frac{R^{(\alpha)2} \gamma^{(\alpha)}}{N^{(\alpha)}} \qquad \Longrightarrow \qquad 0 = 1 - \frac{R^{(\alpha)} \gamma^{(\alpha)}}{\lambda N^{(\alpha)}}$$

$$\begin{aligned}
\frac{R}{\lambda} &= \sum_{\alpha} \frac{n^{(\alpha)}}{N \gamma^{(\alpha)}} N = N \left\langle \frac{1}{\gamma^{(\alpha)}} \right\rangle \\
P &\geq \sum_{\alpha} \lambda R^{(\alpha)} = \lambda R = \frac{R^2}{N} \left\langle \frac{1}{\gamma^{(\alpha)}} \right\rangle^{-1} \\
R &\leq \sqrt{PN \left\langle \frac{1}{\gamma} \right\rangle}
\end{aligned}$$

where $N = \sum_{\alpha} N^{(\alpha)}$ is the total number of computational particles. To maximise the computational rate R, then, we let N scale with the volume of the system, leading once again to the scaling law

$$R \lesssim \sqrt{AV} \sim V^{5/6} \,.$$

II.4. Classical Architectures II: Brownian Machines

Whilst the previous two sections should suffice to cover all physical computational systems, the high mass-low speed limit of the classical system is not ideal from an engineering perspective. For improved practicality, we reformulate this limit in terms of an abstract chemical reaction network (CRN) near equilibrium, and show that we obtain the same result. This result is thus more robust in terms of its attainability.

Entropy generation rate We first seek a general expression for the rate of entropy generation for a chemical reaction network. Consider any such dynamical system consisting of a set of species $\{C_j : j\}$ and a set of reversible reactions $\Gamma = \{\nu_{ij} X_j \longleftrightarrow_{\gamma_i} \nu'_{ij} X_j : i\}$, where we sum over j and the ν_{ij} are stoichiometries. By the convergence theorem for reversible Markov systems, the system will converge to a unique steady state/equilibrium distribution for any initial conditions. As the reactions are reversible, the steady state will also satisfy detailed balance such that the forward and backward rates coincide separately for each reaction, i.e. $\overrightarrow{R}_i = \overleftarrow{R}_i$, where R is a rate of the whole system rather than per unit volume.

Let us look for a quantity H that measures progress towards equilibrium and satisfies the following properties:

1. H is a state function of the system[7], depending only on its instantaneous description;
2. H increases monotonically with time;
3. H is additive, i.e. $H(\cup_i V_i) = \sum_i H(V_i)$ for any set of disjoint regions V_i.

By properties 1 and 2 and the convergence theorem, H will approach a unique maximum at equilibrium. Microscopically, reactions occur discretely and so the rate of change of H can be written

$$\dot{H}_i = \overrightarrow{R}_i \overrightarrow{\Delta h}_i + \overleftarrow{R}_i \overleftarrow{\Delta h}_i = (\overrightarrow{R}_i - \overleftarrow{R}_i)(h_{\text{reactants}} - h_{\text{products}})$$

for any reaction i, where we have used reversibility to identify $\overrightarrow{\Delta h}_i = -\overleftarrow{\Delta h}_i = h_{\text{reactants}} - h_{\text{products}}$ and where $h(\sum_j \nu_{ij} X_j)$ is the entropy of the region associated with precisely ν_{ij} particles of each species X_j. In order to satisfy property 2, the sign of $\overrightarrow{\Delta h}_i$ must equal that of $\overrightarrow{R}_i - \overleftarrow{R}_i$.

Now, consider fixing h_{products} and $\overleftarrow{R}_i$, and varying $\overrightarrow{R}_i$; this is possible unless the reaction is trivial with $\nu_{ij} = \nu'_{ij}$ for all species X_j, i.e. it does nothing. To maintain property 2, it must therefore be the case that $\overrightarrow{\Delta h}_i = h(\overrightarrow{R}_i) - h(\overleftarrow{R}_i)$. To satisfy property 3, we require $h(\alpha \overrightarrow{R}_i) - h(\alpha \overleftarrow{R}_i) = h(\overrightarrow{R}_i) - h(\overleftarrow{R}_i)$ where α is some arbitrary scaling factor of the system. We can therefore infer the functional form $h(x) = \log_b ax$ for some constants a and b.

We have therefore derived an entropy-like quantity, unique up to choice of logarithmic unit via b and entropy at $T = 0$ via a. Without loss of generality we choose $a = 1$ and $b = e$, and therefore find that

$$\dot{H} = \sum_i (\overrightarrow{R}_i - \overleftarrow{R}_i) \log \frac{\overrightarrow{R}_i}{\overleftarrow{R}_i} \equiv 2 \sum_i R_i \beta_i \operatorname{arctanh} \beta_i$$

where $\beta_i = (\overrightarrow{R}_i - \overleftarrow{R}_i)/(\overrightarrow{R}_i + \overleftarrow{R}_i)$ is the effective bias of reaction i and $R_i = \overrightarrow{R}_i + \overleftarrow{R}_i$ is its gross reaction rate.

[7] More accurately, on the joint state of an ensemble of iid systems.

Entropy generation rate for elementary chemical reactions We now calculate our entropy quantity for elementary chemical reactions and compare against the expected value. The rates for elementary chemical reactions are given by collision theory as

$$\vec{r}_i = \vec{k}_i \prod_j [X_j]^{\vec{\nu}_{ij}} \qquad\qquad \overleftarrow{r}_i = \overleftarrow{k}_i \prod_j [X_j]^{\overleftarrow{\nu}_{ij}}$$

where $\vec{k}_i$ and $\overleftarrow{k}_i$ are rate constants and r are rates per unit volume. Assuming no inter-particle interactions, the canonical entropy is given by the Sackur-Tetrode equation which can be obtained simply by considering the spatial distribution of the particles along with any other degrees of freedom. For species X_i, consider an arbitrary volume V_i available to it; if the thermal volume of the particles is $\mathcal{V}_i = \Lambda^3$ where Λ is the thermal *de Broglie* wavelength, then there will be $V_i/\mathcal{V}_i$ loci available to the particle, and so the associated entropy within the volume will be

$$N_i h(X_i) = N_i \log \frac{V_i}{\mathcal{V}_i} - \underbrace{\log N_i!}_{\text{Gibb's factor}} + N_i(\varepsilon_i - 1)$$
$$h(X_i) = \varepsilon_i - \log[X_i]\mathcal{V}_i + \mathcal{O}\left(\frac{\log N_i}{N_i}\right)$$

where $N_i = [X_i]V_i$ is the number of particles in the volume, and $[X_i]$ the concentration. We have also taken into account indistinguishability of the particles with the Gibb's factor, and allowed for additional degrees of freedom with the ε_i term. Strictly speaking the spatial distribution should be calculated combinatorially via the multinomial coefficient

$$\log \frac{(V/\mathcal{V})!}{((V/\mathcal{V} - \sum N_i))! \prod N_i!}$$

where we have assumed all the particles inhabit the same region and have the same wavelength. Applying Stirling's approximation and assuming that $\sum N_i \ll V/\mathcal{V}$, this reduces to $\sum N_i(1 - \log[X_i]\mathcal{V})$ however, so our simplistic derivation is valid.

Therefore we find the canonical entropy change due to reaction i is given by

$$\Delta h_i = \sum_j (\overleftarrow{\nu}_{ij} - \vec{\nu}_{ij}) h(X_j) = \sum_j (\overleftarrow{\nu}_{ij} - \vec{\nu}_{ij})(\varepsilon_j - \log \mathcal{V}_j) - \sum_j (\overleftarrow{\nu}_{ij} - \vec{\nu}_{ij}) \log[X_j]$$

using a slight abuse of notation (the logarithmands are not unitless, but their combination is). Comparing with our quantity $\Delta h = \log \vec{r}/\overleftarrow{r}$, we find

$$\Delta h_i = \log \frac{\vec{k}_i}{\overleftarrow{k}_i} - \sum_j (\overleftarrow{\nu}_{ij} - \vec{\nu}_{ij}) \log[X_j]$$
$$\frac{\vec{k}_i}{\overleftarrow{k}_i} = \left(\prod_j \mathcal{V}_j^{\vec{\nu}_{ij} - \overleftarrow{\nu}_{ij}}\right)\left(\exp \sum_j (\vec{\nu}_{ij} - \overleftarrow{\nu}_{ij})\varepsilon_j\right)$$

which should be compared to the Arrhenius equation. Indeed, the $\mathcal{V}_i$ terms confer the appropriate units to the pre-exponential factor and, by expanding the enthalpy term in terms of k_BT as $\varepsilon_i = \varepsilon_i^{(0)} + \frac{1}{k_BT}\varepsilon_i^{(1)} + \mathcal{O}(\frac{1}{k_BT})^2$, we can group the roughly constant terms of the enthalpy into the pre-exponential factor, leaving the exponential term in the form $e^{-\Delta E/k_BT}$.

Maximising performance in Brownian machines We wish to maximise the net reaction rate of the system by selecting the biases β_i of each individual reaction, subject to bounded total entropy production. If $R = \sum R_i$ is the total gross reaction rate, then the total net rate is $R_c = \sum R_i\beta_i = R\langle\beta\rangle$ where the expectation value is with respect to the fractional contribution of each reaction, i.e. weighted by R_i/R. We denote this by R_c assuming that each reaction contributes to computational progress; if this is not true then R_c will be less than this value. The entropy rate is similarly given by $\dot{H} = 2R\langle\beta \operatorname{arctanh}\beta\rangle$. We use a Lagrange multiplier approach as usual,

$$\Lambda = R\langle\beta\rangle - \lambda R\langle\beta \operatorname{arctanh}\beta\rangle \qquad \Longrightarrow \qquad R_i = \lambda R_i \partial_{\beta_i}\beta_i \operatorname{arctanh}\beta_i;$$

that is, the optimum is achieved by setting all the β_i equal to a constant, β. This gives $\dot{H} = 2R\beta \operatorname{arctanh}\beta = \frac{2R_c^2}{R} + \mathcal{O}(\frac{R_c^4}{R^3})$ and hence

$$R_c \leq \sqrt{\frac{PR}{2kT}}\,,$$

leading once again to the scaling limit $R_c \lesssim \sqrt{AV} \sim V^{5/6}$, as the power P scales with area and the gross computation rate R with volume. Here we have presented a constructive proof of this scaling limit, as any valid reversible CRN implementation will yield this scaling limit.

II.5. Relativistic Effects at Scale

At different scales, from the microscopic to the cosmic, different constraints predominate in the analysis of maximum computation rate. For macroscopic systems at worldly scales, the aforementioned thermodynamic constraints are most directly relevant, yielding an upper bound of $R_C \lesssim \sqrt{AV}$. At sufficiently small scales, however, the surface area to volume ratio will be great enough that this thermodynamic constraint will no longer be limiting, with the volumetric Margolus-Levitin bound instead dominating. This reflects the fact that there is simply insufficient energy enclosed in the system for the resultant rate of computation to saturate the heat dissipation capacity at the boundary. If one were to use a denser computational architecture, the computational capacity of the region could be increased to the point that the power-flux bound is saturated, and thus recovering the more restrictive $\sqrt{AV}$ bound.

It transpires that there are two further regimes. As our systems approach cosmic scales, the threat of gravitational collapse becomes pertinent. In order to avoid this fate,

we must lower the average density such that the system's radius always exceeds its Schwarzschild radius[8]. As the Schwarzschild radius is proportional to the mass of the system, this implies that the computational rate of such large systems varies *linearly* in radius. In fact, there is an intermediate regime: as we have not been utilising the full computational potential of our mass due to the thermodynamic constraint, we can gradually increase said utilisation whilst simultaneously reducing our overall density until the Margolus-Levitin limit is attained, at which point we default to the linear regime. This intermediate regime is thus still described by the $R_C \lesssim \sqrt{PM}$ limit, which in this post-Schwarzschild realm equates to $R_C \lesssim V^{1/2}$. In summary, the scaling laws of these four regimes go as V, $V^{5/6}$, $V^{1/2}$ and $V^{1/3}$, in order of increasing scale.

A more detailed analysis reveals that these large systems are subject to a further consideration. So far we have assumed a Galilean invariance worldview. At small and medium scales, this approximation is very good. At larger scales, however, relativistic effects threaten to reduce our overall computation rate via time dilation, and at even larger scales they can constrain the amount of mass we can fit within a volume lest the system undergo gravitational collapse as mentioned earlier. Special relativistic time dilation is easily avoided by minimising relative motion within the computational parts of the system (this implies that message packets will have reduced computational capacity compared to the static computational background, but message packets can generally be assumed static anyway).

Gravitational time dilation is a more serious issue, which cannot be eluded at large scales. In the case of a hypothetical supermassive computational system, local computation proceeds unabated, but distant users interfacing with the system will observe slower than expected operation, as depicted in Figure II.2c. To calculate this slowdown factor, we shall proceed by modelling the system as spherically symmetric. This is not unreasonable at these scales, where a body's self-gravitation makes maintaining other geometries unstable and impractical. We will not consider rotational systems which could allow for an ellipsoidal shape, as the requisite angular velocities would almost certainly abrogate nearly all computational progress solely due to special relativistic effects.

The metric[9] of an isotropic and spherically symmetric body in hydrostatic equilibrium can be obtained via the *Tolman-Oppenheimer-Volkoff* (TOV) equation [73],

$$\begin{aligned}
\frac{\mathrm{d}P}{\mathrm{d}r} &= -\frac{Gm}{r^2}\rho\Big(1+\frac{P}{\rho c^2}\Big)\Big(1+\frac{4\pi r^3 P}{mc^2}\Big)\Big(1-\frac{2Gm}{rc^2}\Big)^{-1} \\
\frac{\mathrm{d}\nu}{\mathrm{d}r} &= -\Big(\frac{2}{P+\rho c^2}\Big)\frac{\mathrm{d}P}{\mathrm{d}r} \\
&= \frac{2Gm}{r^2c^2}\Big(1+\frac{4\pi r^3 P}{mc^2}\Big)\Big(1-\frac{2Gm}{rc^2}\Big)^{-1} \\
\mathrm{d}s^2 &= e^{\nu}c^2\mathrm{d}t^2 - \Big(1-\frac{2Gm}{rc^2}\Big)^{-1}\mathrm{d}r^2 - r^2\mathrm{d}\Omega^2\,,
\end{aligned} \tag{II.1}$$

[8] In reality, the threshold radius for gravitational collapse exceeds the Schwarzschild radius by a factor not less than $\frac{9}{8}$. The reasons for this shall be discussed in due course.

[9] We use a $(+---)$ signature.

where P is the pressure, $m(r)$ is the mass enclosed by the concentric spherical shell of radius r, and ρ is the local mass-energy density. All masses are as observed by a distant observer.

The rate of dynamics of a point P as observed from a point Q is well known to be

$$\frac{R(Q)}{R(P)} = \sqrt{\frac{g_{00}(P)}{g_{00}(Q)}},$$

where $g_{\mu\nu}$ is the metric tensor. We assume our distant observer to be moving slowly (relative to the computer) in approximately flat space, and thus that $g_{00}(Q) = 1$. This yields a slowdown factor of $f(P) = \sqrt{g_{00}(P)}$. The total slowdown for an extended body is hence found to be

$$\begin{aligned} f &= \frac{1}{M}\int_V \mathrm{d}V \rho\sqrt{g_{00}} \\ &= \frac{1}{M}\int_0^{r_1} \mathrm{d}r \frac{\mathrm{d}m}{\mathrm{d}r} e^{\nu/2}, \qquad \text{(spherically symmetric)} \end{aligned}$$

where M is the total mass and r_1 is the least upper bound of its radial extent.

In order to maximise f, we must clearly maximise ν throughout the body. It can be seen that $\frac{\mathrm{d}\nu}{\mathrm{d}r} \geq 0$, and in fact within solid regions where $\rho > 0$ this inequality is strict. Furthermore, in empty space the TOV equation breaks down; instead, we use our Schwarzschild matching conditions to find

$$\begin{aligned} \frac{\mathrm{d}\nu}{\mathrm{d}r} &= \frac{2Gm}{r^2c^2}\left(1 - \frac{2Gm}{rc^2}\right)^{-1} \\ \Delta\nu &= \Delta \log\left(1 - \frac{2Gm}{rc^2}\right), \end{aligned}$$

which shows that this inequality is always strict whenever $m > 0$. ν at the surface is fixed by the Schwarzschild metric, in accordance with Birkhoff's theorem, and thus for a fixed extent our only control over ν is the distribution m between the surface and core. Approaching the core from the surface, ν strictly decreases until m vanishes. Furthermore $\mathrm{d}\nu/\mathrm{d}r$ is at least as great as in empty space, being strictly greater in massive regions. Therefore it is desirable to concentrate mass towards the surface.

We assume that the density of our architecture of choice is bounded from above, thus limiting the degree to which we can concentrate mass towards the surface. The optimum is then achieved by a thick shell of limiting density from the surface inwards. The two extreme cases of a thin shell and a solid sphere can be treated exactly, but an arbitrarily thick spherical shell must be treated numerically.

Solid Sphere A solid sphere coincides with the Schwarzschild geometry. The usual Schwarzschild metric corresponds to the exterior of the object, instead we must use the interior solution which can be derived from the TOV equation (II.1), yielding

$$\sqrt{g_{00}} = \frac{3}{2}\sqrt{1 - \frac{r_s}{r_1}} - \frac{1}{2}\sqrt{1 - \left(\frac{r}{r_1}\right)^2 \frac{r_s}{r_1}}$$

where $r_s = 2GM/c^2$ is the Schwarzschild radius. Before computing f, note that g_{00} vanishes when $r_s = \frac{8}{9}r_1$. This limit is in fact of great importance; whilst an object of size $r_1 = \frac{9}{8}r_s$ would exceed the Schwarzschild radius and be presumed stable against gravitational collapse, this is not the case. At this point, such an object is unstable towards spontaneous oscillations, its core pressure diverges, and it readily collapses to a black hole [74]. Furthermore, depending on its heat capacity ratio γ, a gaseous hydrostatic sphere may be unstable at even larger radii as tabulated by Chandrasekhar [62]. This $r_1 \geq \frac{9}{8}r_s$ threshold is thus a stronger bound on the size of massive objects. Integrating $\sqrt{g_{00}}$ over the mass of this solid sphere, we find

$$f_{\text{solid}} = \frac{3}{16v_s}((1+6v_s)\sqrt{1-v_s} - v_s^{-1/2}\arcsin\sqrt{v_s})$$

where we have introduced the normalised coordinate $v = r/r_1$. At the $v_s = \frac{8}{9}$ threshold, we get $f_{\text{solid}} \approx 0.1699$.

Thin Shell For a thin shell, we first identify that the pressure on the outer boundary vanishes. Let the inner radius be r_0 and the thickness be $\delta r = r_1 - r_0$ such that $M \approx 4\pi\rho r_1^2\delta r$ and $\delta v \ll 1$. We also introduce the unitless variable $u = P/\rho c^2$,

$$\begin{aligned}
\frac{\mathrm{d}u}{\mathrm{d}r} &= -\frac{4\pi\rho r G}{c^2}(1+u)\Big(\frac{r-r_0}{r}+u\Big)\Big(1-\frac{2Gm}{rc^2}\Big)^{-1} \\
\frac{\mathrm{d}u}{\mathrm{d}v} &= -\frac{4\pi\rho r r_1 G}{c^2}(1+u)\Big(\frac{r-r_0}{r}+u\Big)\Big(1-\frac{2GM}{c^2}\frac{r-r_0}{r\delta r}\Big)^{-1} \\
&\approx -\frac{v_s}{2\delta v}(u+1)(u+\beta\delta v)(1-\beta v_s)^{-1} \\
\Delta u &\approx -\tfrac{1}{2}v_s(u+1)(u+\beta\delta v)(1-\beta v_s)^{-1}\,,
\end{aligned}$$

where $\beta = \frac{r-r_0}{r_1-r_0} \in [0,1]$. Integrating from the surface inwards, we have $u_1 = 0$ and $\beta = 1$, yielding

$$u_0 = \frac{\delta v}{2}\frac{v_s}{1-v_s} + \mathcal{O}(\delta v^2)\ .$$

Therefore in the limit $\delta v \to 0$, the pressure throughout the thin shell vanishes. As a result, we also have that ν is constant for all $v \in [0,1]$, taking the value $1 - v_s$ and giving

$$f_{\text{thin}} = \sqrt{1-v_s}\,.$$

At the threshold, we get $f_{\text{thin}} = \frac{1}{3}$.

Thick Shell For general δv, we integrate numerically. We rewrite the TOV equations in normalised and unitless form thus

$$\frac{\mathrm{d}\log g}{\mathrm{d}v} = \frac{vv_s}{2}\frac{\mu+3u}{\mu_1 - v^2 v_s\mu}\,, \qquad \frac{\mathrm{d}u}{\mathrm{d}v} = -(1+u)\frac{\mathrm{d}\log g}{\mathrm{d}v}\,, \qquad \frac{\mathrm{d}f}{\mathrm{d}v} = -\frac{3v^2 g}{\mu_1}\,,$$

where $\mu = 1-(v_0/v)^3$, $\mu_1 = 1 - v_0^3$ and $g \equiv \sqrt{g_{00}}$. We integrate the system from $v = 1$ down to $v = v_0$, with initial conditions $u_1 = 0$, $g_1 = \sqrt{1-v_s}$ and $f_1 = 0$. Our slowdown factor is given by f_0, and explicit results are given in the following section.

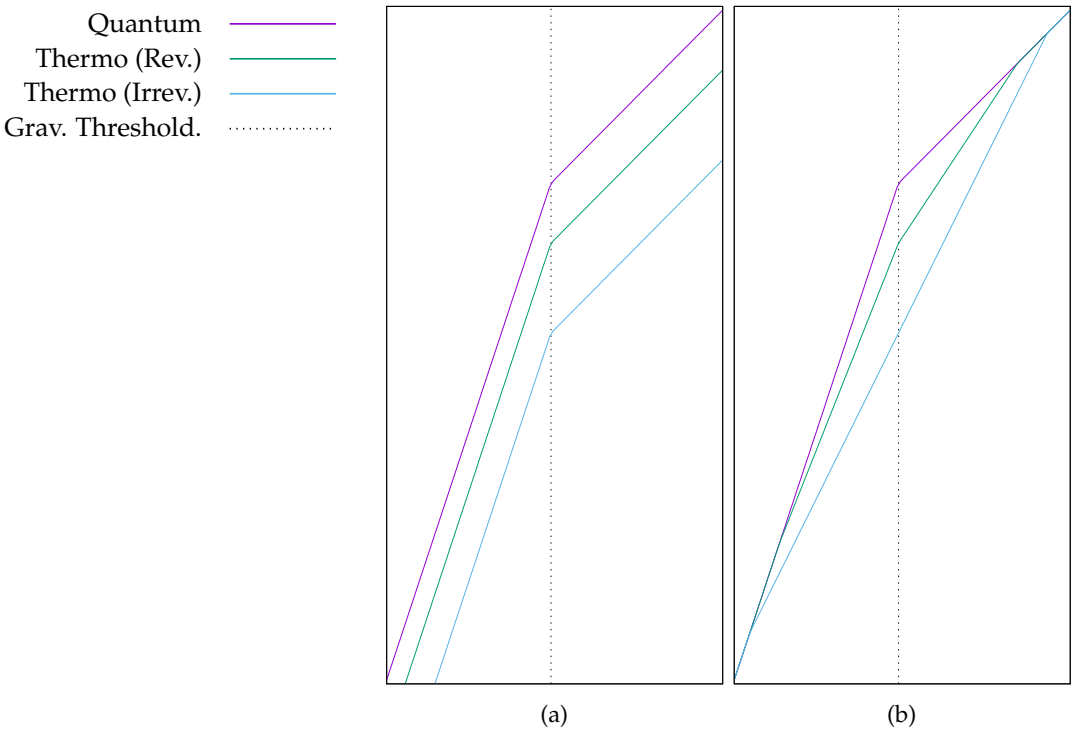
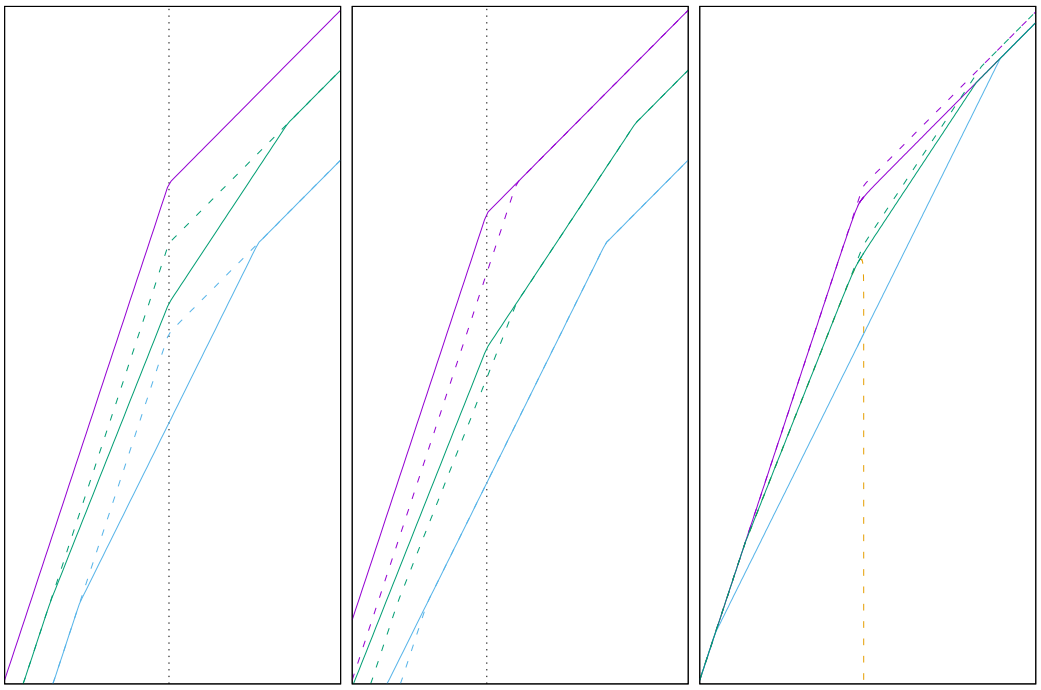


Figure II.3: A series of idealised examples of the relevant bounds on computation rate. In all plots, the vertical axis represents rate of computation ν in arbitrary logarithmic units, and the horizontal axis the radius r of the system in arbitrary logarithmic units. The axes have the same scaling, so that a 45° slope corresponds to a $\nu \propto r$ power law.

(a) If power supply and heat/entropy dissipation is not limiting, then the rate of computation is proportional to the mass of the system. In these examples, the 'technology' is fixed so that the computational systems have uniform density. Therefore we expect a power law $\nu \propto r^3$. Up to a certain threshold radius this is indeed what we see. The *Quantum* line corresponds to the Margolus-Levitin quantum constraint on computational performance, $\nu \leq mc^2/h_P$. The *Thermo (Rev.)* line takes into account that a realistic computer might not achieve this upper bound. The *Thermo (Irrev.)* line takes into account that the computational components of an irreversible computer can be no faster than those of a reversible one, and typically they will be slower as it takes time to erase the excess information. Often this erasure will be via a damping process. At the threshold (marked with a vertical dotted line), the system is on the cusp of gravitational collapse as its radius coincides with its Schwarzschild radius. Beyond this point, the mass scales linearly with radius, hence the power laws all fall to $\nu \propto r$ beyond this point.

(b) In this example we assume the reversible and irreversible architectures saturate the quantum bound at the microscopic level. A realistic computer will be constrained by the rate at which free energy can be supplied and entropy removed. This scales with r^2, and for an irreversible computer this leads to a power law $\nu \propto r^2$ because the rate of entropy generation and free consumption per unit computational mass is bounded from below. At sufficiently small scales, the quantum bound is more limiting and so the power law switches to $\nu \propto r^3$; similarly, at sufficiently large scales the quantum bound is again more limiting and the power law switches to $\nu \propto r$. The case for reversible computers is more complicated, as analysed in this chapter. The power laws are $\nu \propto r^{5/2}$ below the threshold and $\nu \propto r^{3/2}$ above. (continued on next page)



(c) (d) (e)

Figure II.3: (continued) The reversible architecture will always be at least as fast as the irreversible architecture.

(c) This example combines the differing rates of proportionality of (a) with the thermodynamic constraints of (b). The dashed lines show the case where thermodynamics is not limiting.

(d) This example corresponds to (c) except that the density has been increased by a factor 100. The dashed lines are the same as the solid lines of (c). Below the threshold, the Margolus-Levitin limit and reversible computers are faster (by a factor 100 and a factor 10, respectively). The irreversible computer is only faster in the thermodynamically non-limiting range, otherwise the thermodynamic constraint eliminates the benefit of additional computational matter. A second consequence of the increased density is that the Schwarzschild threshold is reached sooner. Nevertheless, for all architectures the computational rate is at least as great as for the lower density case. Therefore increasing density is always helpful at small scales, and not unhelpful at large scales.

(e) This example corresponds to (b) except that gravitational time dilation has been taken into account*. The dashed lines are the same as the solid lines of (b). The orange asymptotically-vertical dashed line shows what happens when the geometry of the computer is maintained as a sphere of uniform density, even as the density is reduced to avoid gravitational collapse. Namely, gravitational time dilation abrogates all computational performance from the perspective of a distant observer. The solid lines optimise the geometry of the system to maximise the observed computational performance from the perspective of a distant observer. Whilst these are reduced in comparison to (b), it is only by a constant factor of at most 3.

*Time dilation corrections were computed by integrating the thick shell system of differential equations using `dopri5`, and the shell thickness was optimised to maximise the rate using a golden section search. The code was written in `Haskell` and is available, complete with integration and optimisation routines, at https://github.com/hannah-earley/revcomp-relativistic-limits.

II.6. Variance in Performance at Different Scales

In order to calculate the maximum performance of a system at different sizes, we compute the optimum speed within each constraint—thermodynamic and Margolus-Levitin—with relativistic corrections, and then pick the least of these upper bounds. To apply the relativistic constraint, we substitute $M = \frac{1}{2}r_s c^2/G$ for mass where r_s is the Schwarzschild radius, and then multiply by our slowdown factor f. For better parametericity, we actually use $r_s = v_s \ell$ where ℓ is the system's radius/linear dimension. This yields the two bounds,

$$R_{\text{thermo.}} = \alpha\sqrt{\ell^3 v_s} f(v_s, v_0)\,, \qquad\qquad R_{\text{marg.lev.}} = \beta \ell v_s f(v_s, v_0)\,,$$

where α and β are constants of proportionality that depend on the system architecture. Notice that there is in fact some choice available in system geometry; we can pick any constant density thick shell solution, with parameters (v_0, v_s). As we increase v_s, the factor f will decrease, and so there will be an optimum pair (v_0, v_s) maximising the given bound. The maximum density constraint is equivalent to setting a maximum value to the mass and hence v_s,

$$\begin{aligned} v_s = \frac{1}{\ell}\frac{2GM}{c^2} &= \ell^2 \frac{8\pi\rho G}{3c^2}(1 - v_0^3) \\ &\leq v_m \equiv \ell^2 \frac{8\pi\rho G}{3c^2} \\ &\leq \frac{8}{9}\,, \end{aligned}$$

where the last constraint is to avoid gravitational collapse. For a given v_m then, we have $v_0 = \sqrt[3]{1 - v_s/v_m}$ and we maximise the two R bounds in $0 < v_s \leq \min(\frac{8}{9}, v_m)$.

An illustrative example of these regimes is depicted in Figure II.3. Notice how the solid sphere, though maximising usable computational matter and thus its local computational rate, has its observable computational rate abrogated as it approaches the point of gravitational self-collapse. This demonstrates how reducing the available matter can enable the observable computational rate to increase, though the difference between the dashed and solid lines shows that there remains a relativistic penalty in the form of gravitational time dilation. We also see that an irreversible system underperforms with respect to a reversible system across a large range of length scales, otherwise matching it at the extremes. The exact range again depends on system architecture specifics, and for some architectures the irreversible system may give transient superior performance at certain scales.

II.7. Discussion

Additional Entropic Sources The preceding analysis has been restricted to dissipating entropy arising in the course of the computation itself. However, a physical computer will be subject to other sources of entropy in addition to this. As mentioned in the

introduction, Bennett calculated that the influence of turbulence in the atmospheres of nearby stars would be sufficient to thermalise a billiard ball computer within a few hundred collisions. Thus, in general, we must aim to shield our computer from such external influences, for example by error correction procedures. Fortunately, the influences of such external sources may only interact with our system via the same boundary through which we dissipate computational entropy, and as such their rate is at most proportional to this surface area[10]. Thus, as long as the external sources are not overwhelmingly strong we should be able to suppress them at all scales with equal ease.

A more challenging source of errors and entropy comes from within. Assuming a non-vanishing system temperature, the entire system volume will generate thermodynamic fluctuations at a commensurate rate. These fluctuations can manifest in a variety of ways, perhaps the most damaging of which are nuclear transitions. Some nuclear transitions, such as radioactive decay leading to the production of damaging radicals, are not of great concern over long timescales as the proportion of unstable isotopes will decay to zero exponentially fast. More problematic is the tendency of all nuclei to decay to the baryonic ground state[11], ^{62}Ni [75] (either by fission or fusion); except for systems composed purely of ^{62}Ni, these spontaneous transitions must be reversed in order to maintain the intended structure. Every event leading to a change in computational state or requiring repair invokes an entropic cost, and given that these events occur with a rate proportional to the computationally active volume, this ultimately recovers an areametric bound to our computational rate,

$$V \leq \frac{P}{kT}\frac{1}{\dot{\eta}_{\text{int.fluc.}}}\,.$$

This does not necessarily render our reversible computation performance gains unattainable, however. Providing $\dot{\eta}_{\text{int.fluc.}}$ is sufficiently small, we can outperform irreversible computers at scales up to

$$\ell \leq \sqrt{\frac{P}{kT}\frac{1}{\dot{\eta}_{\text{int.fluc.}}}\frac{1}{\delta r}}$$

where δr is the thickness of the irreversible computational shell that can be supported by such an architecture. If this threshold size is sufficiently large as to coincide with the Schwarzschild threshold of Figure II.3 then such a reversible architecture would in fact outperform its irreversible analogue at all scales. Furthermore, it is likely that $\dot{\eta}_{\text{int.fluc.}}$ is smaller than it would be in an irreversible system as they have one fewer

[10] Even though such external influences may indeed permeate the entire volume, such as gravitational waves or neutrinos, the *rate* of incidence of these interaction will necessarily scale areametrically. Moreover, gravitational waves and neutrinos only penetrate the entire volume because they are so weakly interacting. A more strongly coupled interaction such as ultraviolet light or particulate matter will primarily affect only the 'skin' of the system, and thus illustrates how such external influences are genuinely only areametric in strength.

[11] In fact, the ground state of nuclear matter depends on density, and at higher densities the equilibrium state may be as high as ^{118}Kr; beyond this, various forms of degenerate matter may dominate [76].

source of fluctuations to combat; namely, irreversible computers expend significant quantities of energy to ensure unidirectional operation, and any deviation from this is a potentially fatal error. In contrast, a reversible computer is intrinsically robust to such 'errors', particularly the Brownian architecture described, which actively exploits this. Expending less energy has the added advantage of permitting lower operating temperatures, further reducing $\dot{\eta}_{\text{int.fluc.}}$.

Mixed Architectures Because of the generality of the $R \lesssim \sqrt{AV}$ scaling law, systems of maximal computational rate can be constructed using any desired mix of architectures in order to meet the requirements of a given problem. Providing entropy and input power can be efficiently transported between the surface and the computational bulk, the entropy generation and power constraints superpose linearly. Thus in principle it is possible to have a mix of quantum and Brownian architectures within a single system. A caveat is that these two architectures may need to operate at different temperatures, and a temperature gradient introduces an additional source of entropy. Thus, the area of the boundary between these subsystems must be at most proportional to the system's bounding area in order that the entropy generated can be effectively countered.

This principle is more general: any non-equilibrium inhomogeneity in the system's structure will result in entropy generation. To quantify this, we assume that such inhomogeneities equilibrate via an uncorrelated diffusive process, and we proceed via a Fokker-Planck approach.

The Fokker-Planck equation (see Section III.2.2 for a more detailed exploration) describes the evolution of a probability distribution in space due to stochastic dynamics. Specifically, it pertains to the influence of a Langevin force with rapidly decaying correlations in time. In this limit, Brownian particles will exhibit a combination of drift and diffusion depending on the properties of the system. The evolution of the probability distribution φ is found [77] to obey[12]

$$\dot{\varphi} = -\nabla \cdot [\mu\varphi - D\nabla\varphi]$$

where μ is a drift vector and D is a diffusion matrix. We also identify the probability current $\mu\varphi - D\nabla\varphi$. In order to establish the rate of entropy generation, we must first determine the steady state distribution. We make the assumption that the steady state probability current is everywhere zero, i.e. there are no persistent current flows or vortices, and therefore find that

$$\frac{\nabla\varphi_0}{\varphi_0} = D^{-1}\mu\,.$$

Now we obtain an expression for the entropy of the system. As there are many particles, we can reinterpret φ as a concentration of particles (as the Fokker-Planck equation does not impose any normalisation), and so each particle will have an associated

[12] In Risken [77], the convention is $\dot{\varphi} = -\partial_i\mu_i\varphi + \partial_i\partial_j D_{ij}\varphi$. We use a different convention here for convenience, wherein $\mu_i' = \mu_i - \partial_j D_{ij}$.

entropy of $1+\varepsilon-\log\lambda$ where $\lambda \propto \varphi$. This yields intensional entropy $\eta = \varphi(1+\varepsilon-\log\lambda)$ and $\dot\eta = \dot\varphi(\varepsilon - \log\lambda)$. At equilibrium, $\dot\eta$ cannot have a leading order linear term in φ, and so $\varepsilon = \log\lambda_0$ which gives

$$\begin{aligned}\eta &= \varphi\Big(1-\log\frac{\varphi}{\varphi_0}\Big)\\ H &= \int \mathrm{d}V\,\varphi\Big(1-\log\frac{\varphi}{\varphi_0}\Big)\\ \dot H &= -\int \mathrm{d}V\,\dot\varphi\log\frac{\varphi}{\varphi_0}\,.\end{aligned}$$

We now substitute the Fokker-Planck equation for $\dot\varphi$, obtaining

$$\begin{aligned}\dot H &= \int \mathrm{d}S\cdot(\mu\varphi - D\nabla\varphi)\log\frac{\varphi}{\varphi_0} - \int \mathrm{d}V\,(\mu\varphi - D\nabla\varphi)\cdot\nabla\log\frac{\varphi}{\varphi_0}\\ &= \int \mathrm{d}V\,(D\nabla\varphi - \mu\varphi)\cdot\nabla\log\frac{\varphi}{\varphi_0}\\ &= \int \mathrm{d}V\,D\varphi\Big(\frac{\nabla\varphi}{\varphi} - D^{-1}\mu\Big)\cdot\nabla\log\frac{\varphi}{\varphi_0}\\ &= \int \mathrm{d}V\,D\varphi\Big(\frac{\nabla\varphi}{\varphi} - \frac{\nabla\varphi_0}{\varphi_0}\Big)\cdot\nabla\log\frac{\varphi}{\varphi_0}\\ &= \int \mathrm{d}V\,\varphi\Big(D\nabla\log\frac{\varphi}{\varphi_0}\Big)\cdot\nabla\log\frac{\varphi}{\varphi_0}\\ \dot H &= \sum_i N_i\Big\langle\,\Big|D_i^{1/2}\nabla\log\frac{\varphi_i}{\varphi_{i,0}}\Big|^2\Big\rangle\,,\end{aligned}$$

where N_i is the number of particles of diffusive species i. In the second line we have assumed that the probability current across the system boundary vanishes and in the last line we have generalised to multiple diffusive species.

The consequence is that a system may only sustain the use of spatial variation in its architecture if at least one of the following three conditions holds for each such species,

1. The region of variation scales with the system's surface area, such as if the gradient is localised to a hemispheric plane,
2. The diffusion rate D is vanishingly small,
3. The average relative strength of the gradient, $\nabla\log(\varphi/\varphi_0)$, has scaling no greater than $\ell^{-1/2} \sim V^{-1/6}$.

These conditions may be violated only to the extent that another condition is exploited to achieve a commensurate reduction in the entropy rate.

Modes of Computation The quantum Zeno architecture is inherently processive, meaning that every transition it makes is useful. This is in contrast to the Brownian

architecture, in which only a vanishingly small fraction of transitions lead to a net advance in computational state. There are a number of consequences to this distinction that make the quantum architecture preferable.

Firstly, a quantum architecture equivalent to a given Brownian architecture will only use $N_{\text{QZE.}} \sim \sqrt{AV}$ active computational elements compared to $N_{\text{Brown.}} \sim V$, which significantly reduces the risk of errors due to internal fluctuation. In addition, the properties of the QZE mean that the fluctuations within the computational subvolume of the quantum architecture can be rectified at all scales in contrast to the classical system. Of course, fluctuations in the remaining bulk of the volume are still problematic, but their rectification is less critical.

Secondly, whilst the two systems would have the same net rate of computation, the computational elements of the quantum system operate at the same maximum speed regardless of scale, and the degree of parallelism can be tuned between maximally parallel ($N \sim \sqrt{AV}$) and fully serial ($N = 1$). This means that the quantum architecture can operate equally well for parallel and serial problems. One caveat is that the maximal attainable parallelism of the quantum system, $\sim \sqrt{AV}$, is lower than for the Brownian, $\sim V$. In practice this is not too problematic as parallel problems can be simulated serially, and because the total computation rate is independent of the degree of parallelism, the quantum system experiences no penalty for this choice. For the Brownian system, however, even a problem maximally exploiting the available parallelism will be limited by the time for each element to perform a single net operation. In addition, whilst the quantum system has fewer computational foci than the Brownian system, the remaining bulk can be used for 'cold' computation such as data storage.

Thirdly, almost all parallel problems will involve inter-element communication and synchronisation. As will be discussed in Chapter III, these processes present significant challenges in the Brownian architecture that in the worst case would throttle the system down to $R \sim A$. This arises because synchronisation processes map to constrictions in phase space, and Brownian diffusion through such a constriction is very slow. As the quantum architecture evolves processively, it is less subject to this effect and thus its computational elements can communicate much more freely.

Fourthly, the quantum architecture permits quantum computation whereas the Brownian architecture is unlikely to be capable of supporting quantum computation. The reason for this is that the average time for each computational transition in the Brownian system is typically large, increasing as the system size does, whereas the decoherence timescale for a quantum state is fixed and typically small. Furthermore, the decoherence timescale is likely significantly smaller than for the QZE architecture as the Brownian system probably operates at a higher temperature. Altogether, these factors make it incredibly unlikely that a quantum state can be reliably maintained through a quantum computation within such an environment.

Despite all these disadvantages, the Brownian architecture is perhaps of more interest as it is ostensibly easier and cheaper to construct with current technology. The most exciting avenue for this may lie in the fields of biological and molecular computing, in which molecules such as DNA are constructed in such a way that their resulting

dynamics encode a computational process [13, 78]. Chemical systems are a natural substrate for Brownian dynamics, and the manipulation of DNA is becoming ever cheaper and more sophisticated. With improvements in synthetic biology, we may even be able to readily prepare self-repairing DNA computers in the near future.

We finish by discussing the timestep for a net computational transition in the Brownian architecture. This timestep is controlled by the biases, b_i. To maximise the system's computation rate, we take $b_i = \beta \propto 1/\sqrt{\ell}$. If, at operating reactant concentrations, the time for any computational step (forwards or backwards) is t_0, then the net timestep is given by

$$\tau = \frac{t_0}{\beta} \propto \ell^{1/2} \sim V^{1/6}$$

which can be seen to get larger as the system gets larger. Fortunately this scaling is sublinear, and so depending on the available power, the achievable bias may be significant. For example, assuming a $500\,\mathrm{W\,m^{-2}}$ radiative capacity and a 1 m radius system consisting of fairly conservative computational particles of size $(10\,\mathrm{nm})^3$ operating at a gross rate of 1 Hz, a bias as high as $\beta \sim 0.4$ is possible.

Another approach to managing modes of computation in a Brownian computer is to institute a hierarchy of bias levels at different concentrations. For example, a viable system may consist of the following tuples $\sim (N, b; R_C)$,

$$\{(V, \ell^{-1/2}; V^{5/6}),\ (V^{14/15}, \ell^{-2/5}; V^{4/5}),\ (V^{5/6}, \ell^{-1/4}; V^{3/4}),\ (V^{2/3}, 1; V^{2/3})\}$$

and computations requiring faster steps may use higher biases at the expense of less computational capacity. Additionally, the highest bias level admits irreversible computation, and so we may enclose our computer with a thin irreversible shell, whilst the internal bulk consists of reversible computations at various bias levels. Going further, we could have a hot inner Brownian core, a cold outer quantum core, and an enveloping irreversible silicon shell. Additionally, a hierarchical Brownian system need not employ spatial variation; by using different species for each bias, the subsystems can be mixed homogeneously.

II.8. Conclusion

We have shown that, subject to reasonable geometric constraints on power delivery and heat dissipation, the rate of computation of a given convex region is subject to a universal bound, scaling as $R_C \lesssim \sqrt{AV} \sim V^{5/6}$, and that this can only be achieved with reversible computation. For irreversible computation, this bound falls to $A \sim V^{2/3}$. The scaling laws persist at all practical scales, only breaking down when the system size approaches the Schwarzschild scale. For typical densities ($\sim 1000\,\mathrm{kg\,m^{-3}}$), this threshold scale is $4 \times 10^{11}\,\mathrm{m} \approx 2.7\,\mathrm{AU}$. Recall that we are considering systems of fixed uniform density; building and maintaining a structure of this density at such sizes may not be

practical without some combination of high tensile-strength materials, orbital motion[13], and propulsion. Beyond this scale, the maximum computational rate is attained by localising mass into as thin a shell as possible, reminiscent of megastructures such as Dyson spheres from the world of science fiction. At very small scales, the rate scales with V as the surface area to volume ratio is negligible. The scaling then falls to $V^{5/6}$, to $V^{1/2}$, and finally to $V^{1/3}$ as the system increases in size. At the Schwarzschild regime and beyond, the computation rate is suppressed by a factor of order unity due to gravitational time dilation.

This analysis has assumed that each computational element acts independently. In our next two chapters we shall investigate the constraints affecting cooperative reversible architectures, in particular the thermodynamic cost of synchronisation processes, such as communication and resource sharing. These costs turn out to be quite significant, even prohibitive.

[13] Orbital motion near the Schwarzschild regime is not sufficient. For such an object, stable circular orbits only exist for $r > \frac{3}{2} r_s$ at which point the orbital velocity approaches the speed of light.

In Chapter II, the limits on the sustained performance of large reversible computers were investigated and found to scale as $\sqrt{AV}$ where A is the convex bounding surface area of the system and V its internal volume, verifying and strengthening a result of Frank [9], compared to A for an irreversible computer. This analysis neglected, however, to consider interactions between subunits of these computers. In particular, a large computer will be composed of many small computational subunits. Each of these subunits will perform independent computation, but for useful programs one often wishes to combine the results of these independent computations. To do so, the subunits must communicate—they must synchronise their individual states. In an irreversible computer with ample free energy density, this is a non-issue. In contrast, when free energy is limiting, the hidden entropic cost of synchronisation is made manifest and must be considered.

To illustrate, consider two reversible Brownian computers which would like to communicate with one another. A reversible Brownian computer is free to wander back and forth along its configuration space, the set of possible computational states it can take, and we can illustrate this by the analogy of two wagons on intersecting train tracks as shown in Figure III.1. In order to synchronise their states, and progress past the intersection, both wagons need to arrive simultaneously. If one wagon arrives before the other, however, then it is almost certain to wander backwards instead of waiting for the other to arrive. If the probability of moving forward is p and backward is q, then the system is said to have 'computational bias' $b = p - q \ll 1$, and the probability that a wagon is at the intersection point is b/p. The rate of synchronisation can then be found approximately by the probability that both are there simultaneously, $\sim b^2 \lll 1$.

Already the rate of independent computation for each subunit is vanishingly small in the limit of large system sizes, and so synchronisation and communication events in Brownian computers are seen to progress even more slowly and appear to 'freeze out' as free energy gets sparser and sparser. In this chapter, we show that there is indeed no way to design a better system for synchronisation of Brownian computers, evaluating the 'time penalty' for a synchronisation event across a broad range of possible designs. We also briefly consider the applicability to non-Brownian reversible computers, conjecturing that all reversible computers whose computational elements evolve independently are subject to the same penalty. We conclude with a discussion of design approaches and mitigations to circumvent the synchronisation time penalty in those reversible computers subject to such a penalty.

III. Performance Trade-offs for Communicating Reversible Computers

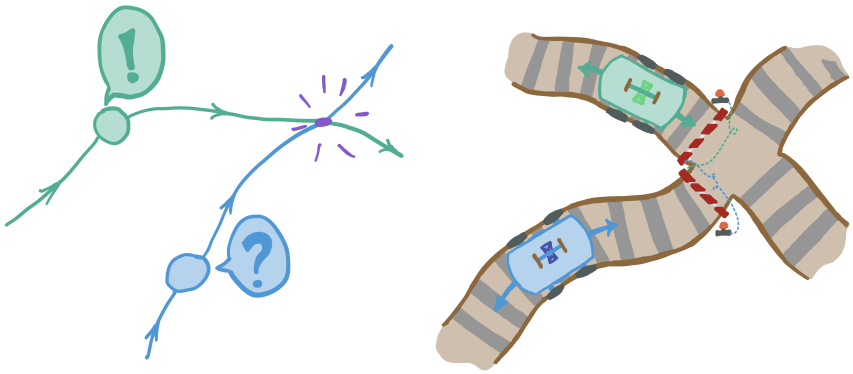


Figure III.1: This chapter concerns synchronisation interactions between reversible Brownian computational particles, which can walk backwards and forwards through phase space. This is illustrated here as two wagons on intersecting train tracks, where there is a barrier at the intersection. Each wagon controls the other's barrier, and so both must be present simultaneously in order to pass through the intersection.

III.1. Introduction

In Chapter II, we investigated how the laws of physics constrain the maximum rate of sustained computation within a given region of space. In fact, this performance measure of operations per unit time is not the only one of interest. As we are considering computers of arbitrary size, interaction between different subvolumes will often be of considerable import. In this chapter and Chapter IV, we evaluate performance within this context. Chapter III concerns the statistical dynamics of communication between two or more distinct computational units, whilst Chapter IV concerns the statistical dynamics of computational units interacting with resources or chemical species, of which there may be a variable number of identical units or particles. Within these analyses we shall work with Brownian computers as our canonical computational system. Nevertheless, we believe our results are more general, and in the conclusion of this chapter we conjecture upon the classes of reversible computational systems to which our results do and do not apply.

To proceed we must define what we mean by a 'Brownian computer'. We already introduced a notion of Brownian computer in Section II.4 in which we used the formalism of Chemical Reaction Networks (CRNs), but the analyses of Chapters III and IV shall require some elaboration. First, it will be helpful to introduce some terminology to distinguish the unique properties of computational particles from those of more traditional chemical species:

Definition III.1 (Mona and Klona)**.** A chemical system is generally comprised of a number of species. Each of these species is present at some count; that is, there is some number of particles of this species in the system, and these are *identical* copies of each other. The study of chemical thermodynamics and statistical mechanics then builds atop this. In contrast, chemical and Brownian computational particles are typically *unique* within the system: each particle represents the state of some computer, and each evolves independently of one another. It may be that occasionally two or more such particles are transiently identical, but this is a coincidental occasion. An alternative instance of effectively unique computational particles arises in fixed lattices of simple elements, such as in *amorphous* computers. Here, the elements at each lattice point may not be globally unique, but as they are uniquely addressable by their lattice location they take on much the same properties. We refer to particles which are generally uniquely distinguishable or addressable *mona* (sg. *monon*), after the Greek for unique, μοναδικός. Similarly, we will call indistinct particles belonging to a species *klona* (sg. *klonon*).

To get a feel for mona, consider the example of a 150 mL solution containing a $1\,\mathrm{mmol\,dm^{-3}}$ concentration of random nucleic acids of length 80 nucleotides. As explained in the introduction, DNA molecules are polymers formed from an arbitrary sequence of four nucleotide monomers. Random nucleic acids are easily obtained from DNA synthesis companies. There are $4^{80} \approx 1.5 \times 10^{48}$ possible distinct 80-mers (nucleic acid of length 80), and the birthday paradox says that we need 1.4×10^{24} of these random molecules before it is likely that there are duplicates present. Our 150 mL, $1\,\mathrm{mmol\,dm^{-3}}$ solution contains just shy of 1×10^{20} such molecules, and it can be calculated that the probability that there are any duplicates in the solution is less than 1 in 350 million. Therefore, it is almost certain that every nucleic acid in said solution is a monon.

We can therefore say that Chapter III studies mona-mona interactions, and Chapter IV mona-klona interactions. We do not cover klona-klona interactions as these are well understood within the domains of chemistry and statistical physics.

Informally, Brownian computers are the sorts of systems embodied by the examples of the DNA-Strand Displacement (DSD) and Tile-Assembly Model (TAM) computer paradigms illustrated in the introduction and shown in Figure I.2. Those DNA computers using DSD as their mechanism have a klonal representation of their computational state, whereas those using the TAM have a monal representation. This is made clearer in Definition III.2 below:

Definition III.2 (Brownian Computers)**.** A Brownian computer is a region of space (typically bounded or otherwise confined) containing particles. The definitions of particles and space are not important, and they may be complex macromolecules or

simple elementary particles or even abstract entities in an abstract topological space. These particles are generally static in isolation, but when two or more particles collide they may alter their states during the interaction, in much the same way as a molecule of ATP and a molecule of H_2O may collide, react, and beget the products ADP, P_i and H^+ (particle number need not be conserved). These interactions must be *reversible*, such that the products may collide and reform the reactants. To enable these interactions, particles must be endowed with velocities, and these velocities are assumed to follow a thermal distribution.

A Brownian computer is characterised by being able to interpret the current state of the particles as a computational state: perhaps some particles are 'computers' in their own right, and thus the evolution of their state over time is isomorphic to some model of computation, or perhaps the joint state of a number of particles must be considered. The former, wherein the computational particles are mona, is more powerful, whereas the latter approach, representing computational state with klona, typically results in the entire reaction volume being dedicated to a single program and thus limits the computational power. Multiple programs can be run if their components are orthogonal in the sense that they don't interact with components of a different program, but designing orthogonal klona is challenging and limited. An alternative mitigation approach is to use multiple bounded sub-volumes containing just the klona for that particular program; these sub-volumes may then be considered as abstract 'particles' or mona, as can the combination of orthogonal sub-klona (forming a virtual sub-volume, if you will). Computational mona can be driven forward in computational phase space by a *computational bias* (Definition III.3, below).

Definition III.3 (Computational Bias)**.** Recall from Chapter II that in a Brownian computer, transitions are mediated by systems of klona representing 'positive' and 'negative' bias. Physically, such klona act as a source of free energy and a real world example is given by ATP and $\mathrm{ADP} + \mathrm{P_i}$ in biochemical systems. In the simplest case, there are $\oplus$ and $\ominus$ klona such that $\forall n.\ |n\rangle + \oplus \rightleftharpoons |n+1\rangle + \ominus$ where $|n\rangle$ is the n^{th} state of some monon, and this is positively biased when the concentration of $\oplus$ exceeds that of $\ominus$. We define the bias in this case to be $b = ([\oplus] - [\ominus])/([\oplus] + [\ominus])$.

To take full advantage of these molecular computers we need to let them communicate and interact. Each interaction event will be between specific mona. If these mona diffuse freely throughout the volume, then communicating with a specific monon will be impracticable except in the case of very small (effective) volumes. On average, the time between collisions of n such mona in a system of volume V will scale as $\sim \mathcal{O}(V^{n-1})$. Moreover, we shall see that a significant problem in reversible communication is that even after mona meet they can go backwards in phase space at which point we must wait for them to collide again; clearly in the freely diffusive case this is untenable. To improve upon this scaling, we are forced to introduce a lattice-like structure into the system. This lattice should be endowed with a coordinate system and the coordinates should be logically chosen so that it is 'easy' to deterministically compute a unique path between any pair of reachable coordinates. Interacting mona should bind to this

lattice, and couple their movement along it to their computational bias (Definition III.3). In so doing, translocation along the lattice becomes just another part of the monon's computational state—the information content of the monon that may be transformed in the course of computation.

In order for two or more mona to communicate, or more generally to synchronise their joint state, we must arrange for their states to coincide at some point. Moreover, no monon will be able to proceed past this synchronisation point without the other(s). In the irreversible limit of computing, this problem is inconsequential as each computational entity can simply wait for the others. In reality, 'waiting' is not a physically reversible process: If the dynamics were reversed it would not be possible for the entity to 'know' how long to wait before going backwards. In an irreversible system, waiting can be simulated by raising the entropy of the environment; for example, the system could continually 'forget' how long it has been since it arrived by resetting some internal state using the Landauer-Szilard principle [8, 7], or the system could transfer its generalised computational 'momentum' to some other computational entity. In the reversible computers we are considering, there is a dearth of free energy[1] as we are distributing the free energy supplied at the computer's surface throughout its entire volume, and so these approaches are not generally possible.

Instead, the simplest approach is to let the mona 'bounce off' the synchronisation point until the other arrives, relying on the (weak) computational bias to drive them towards this. That is, upon reaching the synchronisation point the monon cannot proceed and has a high chance of walking backwards, but the weak bias will keep it within a loose vicinity of the synchronisation point. Some initial estimates (Section III.3.1) will show that, in addition to the time for each monon to reach the synchronisation point independently, there is a 'penalty' term. For computational bias b, each net computational step for lone mona takes time $(b\lambda)^{-1}$ where λ is the rate at which mona collide with bias klona. If each participating monon starts at a phase distance n_i from the synchronisation point, then the expected time for all to have briefly reached the synchronisation point is $\tau_0 = (b\lambda)^{-1} \max_i n_i$. The penalty depends on the joint probability distribution of the mona, and how often they are all found at the synchronisation point. Analysing this in detail turns out to be surprisingly difficult, but we will succeed in showing a range of approximate, empirical and exact results, confirming the initial estimate of a penalty $\Delta\tau \gtrsim (b^d\lambda)^{-1}$ for d mona. As $b \ll 1$ generally, this will be very large compared to the timescales of independent computation by individual mona, and thus potentially rendering reversible communication impracticable.

In this chapter, we shall study this problem in detail for general shapes of synchronisation phase spaces and illustrate with examples of the forms of phase space that would be commonly found in reversibly communicating mona. We divide approaches to synchronisation in two: Section III.3 will concern the 'constrictive' case in which there is a constriction in phase space, as illustrated in Figure III.3. Section III.4 will concern the 'recessive' case in which the phase space is not restricted, and therefore other methods

[1] Free energy is the ability of the system to drive an entropically unfavourable reaction, by increasing the entropy of the environment at least as much as the reaction reduces the local entropy.

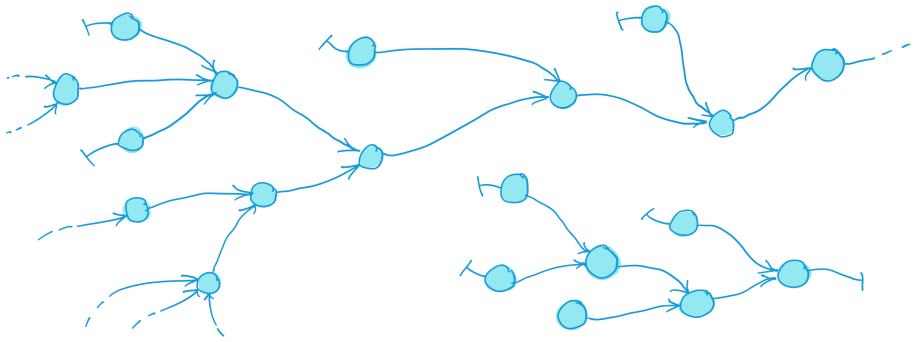
must be used to synchronise the joint state of the mona as illustrated in Figure III.4. In fact, there are two more related cases, dilatory and processive, which are in a sense complementary to the constrictive and recessive cases respectively. In the dilatory case, the phase space widens towards the synchronisation point rather than shrinking, whilst in the processive case the boundary of the post-synchronisation subspace (green quadrant in Figure III.4) is concave rather than convex. In both of these, the penalty will be negligibly small, zero, or even negative as the phase coordinate is pushed entropically towards larger regions of phase space. Nevertheless, they are not desirable because such asymmetry implies an increase in entropy and hence irreversibility. We can therefore make a stronger and more accurate statement: rather than forbidding dilatory cases, we require synchronisation phase spaces to be symmetric in the synchronisation point or subspace.

III.2. Methodology

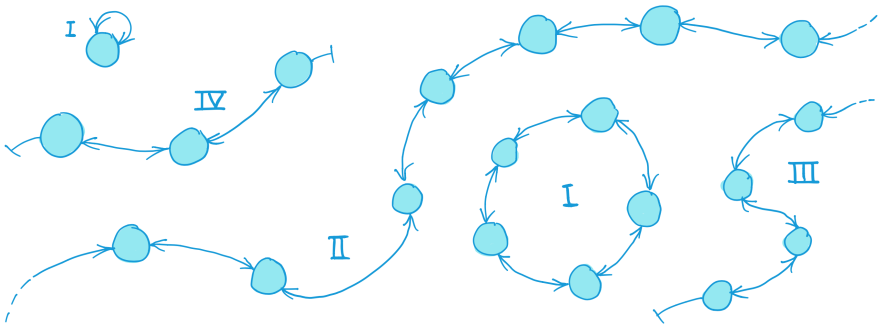
Isolated Mona To formalise the problem statement, we begin by considering computation by isolated mona in more detail. A monon's evolution is best understood by considering the state space of the computation it is simulating as depicted in Figure III.2. Ignoring any possible intermediate states, there will be an isomorphism between the states of the monon and the states of its underlying computation. As was made apparent in the previous chapter, we will want to drive computation forward by coupling the monon's state transitions to some non-equilibrium system of klona, which we term a bias source. Each transition may be coupled to a different bias source, and the strengths of the bias sources may vary over time. The definition of bias sources and their strengths is recalled in Definition III.3. Where possible, we will remain general in our analysis; however, Section II.4 showed that optimal performance is achieved for bias constant in time and uniform in space, and our illustrative examples will reflect this fact.

The dynamics of this system can therefore be seen to be a walk in a 1-dimensional phase space where at any point, there are at most two transitions available: one to the successor state, and one to the predecessor state, should they exist. Depending on whether we use a model with discrete or continuous time[2], these transitions will be labelled with probabilities or rates that depend on the strength of the bias source(s). We note that we expect the strength of the bias sources to be very small, such that the timescale of making one step of net progress is very large compared to that of a single transition. As a result, the monon will experience a time-averaged bias, and any correlations in the system will be expected to have been exponentially suppressed by diffusive processes. Therefore, we can model the monon's dynamics as a Markov process or chain; namely, the dynamics of the system depend only on its current state, rather than its history.

[2]Continuous time is the more apt model, however for generating function approaches discrete time must be employed. Fortunately, when the bias rates are uniform, it is easy to convert between the two by simply dividing the times by the gross bias rate, $\lambda = \lambda_\oplus + \lambda_\ominus$.



(a) Examples of irreversible state spaces



(b) Examples of reversible state spaces

Figure III.2: These figures show some examples of the state spaces of (a) irreversible and (b) reversible computers. The blue nodes represent distinct computational states, and arcs represent computational transitions or operations. Terminal arcs, represented by $\vdash$, indicate that computation either begins or ends here; that is, no predecessor or successor states (respectively) exist. Recall that the dynamics of the system are free to wander forwards and backwards in state space, including away from a terminal state.

The distinction between irreversible and reversible computers is that an irreversible state may (and often *does*) have more than one predecessor state. For example, if two Boolean variables are consumed and replaced with their logical conjunction, $(a, b) \mapsto a \wedge b$, and if this is FALSE, then there are three possible predecessors: (FALSE, FALSE), (FALSE, TRUE), and (TRUE, FALSE). This is loss of information, and it is impossible to determine which path was taken to reach the current state. Where an irreversible computer completely forgets or discards a variable—such as of type `uint32`—it may have a very large number of predecessor states—for `uint32`, $2^{32} \approx 4 \times 10^9$. For abstract models of computation such as a Register Machine (see, e.g., Johnstone [79]), there can even be a countable infinity of predecessors in the case of the 'clear-to-zero' operation.

In contrast, a reversible state may have at most one predecessor and one successor state. We can classify reversible computers into four classes; (I) finite cyclic programs with no termini, (II) bi-infinite programs with no termini, (III) infinite programs with one terminus (comparable to non-halting programs in computability theory), (IV) bi-terminating programs with two termini (i.e. finite linear chains). Bear in mind that the form of this state space is separate from that of the program logic: a reversible program may well contain loops and conditional branching, but for a given program and input or output, evolution is deterministic both forwards and backwards.

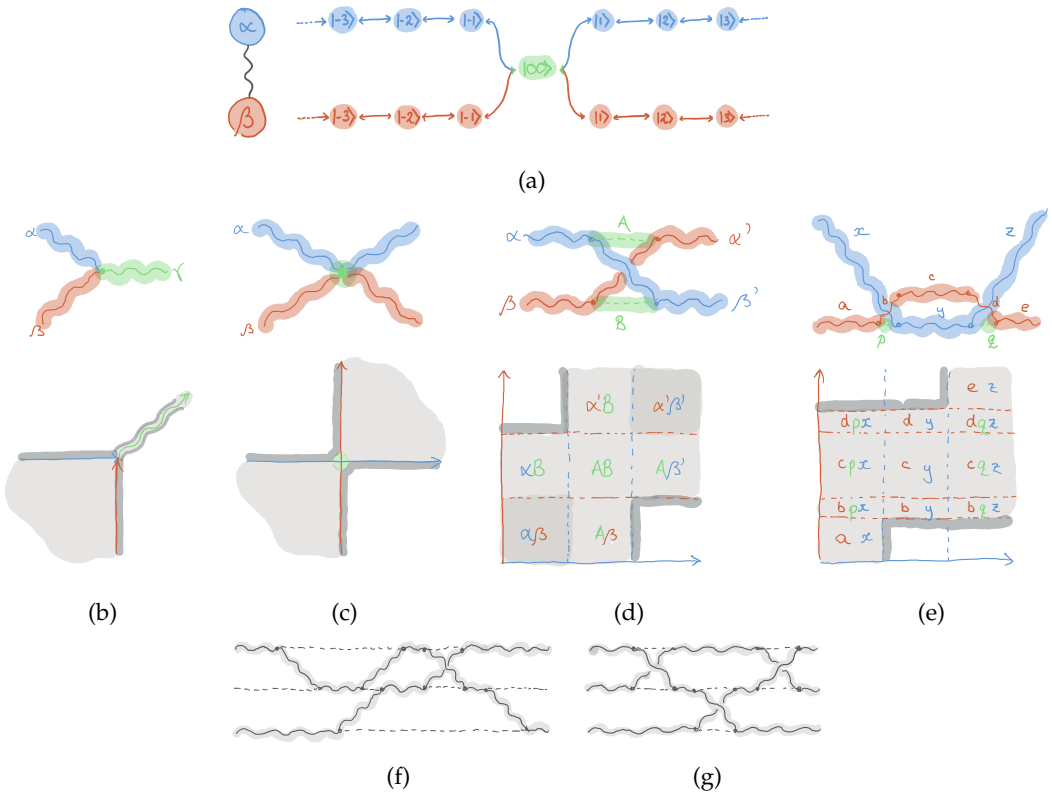


Figure III.3: A range of examples of *constrictive* synchronisation problems, represented in different ways. Figure (a) shows a state diagram for two bound mona α and β; each monon is free to walk back and forth along their state space ($\mathbb{Z}$) independently except for at their *synchronisation point*, $|00\rangle$, through which they must pass together. Figures (b–e) show a range of increasingly complex examples from two perspectives; the top diagrams are abstract state diagrams reminiscent of Feynman diagrams, whilst the bottom diagrams show the joint phase space of the mona. Dashed lines in state space, or 'artefacts', represent static/non-evolving mona; for example, in (d) monon α leaves an artefact A—which may contain some informational payload or data packet—that is then absorbed by monon β, and vice-versa for B. More specifically, an artefact is a packet of data left on the lattice by a computational monon after it departs, ready to be absorbed by another computational monon.

Figure (b) is an asymmetric fusion/fission synchronisation, depending on the direction of time. Figure (c) is symmetric, and equivalent to (a). Figure (d) shows an approach to 'widen the constriction point' of (c) using artefacts, thus making synchronisation easier. Figure (e) shows how two synchronisations close in phase space lead to an apparent 'tunnel'. Finally, figures (f–g) show some more complicated examples suggestive of what a real reversible communicating system might look like.

$$\mathrm{X}_m \rightleftharpoons \mathrm{X}_{m+1} \quad (\textsc{compute}^*) \qquad \mathrm{X}^m_{-0} + \mathrm{Y}^n_{-0} \rightleftharpoons \mathrm{X}^m_{+0} + \mathrm{Y}^n_{+0} \quad (\textsc{sync})$$
$$\mathrm{X}_{-1} \rightleftharpoons \mathrm{X}^0_{-0} \quad (\textsc{sync-begin}) \qquad \mathrm{X}^0_{+0} \rightleftharpoons \mathrm{X}_{+1} \quad (\textsc{sync-end})$$
$$\mathrm{X}^n_{-0} \rightleftharpoons \mathrm{X}^{n+1}_{-0} \quad (\textsc{sync-up}) \qquad \mathrm{X}^{n+1}_{+0} \rightleftharpoons \mathrm{X}^n_{+0} \quad (\textsc{sync-down})$$

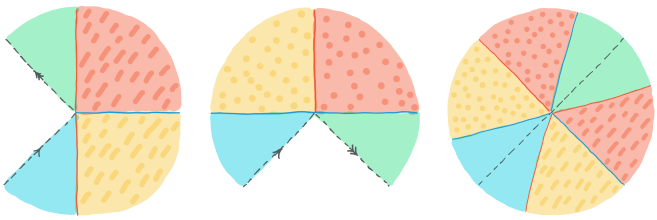
(a) Example recessive 'reaction scheme'. N.B. In the (COMPUTE) rule, $m \leq -2$ or $m \geq 1$.

$$\left.\begin{array}{r} \cdots \mathrm{A}_{-3} \rightleftharpoons \mathrm{A}_{-2} \rightleftharpoons \mathrm{A}_{-1} \rightleftharpoons \mathrm{A}^0_{-0} \rightleftharpoons \mathrm{A}^1_{-0} \rightleftharpoons \mathrm{A}^2_{-0} \rightleftharpoons \mathrm{A}^3_{-0} \\ \cdots \mathrm{B}_{-3} \rightleftharpoons \mathrm{B}_{-2} \rightleftharpoons \mathrm{B}_{-1} \rightleftharpoons \mathrm{B}^0_{-0} \end{array}\right\}$$

$$\mathrm{A}^3_{-0} + \mathrm{B}^0_{-0} \rightleftharpoons \mathrm{A}^3_{+0} + \mathrm{B}^0_{+0}$$

$$\left\{\begin{array}{l} \mathrm{A}^3_{+0} \rightleftharpoons \mathrm{A}^2_{+0} \rightleftharpoons \mathrm{A}^1_{+0} \rightleftharpoons \mathrm{A}^0_{+0} \rightleftharpoons \mathrm{A}_{+1} \rightleftharpoons \mathrm{A}_{+2} \rightleftharpoons \mathrm{A}_{+3} \cdots \\ \mathrm{B}^0_{+0} \rightleftharpoons \mathrm{B}_{+1} \rightleftharpoons \mathrm{B}_{+2} \rightleftharpoons \mathrm{B}_{+3} \cdots \end{array}\right.$$

(b) An example use of the above reaction scheme between two mona, A and B.



(c) (d) (e)

Figure III.4: An example of a recessive synchronisation problem. In a recessive system, there can be no disallowed transitions. This means that if a monon X reaches the synchronisation point prematurely, then it must continue to do 'work' whilst waiting to synchronise. The simplest approach is for X to start counting the net number of steps that it has been waiting to synchronise. When the other monon Y arrives the synchronisation can complete, but the counter must be subsequently consumed. (a) An example reaction scheme implementing recessive synchronisation. For computation prior to synchronisation the states are labelled $\ldots, \mathrm{X}_{-3}, \mathrm{X}_{-2}, \mathrm{X}_{-1}$, and post synchronisation they are labelled $\mathrm{X}_{+1}, \mathrm{X}_{+2}, \mathrm{X}_{+3}, \ldots$. Transitions between these states are implemented by the rule (COMPUTE). A special $\mathrm{X}^n_{\pm 0}$ state is used for the synchronisation-competent state, where $\pm$ represents whether or not synchronisation has yet occurred and n the number of steps counted. The four rules (SYNC–BEGIN,UP,DOWN,END) implement the counter. Finally the rule (SYNC) implements the synchronisation interaction proper. This reaction should be assumed much faster than the counter reactions so that only one monon counts. (b) An example instantiation of the reaction scheme between two mona, A and B. Here, A arrives first and counts for 3 steps. After synchronising, it counts back down. A key feature of this system is that it does not retain a memory of $n = 3$; upon reversal, a higher or lower value of n may be reached or it may even be the case that B counts up instead of A. This effectively sidesteps the issue of 'erasing' the value of n. (c,d) Regions of phase space corresponding to the cases of A (resp. B) reaching the synchronisation point first. The horizontal axis corresponds to the states of A and the vertical to those of B. Therefore, in (c), the yellow quadrant corresponds to A counting up whilst awaiting B's arrival, and the orange quadrant to it counting down. The green quadrant corresponds to both A and B having resumed independent computation. In (d) the yellow quadrant corresponds to B counting up and the orange to it counting down. Note that the bias switches direction; in (c), the bias points NE in the blue and yellow quadrants, and NW in the orange and green. In (d) the bias points NE in the blue and yellow quadrants, and SE in the orange and green. (e) A representation of the entire phase space, consisting of the six 'quadrants' stitched together.

As mentioned in the introduction, we shall incorporate spatial position as just another part of our computational state, such that translocation along a lattice (or indeed any other environmental interaction) can be treated as just another computational transition. This abstraction allows us to greatly simplify our analysis, as we need only track position in phase space; the exact meaning of different phase coordinates is irrelevant, and so our results will remain general in a large class of synchronisation processes.

Interacting Mona For a system of multiple particles, the state of the system is given by a coordinate in their joint phase space. In a classical physical system of N particles, this manifests as a $6N$-dimensional phase space as each particle has six degrees of freedom: three for the particle's position, and three for its momentum vector. Similarly, our system of N mona has as states coordinates in an N-dimensional phase space. When these mona do not interact, the phase space naturally decomposes into a tensor product of each monon's individual phase space. If, however, n of these mona happen to interact with one another, then the subspace formed from their degrees of freedom is irreducible: it cannot be further decomposed into a tensor product. The reason is that a synchronisation manifests as a constraint on the joint state.

To make this more concrete, suppose three mona are to share some information. Let their individual states be indexed by $\mathbb{Z}$, such that their joint phase space is given by a subset of $\mathbb{Z}^3$. Suppose the interaction occurs when each monon is in state 0, i.e. the coordinate at which they interact is $(0,0,0)$. As the future state of each monon depends on this interaction/communication/synchronisation event, we observe that some coordinates are causally impossible. For example, $(3,-4,5)$ would imply that the first and third mona have received information from the second, despite the second having never given them this information. In fact, we find that the only valid states are given by $\{(-a,-b,-c) : a,b,c \in \mathbb{N}\} \cup \{(a,b,c) : a,b,c \in \mathbb{N}\} \equiv (-\mathbb{N})^3 \cup \mathbb{N}^3$. This corresponds to the negative and positive octants of $\mathbb{Z}^3$, and the system must pass through the origin. The equivalent case of two mona is depicted in Figures III.3a and III.3c.

This is not the only way for mona to interact as illustrated by other examples in Figure III.3. The common factor is that a synchronisation event between mona with individual phase spaces $\mathcal{R}_i$ corresponds to their joint state $\mathcal{R}$ being a strict subset of $\mathcal{R}' = \bigotimes_i \mathcal{R}_i$. Our aim is to analyse the mean time to get from some region of phase space to another, and for this we shall need to specify some regions: $\mathcal{I}$ will be the initial region of phase space (or more generally, a distribution over phase space), $\mathcal{P}$ will be the region 'before' synchronisation, $\mathcal{P}'$ the region 'after', and $\mathcal{S}$ the interfacial region corresponding to the process of synchronisation. These phase regions must satisfy the relations $\mathcal{I} \subset \mathcal{P}$, $\mathcal{P} \cap \mathcal{P}' = \mathcal{S}$, and $\mathcal{P} \cup \mathcal{P}' = \mathcal{R}$. It should be noted that the synchronisation region $\mathcal{S}$ is arbitrary, although in practice there is usually a natural choice.

The question naturally arises of what happens at the boundaries of this phase space. Quite simply, transitions which would take the mona outside of the valid phase region are blocked; that is, there is no reaction with reactants $|0\rangle$ and $\oplus$ if the other mona are not all receptive to synchronisation. A more abstract but more general answer is that the boundaries of $\mathcal{R}$ are *reflective*: the probability flux/current normal to them vanishes.

We now introduce a canonical slicing of $\mathcal{P}$ into hypersurfaces $\{\mathcal{P}_s : s \in \mathbb{N}\}$. The hypersurface $\mathcal{P}_s$ is given by the locus of nodes whose shortest path (i.e. the minimum number of transitions) to $\mathcal{S}$ is s. By definition, $\mathcal{P}_0 \equiv \mathcal{S}$, and it can be shown that to get from a node in $\mathcal{P}_{s+k}$ to $\mathcal{P}_s$, one must pass through the hypersurfaces $\{\mathcal{P}_{s+k-1}, \mathcal{P}_{s+k-2}, \ldots, \mathcal{P}_{s+1}\}$ in turn. We will usually take $\mathcal{I} \subseteq \mathcal{P}_s$ for some 'distance' s.

These definitions allow us to categorise the forms of synchronisation events. If $|\mathcal{P}_{s+1}| \geq |\mathcal{P}_s|$ for all s within an appropriate vicinity of $\mathcal{S}$ where $|\mathcal{P}_s|$ is the number of nodes in said hypersurface, then we call this a constrictive synchronisation. If instead $|\mathcal{P}_{s+1}| \leq |\mathcal{P}_s|$ we call it dilatory. These categories intersect in the special case of $|\mathcal{P}_s|$ constant; this can be likened to a tube, and has zero penalty. For uniformly positive bias, a dilatory synchronisation problem does not present an impediment to synchronisation and in fact may even aid the process, as the reflective boundary serves to 'push' probability density towards $\mathcal{S}$. In practice, dilatory synchronisations only occur in contrived scenarios, if at all, and synchronisations can be assumed constrictive. If a synchronisation geometry does not fit into either of these categories, then it may need to be partitioned into constrictive and dilatory regions, each of which can be analysed separately. Alternatively, if the geometry is 'on average' constrictive then one should be able to cautiously apply the relevant results.

Locally Unconstrained Synchronisation As alluded to in the introduction and Figure III.4, a synchronisation problem can be modified to 'fill in' the missing regions of phase space. This is achieved by introducing a 'dummy' counter for each monon. Whenever a monon reaches the synchronisation subspace (i.e. it is receptive to synchronisation) it begins counting the time it has waited. This is not quite accurate, as the counter is just another part of the monon's computational state and so it can go down as well as up; nevertheless, on average it counts up. The consequence is that every monon can continue to evolve unimpeded. Once the last monon becomes receptive to synchronisation, the mona finally synchronise. At this point, the latent mona begin to decrement their counters. When a monon reaches zero, it may resume its original computation. Though a subtle point, this is indeed a reversible computation despite appearing to erase information, and is made clearer in Figure III.4.

In return for the elimination of phase space boundaries, the phase space becomes effectively larger, and control over the state of the mona is more limited. In particular, with a weak bias the mona are free to wander into these dummy regions. We will analyse this case in Section III.4 in order to compare its performance to the constrictive case. As with the constrictive/dilatory cases, we can classify these problems into recessive and processive cases. Recessive cases correspond to $\mathcal{P}'$ being convex, and processive to it being concave. Again, it is not obvious how a processive case could be constructed, if indeed it can be. Moreover, it is unclear how to analyse a case that does not fall into these categories.

Problem Statement With the synchronisation phase space well defined, we are now in a position to formulate a question. Our goal is to determine the mean time to get from

$\mathcal{I}$ to $\mathcal{S}$, and to compare this to the unsynchronised system $\mathcal{R}'$. For homogeneous constant bias b, it is straightforward to show that unconstrained evolution from $\mathcal{P}_s$ to $\mathcal{P}_0 = \mathcal{S}$ takes mean time $\tau = s/b\lambda$. That is, the computational 'speed' is $1/b\lambda$. More generally, we can define this quantity by $\tau = \inf\{t : \forall u > 0.\langle x(t+u)\rangle \in \mathcal{P}'\}$. That is, it is the least time for which the mean position in phase space never leaves the post-synchronisation domain. For many systems, this can be identified with the first time at which the mean position in phase space enters the post-synchronisation domain, $\tau = \inf\{t : \langle x(t)\rangle \in \mathcal{P}'\}$.

To compute this latter quantity τ in general cases, we consider the evolution of the phase space probability distribution, $W(t) = W(\vec{x}; t)$. This quantity τ is well known in the literature, and is termed the Mean First-Passage Time, or MFPT. We will generally follow the approach of Risken [77], in particular Chapter 8. The first observation we can make is that we need not track particles that cross the boundary between the two domains; that is, we can place an absorbing boundary at $\mathcal{S}$. The total extant probability, $G(t) = \int_{\mathcal{P}\setminus\mathcal{S}} \mathrm{d}\vec{x}\, W(t)$, can then be interpreted as a survival probability, $\mathbb{P}(\tau > t)$. Introducing a probability density $\varrho(t)$ for the MFPT, such that $\tau = \int_0^\infty \mathrm{d}t\, t\varrho(t)$, we can see that $G(t) = \mathbb{P}(\tau > t) = \int_t^\infty \mathrm{d}t'\, \varrho(t')$ and hence $\tau = -\int_0^\infty \mathrm{d}t\, t\frac{\mathrm{d}G}{\mathrm{d}t} = \int_0^\infty \mathrm{d}t\, G(t)$.

We proceed by reducing the dimension of the problem. Surprisingly, it transpires that we can in fact eliminate the time variable. We recall the standard approach from Risken [77]. The n^{th} moment of τ is found to be

$$\langle \tau^n \rangle = -\int_0^\infty \mathrm{d}t\, t^n \frac{\partial}{\partial t} \int_{\mathcal{P}} \mathrm{d}\vec{x}\, W(\vec{x}; t) = \int_{\mathcal{P}} \mathrm{d}\vec{x} \underbrace{\left(-\int_0^\infty \mathrm{d}t\, t^n \frac{\partial}{\partial t} W(\vec{x}; t) \right)}_{w_n(\vec{x})}.$$

Let $\mathcal{L} = \mathcal{L}(\vec{x})$ be an operator describing the time evolution of the phase distribution, i.e. $\dot{W} = \mathcal{L}W$. Here we require that $\mathcal{L}$ not depend on time, and so we will not consider time-variable biases from here on. Integrating by parts and then applying $\mathcal{L}$, we can relate the w_n to each other:

$$\begin{aligned} w_{n+1}(\vec{x}) &= n \int_0^\infty \mathrm{d}t\, t^n W(\vec{x}; t), \\ \mathcal{L}w_{n+1}(\vec{x}) &= n \int_0^\infty \mathrm{d}t\, t^n \mathcal{L}W(\vec{x}; t) \\ &= n \int_0^\infty \mathrm{d}t\, t^n \frac{\partial}{\partial t} W(\vec{x}; t) \\ &= -n w_n(\vec{x}). \end{aligned}$$

Evaluating the base case, $w_0(\vec{x}) = W(\vec{x}; 0)$, we thus have an inductive definition of the w_n, from which we can obtain each moment of τ. As these turn out to be non-trivial to evaluate, we will focus on the first moment—the mean—in this chapter, i.e. $\tau = \int_{\mathcal{P}} \mathrm{d}\vec{x}\, w_1$ with $\mathcal{L}w_1 = -W(0)$ where $W(0)$ is the initial distribution on $\mathcal{I}$. That is, we merely need to compute the steady state distribution of the system $\mathcal{P}$ with absorbing boundary at $\mathcal{S}$, subject to forcing $W(0)$. Note that, whilst $W(0)$ is a normalised probability distribution with total density 1, w is not. In fact, its total density is precisely the MFPT.

A useful observation is that phase density is conserved by the operator $\mathcal{L}$. This means we can write $\mathcal{L} = -\vec{\Delta} \cdot \vec{S}$ for discrete phase space, where $\vec{\Delta}$ is the discrete vector derivative (or finite difference) and $\vec{S}$ is the phase current. For continuous phase space, we have instead $\mathcal{L} = -\vec{\nabla} \cdot \vec{S}$. Current will be very important to our analysis, and is informed by the sources and sinks due to the $W(0)$ forcing and absorbing boundary respectively.

III.2.1. Techniques for Discrete Phase Space

Recurrence Relations There are a number of ways to represent the operator $\mathcal{L}$ for a discrete system. The canonical approach to describe a Markov chain is by introducing a distribution vector with indices corresponding to loci in phase space. For example, the phase space corresponding to Figure III.3a would be represented by a vector in the infinite dimensional vector space indexed by the set $\{(-i,-j) : i,j \in \mathbb{N}\} \cup \{(i,j) : i,j \in \mathbb{N}\}$. The operator is then an infinite matrix for this space. Whilst a convenient approach due to the wealth of techniques built around this representation, in addition to the ability to solve for the steady state as an eigenvector problem, it is too unwieldy to make much progress here.

A better representation is given by considering the local structure of $\mathcal{L}$, as alluded to earlier when we introduced current. For the one-dimensional phase space of a bi-infinite monon with transition rates $\lambda(i \mapsto i+1)$ and $\lambda(i+1 \mapsto i)$, we can write the system in the following two convenient ways:

$$\begin{aligned} \dot{x}_i &= -[\lambda(i \mapsto i+1) + \lambda(i \mapsto i-1)]x_i \\ &\quad + \lambda(i-1 \mapsto i)\, x_{i-1} + \lambda(i+1 \mapsto i)\, x_{i+1}, \\ S(i \mapsto i+1) &= \lambda(i \mapsto i+1)\, x_i - \lambda(i+1 \mapsto i)\, x_{i+1}. \end{aligned}$$

Note how $\dot{x}_i \equiv -S(i \mapsto i+1) + S(i-1 \mapsto i)$, as expected. Also keep in mind that the expression for $\dot{x}_i$ needs to be modified to incorporate sources and sinks. These representations are just recurrence relations, and methods to solve these include elimination, induction, and generating functions. To demonstrate, we solve for the MFPT from $|-s\rangle$ to $|0\rangle$.

To set up the problem, we place an absorbing boundary at $|0\rangle$ such that $x_0 = 0$ and we force the system by placing a negative unit-strength source at $|-s\rangle$ such that $S(-s \mapsto -s+1) - S(-s-1 \mapsto -s) = 1$. The boundary condition at $-\infty$ is $x_{-\infty} = 0$, as is the current. Therefore, we find that $S(-s-k-1 \mapsto -s-k) = 0$ for $k > 0$ and $S(-s+k \mapsto -s+k+1) = 1$ for $s > k \geq 0$. That is, the current transports density from

the source to the sink. Writing out the current equations,

$$\begin{aligned}
1 = S(-1 \mapsto 0) &= \lambda(-1 \mapsto 0)\, x_{-1}, \\
1 = S(-2 \mapsto -1) &= \lambda(-2 \mapsto -1)\, x_{-2} - \lambda(-1 \mapsto -2)\, x_{-1}, \\
1 = S(-3 \mapsto -2) &= \lambda(-3 \mapsto -2)\, x_{-3} - \lambda(-2 \mapsto -3)\, x_{-2}, \\
&\vdots \\
0 = S(-s-k-1 \mapsto -s-k) &= \lambda(-s-k-1 \mapsto -s-k)\, x_{-s-k-1} \\
&\quad - \lambda(-s-k \mapsto -s-k-1)\, x_{-s-k}, \\
&\vdots,
\end{aligned}$$

we can find the densities in the vicinity of the boundary;

$$\begin{aligned}
x_0 &= 0, \\
x_{-1} &= \frac{1}{\lambda(-1 \mapsto 0)}, \\
x_{-2} &= \frac{1}{\lambda(-2 \mapsto -1)}\Big[1 + \frac{\lambda(-1 \mapsto -2)}{\lambda(-1 \mapsto 0)}\Big], \\
x_{-3} &= \frac{1}{\lambda(-3 \mapsto -2)}\Big[1 + \frac{\lambda(-2 \mapsto -3)}{\lambda(-2 \mapsto -1)}\Big[1 + \frac{\lambda(-1 \mapsto -2)}{\lambda(-1 \mapsto 0)}\Big]\Big], \\
&\vdots
\end{aligned}$$

More generally, we can write

$$x_{-n} = \begin{cases} \frac{1}{\lambda(-1 \mapsto 0)}\big(\sum_{k=1}^{n} \prod_{r=2}^{k} \frac{\lambda(-r+1 \mapsto -r)}{\lambda(-r \mapsto -r+1)}\big), & n \le s, \\ \frac{1}{\lambda(-1 \mapsto 0)}\big(\sum_{k=1}^{s} \prod_{r=2}^{k} \frac{\lambda(-r+1 \mapsto -r)}{\lambda(-r \mapsto -r+1)}\big)\big(\prod_{r=s+1}^{n} \frac{\lambda(-r+1 \mapsto -r)}{\lambda(-r \mapsto -r+1)}\big), & n > s. \end{cases}$$

That is, we have exact expressions for the entire steady state distribution. Adding these up will yield the MFPT, $\tau = \sum_{i=0}^{\infty} x_i$. This expression is not particularly convenient; we can instead rewrite the recurrence relation as

$$x_{-n} = \frac{S(-n \mapsto -n+1)}{\lambda(-n \mapsto -n+1)} + \frac{\lambda(-n+1 \mapsto -n)}{\lambda(-n \mapsto -n+1)} x_{-n+1},$$

from which we find

$$\begin{aligned}
\tau &= \sum_{n=1}^{s} \frac{1}{\lambda(-n \mapsto -n+1)} + \tau \sum_{n=1}^{\infty} \frac{\lambda(-n+1 \mapsto -n)}{\lambda(-n \mapsto -n+1)} \frac{x_{-n+1}}{\tau} \\
&= \Big(1 - \Big\langle \frac{\lambda(-n+1 \mapsto -n)}{\lambda(-n \mapsto -n+1)} \Big\rangle\Big)^{-1} \sum_{n=1}^{s} \frac{1}{\lambda(-n \mapsto -n+1)},
\end{aligned}$$

where the expectation value is taken over the steady state probability distribution. Even if the steady state distribution is unknown, we can nevertheless use this to obtain bounds

on τ. In the specific case of homogeneous bias, $\lambda(n \mapsto n+1) = p\lambda$ and $\lambda(n+1 \mapsto n) = q\lambda$, the bounds coincide and thus we can obtain an exact expression without knowledge of the exact distribution (though the distribution also reduces to a reasonably simple form), finding

$$\tau = \left(1 - \frac{q}{p}\right)^{-1} \frac{s}{p\lambda} = \frac{s}{b\lambda},$$

as expected.

Generating Functions When the values of λ take on a particularly nice form, such as constant uniform, generating functions provide a tempting approach. A generating function is used to represent a series as coefficients of a power series. For example, the series $\{0, 1, 2, 3, \ldots\}$ has generating function $t(1-t)^{-2}$ because its series expansion is $t + 2t^2 + 3t^3 + 4t^4 + \cdots$. Using multiple variables, complicated structures such as walks can be encoded in a generating function[3]. We will seek a generating function, W, whose terms correspond to walks starting at each position $|s\rangle$, and end when they reach $|0\rangle$ for the first time. For convenience, we are walking down towards 0 from positive states, rather than up from negative states. It turns out that the easiest way to construct these walks is in reverse: we start from $|0\rangle$ and add either an 'up' step or a 'down' step, providing we never go down to $|-1\rangle$. Stated another way, a walk is either the empty walk (with generating function 1), another walk w followed by an 'up' step (wu), or another (positive) walk w_+ followed by a 'down' step (w_+d). This way, the walks described are only those right up to the point they cross the origin.

To get meaningful statistical information, we label each step with its probability. In the reversed walk, an 'up' step has probability p and a 'down' step q. This leads to the implicit functional equation

$$W = 1 + pxtW + q\bar{x}t(W - W|_{x=0})$$

where $\bar{x} \equiv 1/x$, x is a variable that records the final position of the walk, and t is a variable that records the number of steps in the walk. To prevent a downward step at $|0\rangle$, we ignore these walks when constructing the next step. Expanding W, we can see that it has the terms we expect:

$$W = t^0(x^0) + t^1(px^1) + t^2(ppx^2 + pqx^0) + t^3(pppx^3 + ppqx^1 + pqpx^1) + \cdots$$

To proceed, we seek a closed form expression. Rewriting, we have $(x - t(px^2 + q))W = x - qtW|_{x=0}$. It would appear that we are now stuck, as there is an unknown $W|_{x=0}$ which depends on the complete generating function W. Fortunately there is a powerful technique that can be used here, the Kernel method [83]. We require that W have no negative powers of x, and therefore the *kernel*, $x - t(px^2 + q)$, must be a factor of the

[3] For introductions to generating functions and enumerative combinatorics, see Wilf [80] and Stanley [81]. For a review of techniques used to analyse walks in two or more dimensions with generating functions, see Bousquet-Mélou and Mishna [82].

right hand side of the equation. Among other things, this means that when the kernel vanishes, so must the right hand side. The kernel has two roots, $X_\pm$, although only one of these turns out to be appropriate (in the sense of not containing negative powers of t):

$$X_\pm = \frac{1 \pm \sqrt{1-4pqt^2}}{2pt}, \qquad 0 = X_- - qtW|_{x=0}.$$

Substituting into our functional equation, we obtain

$$W = \frac{1-\bar{x}X_-}{1-(px+q\bar{x})t}.$$

In fact, this is not quite the generating function we want. We want the final step to be into an absorbing boundary, so we define the 'true' generating function to be $V = pxtW$. A little thought shows that the coefficient of x^s in V is the probability generating function of the first passage time; that is, the coefficient of $t^n x^s$ is the probability that a walk starting at $|s\rangle$ takes n steps to reach $|0\rangle$. In order to find the MFPT, we want to calculate $\sum_{n=0}^{\infty} n[t^n x^s]V$ where $[t^n x^s]V$ is the coefficient of $t^n x^s$ in V. In fact, we can obtain a generating function encoding this information by taking the t-derivative of V, exploiting the fact that $\partial_t t^n = nt^{n-1}$. If we then set $t = 1$, we sum up all the t coefficients and therefore obtain the generating function for the MFPTs, parametrised by s. That is, $\dot{V}|_{t=1}$. Evaluating this gives

$$\dot{V}|_{t=1} = \frac{1}{b}\frac{x}{(1-x)^2} = \sum_{s=0}^{\infty}\frac{s}{b}x^s,$$

i.e. the MFPT for $|s\rangle$ is s/b as expected for discrete time. Converting to continuous time recovers $s/b\lambda$.

Unfortunately, these techniques do not extend easily to multiple dimensions. Nevertheless, there is a growing selection of techniques applicable to two-dimensional quadrant walks. See Bousquet-Mélou and Mishna [82] for a review. We had partial success in applying these techniques, and some of our results are summarised in Section III.5, but ultimately we were unable to use these to obtain expressions for the MFPT. Instead, we focussed on obtaining lower and upper bounds by making use of the recurrence relations and properties such as detailed balance; these results are presented in Section III.3.2.

III.2.2. Techniques for Continuous Phase Space

Fokker-Planck Equation The phase spaces we are interested in are discrete, but for completeness we shall also consider continuous phase spaces. Many of the definitions for discrete phase space generalise naturally such as the phase regions $\mathcal{R}'$, $\mathcal{R}$, $\mathcal{P}$, $\mathcal{P}'$, $\mathcal{I}$, and $\mathcal{S}$. The distribution W is to be interpreted as a density, but otherwise W, G, w_n, ϱ, etc. generalise as expected. The boundaries of $\mathcal{R}$ are enforced by a no-flux boundary condition. The hypersurface slicing is less obvious, and will require us to first define the dynamics of the system.

Being stochastic, the dynamics are conventionally described by a Langevin stochastic differential equation. If $\vec{\xi}(t)$ is the phase coordinate, then we can write $\frac{\mathrm{d}}{\mathrm{d}t}\vec{\xi}(t) = \vec{\eta}(\vec{\xi};t)$ where $\vec{\eta}(\vec{\xi};t)$ is the noise term, a family of vectors of random variables indexed by the time coordinate. It is standard to use the decomposition $\vec{\eta} = \vec{h} + \vec{\theta}$ where $\vec{h} = \langle\vec{\eta}\rangle$ is a deterministic 'forcing' term, and $\vec{\theta} = \vec{\eta} - \langle\vec{\eta}\rangle$ is a noise term with zero mean. Moreover, we expect the noise term to be uncorrelated in time, i.e. our dynamics should have the Markov property. Following the approach of Risken [77] in Chapter 3.4, we find we can write $\vec{\theta}(\vec{\xi};t) = \mathbf{g}(\vec{\xi};t)\vec{\Gamma}(t)$ where $\vec{\Gamma}$ is a vector of independent unit-variance zero-mean Gaussian variables, and $\mathbf{g}$ is a noise strength matrix. Using the Kramers-Moyal expansion, we can derive an equation for the evolution of the probability density W,

$$\dot{W} = -\vec{\nabla}\cdot\underbrace{(\vec{\mu} - \vec{\nabla}\cdot\mathbf{D})W}_{\vec{S}},$$

where $\vec{\mu}$ is the drift coefficient and $\mathbf{D}$ the diffusion matrix, both deriving from $\vec{h}$ and $\mathbf{g}$. This equation is known as the *Fokker-Planck* equation (FPE).

We can now define hypersurface slicing for continuous phase space, remaining robust against coordinate transforms. Geodesic paths can be found by integrating the drift coefficient, $\dot{\vec{x}} = \vec{\mu}(\vec{x})$, from some initial coordinate $\vec{x}_0$. A hypersurface $\mathcal{P}_s$ is defined by the locus of points which take the same time t to reach $\mathcal{S}$. The label s is arbitrary as long as it increases monotonically with t; typically s will be chosen to correspond to geodesic path lengths in the 'physically relevant' coordinate system, e.g. $s = b\lambda t$ in the case of uniform constant bias.

To relate the drift coefficient and diffusion matrix to the computational bias, we need to compare the statistical properties of the two systems. For constant bias, this will be straightforward and will result in a constant drift coefficient and diffusion matrix. Where the bias varies, more care is required to ensure the desired properties are replicated in the continuous case.

We match the statistical properties for a single degree of freedom. In the discrete case, the relevant distribution is given by the difference between two Poisson distributions. One Poisson distribution represents the number of $\oplus$ tokens received, and the other the number of $\ominus$ tokens. This is known as the *Skellam* distribution, and our parameters are $p\lambda t$ and $q\lambda t$, yielding mean $b\lambda t$ and variance λt. In the continuous case, the FPE is given by $\dot{W} = -\mu W' + DW''$ and its exact solution can be obtained by a routine application of

Fourier transforms:

$$\begin{aligned}
\dot{\widetilde{W}} &= -ik\mu\widetilde{W} - k^2 D\widetilde{W} \\
\partial_t \log \widetilde{W} &= -(k^2 D + ik\mu) \\
W &= \frac{1}{\sqrt{2\pi}} W_0 * \frac{1}{\sqrt{2\pi}} \int_{\mathbb{R}} \mathrm{d}k \exp\Big(ikx - (k^2 D + ik\mu)t\Big) \\
&= \frac{1}{\sqrt{2\pi}} W_0 * \frac{1}{\sqrt{2\pi}} \int_{\mathbb{R}} \mathrm{d}k \exp\Big(-Dt\Big(k + \frac{i(x-\mu t)}{2Dt}\Big)^2 - \frac{(x-\mu t)^2}{4Dt}\Big) \\
&= W_0 * \frac{1}{\sqrt{4\pi Dt}} \exp\Big(-\frac{(x-\mu t)^2}{4Dt}\Big).
\end{aligned}$$

That is, the initial distribution W_0 is convolved with a Gaussian of mean μt and variance $2Dt$. Comparing with the Skellam distribution, we identify $\mu = b\lambda$ and $D = \frac{1}{2}\lambda$.

III.3. Constrictive Case

Using these techniques, we will be able to obtain a variety of exact, approximate, and numeric results for the constrictive case for general geometries. In practice, we are more interested in truncated simplicial geometries (Definition III.4) as their construction is more 'natural'. The construction is natural because it comprises d mona which evolve independently at all times except for the moment of synchronisation.

Definition III.4 (Truncated Simplex)**.** We refer to our primary geometry of interest as a 'truncated simplex'. A simplex is a generalisation of the concept of a triangle to arbitrary dimensions. That is, in dimension one it is a line segment, and in dimension three it is a tetrahedron. The pre-synchronisation subspace $\mathcal{P}$ of Figure III.3c can be considered to be a right simplex in two dimensions, i.e. a right triangle, of infinite extent. It is defined by the set $\{(x, y) : x \leq 0 \wedge y \leq 0\}$ (restricted to $\mathbb{Z}^2$ for the discrete case or $\mathbb{R}^2$ for the continuous). The pre-synchronisation subspace $\mathcal{P}$ of Figure III.3d, meanwhile, is a *truncated* simplex in two dimensions. Its set definition is $\{(x, y) : x \leq 0 \wedge y \leq 0 \wedge |x+y| \geq w\}$ where w is the side-length of the constriction surface $\mathcal{S}$. In d dimensions, the set is given by

$$\left\{\vec{x} : \bigwedge_{i=1}^{d} x_i \leq 0 \wedge \left|\sum_{i=1}^{d} x_i\right| \geq w\right\}.$$

It will be useful to determine the sizes of hypersurfaces $|\mathcal{P}_n|$ for truncated simplices. These hypersurfaces are in fact regular $d-1$ simplices, which in $\mathbb{R}^d$ have hypervolume

$$\frac{\sqrt{d}}{(d-1)!}\left(\frac{n+w}{\sqrt{2}}\right)^{d-1}.$$

In $\mathbb{Z}^d$ we can use generating functions. The size of $|\mathcal{P}_n|$ is equivalent to the number of ways $n + w - 1$ can be made from the sum of d natural numbers. This is conveniently

given by the coefficient of x^{n+w-1} in $(1-x)^{-d}$,

$$\binom{d+n+w-2}{n+w-1}.$$

III.3.1. Initial Estimates

Information Erasure Model We shall begin first with some quick initial estimates in order to better understand the qualitative behaviour of our results. Consider the lateral evolution of some initial distribution $\mathcal{I}$; whilst this initial distribution can in principle be chosen arbitrarily, it will tend to diffuse to fill the entire hypersurface. Moreover, the timescale of this lateral diffusion will be much shorter than that of the approach of the mean position towards the interfacial region $\mathcal{S}$ in the case of vanishing bias. To clarify, the bias between two adjacent nodes x and y is given by

$$b(x \mapsto y) = \frac{\lambda(x \mapsto y) - \lambda(y \mapsto x)}{\lambda(x \mapsto y) + \lambda(y \mapsto x)}.$$

Therefore, without significant loss of generality, we assume a substantially delocalised initial distribution. This distribution need not be uniform, but it will have an entropy that scales roughly proportional to $\log|\mathcal{P}_s|$, where $|\mathcal{P}_s|$ is the size of the initial hypersurface. In order for an interaction to occur, the distribution must pass through the interfacial hypersurface $\mathcal{S} \equiv \mathcal{P}_0$ whereupon it will have a distributional entropy proportional to $\log|\mathcal{P}_0|$. As the geometry is constrictive, $|\mathcal{P}_0| \leq |\mathcal{P}_s|$ and therefore this process requires an erasure of information. For constant bias b, information can be erased (see Chapter II) at a maximum rate $2b^2\lambda$ where λ is the gross transition rate. We can therefore bound the synchronisation time from below by

$$\frac{1}{2db^2\lambda}\log\frac{|\mathcal{P}_s|}{|\mathcal{P}_0|}$$

which, for a truncated simplex, is

$$\frac{1}{2db^2\lambda}\sum_{k=0}^{d-2}\log\Big(1+\frac{s}{w+k}\Big) = \underbrace{\frac{1}{4b^2\lambda}\log\Big(1+\frac{s}{w}\Big)}_{d=2}.$$

Note the factor d in the denominator which arises because each monon is able to independently erase information at the given rate.

This neglects the time for the individual mona to reach the synchronisation surface, which is $s/b\lambda$. For $s \lesssim 1/b$, the erasure time will dominate and so the MFPT can be approximated as

$$\frac{s}{b\lambda} + \frac{1}{4b^2\lambda}\log\Big(1+\frac{s}{w}\Big),$$

which shows that there is a 'penalty' term for synchronisation. When $s \gtrsim 1/b$ it is less clear as it is possible that the mona could perform some or all of the erasure along their

journey leading to no penalty at all. From a practical point of view, any penalty in this case is negligible as the journey time will be larger, but it is instructive to understand what occurs in this case in order to understand how the system erases information. If the penalty were eliminated, then it would suggest that the lateral distribution of the mona should collimate into a narrow 'beam' in anticipation of the synchronisation surface. This is clearly nonsensical, as the tendency is for the mona to diffuse laterally. A simple way to model the interaction is to divide the d mona into 1 'latent' monon and $d-1$ 'precocious' mona. The latent monon is identified as the last monon to arrive, and as such the precocious mona can be assumed to have reached a steady state distribution. For uniform constant bias, each monon's steady state distribution is geometric with $\mathbb{P}(\mathcal{P}_n) = \frac{b}{p}(\frac{q}{p})^n$ and entropy $\log \frac{p}{b} + 1 + \mathcal{O}(\frac{b}{q})$. For the latent monon to pass through the constriction point, each of the precocious mona's distributions must be erased. As these erasures could in principle occur simultaneously, this gives a lower bound for such a penalty as

$$\frac{d-1}{2db^2\lambda} \log \frac{p}{bw}.$$

Putting the two results together, we see that the lower bound on the penalty should increase logarithmically with distance before reaching a plateau as $s \gtrsim 1/b$.

Quasi-Steady State Approximation Whilst the above is reasonable from an information theory perspective, it does not provide a mechanism for information erasure. A simple mechanistic model for synchronisation can be given by the Quasi-Steady State Approximation. This approximation consists of two phases; first, the initial distribution $\mathcal{I}$ evolves and approaches the interfacial surface $\mathcal{S}$, which is taken to be reflective. We then assume the distribution to have roughly attained its steady state form, at which point the second phase begins. In the second phase, there is a steady leak of density through $\mathcal{S}$ per the transition rates of the system which is assumed to not significantly affect the form of the distribution in $\mathcal{P}$.

For constant uniform bias, the steady state is given by $\mathbb{P}(\mathcal{P}_n) = |\mathcal{P}_n|(\frac{q}{p})^n / \sum_{k=0}^{\infty} |\mathcal{P}_k|(\frac{q}{p})^k$. Evaluating the normalisation constant is non-trivial for arbitrary $|\mathcal{P}_k|$, but we find approximately that $\mathbb{P}(\mathcal{P}_0) \sim b^d w^{d-1}$. The leak rate is $p\lambda\mathbb{P}(\mathcal{P}_0)$, and therefore the Quasi-Steady State approximation gives an exponential decay process with half life $\sim b^{-d} w^{1-d} \lambda^{-1}$. This is the approximate penalty term, giving an overall MFPT

$$\tau \sim \frac{s}{b\lambda} + \frac{1}{b^d w^{d-1} \lambda}$$

and showing that, whilst in principle an order $1/b^2$ penalty term is possible via information erasure, in practice a d-dimensional synchronisation will incur a substantially greater penalty for $d > 2$ (as $b \ll 1$). Fortunately this can be mitigated, either by replacing the synchronisation with a series of 2-dimensional interactions or by making the information erasure explicit. Explicating the information erasure in such a case can be achieved using the 'naive' approach of making the interfacial hypersurface sticky at the cost of erasing at least 1 bit of information (representing the state `mononIsMoving`).

III.3.2. Discrete Phase Spaces

Whilst the preceding approximations are reasonable, they make significant simplifications that neglect much of the specific dynamics of the system. In order to be more confident in their conclusions, we shall proceed by making use of the analytical techniques discussed in Section III.2. We begin with the discrete case, being that this corresponds to the underlying geometry of the phase spaces of interest.

III.3.2.1. Exact results

Recall that the MFPT can be obtained as the total phase density at steady state of the modified system, wherein the initial distribution $\mathcal{I}$ is instead supplied as a constant forcing term. Equivalently, this modified system can be understood as the 'teleportation' of density absorbed at $\mathcal{S}$ back into the system according to $\mathcal{I}$. This is because, at steady state, the rate of density loss at the absorbing boundary must exactly balance the rate of the forcing term. Observe now that, if $s = 1$ (i.e. $\mathcal{I} \subseteq \mathcal{P}_1$), then the 'teleportation' of density from $\mathcal{S}$ to $\mathcal{I}$ resembles a reflective boundary. An unforced system with reflecting boundaries and whose underlying Markov chain is reversible has the convenient property that the current is everywhere vanishing. Such a reversible Markov system with vanishing current permits us to readily obtain the steady state using the principle of *detailed balance*: the densities of each pair of states x and y are such that $[x]\,\lambda(x \mapsto y) = [y]\,\lambda(y \mapsto x)$.

Before continuing, we must first clarify the constraint on $\mathcal{I}$. We begin by introducing a labelling for the states in each hypersurface $\mathcal{P}_n$ as $\{(n, c) : c \in \mathcal{P}_n\}$. It is important to note that the steady state densities in the initial hypersurface, $[(s, c)]$, do not generally correspond in a simple way to the initial distribution $[(s, c)]_0 \sim \mathcal{I}$. In the reflective system, the effective forcing term is given simply by the reflection rates,

$$\begin{aligned}[(1, c)]_0 &= \sum_{c' \in \mathcal{S}} [(0, c')]\,\lambda((0, c') \mapsto (1, c)) \\ &\equiv [(1, c)] \sum_{c' \in \mathcal{S}} \lambda((1, c) \mapsto (0, c')).\end{aligned}$$

That is, the only permissible initial distribution is prescribed by the steady state distribution of $\mathcal{P}_1$ and is in proportion to the forward transition rates. As shall be seen shortly, for the example geometry shown in Figure III.3d this yields the initial distribution

$$[(1, c)]_0 = \frac{1}{w} \begin{cases} \frac{1}{2}, & c \text{ on boundary}, \\ 1, & \text{otherwise}, \end{cases}$$

which, for large w, is effectively uniform. Finally, to ensure that the density sum yields the MFPT, we pick the normalisation such that $[\mathcal{I}] = \sum_{cc'} [(1, c)]\,\lambda((1, c) \mapsto (0, c')) = \sum_{cc'} [(0, c')]\,\lambda((0, c') \mapsto (1, c)) = 1$. Bear also in mind that the reflecting boundary is an artefact of this representation, with its density in the original system being zero; therefore, in computing the MFPT, we must remember to neglect its contribution.

Truncated Simplices In addition to being a relatively simple geometry, the truncated simplex systems have constant uniform bias. This property gives rise to a particularly simple steady state. Suppose the forward and backward rates are given by $p\lambda/d$ and $q\lambda/d$ respectively where $p - q = b$ and d is the dimension of the simplex. More precisely, given that each non-boundary state (n, c) is adjacent to d states $(n-1, c'_i)$ in the forward direction and d states $(n+1, c'_i)$ in the backward direction for $i = 1 \dots d$, the rates may be written as $\lambda((n, c) \mapsto (n-1, c'_i)) = p\lambda/d$ and $\lambda((n, c) \mapsto (n+1, c'_i)) = q\lambda/d$. Using detailed balance, we can deduce therefore that $[(n, c)] = \frac{q}{p}[(n-1, c'_i)]$ for *any* $i = 1 \dots d$ and hence $[(n-1, c'_i)] = \frac{p}{q}[(n, c)]$. As each pair of adjacent hypersurfaces in a truncated simplex forms a connected subgraph, one can show inductively that $[(n, c)] = [(n, c')]$ for all $c, c' \in \mathcal{P}_n$ (including boundary states). Combining the results thus far, we obtain $[(n+m, c)] = (\frac{q}{p})^m[(n, c')]$. Finally, we find the normalisation as $\forall c.\, 1 = [(0, c)]|\mathcal{S}|q\lambda$ which leads to the complete description of the steady state distribution and hence the MFPT,

$$[(n, c)] = \frac{(\frac{q}{p})^n}{q\lambda|\mathcal{S}|}, \qquad \tau = \sum_{n=1}^{\infty} \frac{(\frac{q}{p})^n}{q\lambda} \frac{|\mathcal{P}_n|}{|\mathcal{S}|},$$

where we have neglected the contribution of the reflective boundary density, $[\mathcal{S}]$ as this is just an artefact of our representation.

Using the expression for the hypersurface sizes, we can compute the MFPT for an arbitrary dimension d. The factor $|\mathcal{P}_n|/|\mathcal{S}|$ reduces to $\prod_{k=0}^{d-2}(1 + \frac{n}{w+k})$ and therefore the MFPT is given by $\frac{1}{q\lambda}\sum_{n=1}^{\infty}(\frac{q}{p})^n \prod_{k=0}^{d-2}(1 + \frac{n}{w+k})$. Enumerating for dimensions $d = 1, 2, 3$, we find the d-dimensional MFPTs τ_d,

$$\begin{aligned}
\tau_1 &= \frac{1}{q\lambda}\sum_{n=1}^{\infty}\left(\frac{q}{p}\right)^n && = \frac{1}{b\lambda}, \\
\tau_2 &= \frac{1}{q\lambda}\sum_{n=1}^{\infty}\left(\frac{q}{p}\right)^n\left(1 + \frac{n}{w}\right) && = \frac{1}{b\lambda} + \frac{1}{w}\frac{p}{b^2\lambda}, \\
\tau_3 &= \frac{1}{q\lambda}\sum_{n=1}^{\infty}\left(\frac{q}{p}\right)^n\left(1 + \frac{n}{w}\right)\left(1 + \frac{n}{w+1}\right) && = \frac{1}{b\lambda} + \frac{2w+1}{w(w+1)}\frac{p}{b^2\lambda} + \frac{1}{w(w+1)}\frac{p}{b^3\lambda}.
\end{aligned} \tag{III.1}$$

Obtaining an expression for general d is non-trivial, but we can see that the leading order term will be $\sim b^{-d}w^{1-d}\lambda^{-1}$ whenever $w \lesssim 1/b$. When $d \lesssim w$, we can also obtain

an approximate expression,

$$\begin{aligned}\tau_d &\approx \frac{1}{q\lambda}\sum_{n=1}^{\infty}\Big(\frac{q}{p}\Big)^n\Big(1+\frac{n}{w}\Big)^{d-1}\\ &= \frac{1}{q\lambda}\sum_{k=0}^{d-1}\binom{d-1}{k}\Big[\Big(\frac{t}{w}\frac{\partial}{\partial t}\Big)^k\frac{1}{1-t}\Big]_{t=\frac{q}{p}}\\ &\approx \frac{1}{q\lambda}\sum_{k=0}^{d-1}\binom{d-1}{k}\Big[\Big(-\frac{1}{w}\frac{\partial}{\partial s}\Big)^k\frac{1}{s}\Big]_{s=2b}\\ &\approx \sum_{k=0}^{d-1}\Big(\frac{d}{2w}\Big)^k\frac{1}{b^{k+1}\lambda}.\end{aligned}$$

General Geometries Other geometries can be evaluated in the same way as for truncated simplices. We will now generalise the preceding argument as far as possible. First, we use detailed balance to compute the average state density for each hypersurface,

$$\begin{aligned}\frac{[\mathcal{P}_n]}{|\mathcal{P}_n|} &= \frac{1}{|\mathcal{P}_n|}\sum_{c\in\mathcal{P}_n}[(n-1,c')]\frac{\lambda((n-1,c')\mapsto(n,c))}{\lambda((n,c)\mapsto(n-1,c'))}\\ &= \frac{[\mathcal{P}_{n-1}]}{|\mathcal{P}_{n-1}|}\underbrace{\frac{1}{|\mathcal{P}_n|}\sum_{c\in\mathcal{P}_n}\frac{[(n-1,c')]}{[\mathcal{P}_{n-1}]/|\mathcal{P}_{n-1}|}\frac{\lambda((n-1,c')\mapsto(n,c))}{\lambda((n,c)\mapsto(n-1,c'))}}_{\hat{t}_n},\end{aligned}$$

where for each $c\in\mathcal{P}_n$, $c'\in\mathcal{P}_{n-1}$ is any node adjacent to c (i.e. such that the transition rates between them are non-zero). In so doing, we have generalised the factor $t=\frac{q}{p}$, used for constant uniform bias, to $\hat{t}_n$. These densities can then be summed to obtain an expression for the MFPT,

$$\begin{aligned}\tau &= [\mathcal{S}]\sum_{n=1}^{\infty}\frac{|\mathcal{P}_n|}{|\mathcal{S}|}\prod_{k=1}^{n}\hat{t}_k\\ &= \Big\langle\sum_{c'}\lambda((0,c)\mapsto(1,c'))\Big\rangle_{c\in\mathcal{S}}^{-1}\sum_{n=1}^{\infty}\frac{|\mathcal{P}_n|}{|\mathcal{S}|}\prod_{k=1}^{n}\hat{t}_k,\end{aligned}$$

where we have used the normalisation condition to solve for $[\mathcal{S}]$.

We can now try to extract general scaling behaviour for a broad class of systems. As we are considering constrictive geometries, $|\mathcal{P}_n|$ is a monotonically increasing function as $n\to\infty$. Therefore, we must have $\prod^n\hat{t}_k\to 0$ as $n\to\infty$ or else the MFPT will diverge (the physical reason being that $\hat{t}_k>1$ implies negative bias). Moreover, the $\prod^n\hat{t}_k$ must, in the limit, decrease faster than $|\mathcal{P}_n|$ grows. Provided that (most) of the $\hat{t}_k$ are less than 1 this is usually true as it ensures exponential decay whereas we expect $|\mathcal{P}_n|$ to grow only polynomially. It is reasonable to assume that regions of $\hat{t}_k>1$ (negative bias) are

brief and rare. Given this assumption, the $\prod^n \hat{t}_k$ can be characterised by a decay length $\ell_n = \min\{m : -\sum_{k=n}^{n+m} \log \hat{t}_k \geq 1\}$, i.e. the distance from n over which it decays by a factor e. This decay length is an estimator of the bias, $\ell_n \sim 1/2b$.

Consider the function $n^{p-1}t^n$ for $p, n > 0$ and $0 < t < 1$; the function increases monotonically until its one turning point, before decaying inexorably towards zero. The turning point can be shown to occur at $n = (p-1)\ell$ where $\ell = -1/\log t$; consequently, the integral under the curve is dominated by the range $[0, (p - 1 + \mathcal{O}(1))\ell)$. The integral can therefore be approximated as $\sim ((p - 1 + \mathcal{O}(1))\ell)^p/p \sim p!\ell^p$ by assumption that within this range $t^n \sim \mathcal{O}(1)$. Returning to our general MFPT expression, and using the fact that $|\mathcal{P}_n|$ is monotonic, we can come to the same approximate conclusion. Interpolating quantities to be continuous in n, the turning point occurs when $\partial_n \log |\mathcal{P}_n| = -\log \hat{t}_n$. Expanding $|\mathcal{P}_n|$ as a polynomial in n over this range, we can approximate it by its leading order term as $|\mathcal{P}_n| = an^{p-1} + \mathcal{O}(n^{p-2})$, and therefore obtain the same range $\sim [0, p\ell)$. Finally then, the MFPT can be approximated to leading order (up to a constant multiplicative factor of order unity) as

$$\tau \sim a\Big\langle \sum_{c'} \lambda((0,c) \mapsto (1,c')) \Big\rangle_{c\in\mathcal{S}}^{-1} \frac{p!}{2^p} \frac{1}{b^p}$$

and we identify p as the effective dimensionality d of the system.

III.3.2.2. Bounded Results

To extend to the case of $s > 1$ we proceed inductively. The MFPT from a node x is given by the recurrence relation $\tau(x \mapsto \mathcal{S}) = (\sum_{x'} \lambda(x \mapsto x'))^{-1} + \sum_{x'} \lambda(x \mapsto x')\tau(x' \mapsto \mathcal{S})$ where the x' are adjacent to x. To rewrite this inductively, first pick a contiguous hypersurface $\mathcal{X}$ lying between $\mathcal{I}$ and $\mathcal{S}$ such that all paths from $\mathcal{I}$ to $\mathcal{S}$ pass through $\mathcal{X}$ at least once. It can then be seen that $\tau(i \mapsto \mathcal{S})$ where $i \in \mathcal{I}$ can be written as $\tau(i \mapsto \mathcal{X}) + \sum_{x\in\mathcal{X}} p_{x|i}\tau(x \mapsto \mathcal{S})$ where $\sum_{x\in\mathcal{X}} p_{x|i} = 1$. This itself can be proven inductively by assumption that any node in the region before $\mathcal{X}$ can be written thus and using the trivial base case of $\tau(x \mapsto \mathcal{X}) = 0$ where $x \in \mathcal{X}$. We then introduce the distributional quantity $\tau(\mathcal{I} \mapsto \mathcal{S}) = \sum_{i\in\mathcal{I}} \mathbb{P}(i|\mathcal{I})\tau(i \mapsto \mathcal{S})$ to write this as $\tau(\mathcal{I} \mapsto \mathcal{S}) = \tau(\mathcal{I} \mapsto \mathcal{X}_\mathcal{I}) + \tau(\mathcal{X}_\mathcal{I} \mapsto \mathcal{S})$ where $\mathcal{X}_\mathcal{I}$ is the *unique* distribution of states in $\mathcal{X}$ as they are first reached from $\mathcal{I}$.

Using this inductive form, we can rewrite the MFPT as the sum of a series of $s = 1$ MFPTs, $\tau(\mathcal{I}_s \mapsto \mathcal{S}) = \sum_{k=0}^{s-1} \tau(\mathcal{I}_{k+1} \mapsto \mathcal{I}_k)$ where $\mathcal{I}_k$ is the unique distribution over $\mathcal{P}_k$ as first reached from $\mathcal{I}_{k+1}$. This immediately yields a first approximation to the MFPT for arbitrary s by assuming that $\mathcal{I}_k$ is similar to the reflective distribution used in Section III.3.2.1. Calculating for the general geometry restricted to uniform constant

bias, we find

$$\begin{aligned}\tau &\approx \sum_{k=1}^{s} \frac{1}{q\lambda} \sum_{n=1}^{\infty} \frac{|\mathcal{P}_{k+n}|}{|\mathcal{P}_{k-1}|} \Big(\frac{q}{p}\Big)^n \\ &= \frac{1}{p\lambda} \sum_{n=0}^{\infty} \Big(\frac{q}{p}\Big)^n \sum_{k=0}^{s-1} \frac{\mathcal{P}_{k+n+1}}{\mathcal{P}_k} \\ &\equiv \frac{1}{p\lambda}\, \mathcal{Z}\Big\{\sum_{k=0}^{s-1} \frac{\mathcal{P}_{k+n+1}}{\mathcal{P}_k}\Big\}\Big[\frac{p}{q}\Big]\end{aligned} \tag{III.2}$$

where $\mathcal{Z}$ is the unilateral $\mathcal{Z}$-transform.

Unfortunately we have been unable to generally determine $\mathcal{I}_{n-1}$ from $\mathcal{I}_n$, but we can extract sufficient information to be able to bound the MFPT from above and below. In the reflective approach for $s = 1$, recall that the steady state distribution is given by $[(n+m, c)] = (\frac{q}{p})^m[(n, c')]$ for all c, c'. Notice also that the transitions to $\mathcal{S}$ are precisely those that are to be 'absorbed', and those from $\mathcal{S}$ are precisely those to be 'teleported'. The consequence is that the distribution on $\mathcal{S}$ is the unique distribution $\mathcal{I}_0$ as earlier defined, induced by $\mathcal{I} = \mathcal{I}_1$. Consider then the case $s = 2$ where the initial distribution $\mathcal{I}$ is this 'reflective' distribution. The MFPT can be decomposed into $\tau(\mathcal{I}_2 \mapsto \mathcal{I}_1) + \tau(\mathcal{I}_1 \mapsto \mathcal{S})$; we know the first term as it is just that given by our $s = 1$ calculation, but the second term requires us to know the $s = 1$ MFPT for a uniform initial distribution.

The 'reflective' distribution is defined by $[(n, c)] \propto$ number of forward transitions; for example, the distribution for the Truncated 2-Simplex is given ratiometrically as $1 : 2 : 2 : \cdots : 2 : 1$. The uniform $1 : 1 : 1 : \cdots : 1 : 1$ distribution can thus be considered as a normalised superposition of the reflective distribution and a pure-boundary distribution $1 : 0 : 0 : \cdots : 0 : 1$. This is useful because the MFPT is linear, and therefore we can apply the principle of linear superposition. Now, for uniform constant bias in a constrictive geometry, the boundary nodes within a given hypersurface $\mathcal{P}_n$ must have maximal MFPT because the expected transition direction from the boundary nodes is backwards whereas for all other nodes it is forwards. As a result, the MFPT for a uniform distribution is greater than for a reflective distribution. This applies for each subsequently induced hypersurface distribution, as the effective negativity of the boundary transitions causes it to be 'sticky' and attract density to itself.

Lower Bound Therefore, for an initially reflective distribution, the estimate of the MFPT given by Equation (III.2) is in fact a lower bound. Moreover, it is a lower bound for any distribution lying between the reflective distribution and the limiting distribution induced by the aforementioned boundary attraction phenomenon.

Upper Bound A trivial upper bound is given by the sum of the MFPTs for pure-boundary distributions, but this is needlessly loose. To obtain a tighter upper bound, we

appeal to the distributional superposition,

$$\begin{aligned}\tau(\mathcal{I}_n^{\text{refl.}}) &= \tau(\mathcal{I}_n^{\text{refl.}} \mapsto \mathcal{I}_{n-1}^{\text{unif.}}) + \alpha\tau(\mathcal{I}_{n-1}^{\text{refl.}}) + (1-\alpha)\tau(\mathcal{I}_{m-1}^{\text{bound.}}) \\ &= \frac{1}{p\lambda}\sum_{r=0}^{\infty}\frac{|\mathcal{P}_{n+r}|}{|\mathcal{P}_{n-1}|}\Big(\frac{q}{p}\Big)^r + \alpha\tau(\mathcal{I}_{n-1}^{\text{refl.}}) + (1-\alpha)\tau(\mathcal{I}_{m-1}^{\text{bound.}}).\end{aligned}$$

The value of α can be found by taking the ratiometric vector for the reflective distribution, say $1:2:\cdots:2:1$, and then finding the boundary distribution which makes this up to uniform, i.e. $1:0:\cdots:0:1$. The uniform distribution would then be $2:2:\cdots:2:2$, and so the proportion of density in the reflective component is $(1+2+\cdots+2+1)/(2+2+\cdots+2+2)$, or the normalised average $\langle f_x\rangle/\max_x f_x$ where f_x is the number of forward transitions for node x.

The boundary MFPT is harder to find, but we can bound it from above. First consider the exact $s=1$ MFPT when $w=1$; in this case, $\mathcal{P}_1$ is formed entirely from boundary nodes and therefore its MFPT is precisely the boundary MFPT. In fact, this boundary MFPT is an upper bound on all the other boundary MFPTs as it corresponds to the special case of no interior nodes; when interior nodes do exist, they will have lower MFPTs than the boundary and density in the boundary will diffuse into these interior nodes, thus reducing the boundary MFPT. These definitions of α and boundary MFPTs will become clearer with a concrete example.

Truncated Simplices Specialising to truncated simplices, we can quantify these lower and upper bounds. The $s=1$ MFPTs have already been obtained earlier for $d=1,2,3$ in Equation (III.1), and so the MFPT lower bounds can be obtained thus,

$$\begin{aligned}\tau_1 &= \frac{s}{b\lambda}, \\ \tau_2 &\geq \frac{s}{b\lambda} + \frac{p}{b^2\lambda}\Delta\psi, \\ \tau_3 &\geq \frac{s}{b\lambda} + \frac{p}{b^2\lambda}\Big(2\Delta\psi + \frac{1}{s+w} - \frac{1}{w}\Big) + \frac{p}{b^3\lambda}\Big(\frac{1}{w} - \frac{1}{s+w}\Big),\end{aligned}$$

where $\Delta\psi = \sum_{k=w}^{w+s-1}\frac{1}{k} = \psi(s+w) - \psi(w) \approx \log\big(1+\frac{s}{w}\big)$ with ψ the digamma function. The factor $\Delta\psi$ closely resembles the information loss in the synchronisation interaction for 2-simplices, and the coefficient of $p/b^2\lambda$ in τ_3 resembles that for 3-simplices, as expected. Notice that these lower bounds do not plateau as $s \to 1/b$ as conjectured in Section III.3.1; whilst it is in principle possible from an information theoretic perspective to reach this plateau penalty, the passive dynamics of these interactions preclude this possibility. Nevertheless, at the point where the plateau becomes relevant, the penalty becomes insignificant in comparison to the $s/b\lambda$ for τ_2 and therefore this is of little concern. For τ_3 and above, penalties of order $\mathcal{O}(b^{-3})$ and above are present and remain significant beyond $s/b\lambda$.

For the upper bound, we first obtain the upper bound on the boundary MFPT as $\frac{s}{b\lambda} + \frac{ps}{b^2\lambda}$ for $d=2$ and $\frac{s}{b\lambda} + \frac{3}{2}\frac{ps}{b^2\lambda} + \frac{1}{2}\frac{ps}{b^3\lambda}$ for $d=3$. One needs to be particularly careful for

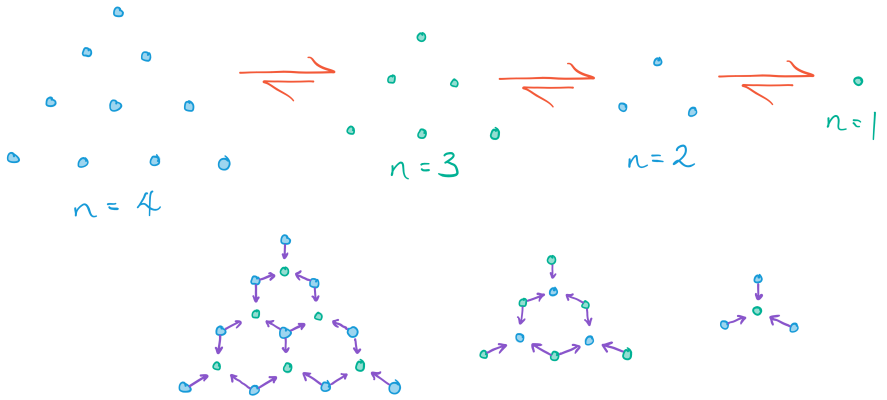


Figure III.5: An illustration of a series of hypersurfaces in the $d = 3$ simplex to demonstrate the different classes of boundary nodes. The top row shows the hypersurfaces on their own, whilst the bottom row shows adjacent hypersurfaces superimposed and the forward transitions between them.

$d = 3$ and above as there are different 'classes' of boundary nodes: for $d = 3$ (as shown in Figure III.5), there are 'vertex' boundary nodes which have only one forward transition, and 'edge' boundary nodes which have two forward transitions, whilst interior nodes have three. More generally, in dimension d there are $d - 1$ classes of boundary nodes with $1, 2, \ldots, d - 1$ forward transitions respectively (and interior nodes with d forward transitions). The vertex nodes will have the highest MFPTs and correspond precisely to the $w = s = 1$ MFPT, and are therefore what we shall use for MFPT bounds in higher dimensions. The values of α can be found by counting transitions, and are $\alpha_1 = 1$, $\alpha_2 = 1 - (n + w - 1)^{-2}$ and $\alpha_3 = 1 - 2(n + w)^{-1}$.

Using these values of the boundary MFPTs and superposition fractions, we can write a recurrence relation for each dimension in $\tau_d(n)$,

$$\begin{aligned}
\tau_1(w+k+1) &= \frac{1}{b\lambda} + 1 \cdot \tau_1(w+k) + 0 \cdot \left(\frac{k}{b\lambda}\right), \\
\tau_2(w+k+1) &\leq \frac{1}{b\lambda} + \frac{w+k-1}{w+k} \cdot \tau_2(w+k) + \frac{1}{w+k} \cdot \left(\frac{k}{b\lambda} + \frac{pk}{b^2\lambda}\right), \\
\tau_3(w+k+1) &\leq \frac{1}{b\lambda} + \frac{w+k-1}{w+k+1} \cdot \tau_3(w+k) + \frac{2}{w+k} \cdot \left(\frac{k}{b\lambda} + \frac{3}{2}\frac{pk}{b^2\lambda} + \frac{1}{2}\frac{pk}{b^3\lambda}\right),
\end{aligned}$$

with initial conditions

$$\begin{aligned}
\tau_1(w+1) &= \frac{1}{b\lambda}, \\
\tau_2(w+1) &= \frac{1}{b\lambda} + \frac{1}{w}\frac{p}{b^2\lambda}, \\
\tau_3(w+1) &= \frac{1}{b\lambda} + \frac{2w+1}{w(w+1)}\frac{p}{b^2\lambda} + \frac{1}{w(w+1)}\frac{p}{b^3\lambda}.
\end{aligned}$$

These can finally be solved to yield

$$\begin{aligned}
\tau_1 &= \frac{s}{b\lambda}, \\
\tau_2 &\leq \frac{s}{b\lambda} + \frac{\frac{1}{2}s(s+1)}{s+w-1}\frac{p}{b^2\lambda}, \\
\tau_3 &\leq \frac{s}{b\lambda} + \frac{\frac{1}{2}s[2s^2+(3s+1)(w+1)-4s]}{(w+s)(w+s-1)}\frac{p}{b^2\lambda} + \frac{\frac{1}{6}s(2s^2+3(s-1)(w-1)+4)}{(w+s)(w+s-1)}\frac{p}{b^3\lambda},
\end{aligned}$$

and, in the limit of large s, these simplify to

$$\tau_1 = \frac{s}{b\lambda}, \qquad \tau_2 \leq \frac{s}{b\lambda} + \frac{s+1}{2b^2\lambda}, \qquad \tau_3 \leq \frac{s}{b\lambda} + \frac{2s+3w}{2b^2\lambda} + \frac{2s+3w}{6b^3\lambda}.$$

In summary, the MFPT penalties for $d = 1, 2, 3$ truncated simplices are bounded above and below by

$$\begin{aligned}
\Delta\tau_1 &= 0, \\
\frac{p}{b^2\lambda}\Delta I_2 \leq \Delta\tau_2 &\leq \frac{p}{b^2\lambda}\frac{\frac{1}{2}s(s+1)}{s+w-1}, \\
\frac{p}{b^2\lambda}\Delta I_3 + \frac{p}{b^3\lambda}\frac{s}{w(s+w)} \leq \Delta\tau_3 &\leq \frac{p}{b^2\lambda}\frac{\frac{1}{2}s[2s^2+(3s+1)(w+1)-4s]}{(w+s)(w+s-1)} \\
&\quad + \frac{p}{b^3\lambda}\frac{\frac{1}{6}s(2s^2+3(s-1)(w-1)+4)}{(w+s)(w+s-1)},
\end{aligned}$$

where $\Delta I_2 = \psi(s+w) - \psi(w)$ and $\Delta I_3 = 2\Delta I_2 + \frac{1}{s+w} - \frac{1}{w}$. Expressions for higher dimensions can be obtained using the same approach, but the key point to note is that the terms in each order of b^{-1} are present and significant in both the lower and upper bounds and therefore a synchronisation interaction in dimension d will be subject to a penalty of order $\mathcal{O}(b^{-d})$, with the coefficient of the $\mathcal{O}(b^{-2})$ penalty term bounded below by the information loss in the interaction.

III.3.2.3. Numerical Simulation

In the course of investigating this synchronisation problem, a suite of tools[4] for computing MFPTs was developed, as well as an assortment of other tools[5][6]. Computing MFPTs in regimes of low bias and unbounded state space is challenging because the probability density tends to diffuse very far from $\mathcal{S}$. Attempting to compute it in a deterministic way via the explicit steady state distribution has a high spatial complexity as one must make sure not to prematurely truncate the state space, or else risk underestimating the MFPT. A rough estimate gives a spatial complexity of $\mathcal{O}(1/b^2 + (s+w)/b)$, and the time

[4] https://github.com/hannah-earley/revcomp-synch-rw-mfpt
[5] https://github.com/hannah-earley/revcomp-synch-rw-distribution
[6] https://github.com/hannah-earley/revcomp-synch-rw-distribution-2

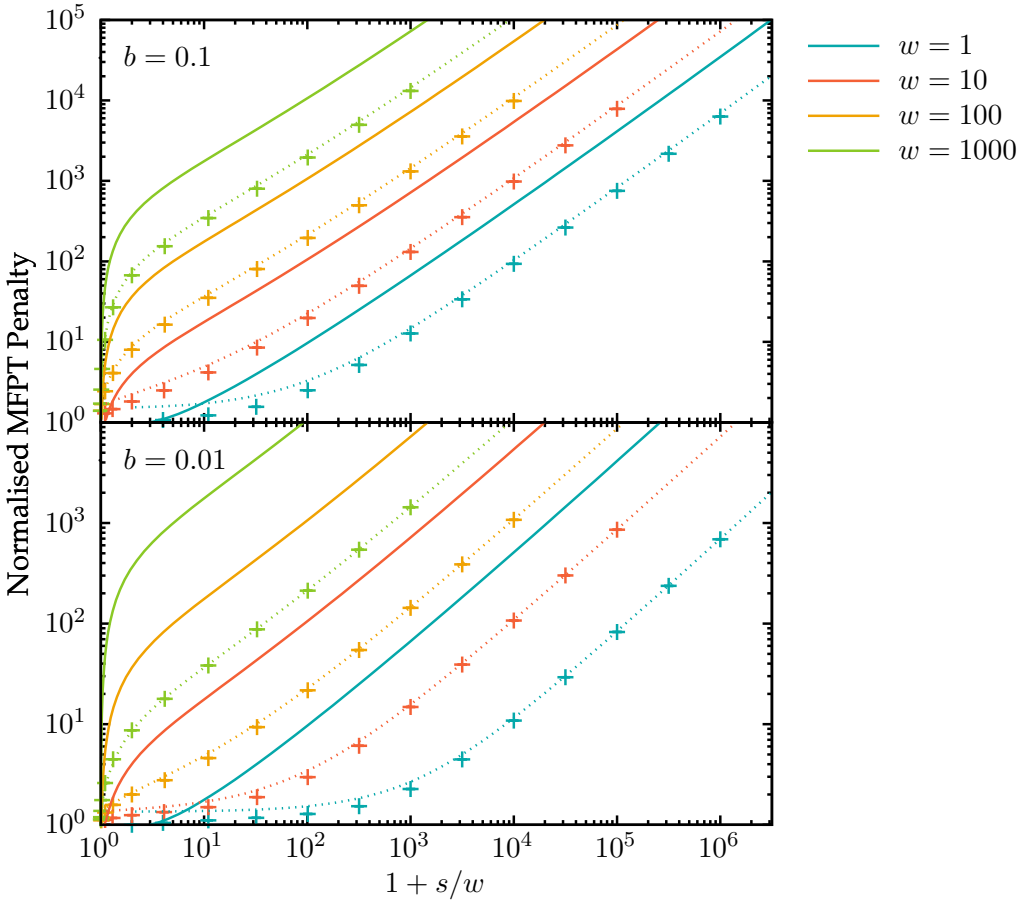


Figure III.6: Simulation results for the $d = 2$ truncated simplex case, as depicted in Figure III.3d, with varying initial hyperplane distances s and constriction widths w, and subject to a uniform bias $b \in \{0.1, 0.01\}$. The *Normalised MFPT Penalty* corresponds to $(\tau - \frac{s}{b\lambda})/(\frac{p}{b^2\lambda}\Delta I_2)$, i.e. it is the MFPT penalty divided by the lower bound as computed in Section III.3.2. As a result, the horizontal axis corresponds to the lower bound. The plotted points show simulation data with (negligible) error bars, the solid lines show the (normalised) upper bound, and the dashed lines show the empirical approximation given by Equation (III.3).

complexity will be the product of this spatial complexity and the number of simulation steps to convergence, which will be at least $\mathcal{O}(1/b)$ to allow particles from the initial distribution to explore the entire state space.

The alternative is to exploit the Markov property of the system in order to use a stochastic Markov-Chain Monte-Carlo simulation approach, or MCMC. In an MCMC approach we essentially apply the stochastic dynamics of the system to an ensemble of instances of the system. In this case, each instance is a single phase particle, whose coordinate represents the joint state of the underlying mona. Therefore, for an ensemble of size m, the spatial complexity will be $\mathcal{O}(m)$. Again, the goal is to obtain the steady state distribution of the forced absorbing-boundary system; to ensure conservation of density, we use the teleporting form in which there is a one-to-one correspondence between each particle incident on the absorbing boundary and a particle injected into the initial distribution.

The naive approach is to record the number of time-steps between injection and absorption for each particle. Unfortunately this approach is not robust enough here as the high-diffusion negligible-drift regime leads to substantial variance in the FPT distribution such that a significant proportion of particles in the ensemble will get 'lost' in the phase space for an unboundedly long time. Therefore the mean of the particles' injection-absorption times will typically be divergent, or the program will run indefinitely as it waits for each particle to be absorbed. A possible solution is to exclude particles which have not returned after some predefined number of time-steps, but this will lead to an underestimate of the MFPT.

The correct solution, it transpires, is to instead record the absorption current: that is, within some time-step window Δt, count the number of particles n incident on the absorbing boundary in order to obtain an unbiased estimate of the current $S = n/m\Delta t$ where m is the ensemble size. This process can be repeated as many times as one likes to obtain a series of measurements of the current, from which basic statistics can be used to find an improved estimate of the mean current and its standard error.

There remains, however, a caveat common to all MCMC simulations. Assuming our particles are distributed according to the desired steady state, then their distribution will remain so as they evolve under the stochastic dynamics of the system. The problem is in ensuring that the initial distribution of the system is the steady state, despite not knowing what that is. To address this, one typically initialises the system to an approximation of the steady state and then lets the distribution 'burn in' by running the simulation for some number of initial time-steps, after which it is hoped that the approximation will have converged to the steady state. The poorer the quality of the initial approximation to the steady state, the longer this burn-in phase will take. Moreover, the length of the burn-in time can be hard to ascertain and so often one must resort to a heuristic judgement.

Oft forgotten can be the quality of the random number generator used to simulate the stochastic dynamics. In an earlier iteration of this tool, we neglected to consider this and as a result used the standard `rand` function: a *linear congruential generator* with period 2^{32}. As the simulation times required to obtain good quality MFPT estimates for

many of our parameters exceeded this period, the data obtained was invalid. It can be hard to detect this, and we only realised when a debug trace of the intermediate outputs showed a noticeable and unexpected periodicity. As a result, our MFPT suite now uses the PCG family of PRNGs [84] which have an internal state space size and period of 2^{64} whilst also being very fast.

Whilst the dynamics of this system are very simple, some several quadrillion time-steps were required to obtain the data shown in Figure III.6 and so a sophisticated toolchain was built around the MCMC routine to improve robustness and automatability. The core program, `./walk`, is written in `C++` and makes use of `OpenMP` for parallelisation. Simulating an ensemble of random walkers is an *embarrassingly parallel* problem, meaning that performance can generally scale linearly with the number of available CPUs due to the minimal amount of inter-thread synchronisation necessary. Unfortunately the `GCC` and `LLVM` compilers[7] were not able to fully optimise the inner loop responsible for executing the stochastic dynamics, and so some hand tuning was necessary to improve the assembly output and bring down the iteration time to $\lesssim 5\,\mathrm{ns}$ on the machines used. To ensure resumable operation and cooperation with cluster scheduling systems such as `slurm`, the program supports checkpointing (both intermittent and in response to trappable signals). The program is also reasonably modular, making it straightforward to adapt to new random walk systems.

To coordinate simulating the large number of parameters across different topologies of networked systems and to minimise need for manual intervention, a batched job runner was developed in the form of a `python` script. The job runner has three modes of operation; it can take a description of a set of jobs and generate specific instructions for each job, it can connect to a distributed job queue to request and run these jobs, and it can query the status of all the jobs and job runners (optionally sending updates by email at regular intervals). A key feature is that it can analyse the output of `./walk` to determine whether the error has converged to a sufficiently small value, whereupon it will move onto the next job. Finally, a number of tools were created to analyse the resulting data, including the ability to infer and inspect the approximate steady state distribution over the phase space.

The results of these simulation tools are shown in Figure III.6. As well as serving as a sanity check on the derived MFPT bounds, we were also able to obtain an estimated empirical equation for the true MFPT as

$$\tau = \frac{s}{b\lambda}\left(1 + p\frac{3+s}{w+s}\right) + \frac{4}{3}\frac{p}{b^2\lambda}\Delta I_2, \tag{III.3}$$

where the $(3+s)/(w+s)$ term is inspired by the exact MFPT for $b = 1$, given by $\tau = s(1 + \frac{1}{2}\frac{3+s}{w+s})$. In particular these tools were very helpful in getting an intuitive feel for the parametric dependence of the MFPT when analytic approaches seemed unyielding.

[7] In our experience, `LLVM`'s optimisation was far better than that of `GCC` in this case.

III.3.3. Continuous Phase Spaces

The approach for continuous phase spaces will largely follow that for discrete, except that we shall make use of the Fokker-Planck equation as introduced in Section III.2.2. In order to solve for the reflective steady state, we shall use the fact that its current vanishes. Recall that the continuous current is given by $\vec{S} = (\vec{\mu} - \vec{\nabla} \cdot \mathbf{D})W$ where W is the phase density, $\vec{\mu}$ the drift coefficient and $\mathbf{D}$ the diffusion matrix. When $\mathbf{D}$ is non-singular, this can be rewritten as $-\mathbf{D}(-\mathbf{D}^{-1}\vec{\mu}' + \vec{\nabla})W = -\mathbf{D}e^{-\varphi}\vec{\nabla}e^{\varphi}W$ where $\vec{\mu}' = \vec{\mu} - [\vec{\nabla} \cdot \mathbf{D}]$ and φ is defined by $\vec{\nabla}\varphi = -\mathbf{D}^{-1}\vec{\mu}'$. If the current is identically zero, then it immediately follows that the density is given by $W = Ne^{-\varphi}$ where N is a normalisation constant. This is fairly standard, and a similar result is obtained in Chapter 6 of Risken [77].

To connect this reflective-boundary distribution with the absorbing-boundary distribution, we must find the absorption current. We assume that our reflective-boundary distribution is approximately equivalent to an absorbing-boundary distribution with forcing applied at the $\mathcal{P}(-\delta x)$ hypersurface, where x is a coordinate orthogonal to the hypersurfaces; in the limit $\delta x \to 0$ this becomes exact, in analogy with the discrete case. The true absorbing-boundary distribution has vanishing density at $\mathcal{P}(0) = \mathcal{S}$ by definition, and therefore we can recover our desired distribution by introducing an appropriate decay from $\mathcal{P}(-\delta x)$ to $\mathcal{S}$. In the limit $\delta x \to 0$ any higher order polynomial terms in this decay will vanish, and therefore the decay can be assumed linear. The absorption current is then given by

$$\vec{S}(0) = \vec{\mu}W|_{x=0} - \sum_i \vec{D}_i \lim_{\delta x_i \to 0} \frac{W|_{x_i} - W|_{x_i - \delta x_i}}{\delta x_i} = \vec{D}_x \frac{W(-\delta x)}{\delta x} = \vec{D}_x \frac{Ne^{-\varphi}}{\delta x}.$$

Using the fact that the current is along the x direction, we can find the normalisation constant subject to the constraint that the total absorption current is 1 as $N = \delta x / \int_{\mathcal{S}} \mathrm{d}^{d-1}\vec{x}\, D_{xx}e^{-\varphi}$. Therefore we find that the MFPT from $s = -\delta x$ is given by $\delta\tau = \delta x \int_{\mathcal{P}} \mathrm{d}^d\vec{x}\, e^{-\varphi} / \int_{\mathcal{S}} \mathrm{d}^{d-1}\vec{x}\, D_{xx}e^{-\varphi}$, from which we find the lower bound for the MFPT from arbitrary s,

$$\tau(s) \geq \int_{-s}^{0} \mathrm{d}x \, \frac{\int_{-\infty}^{x} \mathrm{d}x' \int_{\mathcal{P}(x')} \mathrm{d}^{d-1}\vec{x}\, e^{-\varphi}}{\int_{\mathcal{P}(x)} \mathrm{d}^{d-1}\vec{x}\, D_{xx}e^{-\varphi}} = \int_0^{\infty} \mathrm{d}t \int_{-s}^{0} \mathrm{d}x \, \frac{[\langle e^{-\varphi}\rangle A]_{x-t}}{[\langle D_{xx}e^{-\varphi}\rangle A]_x}$$

where $\langle e^{-\varphi}\rangle$ is averaged over the given hypersurface and $A = \int_{\mathcal{P}(x)} \mathrm{d}^{d-1}\vec{x}$ is the area of that hypersurface. We see that the MFPT for general phase space is easier to express in the continuum limit, and in the particular case of φ linear in x, i.e. $\varphi = \phi - \mu x/D$, this reduces to a Laplace transform:

$$\tau \geq \mathcal{L}_t\Big\{ \int_{-s}^{0} \mathrm{d}x \, \frac{[\langle e^{-\phi}\rangle A]_{x-t}}{[\langle D_{xx}e^{-\phi}\rangle A]_x} \Big\}\Big(\frac{\mu}{D}\Big).$$

Truncated Simplices For the Truncated Simplex example in d dimensions, the drift coefficient is constant uniform in the x direction and the diffusion matrix is the constant

uniform and isotropic $\mathbf{D} = D\mathbf{1}$. Therefore, $\phi = 0$ and we have from earlier that

$$A(-n) = \frac{\sqrt{d}}{(d-1)!}\Big(\frac{n+w}{\sqrt{2}}\Big)^{d-1}.$$

Hence, the MFPT lower bound for arbitrary d may be obtained thus:

$$\begin{aligned}
\tau &\geq \frac{1}{D}\mathcal{L}_t\Big\{\int_{-s}^{0}\mathrm{d}x\,\Big(\frac{w-x+t}{w-x}\Big)^{d-1}\Big\}\Big(\frac{\mu}{D}\Big)\\
&\geq \frac{1}{D}\mathcal{L}_t\Big\{\int_{0}^{s}\mathrm{d}x\sum_{k=0}^{d-1}\binom{d+1}{k}\Big(\frac{t}{w+x}\Big)^{k}\Big\}\Big(\frac{\mu}{D}\Big)\\
&\geq \frac{1}{D}\sum_{k=0}^{d-1}\binom{d+1}{k}\mathcal{L}_t\Big\{t^k\Big\}\Big(\frac{\mu}{D}\Big)\int_{0}^{s}\mathrm{d}x\,\Big(\frac{1}{w+x}\Big)^{k}\\
&\geq \frac{s}{\mu}+(d-1)\frac{D}{\mu^2}\log\Big(1+\frac{s}{w}\Big)+\sum_{k=2}^{d-1}\frac{D^k}{\mu^{k+1}}\frac{1}{k-1}\frac{(d-1)!}{(d-1-k)!}\Big[\frac{1}{w^{k-1}}-\frac{1}{(w+s)^{k-1}}\Big].
\end{aligned}$$

III.4. Recessive Case

We turn now to the recessive case, for which we shall assume a continuous phase space for simplicity. We shall also restrict our attention to recessive systems whose diffusion matrices are uniform and isotropic, $\mathbf{D} = D\mathbf{1}$, across $\mathcal{R} = \mathcal{P} \cup \mathcal{P}'$ and whose drift vectors are constant uniform in $\mathcal{P}$; the condition on $\vec{\mu}$ in $\mathcal{P}'$ will become apparent later, but for now we shall assume it takes the same value as in $\mathcal{P}$. Finally, we adopt an orthogonal coordinate system $(u; \vec{v})$ where u is chosen such that $\vec{\mu} = \mu\hat{u}$. It may appear that u indexes the hypersurfaces, but this is in fact not the case; a coordinate transform that would render u a hypersurface index would apply a shear and therefore make $\mathbf{D}$ anisotropic.

MFPT Approach Unfortunately the reflective steady state approach used for the constrictive case is inapplicable here as the unbounded transverse width of the space means the steady state is everywhere vanishing. Moreover, ignoring the zero normalisation, the uniform initial distribution is undesirable. For the canonical example shown in Figure III.4, initial distributions $\mathcal{I}$ of interest are typically centered on the line $x = y$ (corresponding to the white dashed line in the Figure). Nonetheless, we can apply[8] the

[8] Our simulation suite is specialised to the case of walks in a single quadrant (with a possibly truncated boundary). Whilst it would not be too difficult to handle more general geometries, this particular three-quadrant example can be easily transformed into a single quadrant walk. First, we identify that the geometry is symmetric in the line $x = y$ and therefore the three quadrants are mapped to one and a half, with a reflective boundary placed at $x = y$. That is, $\mathcal{P} = \{(-x, y) : x \in \mathbb{N} \wedge y \in \mathbb{Z} \wedge y > -x\}$. Next, we apply a shear $(-x, y) \mapsto (-x, y + x)$ to map the region to the top-left quadrant. The transformed transitions turn out to correspond to those of Gessel walks, a summary of which is presented by Bousquet-Mélou [85] along with an investigation of their generating function.

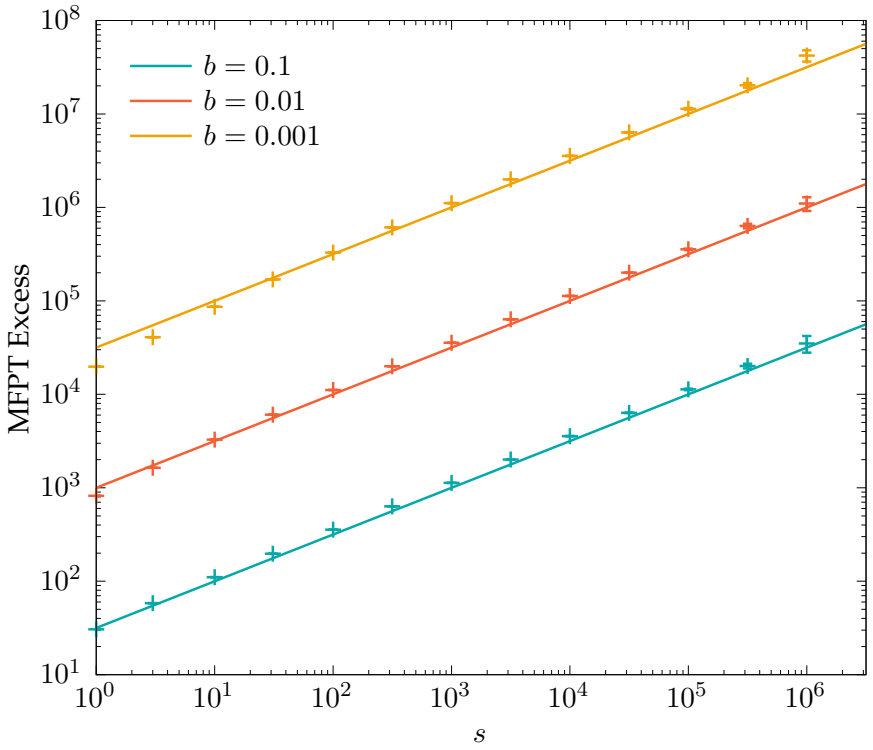


Figure III.7: Simulation results for the $d = 2$ recessive case, as depicted in Figure III.4, starting on the line $x = y$ and with varying initial distances s, and subject to a uniform bias $b \in \{0.1, 0.01, 0.001\}$. The *MFPT Excess* corresponds to $\tau - \frac{2s}{b\lambda}$ (the coefficient 2 was chosen empirically). The plotted points show simulation data with error bars, whilst the solid lines show $\frac{\sqrt{s}}{b^{3/2}\lambda}$ hence demonstrating an empirical MFPT of $\frac{2s}{b\lambda} + \frac{\sqrt{s}}{b^{3/2}\lambda}$.

simulation suite introduced in Section III.3.2.3. The results for an initial delta distribution on the line $x = y$ and a distance s from $\mathcal{S}$ are shown in Figure III.7 and reveal an empirical MFPT of $\tau \approx \frac{2s}{b\lambda} + \frac{\sqrt{s}}{b^{3/2}\lambda}$ and therefore a penalty of order $b^{-3/2}$.

This penalty would appear to be a significant improvement over that for the constrictive case of b^{-2}. Unfortunately this penalty is invalid because in fact the MFPT is not an appropriate quantity to use here. The reason is that the time quantity we are seeking is $\inf\{t : \forall u > 0.\langle n(t+u)\rangle > 0\}$ where n is a hypersurface index, and the use of the MFPT is predicated upon the idea that an initial distribution within $\mathcal{P}'$ has $\langle n(t)\rangle > 0$ for all positive t. It is straightforward to see that this property does not apply in the recessive case: Consider a simplified view of Figure III.4e as just the four quadrant domain $\mathcal{R} = \mathbb{R}^2$ such that $\mathcal{P} = \{(x, y) : x < 0 \wedge y < 0\}$ and $\mathcal{P}' = \{(x, y) : x > 0 \wedge y > 0\}$ and with $\vec{\mu} = \mu\frac{1}{\sqrt{2}}(\hat{x} + \hat{y})$; the hypersurfaces are given by $\mathcal{P}(\sqrt{2}n) = \{(n, n + y) : y \in \mathbb{R}_+\} \cup \{(n + x, n) : x \in \mathbb{R}_+\}$, i.e. $n(\vec{x}) = \sqrt{2}\min(x, y)$. Now, the evolution of a phase particle may be decomposed into that along the line $x = y$ and perpendicular to this line. Calling these coordinates u and v respectively, i.e. $(u, v) = \frac{1}{\sqrt{2}}(x + y, x - y)$ and $n = u - |v|$, we see that the distribution is given by the product of two Gaussians, with probability density

$$\frac{1}{\sqrt{4\pi Dt}}\exp\Big(-\frac{(u + s - \mu t)^2}{4Dt}\Big) \cdot \frac{1}{\sqrt{4\pi Dt}}\exp\Big(-\frac{v^2}{4Dt}\Big).$$

Now, the expected hypersurface is given by

$$\langle n\rangle = \langle u\rangle - \langle|v|\rangle = -s + \mu t - \sqrt{\frac{4Dt}{\pi}}$$

and therefore an initial distribution $\delta(n_0, 0)$ will tend to go backwards in n-space for a time $t = \frac{D}{\pi\mu^2}$, retreating as far back as $\langle n\rangle = n_0 - \frac{D}{\pi\mu}$ before it begins to advance; at time $t = \frac{2D}{\pi\mu^2}$ it returns to its initial position $\langle n\rangle = n_0$ whereupon it begins to make net progress. The consequence is that there is a time penalty for all $n_0 < +\frac{D}{\pi\mu}$, and hence the appropriate boundary to consider for the MFPT calculation is given by $n = +\frac{D}{\pi\mu}$.

Of course, in the case of $\vec{\mu}$ constant uniform across $\mathcal{R}$ we are able to use the far simpler approach of considering the exact probability distribution, as detailed above. We can also extend easily to arbitrary boundary shape. Unfortunately, as mentioned, $\vec{\mu}$ may well differ between $\mathcal{P}$ and $\mathcal{P}'$. For our canonical example in (u, v) coordinates,

$$\vec{\mu} = \begin{cases} \mu\hat{u}, & (u, v) \in \mathcal{P}, \\ -(\operatorname{sign} v)\mu\hat{v} & (u, v) \in \mathcal{P}'. \end{cases}$$

That is, in $\mathcal{P}'$ the drift direction is towards the u axis. Recall also that we have ignored the six-quadrant structure, and so there is in a sense a lot more 'space' in $\mathcal{P}'$. Nevertheless, we will be able to obtain lower and upper bounds.

Probability Distribution Approach Ignoring the caveat about $\vec{\mu}$ for now, we proceed for a general boundary shape $u = f(v)$. For the canonical example, $f(v) = |v|$. Without loss of generality, we pick $f(0) = 0$ and we note that a recessive geometry implies f increases monotonically away from $v = 0$. We then have $\langle n \rangle = \langle u - f(v) \rangle$; decomposing f into its even and odd parts, $f_e(v) = \frac{1}{2}[f(v) + f(-v)]$ and $f_o(v) = \frac{1}{2}[f(v) - f(-v)]$ respectively, we see that $\langle n \rangle \equiv \langle u - f_e(v) \rangle$ whenever the initial distribution is even because the transverse dynamics respect evenness. As we are considering the initial distribution $\delta(-s, 0)$, we discard the odd part of f without loss of generality.

If f has a power series expansion about $v = 0$, it will take the form $f(v) = \sum_{p=1}^{\infty} f_{2p} v^{2p}$. Series representations do not exist for many boundary shape functions of interest however, in particular $|v|$ has no such expansion. Consider instead a smooth function $g(v)$ such that $f(v) = g(|v|)$; if g exists, it has series expansion $\sum_{p=1}^{\infty} f_p v^p$ and hence f will have series expansion $\sum_{p=1}^{\infty} f_p |v|^p$. We can then compute $\langle f \rangle$ as $\sum_{p=0}^{\infty} \alpha_p f_p (2Dt)^{p/2}$ where $\alpha_{2p} = (2p-1)!!$ and $\alpha_{2p+1} = \sqrt{2/\pi}(2p)!!$, and $n!! = n(n-2)(n-4)\cdots$ is the double factorial.

For the case $f = f_1|v|$, $\langle n \rangle = -s + \mu t - 2f_1\sqrt{Dt/\pi}$ and we will find a penalty of order $1/\mu^2$ as earlier. If instead f is quadratic, then $\langle n \rangle = -s + (\mu - 2f_2 D)t$ and there are two subcases: if $\mu > 2f_2 D$ then the rate of computation is reduced to $\mu - 2f_2 D$ within a large vicinity of the synchronisation interaction; if $\mu \leq 2f_2 D$ then computation halts or goes backwards, and the synchronisation never[9] occurs. As we are considering systems with arbitrarily low bias, and hence μ, this effectively means that quadratic boundaries are untenable. For f cubic and above, $\langle n \rangle$ briefly increases before decreasing forever. From these, we can deduce that f should ideally be (absolute) linear, though it is admissible to also have a small quadratic component if μ is bounded from below by $2f_2 D$.

We now show how to find lower and upper bounds for $\vec{\mu}$ per our canonical example, that is $\vec{\mu}(\mathcal{P}) = \mu\hat{u}$ and $\vec{\mu}(\mathcal{P}') = -(\operatorname{sign} v)\mu\hat{v}$. In both regions, the action of the drift vector alone is to increase n. The direction of $\vec{\mu}$ in $\mathcal{P}'$ acts to pull the distribution further away from $\mathcal{P}$ than would be the case if $\vec{\mu}$ pointed u-wards; therefore, picking constant uniform $\vec{\mu}$ will retard the process of synchronisation and thereby result in an upper bound. Alternatively we can advance the process by picking a uniform superposition of the two, $\vec{\mu} = \mu(\hat{u} - (\operatorname{sign} v)\hat{v})$, letting the extra 'space' in the v axis apply to $\mathcal{P}$ too. This extra 'space' would appear to complicate matters further, but we can avoid this complication by realising that the action of this v-wards drift is simply to increase the value of n and therefore it is equivalent to setting $\vec{\mu} = 2\mu\hat{u}$. Solving for the bounds finally yields

$$\begin{aligned}\tau &\geq \frac{s}{2\mu - 2f_2 D} + \frac{2f_1^2 D}{\pi(2\mu - 2f_2 D)^2}\left[1 + \sqrt{1 + \frac{\pi s}{f_1^2 D(2\mu - 2f_2 D)}}\right] \\ &\leq \frac{s}{\mu - 2f_2 D} + \frac{2f_1^2 D}{\pi(\mu - 2f_2 D)^2}\left[1 + \sqrt{1 + \frac{\pi s}{f_1^2 D(\mu - 2f_2 D)}}\right],\end{aligned}$$

[9] It is possible to escape if the phase geometry eventually changes to a permissible form, but the penalty will be substantial.

which for our canonical example reduces to

$$\frac{s}{2\mu}+\frac{D}{2\pi\mu^2}\left[1+\sqrt{1+\frac{\pi s}{2\mu D}}\right]\leq\tau\leq\frac{s}{\mu}+\frac{2D}{\pi\mu^2}\left[1+\sqrt{1+\frac{\pi s}{\mu D}}\right].$$

This shows that the penalty is at least of order μ^{-2} and, for small s, is as high as $\mu^{-5/2}$. In other words, the penalty is worse than for the constrictive case. Interestingly however, the penalty is almost exactly the same for higher dimensions if the boundary is replaced by a hypersurface of revolution of f; in fact, in dimension d we find

$$\alpha_p=\frac{(p+d-3)!!}{(d-3)!!}\begin{cases}1 & p,d\text{ even}\\ \sqrt{\frac{2}{\pi}} & p\text{ odd}\\ \frac{2}{\pi} & p\text{ even, }d\text{ odd}\end{cases}$$

but otherwise the form of the synchronisation times remains the same. As a result, using a recessive synchronisation geometry is an alternative way of efficiently synchronising in dimensions $d=3$ and above. Of course, a sequence of $d=2$ constrictive synchronisations will limit the penalty to order μ^{-2} and so constrictive synchronisation geometries remain the most effective approach.

III.5. Constrictive Generating Functions

Finally, we present some limited successes towards generating functions for the MFPT for arbitrary initial distribution. We shall consider the truncated simplex for $d=2$ and $w=1$, as shown in Figure III.3c. To do so, we must construct a function to enumerate all possible walks from each initial position and weight them by their probabilities. A walk starts at some initial position (for convenience we use the upper right quadrant), takes a number of up, down, left and right steps, and terminates at the origin. It is also subject to the constraint that it must not reach the origin until its final step, and if the walker is on a reflective boundary then a forbidden step (left or down, depending on which boundary) is replaced by a null step with the same probability but which leaves its position unchanged.

Constructing a generating function subject to these constraints is easier in reverse. Let the variables of the generating function be x, y and t, such that a term $\rho x^m y^n t^s$ indicates that a walk starting at (m,n) reaches the origin in s steps with probability $\rho\equiv\mathbb{P}(s|m,n)$. As all walks almost surely reach the origin in a finite time, $\sum_{s=0}^{\infty}\mathbb{P}(s|m,n)=1$ and so if $W(x,y;t)=\sum_{mns}\mathbb{P}(s|m,n)x^m y^n t^s$ then $W(x,y;1)=\sum_{mn}x^m y^n\equiv\frac{1}{1-x}\frac{1}{1-y}$. To construct such a W from walks in reverse, we start from the origin and walk anywhere in the plane for any number of steps. As we shall be handling the origin specially, we start with the last step which must be from either $(0,1)$ or $(1,0)$. The probability that the next step from either of these reaches the origin is $\frac{1}{2}p$, and therefore these two particular walks are given by the initial term $\frac{1}{2}pt(x+y)$. Given a walk w of length s starting at (m,n), we can construct a walk of length $s+1$ by prepending it with each of

the four possible steps. Ignoring the boundary conditions, this means we can generate a longer walk as $\frac{1}{2}pt(x+y)w + \frac{1}{2}qt(\bar{x}+\bar{y})w$ where $\bar{x} \equiv x^{-1}$. Note that $\frac{1}{2}ptyw$ corresponds to prepending a downward step with probability $\frac{1}{2}p$, but we increase the power of y because the power of y is the *starting* position. If w is of the form $\rho x^m t^s$ or $\rho y^n t^s$, then we replace respectively the steps $\frac{1}{2}qt\bar{y}$ or $\frac{1}{2}qt\bar{x}$ with $\frac{1}{2}pt$ (this is not a typo; to see that p is correct it is helpful to work through an example walk). Lastly we must ensure walks avoid the origin, or at least that we ignore those walks which prematurely reach the origin. We do so by 'trapping' such walks so that they get stuck there by excluding these walks from the $\frac{1}{2}pt(x+y)w$ transitions. Assembling these, we obtain the functional equation

$$\begin{aligned} W(x,y;t) = {} & \tfrac{1}{2}pt(x+y) && \text{(last step)} \\ & + \tfrac{1}{2}qt\bar{x}[W(x,y;t) - W(0,y;t)] + \tfrac{1}{2}ptW(0,y;t) && \text{(right step)} \\ & + \tfrac{1}{2}qt\bar{y}[W(x,y;t) - W(x,0;t)] + \tfrac{1}{2}ptW(x,0;t) && \text{(up step)} \\ & + \tfrac{1}{2}pt(x+y)[W(x,y;t) - W(0,0;t)], && \text{(left/down step)} \end{aligned} \tag{III.4}$$

which can be written more usefully in the form

$$\begin{aligned} \tfrac{2}{p}\bar{t}KW &= (x+y)(1-S) + (1-r\bar{y})X(x) + (1-r\bar{x})X(y) \\ K &= 1 - \tfrac{1}{2}pt(x + r\bar{x} + y + r\bar{y}) \end{aligned} \tag{III.5}$$

where $r = \frac{q}{p}$, $S = W(0,0;t)$, $X(x) = W(x,0;t) \equiv W(0,x;t)$, and $K \equiv K(x,y;t)$ is known as the kernel.

Orbital Sum Approach We now apply the orbital sum approach as detailed by Bousquet-Mélou and Mishna [82]. The group of the kernel is defined to be the set of maps $(x,y) \mapsto (x',y')$ that leave the kernel unchanged, with the group operation taken to be function composition, $\circ$. For this kernel it is generated by the orthogonal involutions $\chi : (x,y) \mapsto (r\bar{x},y)$ and $\upsilon : (x,y) \mapsto (x,r\bar{y})$, and hence is finite with order 4. Explicitly, the elements of the group are $G = \{1, \chi, \chi\upsilon, \upsilon\}$. We also define the 'sign' of each element, σ_γ, as $\sigma_1 = \sigma_{\chi\upsilon} = +1$ and $\sigma_\chi = \sigma_\upsilon = -1$. When finite, this group has the interesting property that the 'orbital sum' of a function of just x or y under this group vanishes, i.e. $\sum_{\gamma\in G} \sigma_\gamma \gamma(f(x)) = \sum_{\gamma\in G} \sigma_\gamma \gamma(f(y)) = 0$.

Looking at our functional equation III.5, all of the terms on the right hand side are functions of both x and y. However, by dividing through by $(1-r\bar{x})(1-r\bar{y})$ we can make two of these terms functions of only one of x and y. We need to be careful however, as generating functions have ring structure but don't generally admit an operation corresponding to division. Moreover, we are dividing through by series in negative powers of x and y, and so we need to consider the Laurent series structure. Specifically our series belong to the ring $\mathbb{R}(x,y)[\![t]\!]$ of formal power series in t whose coefficients are Laurent polynomials in x and y, and the series W belongs to the ring $\mathbb{R}[x,y][\![t]\!]$ of formal power series in t whose coefficients are polynomials in x and y. To perform the division, we observe that the series $-\bar{r}x(1-\bar{r}x)^{-1} = -\sum_{n=1}^{\infty}(\bar{r}x)^n$ annihilates $(1-r\bar{x})$ under multiplication and therefore is an appropriate multiplicative inverse to use.

Multiplying through then and taking the orbital sum, we find

$$\sum_{\gamma\in G}\sigma_\gamma\gamma\Big(\frac{\bar{r}x}{1-\bar{r}x}\frac{\bar{r}y}{1-\bar{r}y}\tfrac{2}{p}\bar{t}KW\Big)=(1-S)\sum_{\gamma\in G}\sigma_\gamma\gamma\Big((x+y)\frac{\bar{r}x}{1-\bar{r}x}\frac{\bar{r}y}{1-\bar{r}y}\Big).$$

Recalling that K is invariant with respect to the group, we can rewrite as

$$\sum_{\gamma\in G}\sigma_\gamma\gamma\Big(\frac{\bar{r}x}{1-\bar{r}x}\frac{\bar{r}y}{1-\bar{r}y}W\Big)=\tfrac{1}{2}pt(1-S)\underbrace{\frac{1}{K}\sum_{\gamma\in G}\sigma_\gamma\gamma\Big((x+y)\frac{\bar{r}x}{1-\bar{r}x}\frac{\bar{r}y}{1-\bar{r}y}\Big)}_{R(x,y;t)}.$$

To proceed, observe that $\frac{\bar{r}x}{1-\bar{r}x}\frac{\bar{r}y}{1-\bar{r}y}W$ remains part of the ring $\mathbb{R}[x,y][\![t]\!]$. Moreover, every term has a positive power in x and y. Therefore, the orbital sum yields the sum of series in the rings $\mathbb{R}[x,y][\![t]\!]$, $\mathbb{R}[\bar{x},y][\![t]\!]$, $\mathbb{R}[x,\bar{y}][\![t]\!]$ and $\mathbb{R}[\bar{x},\bar{y}][\![t]\!]$, and so if we extract the positive part consisting of only the terms with positive powers of x and y, then we can find an expression for W

$$\begin{aligned}\frac{\bar{r}x}{1-\bar{r}x}\frac{\bar{r}y}{1-\bar{r}y}W&=\tfrac{1}{2}pt(1-S)[x^>y^>]R\\ W&=\tfrac{1}{2}pt(1-r\bar{x})(1-r\bar{y})(1-S)R^>\end{aligned}$$

where $[x^>y^>]R\equiv R^>$ is the positive part of R. There is a limitation here: if W can be written as $f(x,y;t)+g(t)\frac{1}{1-x}\frac{1}{1-y}$ then the orbital sum of the g term in this scheme will vanish because $\frac{1}{1-x}\frac{\bar{r}x}{1-\bar{r}x}$ is invariant under the group, as is the y part. Moreover, any other functions which vanish under the orbital sum will also be excluded from our apparent W. This means, for example, that evaluating the above expression for $W(x,y;1)$ will yield 0 because $W(x,y;1)$ is in fact $\frac{1}{1-x}\frac{1}{1-y}$.

The MFPT starting from (m,n) is given by $\sum_s s\mathbb{P}(s|m,n)$, and its generating function is $T(x,y)=\sum_{mns}s\mathbb{P}(s|m,n)x^my^n$. It is easy to show, therefore, that $T(x,y)=\dot{W}(x,y;1)$ and so we find

$$T(x,y)=\tfrac{1}{2}p(1-r\bar{x})(1-r\bar{y})(-\dot{S}(1))R^>(x,y;1),$$

assuming that T has no terms which vanish under the orbital sum. It remains to find S, and to determine whether or not T has any such terms.

Obstinate Kernel Method Another approach is given by the obstinate kernel method, wherein we exploit roots of the kernel. A pair of such roots is given by $(x,Y_\pm(x))$ where $Y_\pm=B\pm\sqrt{B^2-r}$, $B=\frac{1}{pt}-\frac{1}{2}z$, and $z=x+r\bar{x}$ is a symmetric function of x and $\chi(x)\equiv r\bar{x}$. It will be helpful to note the elementary symmetric polynomials of $Y_\pm$ as $\frac{1}{2}(Y_++Y_-)=B$ and $Y_+Y_-=r$. Only one of these roots, Y_-, is a power series in t; the other includes a term in t^{-1}, and so we shall substitute Y_- into the functional equation III.5 to get

$$0=(x+Y_-)(1-S)+(1-Y_+)X(x)+(1-r\bar{x})X(Y_-).$$

To solve for $X(x)$, and thence $W(x,y)$, we eliminate $X(Y_-)$ by making use of the kernel's group; in particular applying χ yields a second equation in $X(Y_-)$,

$$0 = (r\bar{x} + Y_-)(1-S) + (1-Y_+)X(r\bar{x}) + (1-x)X(Y_-),$$

from which we find

$$Q(x) - Q(r\bar{x}) = (1-S)(r\bar{x}-x)\Big(\frac{1-Y_- - z}{1-Y_+}\Big)$$

with $Q(x) \equiv (1-x)X(x)$. Extracting the positive part then gives

$$Q(x) - S = (1-S)[x^>](r\bar{x}-x)\Big(\frac{1-Y_- - z}{1-Y_+}\Big).$$

From Q, X can be obtained and thereby W and $T = \dot{W}|_{t=1}$. Alternatively, the MFPTs for arbitrary positions can be computed using the recurrence relations and the boundary MFPTs $\dot{X}|_{t=1}$. In order to do so, we shall need to obtain an expression for S. Letting $U(t) = [x^1]X(x;t)$, we see that $[x^1](Q-S) = U - S$ and we compute $[x^0y^0]W$ using Equation (III.4) to find $(1-pt)S = qtU$. Therefore we can obtain an expression for S,

$$\frac{S}{1-S} = \frac{qt}{1-t}[x^1](r\bar{x}-x)\Big(\frac{1-Y_- - z}{1-Y_+}\Big).$$

Using the symmetric variable x helps to evaluate the positive parts of the antisymmetric series $(r\bar{x}-x)\big(\frac{1-Y_- - z}{1-Y_+}\big)$, but we were unable to proceed beyond this point.

III.6. Applicability to Other Architectures

Finally, we comment on the applicability of these results to non-Brownian reversible computers in the limit of vanishing free energy density. The concepts of mona and synchronisation phase space can be readily extended to other models of communicating reversible computer, but the time evolution of the phase space distribution may differ. In many architectures, the probability of a backward transition approaches 0. Nevertheless, it is still possible for the joint distribution of mona in phase space to expand over time. In particular, if the eigenstates of the Hamiltonian change over time due to thermal fluctuations, then the rate of computation at different *independent* loci will fluctuate and so synchronisation will be lost. Moreover, universal phenomena such as the Heisenberg uncertainty principle and general relativistic time dilation effects suggest that even a system at absolute zero would experience desynchronisation.

An important assumption in the above, and indeed this chapter, was that the mona evolve independently except during synchronisation events. As argued by Frank [86] in a thought experiment, however, it is possible to couple the mona in a system with a potential that increases with increasing desynchronisation. In particular he imagined a lattice of clockwork computational units driven by a series of motors, with adjacent

axles connected by bicycle chains. At low speeds $|v|$, the power supplied to counter frictional losses in the system would scale with $\sim N|v|^2$ where N is the number of mona, which recovers the $V^{5/6}$ scaling law. Any tension due to desynchronisation between adjacent mona would not substantially increase this power requirement, not least because adjacent axles experience equal and opposite tensions. Therefore the introduction of such potentials may be used to constrain the size of the joint phase distribution of adjacent mona; that is, mona remain locally in approximate synchronisation. Globally, remote mona may accumulate more significant desynchronisation, scaling with $\sqrt{RT}$ where R is linear dimension of the system and T the temperature, but this would not pose any problems to long-distance communication as long as message packets remain in local synchronisation at all points along their routes.

As a corollary, we extend Frank's thought experiment to a general conjecture about which classes of reversible computer are and are not subject to a synchronisation penalty:

Conjecture III.5 (Inter-monon synchronisation potentials are a necessary condition for cheap communication)**.** *By the above arguments, we conjecture that the joint phase space of independently evolving mona cannot remain localised, and hence that the synchronisation penalties derived in this chapter apply for all such reversible computer architectures. Conversely, mona whose evolution is coupled by a potential that increases with increasing desynchronisation will remain localised, and therefore can in principle pass through synchronisation windows with arbitrarily low penalty. If the distribution can be constrained to be compact, then the penalty may be eliminated.*

However, this is predicated on communicating mona being carefully programmed. Communicating mona must anticipate each other's computational states, such that they are both simultaneously (up to the width of the synchronisation window) receptive to communication. If a monon A attempts to send information to another monon B that is not expecting to receive such information, then this will incur an entropic synchronisation penalty by the mechanisms discussed in this chapter. Alternatively, a radically different approach to eliminating communication penalties is given by fully serial architectures, such as is *theoretically* permitted by a Quantum Zeno architecture (Section II.2).

An interesting example of an architecture where careful ordering is intrinsic to the computation itself is the BARC[10] (Ballistic Asynchronous Reversible Computing) model of Frank [87]. BARC systems consist of devices with an internal state, and wires between them along which pulses can travel. Devices are designed to interact with at most one pulse at a given time, the effect of which may be to change the device's internal state and to re-emit the pulse along a different wire. As the order of signals is important to the computation, any well-designed BARC computer will satisfy the synchronisation simultaneity condition. In this case, the synchronisation window depends on pulse spacing. Additionally it is an architecture without backwards transitions. However, the architecture—as presented—does not couple pulses to each other with a synchronisation potential, and so it is subject to the conclusions of this chapter as the ordering between

[10]BARC was historically referred to as ABRC, the Asynchronous Ballistic Reversible Computing model.

free signals is subject to change. Nevertheless, this desynchronisation can be kept impressively minimal through the high efficiencies and engineering tolerances afforded by superconducting circuits (see Frank et al. [88] for progress towards a Single Flux Quanta implementation of the BARC model).

Petri Nets Another model of asynchronous concurrency that is quite popular is that of Petri Nets (see [89] for a review). A Petri Net is a weighted directed graph with two types of nodes, 'places' and 'transitions'. A place contains tokens, and the configuration of tokens in all places—a 'marking'—represents the state of the system. Tokens can be consumed, generated, and moved by the activation of a transition. A transition node consumes some tokens from each place connected to it by an incoming edge, and deposits some tokens to each place connected by an outgoing edge. Edges may only be drawn between a transition and a place, and the weight of the edge dictates the number of tokens involved in the transition. A transition may only activate if there are enough input tokens.

Whilst a Petri Net is not necessarily reversible in the sense of having a corresponding inverse transition for every transition, it is always possible to construct an 'inverse Petri Net' by inverting each edge's direction. From this perspective, a Petri Net along with its sequence of transitions form a logically reversible system. It is not immediately obvious where entropy costs manifest in a Petri Net, and so it may appear that they offer a route to avoid the conclusions of this chapter. In order to accommodate entropy costs, the sequence of transitions needs closer attention. Without loss of generality, assume discrete time where at each time step a single[11] transition is attempted. If every transition is always successful, then there is no entropy increase. If, however, a transition may fail, then reversibility requires us to record a binary outcome of success or failure. That is, the input is a stream of symbols—each corresponding to a particular transition—and the output is a stream of symbol-outcome pairs. This is where entropy may increase, and is another example of a breakdown of simultaneity in synchronisation. Moreover, such a system—in which the transition order is sequential—does not allow for separate independent entities, let alone spatial structure. Therefore the notion of synchronisation/communication is not particularly relevant. To accommodate this, multiple transition streams can be introduced. For avoidance of conflict, the streams must satisfy the constraint that no two transitions sharing a place fire at the same time. Now the issues of synchronicity of signals and events become manifest: if these transition streams can be coupled to each other to maintain some local synchronisation then low cost communication can be supported, otherwise there will be a synchronisation penalty. Therefore, whilst Petri Nets are a convenient framework for modelling asynchronous systems, they are not particularly convenient for modelling the possibility of desynchronisation and the resultant resynchronisation entropy costs.

[11] Multiple transitions can occur simultaneously only if they do not share any places, but then they can always be assigned an arbitrary order and treated as sequential.

III.7. Conclusion

The primary result of this chapter is that communication between distinct reversible computers in a Brownian system (and arguably any reversible architecture whose mona evolve independently) with a limiting supply of free energy is very expensive in comparison to independent computation. A single computational step takes time $1/b\lambda$ for gross computational rate λ and bias $b = (\dot{G}/k_BT\lambda N)^{1/2}$, where $\dot{G}$ is the rate of supply of free energy, N is the number of mona, k_B is Boltzmann's constant, and T is the system's average temperature. In contrast, an interaction event such as communication takes at least a time $\sim 1/b^2\lambda$. As the bias scales as $\sim \sqrt{A/V} \sim R^{-1/2}$ for a system of convex bounding surface area A, internal volume V and radius R, communication will tend to 'freeze' out as the system gets bigger. This cost is unavoidable as it is due to the implicit erasure of information involved in synchronising divergent computational states; however, if synchronisation is rare then the temporal cost can be reduced. Suppose that the average proportion of mona wishing to communicate at a given time is $\nu \ll 1$, then the free energy can be divided into two supplies: $\dot{G}_{\text{indep.}}$ and $\dot{G}_{\text{synch.}}$ for independent computation and synchronisation events respectively, as intimated in Section II.7. A separate set of bias klona can then be introduced for the synchronising mona,

$$b_{\text{indep.}} = \sqrt{\frac{\dot{G}_{\text{indep.}}}{k_BT}\frac{1}{\lambda N}}, \qquad b_{\text{synch.}} = \sqrt{\frac{\dot{G}_{\text{synch.}}}{k_BT}\frac{1}{\lambda \nu N}}.$$

In order for the synchronisation to be viable we therefore require $1/b^2_{\text{synch.}} \lesssim 1/b_{\text{indep.}}$ and hence find

$$\nu \lesssim \frac{\dot{G}_{\text{synch.}}}{k_BT}\sqrt{\frac{k_BT}{\dot{G}_{\text{indep.}}}\frac{1}{\lambda N}}.$$

If we want to maintain our asymptotic independent computation rate of $\sim \sqrt{AV}$, then we will have $\dot{G}_{\text{synch.}} \sim \dot{G}_{\text{indep.}} \sim A$ and $N \sim V$, giving $\nu \lesssim \sqrt{A/V} \sim b$. That is, the number of synchronising mona νN can be as high as $\sim \sqrt{AV}$. In fact, this subpopulation of mona are also permitted to perform arbitrary operations subject to entropic costs, with the proviso that their individual net transition rates will be $b\lambda \sim R^{-1/2}$. The consequence is that, for a very large reversible Brownian computer, any parallel computation should be structured to minimise the need to synchronise the joint state of the system. Equivalently, the proportion of computational steps that are permitted to synchronise is $\lesssim b$. If one wishes to increase the proportion of synchronising mona, then the total computational rate must necessarily be reduced or an architecture satisfying Conjecture III.5 must be employed.

Impact on Algorithms The consequence of this 'freezing' out of synchronisation is that the most natural computational problems for these computers are those which are 'embarrasingly parallel'. An embarrasingly parallel problem is one which can be easily

divided into many tasks with very infrequent synchronisations between them. Some examples of such problems are Monte Carlo algorithms, brute force searches, and ray tracing. In contrast, highly serial problems, or parallel problems involving significant synchronisation, are not well accommodated. Serial programs suffer because the rate of computation by individual mona is asymptotically vanishing. Highly synchronous parallel programs suffer because of the time penalty for synchronisations. In limiting cases where no parallelisation is possible or where synchronisation cannot be made arbitrarily infrequent, the rate of these computations will fall to $\sim A$ as with irreversible computers. Nevertheless, as discussed in Section II.7, hybrid computational systems supporting both high-bias and low-bias subsystems can be constructed. Therefore serial/highly synchronous problems can be executed on a thin outer shell of the computer, and embarrassingly parallel problems can make use of the inner bulk of the computational matter.

A common example of highly synchronous parallel problems is that of simulating systems involving local interactions on a lattice, such as cellular automata, Ising models, and molecular dynamics simulations. Indeed, in Frank [9] (Section 6.3.2.1), he examines local computation by a three dimensional array of computing elements. For the class of non-Brownian computers under consideration, he finds an asymptotic benefit for reversible architectures over irreversible architectures. Unfortunately the results of this chapter show that problems like these are not well-supported by Brownian architectures, and so this asymptotic benefit is not seen in this case. However, the thought experiment of Frank [86] and Conjecture III.5 do support the realisation of such asymptotic benefit in architectures in which the computing elements are appropriately coupled.

This chapter concludes our analysis of the limits the laws of physics place on the sustained performance of reversible computers. Chapter II concerned aggregate performance in terms of computational operations per unit time, but neglected to consider interactions among computational sub-units or between computational sub-units and shared resources such as memory or chemical species. Chapter III extended the analysis to consider the former set of interactions. In this chapter we extend the analysis to consider the latter set, which in general will be governed by the laws of thermodynamics and statistical mechanics. When the free energy available to the computational subunits is limiting, this becomes a challenging endeavour as perturbations to these subsystems may have a corresponding entropy cost for which our available free energy is insufficient.

In the first half we pay particular attention to resource distribution: Our Brownian computational subunits—themselves being small and particulate—will likely lack adequate internal memory for many programs of interest; as such, it can be expected that a realistic Brownian system will make heavy use of a shared pool of memory resources that can be dynamically distributed amongst the computational subunits according to their need. It is found that most schemes imaginable fail to function effectively in the limit of vanishing 'computational bias' b, which measures the net fraction of transitions which are successful and which falls as the system grows in size.

Driving thermodynamically unfavourable reactions, such as resource distribution, is a very general problem for such systems and can be solved by supplying a sufficient excess of free energy. In order to make these interactions viable, we propose an abstract chemical scheme to dynamically infer how much free energy is needed, and to supply it by accumulating just enough from the (weak) ambient free energy. The scheme takes any reaction, arbitrarily far from equilibrium, and sequesters some of the reactants and products such that the extant reactants and products are in equilibrium. The scheme rapidly adjusts to changes in the equilibrium position, and makes use of molecular computation for its inference algorithm. Unfortunately there is a cost in applying this scheme: the reaction is slowed by a factor inversely proportional to the 'computational bias', $b \ll 1$ (the net proportion of computational transitions that are successful). In fact, this overhead is comparable to that found for synchronisation interactions in Chapter III.

IV. Performance Trade-offs for Reversible Computers Sharing Resources

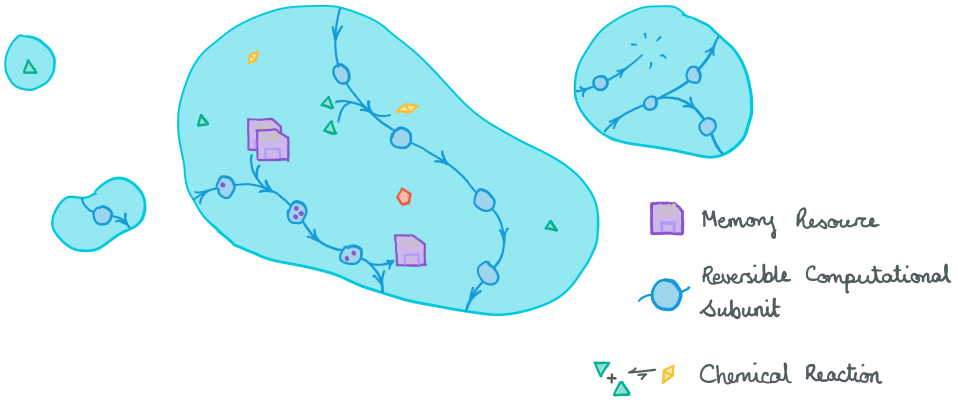


Figure IV.1: This chapter concerns interactions between reversible Brownian computational particles and populations of other particles such as species involved in chemical reactions or shared memory resources. Populations of particles are subject to well known statistical mechanical laws, and these laws can cause problems when computational particles attempt to interact with these populations.

IV.1. Introduction

This is the final chapter investigating the influence of the laws of physics on computation, particularly as they apply to large reversible Brownian computers (see Definition III.2). As discussed in Chapter III, the net computational rate (in terms of primitive computational operations per unit time) studied in Chapter II is not the only performance metric of interest. Therefore, in Chapter III we evaluated computational performance of reversible Brownian computers (and argued for the applicability of the results to other reversible computers whose computational units evolve independently) whose individual computational components were permitted to interact with one another, i.e. to communicate and to perform concurrent/parallel computation. A key property of these interactions is that computational 'particles' can be uniquely referenced, and therefore the traditional thermodynamic principles that apply to chemical systems were inapplicable. In this chapter, we now consider the interface between traditional chemical

species, of which there may be a variable number of identical units or particles, and the computational particles. See Figure IV.1 for an illustration. For convenience, we shall refer to these as *klona* and *mona* respectively, following the formulation introduced in Definition III.1. Recalling the definitions, we call particles which are uniquely distinguishable or addressable *mona* (sg. *monon*), after the Greek for unique, μοναδικός. Similarly, we call indistinct particles belonging to a species *klona* (sg. *klonon*). Klona may be likened to traditional chemical species such as glucose and ATP. Mona are less obvious; an example would be a 150 mL solution containing a 1 mmol dm^{-3} concentration of random nucleic acids of length 80 nucleotides.

Interactions between mona and klona are of great practical importance for Brownian computers; whilst mona by themselves may be capable of arbitrary computation, this computation only becomes useful when used to actuate some other system or when it otherwise produces some observable output. Examples may include the activation of a fluorophore upon reaching some logical condition, or directing the translation of a particular messenger RNA into a protein. Moreover, interaction with klona may be essential for computation: general computation requires access to an unbounded amount of memory, but by their very nature mona are limited in size and thus memory. Therefore it is important for mona to be able to recruit more memory as required, and release it for use by other mona when the need passes. Resources other than memory may also be considered, such as structural monomers for use in constructing and deconstructing molecular machines or computers.

In this chapter we shall be most interested in the case of resource distribution, and will consider various such schemes in Section IV.2. One approach is a centralised store which mona can access, either by travelling there themselves or by sending a 'courier' monon on their behalf to fetch and retrieve (or carry and return) a resource. This approach is undesirable. If the couriers float freely in solution, then reaching the central store and returning will both take a time $\sim \mathcal{O}(V)$; if on the other hand we introduce a lattice as in Chapter III, then the average distance will be proportional to the radius R and, given $b \sim R^{-1/2}$, the average time will be $\sim \mathcal{O}(b^{-3})$. As the time for a single net computational step is $\sim \mathcal{O}(b^{-1})$, both of these are untenable. Instead we shall seek a decentralised approach by making use of klona; intuitively it must become harder to access a resource as its scarcity increases, but we would like the timescale to do so to not be substantially more costly than $\sim (b[\mathrm{X}])^{-1}$ where $[\mathrm{X}]$ is the available concentration of the resource.

We will unfortunately find that the reactions involved in all possible schemes are unfavourable except in a very limited range of resource availability. In Section IV.3 we shall return to considering general mona-klona interactions, and demonstrate how mona can dynamically drive any unfavourable reaction with klona by inferring the number of bias tokens required to balance the entropic cost of the reaction. Alas, there will turn out to be an overhead in achieving this even in the case when the driven reaction is at equilibrium and hence cost-free.

IV.2. Resource Distribution Schemes

A resource distribution scheme for a resource X is a set of klona from which X can be extracted and into which it can be deposited. Formally, resources are conserved discrete quantities analogous to charges in physics. Letting the set of resource species be indexed by $\mathcal{R}$ and the set of resource klona by Σ, then to each klonon $\sigma \in \Sigma$ is associated a resource charge $\vec{q}_\sigma \in \mathbb{N}^{\mathcal{R}}$: a vector over the field of naturals and indexed by the resource species. In any reaction, the net resource charge must be conserved; for example, for the reaction $|n\rangle + \sum_{\sigma\in\Sigma} \nu_\sigma \sigma \rightleftharpoons |n+1\rangle + \sum_{\sigma\in\Sigma} \nu'_\sigma \sigma$ involving the transition of a monon between states $|n\rangle$ and $|n+1\rangle$ and the interaction of said monon with resource klona according to the stoichiometries $\vec{\nu} \rightleftharpoons \vec{\nu}'$, this conservation law can be expressed as $\vec{q}_{|n+1\rangle} - \vec{q}_{|n\rangle} = \sum_{\sigma\in\Sigma}(\nu_\sigma - \nu'_\sigma)\vec{q}_\sigma$.

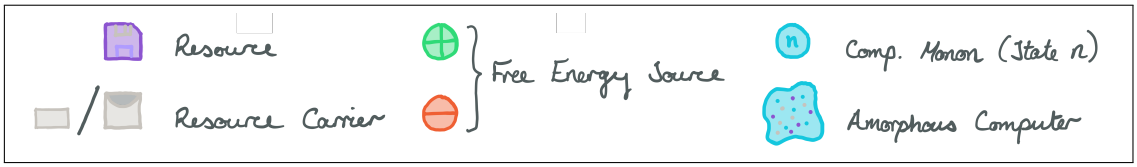
Free Klona The simplest possible scheme is that in which the resource is freely present in solution, as shown in Figure IV.2b. That is, a system with $\mathcal{R} = \{\mathrm{X}\}$, $\Sigma = \{\mathrm{X}\}$ and $(\vec{q}_\mathrm{X})_\mathrm{X} = 1$. This system will be a viable resource pool so long as the free energy change involved in both extraction and deposition of X is less than the available computational free energy, $\log \frac{p}{q} = 2\,\mathrm{arctanh}\, b$. Note that our notion of 'free energy' uses a different convention than is normally taken, namely we are giving the change in information entropy of the universe Δh, whereas normally the free energy change is expressed as $\Delta G = -kT\Delta h$. The free energy change vanishes when the system is in equilibrium, which for X an ideal gas can be calculated (via the Sackur-Tetrode equation) to occur at a concentration

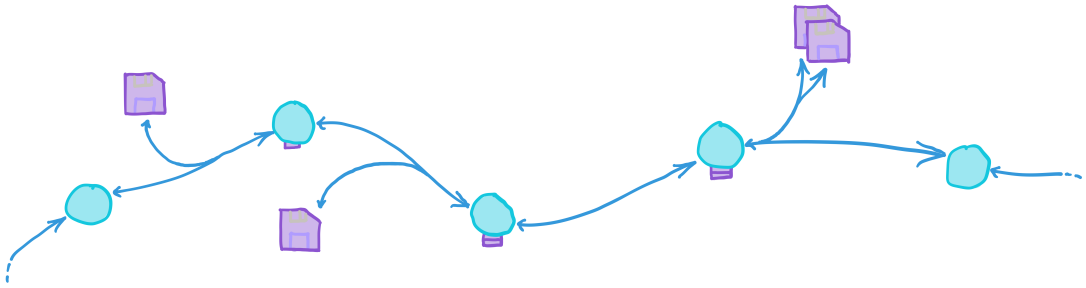
$$[\mathrm{X}]_{(\mathrm{eq.})} = e^{5/2}\Big(\frac{4\pi m u}{3h_P^2}\Big)^{3/2},$$

where h_P is Planck's constant, m is the mass of X, and u is its average internal energy. As we interact with the system, the concentration will be drawn away from this equilibrium position and as a result the absolute free energy difference will increase. For this scheme, the free energy difference corresponds to that due to adding or removing a particle of X and can therefore be identified with the *Chemical Potential*, μ_X. In units of information, this can be expressed as $\mu_\mathrm{X} = \log \frac{[\mathrm{X}]\gamma_\mathrm{X}}{[\mathrm{X}]\gamma_\mathrm{X}|_{(\mathrm{eq.})}}$ where γ_X is the activity coefficient of X that quantifies the deviation of the properties of X from an 'ideal' substance. If Q is the concentration of resource X (here trivially equal to $[\mathrm{X}]$) then we can now determine how far from equilibrium the resource system can be brought, and therefore how much of the resource Q is accessible to the computational mona. For a small change δQ, we find

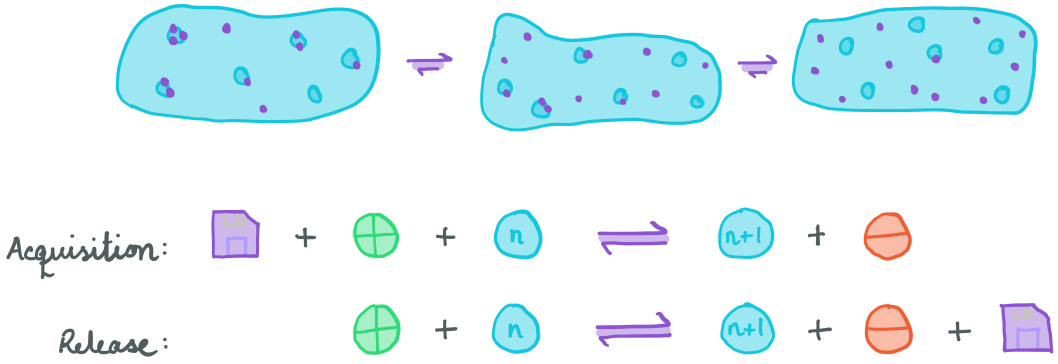
$$\Big|\frac{\delta Q}{Q}\Big| < 2b\Big|1 + \frac{\partial \log \gamma_\mathrm{X}}{\partial \log Q}\Big|^{-1} + \mathcal{O}(b^2)\,.$$

We don't expect the activity coefficient to depend strongly on Q, and certainly not to the extent of $\frac{\partial \log \gamma}{\partial \log Q} = -1$, which would imply a range of isentropic equilibria, and therefore we can only usefully access a proportion $\mathcal{O}(b) \ll 1$ of the available resource X under this scheme.



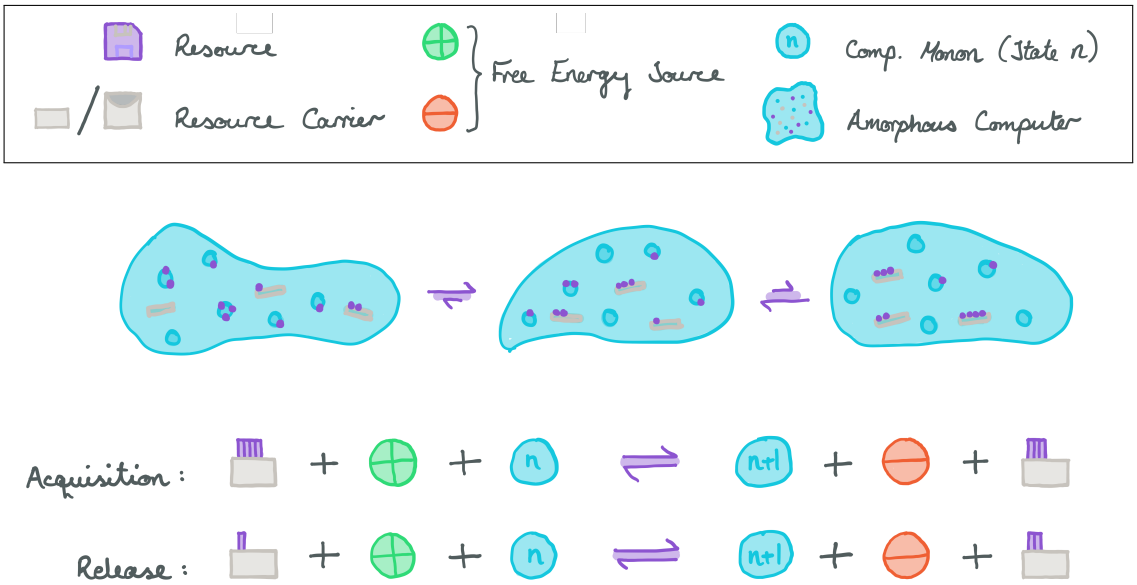


(a) A possible state diagram for a monon interacting with a resource pool; the particular resource distribution scheme is left abstract.

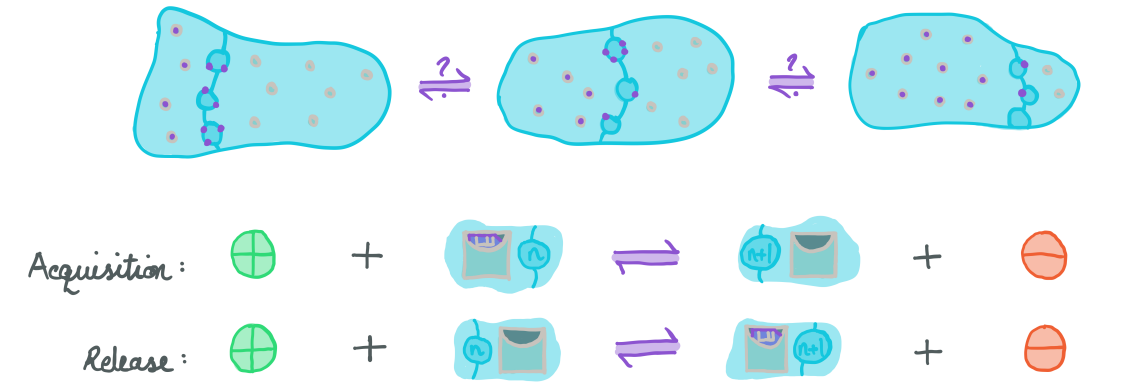


(b) An illustration of the statistical states and reactions for the Free Klona resource distribution scheme.

Figure IV.2: An assortment of illustrative examples of some of the resource distribution schemes for a Brownian amorphous computer as considered in this chapter.



(c) An illustration of the statistical states and reactions for the Bounded Carrier resource distribution scheme.



(d) An illustration of the statistical states and reactions for the Isobaric Compartment resource distribution scheme. The question marks serve to indicate that the apparent isergonicity of the resource reactions is in question, and as we find, is unfortunately not the case.

Figure IV.2: (continued) An assortment of illustrative examples of some of the resource distribution schemes for a Brownian amorphous computer as considered in this chapter.

Bounded Carriers Perhaps the primary issue with free klona is that there is a change in particle number, implying a substantial entropy change when resources are traded with the pool. A possible improvement is suggested in ensuring total particle number remains constant by binding the resource to carrier klona, as shown in Figure IV.2c. This may manifest, for example, as $\Sigma = \{\mathrm{X}^0, X^1\}$ with $(\vec{q}_{\mathrm{X}^0})_{\mathrm{X}} = 0$ and $(\vec{q}_{\mathrm{X}^1})_{\mathrm{X}} = 1$. More generally, we have polymeric klona $\Sigma = \{\mathrm{X}^k : k \in [0, n] \cap \mathbb{N}\}$ with $(\vec{q}_{\mathrm{X}^k})_{\mathrm{X}} = k$ for some integer $n > 1$. Once again, we shall consider a perturbation to the equilibrium distribution of the pool; it will be convenient to introduce a partition function $\mathcal{Z}(\beta)$ where the thermodynamic variable β correlates with the amount of resource stored by the pool. The partition function will take the form $\mathcal{Z} = \sum_{k=0}^{n} e^{-\varepsilon_k - \beta k}$ where ε_k contains all the thermodynamically pertinent information about X^k klona, such as its energy and internal degrees of freedom. In fact, we will take $\varepsilon_k = 0$ as any k-dependence of the ε_k would only serve to reduce the effective value of n by narrowing the distribution[1].

The free energy change for a reaction is given by the logarithm of the ratios of reaction rates (see, e.g., Section II.4 for a derivation). Therefore in this case it is given by $\Delta h = \log \sum_{k=1}^{n} e^{-\beta k} - \log \sum_{k=0}^{n-1} e^{-\beta k}$, and so at equilibrium (when $\Delta h = 0$) we find $\beta = 0$. For a perturbation from equilibrium, this reduces (exactly) to $\delta \Delta h = -\delta\beta$. Now, the average X-charge of each klonon is given by $\bar{k} = -\partial_\beta \log \mathcal{Z}$, and the charge density by $Q = \sum_{k=0}^{n} k[\mathrm{X}^k] \equiv \bar{k}[\mathrm{X}]$. The proportional resource accessibility is then given by

$$\left|\frac{\delta Q}{Q}\right| = \left|\frac{-\delta\beta \operatorname{var} k}{\bar{k}}\right| < \tfrac{1}{3} b(n+2)$$

using $\bar{k} = \frac{1}{2}n$ and $\operatorname{var} k = \frac{1}{12}n(n+2)$. This suggests taking $n \gtrsim b^{-1}$ to ensure full resource availability. Unfortunately, doing so would mean the total system charge scales as $V/b \sim V^{7/6}$ which clearly cannot be sustained in the limit of large systems. One potential resolution is to reduce the carrier concentration to $[\mathrm{X}] \sim b^{1/2}$ and then take $n \sim b^{-1/2}$ so that the total charge scales with V; whilst this would slow down resource reactions commensurately, mona could use this additional time to accumulate $\sim b^{-1/2}$ tokens in order to raise their free energy budget to $\Delta h \lesssim b^{1/2}$, thereby obtaining a resource accessibility of $|\delta Q/Q| \sim 1$ at a time penalty of $\sim b^{-3/2}$.

Unbounded Carriers It is also possible to consider polymeric klona of unbounded size, i.e. the limit $n \to \infty$, in which case the partition function is given by $\mathcal{Z} = (1 - e^{-\beta})^{-1}$. It should be noted that in this limit there is no equilibrium distribution of the resources, as $\mathcal{Z}$ diverges as $\beta \to 0$. In fact, distributions only exist for $\beta > 0$ whilst the free energy constraint bounds it from above and hence $0 < \beta < 2b$. This range corresponds to $\bar{k} > (2b)^{-2} - \mathcal{O}(1)$ and therefore a total enclosed charge $\gtrsim V^{4/3}$; for lower densities, the only favourable reaction is resource deposition. Therefore, whilst conceptually simpler, unbounded carriers are to be avoided.

[1] Assuming that the equilibrium distribution is unimodal, this corresponds to a reduction in variance. If the distribution is multimodal then instead there is a reduced 'local' variance about each mode.

Isobaric Compartments The problem inherent to seemingly all schemes is that a resource reaction must necessarily reduce the quantity of reactants and increase the quantity of products, and this will in turn bias the resource reactions in the opposite direction per *Le Chatelier*'s principle. As our last attempt to circumvent these issues, we introduce a scheme exploiting varying partial volumes in order to maintain equal concentrations of reactants and products. This 'Isobaric Compartment' scheme is illustrated in Figure IV.2d and consists of dividing the system into two compartments by means of a massless and infinitely flexible membrane, impermeable to the carrier species X^0 and X^1. Referring to the two compartments by $\Gamma_\oplus$ and $\Gamma_\ominus$, we then establish the invariants that $\Gamma_\oplus$ only contains X^1 and $\Gamma_\ominus$ only contains X^0. A monon wishing to exchange resources with the pool must first bind to the membrane, and then to maintain the invariant it will translocate the carrier species between the two compartments. At equilibrium, the concentrations of carrier species in each compartment must be equal and thus it would appear that we have been successful in eliminating the free energy cost for this mona-klona interaction.

Unfortunately, whilst the carrier species concentrations are equal, concentration is not the correct quantity for determining the reaction rates and thence the free energy difference. From a collision theory perspective, the reaction rates are given by the frequency with which mona and klona collide (in the correct orientation and conformation). Consider the case where all but one carrier reside within $\Gamma_\oplus$: if the total number of carriers is N then the mean volume of $\Gamma_\ominus$ will be V/N and the mean concentration $1/(V/N) = N/V$. This would suggest that a monon bound to the interfacial membrane will encounter the X^0 klonon as often as it does an X^1 klonon, but this clearly cannot be the case. If we imagine the X^0 klonon as residing within a small bubble of volume V/N, then whilst its local concentration is indeed the same as for the X^1 klona, the bubble itself is free to migrate anywhere along the membrane. Consequently, in a system in which particles can bind to the boundary of the system in addition to exploring their volume, the effective concentration becomes $N/(\alpha + \gamma)$ where α is the effective 'volume' of the boundary (proportional to its area) and γ is the enclosed conventional volume. For our isobaric system, $\gamma_\oplus = N_\oplus \gamma_0$ (and similarly for $\gamma_\ominus$) where the constant $\gamma_0 = V/(N_\oplus + N_\ominus)$ is the volume available to each particle. We can therefore write $[\mathrm{X}^0]_{\text{eff.}} = [\mathrm{X}^0]/(\alpha/V + [\mathrm{X}^0]\gamma_0)$ where $[\mathrm{X}^0] = N(\mathrm{X}^0)/V$ is the 'true' concentration; redefining terms, we let $[\mathrm{X}^0]_{\text{eff.}} = [\mathrm{X}^0]/(\alpha + [\mathrm{X}^0]\gamma)$.

The system can be generalised to carriers for $n > 1$ by letting the membrane be semipermeable to $\{\mathrm{X}^k : 0 < k < n\}$, as these klona can participate as both reactants and products in either reaction. We can then write $\mathcal{Z}_\ominus = \sum_{k=0}^{n-1} e^{-\beta_\ominus k}$ for the partition function of $\Gamma_\ominus$ and similarly for $\Gamma_\oplus$. The effective concentrations of X^k in each compartment must be equal for $0 < k < n$, i.e. $(e^{-\beta_\ominus k}/\mathcal{Z}_\ominus)[\mathrm{X}]_{\ominus\text{eff.}} = (e^{-\beta_\oplus k}/\mathcal{Z}_\oplus)[\mathrm{X}]_{\oplus\text{eff.}}$. As this must apply for each $0 < k < n$, $(\beta_\oplus - \beta_\ominus)k$ must be constant with respect to k and therefore $\beta_\oplus = \beta_\ominus = \beta$. Moreover, $[\mathrm{X}]_{\oplus\text{eff.}}/[\mathrm{X}]_{\ominus\text{eff.}} = \mathcal{Z}_\oplus/\mathcal{Z}_\ominus = e^{-\beta}$. Equilibrium occurs once again for $\beta = 0$ with corresponding 'true' concentrations $[\mathrm{X}]_\oplus = [\mathrm{X}]_\ominus = \frac{1}{2}[\mathrm{X}]_{\text{total}}$. The

free energy difference is then given by

$$\delta\Delta h = -\delta\beta = \frac{\delta[\mathrm{X}]_\oplus}{\frac{1}{2}[\mathrm{X}]}\frac{2\alpha}{\alpha + \frac{1}{2}\gamma[\mathrm{X}]}.$$

Finally, the resource charge is $Q = \bar{k}_\oplus[\mathrm{X}]_\oplus + \bar{k}_\ominus[\mathrm{X}]_\ominus$ and hence the proportional availability is given by

$$\begin{aligned}\Big|\frac{\delta Q}{Q}\Big| &= \Big|\frac{-\delta\beta\frac{1}{2}[\mathrm{X}](\mathrm{var}_\oplus\, k + \mathrm{var}_\ominus\, k) + \delta[\mathrm{X}]_\oplus}{\frac{1}{2}n[\mathrm{X}]}\Big| \\ &< 2b\Big(\frac{\frac{1}{6}(n^2-1)}{n} + \frac{\alpha + \frac{1}{2}\gamma[\mathrm{X}]}{2\alpha n}\Big) \\ &< b(\tfrac{1}{3}n + \tfrac{1}{6n} + \tfrac{1}{4\alpha n}),\end{aligned}$$

recalling that $\gamma \equiv 1/[\mathrm{X}]$.

As with the Bounded Carrier scheme we can let $n \sim 1/b$ to ensure full resource availability, but we can also choose to let $\alpha \sim b$ with $n = 1$. The constant α is proportional to the ratio of the membrane area to the system volume. A naive implementation would of course have $\alpha \sim b^2$ which, whilst ensuring full resource availability, would limit the number of mona able to interact with the resource scheme proportional to A. Instead, consider replacing the system's volume with a space-filling tubule (or perhaps a tubule-lattice) and partitioning the tubule along its length; this system is still an instance of the Isobaric Compartment scheme, but allows an α as large as ~ 1 (if we wanted to allow all mona to interact with the resource system simultaneously). By making the tubule diameter $\sim b^{-1}$, a value of $\alpha \sim b$ can be achieved in which case the proportion of mona able to simultaneously interact with the resource system will be $\sim V^{5/6}$. This is, however, a time penalty of $\sim b^{-2}$ and hence is strictly worse than the Bounded Carrier scheme.

IV.3. Driving Unfavourable Reactions

We conjecture that the results in the previous section are general; that is, no resource scheme can achieve better resource availability under the same constraints. In order to make use of a greater proportion of the resource pool, more free energy must be supplied and, ideally, the amount of free energy supplied should be the minimum necessary to avoid wasting the supply of negentropy. As alluded to in the introduction, we shall now consider general mona-klona interactions of arbitrary free energy cost.

Reactions of Known or Bounded Cost Consider an arbitrary reaction $\sum_i \nu_i \mathrm{X}_i \rightleftharpoons \sum_i \nu'_i \mathrm{X}_i$ with forward reaction rate α and backward rate β. Recalling that the free energy change for the forward reaction is given by $\Delta h = \log\frac{\alpha}{\beta}$, it can be seen that $n > -\Delta h/2\,\mathrm{arctanh}\, b$ computational bias tokens are needed to drive the reaction forward in the case $\Delta h < 0$; similarly, if $\Delta h > 0$ then $n > \Delta h/2\,\mathrm{arctanh}\, b$ tokens are needed to

drive the reaction backward. In addition, if we wish to couple such an unfavourable reaction to our computational mona then we should ensure by some means that the reaction does not occur in uncoupled isolation, such as raising the activation energy of the reaction or by employing a reaction mechanism that requires interaction with the mona. This kind of coupling abounds in nature, wherein a variety of bias systems are exploited. In fact, the $\mathrm{ATP} : \mathrm{ADP} + \mathrm{P}_i$ bias system is itself generated from a number of other bias sources. One particular example involves the conversion of the free energy of a transmembrane proton gradient for which $n \gtrsim 4$; the molecular machine responsible for this coupling, ATP synthase, is a molecular motor which translocates 11 protons[2] across the membrane per revolution, in the process converting 3 molecules of ADP to ATP, an entropically unfavourable process.

When n is known and fixed (or, at least, bounded from above) an optimal approach is provided by the simple scheme that performs n transitions between each unfavourable reaction. Indexing the mona states, M_k, modulo $n+1$, the scheme can be written thus

$$\mathrm{M}_0 \underset{q\lambda}{\overset{p\lambda}{\rightleftharpoons}} \cdots \underset{q\lambda}{\overset{p\lambda}{\rightleftharpoons}} \mathrm{M}_k \underset{q\lambda}{\overset{p\lambda}{\rightleftharpoons}} \cdots \underset{q\lambda}{\overset{p\lambda}{\rightleftharpoons}} \mathrm{M}_n,$$
$$\mathrm{M}_n + \sum_i \nu_i \mathrm{X}_i \underset{\beta'}{\overset{\alpha'}{\rightleftharpoons}} \mathrm{M}_0 + \sum_i \nu'_i \mathrm{X}_i.$$

The reactions between M_k and M_{k+1} serve to increase the ratio $[\mathrm{M}_n]/[\mathrm{M}_0]$ such that it counteracts the ratio $\alpha'/\beta' = \alpha/\beta < 1$. To solve for the steady state dynamics of this system, we write the currents

$$S(\mathrm{M}_k \mapsto \mathrm{M}_{k+1}) = p\lambda[\mathrm{M}_k] - q\lambda[\mathrm{M}_{k+1}]$$
$$S(\mathrm{M}_n \mapsto \mathrm{M}_0) = \alpha'[\mathrm{M}_n] - \beta'[\mathrm{M}_0]$$

and use the fact that, at steady state, the currents must be equal. Moreover, the current will correspond to the net rate at which the target reaction is driven, and therefore we can find the optimal choice of $n > |\Delta h/2 \operatorname{arctanh} b|$ to maximise this rate (as for excessively large n, the chain of intermediate monon states will dilute the proportion of transitions performing the desired reaction).

We first show that the system makes optimal use of the consumed bias tokens by calculating the value of n for which the current vanishes, and hence for which the reaction is exactly balanced. For vanishing current, the principle of detailed balance applies and the steady state distribution is obtained simply by $[\mathrm{M}_k]/[\mathrm{M}_0] = (\frac{p}{q})^k$ and $[\mathrm{M}_0]/[\mathrm{M}_n] = \alpha/\beta$. This yields $n = \log \frac{\beta}{\alpha} / \log \frac{p}{q} = -\Delta h/2 \operatorname{arctanh} b$, precisely the threshold value of n. The attentive reader may point out that this value of n is not necessarily integral, and therefore a cyclic chain of length $n+1$ may not exist; this is true, but we shall be seeking larger values of n to increase the rate of the reaction and for which picking the nearest integral value shall be acceptable. Alternatively, one could consider the superposition of two such chains for $\lfloor n \rfloor$ and $\lceil n \rceil$ in the appropriate proportions.

[2] The exact number varies between species, and even between different organelles of the same species, but is typically between 10 and 14.

To solve for non-vanishing current, we perform two telescopic sums

$$nS = \sum_{k=0}^{n-1} S(\mathrm{M}_k \mapsto \mathrm{M}_{k+1}) \qquad \frac{1}{p}S\sum_{k=0}^{n-1} t^{-k} = \sum_{k=0}^{n-1}(t^{-k}\lambda[\mathrm{M}_k] - t^{-k-1}\lambda[\mathrm{M}_{k+1}])$$
$$= b\lambda[\mathrm{M}] + q\lambda[\mathrm{M}_0] - p\lambda[\mathrm{M}_n], \qquad S\frac{p^n - q^n}{p-q} = p^n\lambda[\mathrm{M}_0] - q^n\lambda[\mathrm{M}_n],$$

where $t \equiv \frac{p}{q}$, and then substitute $S(\mathrm{M}_n \mapsto \mathrm{M}_0)$ for the current:

$$b\lambda[\mathrm{M}] = (p\lambda + n\alpha')[\mathrm{M}_n] - (q\lambda + n\beta')[\mathrm{M}_0],$$
$$\Big(q^n\lambda + \frac{p^n - q^n}{p-q}\alpha'\Big)[\mathrm{M}_n] = \Big(p^n\lambda + \frac{p^n - q^n}{p-q}\beta'\Big)[\mathrm{M}_0].$$

Hence, solving for $[\mathrm{M}_0]$ and $[\mathrm{M}_n]$, we can obtain the steady state current thus:

$$S = b\lambda[\mathrm{M}]\frac{q^n\beta' - p^n\alpha'}{(q^{n+1} - p^{n+1})\lambda + n(q^n\beta' - p^n\alpha') + (q\alpha' - p\beta')\frac{1}{b}(p^n - q^n)}.$$

After some rearrangement, we find that the current is maximised when

$$(t^{n/2} - t^{-n/2}\gamma)^2 = \frac{q\log t}{b}(\gamma - 1)(t\gamma - 1) + \frac{q\lambda\log t}{\alpha}(t\gamma + 1)$$

where $\gamma \equiv \beta/\alpha$. For small bias, this reduces to $n = 2n_0 + \mathcal{O}(b)$ where $n_0 = |\Delta h|/2\operatorname{arctanh} b$ is the threshold value. When n is large, this gives a current $S \to [\mathrm{M}]\min(2b^2\lambda\frac{\alpha}{\beta}, 2b\alpha)$. This assumes that the intermediate monon transitions are not useful; if the intermediate transitions in fact perform useful work then this current can be multiplied by n and, in the limit of large n, the current tends to $b\lambda$: the net transition rate for uncoupled mona.

Reactions of Unknown or Unbounded Cost The preceding scheme suffers from a number of shortcomings due to the fact that n is fixed; when n_0 is lower than that designed for, the system expends more bias tokens than necessary, whereas when it is greater the system will stall. Moreover the system is asymmetric with respect to direction of the driven reaction: if $\beta > \alpha$ then the forward reaction should be driven with n tokens whilst the reverse reaction needs no additional bias, but this complicates the act of running a subroutine in reverse as each bias chain for the forward reaction must be removed, and a bias chain must be introduced for each instance of the reverse reaction. This is further complicated if the intermediate transitions along the chain perform useful computation. Whilst these problems are not insurmountable, it is possible to solve them all with a revised scheme in which n is permitted to vary.

A scheme with variable n is incompatible with the specific implementation of that for fixed n, and so we first migrate the mona chains into a distinct system of klona consisting of the species $\{\mathrm{X}_k^{(n)} : n, k \in \mathbb{N} \wedge 0 \le k \le n\} \cup \{\mathrm{X}_{-k}^{(-n)} : n, k \in \mathbb{N} \wedge 0 \le k \le n\}$ and subject to reactions $\mathrm{X}_k^{(n)} \xrightarrow[\overleftarrow{q}]{p} \mathrm{X}_{k+1}^{(n)}$, with steady state $[\mathrm{X}_k^{(n)}]/[\mathrm{X}_0^{(n)}] = (\frac{p}{q})^k$. To

demonstrate practicability, an abstract molecular realisation is provided in Section IV.4. The idea is to sequester some of the reactants and products in an inactive state, with those remaining in an active state being at equilibrium (for an isenthalpic reaction, this manifests as equal concentrations). Representing the reactants by A and the products by B, this effective sequestration behaviour is achieved by the reaction $\mathrm{M}_i + \mathrm{A{:}X}_{n_0}^{(n_0)} \xrightleftharpoons[q]{p} \mathrm{M}_{i+1} + \mathrm{B{:}X}_0^{(n_0)}$ with $n_0 = -\Delta h/2 \operatorname{arctanh} b$. For fixed n_0, this scheme already solves all the aforementioned problems as the complexed system $\mathrm{A{:}X}_{n_0}^{(n_0)} \rightleftharpoons \mathrm{B{:}X}_0^{(n_0)}$ is an equilibrated abstraction over the underlying $\mathrm{A} \rightleftharpoons \mathrm{B}$ reaction, and hence can be coupled directly to the computational mona system.

In the course of system operation, the value of n_0 will almost certainly vary. Introducing the sequestration klona for all values of $n \in \mathbb{Z}$, and letting $\alpha' \equiv \alpha \sum[\mathrm{X}_n^{(n)}]$ and $\beta' \equiv \beta \sum[\mathrm{X}_0^{(n)}]$, we see that the desired equilibrium condition is given by $\alpha' = \beta'$. Therefore, if the values of α' and β' deviate from equilibrium then we should counteract this deviation by perturbing the n-distribution of the sequestration klona. Ideal kinetics are attained if this n-distribution converges to $\alpha' = \beta'$ exponentially fast, i.e. $\dot{n} \propto \beta' - \alpha'$. This target kinetics arises because $[\mathrm{X}_n^{(n)}] = (\frac{p}{q})^n [\mathrm{X}_0^{(n)}]$ and $\frac{p}{q} > 1$, hence increasing n increases α'/β'. A first attempt at implementation might be given by $\forall n \in \mathbb{Z}.\mathrm{M}_i + \mathrm{A{:}X}_n^{(n)} \xrightleftharpoons[q]{p} \mathrm{M}_{i+1} + \mathrm{B{:}X}_0^{(n-1)}$, but this doesn't quite replicate the desired kinetics. Instead we separate the concerns of the computational bias and the reaction coupling thus:

$$\left.\begin{array}{l} \mathrm{M}_i \xrightleftharpoons[p]{p} \mathrm{M}_{i+\frac{1}{3}} \\ \mathrm{M}_i \xrightarrow[q]{p} \mathrm{M}_{i+\frac{1}{3}} \end{array}\right\} \quad \mathrm{M}_{i+\frac{1}{3}} + \mathrm{A{:}X}_n^{(n)} \rightleftharpoons \mathrm{M}_{i+\frac{2}{3}} + \mathrm{B{:}X}_0^{(n-1)} \quad \left\{\begin{array}{l} \mathrm{M}_{i+\frac{2}{3}} \xrightleftharpoons[q]{q} \mathrm{M}_{i+1} \\ \mathrm{M}_{i+\frac{2}{3}} \xrightarrow[q]{p} \mathrm{M}_{i+1} \end{array}\right. \tag{IV.1}$$

Assuming that at steady state $[\mathrm{M}_{i+\frac{1}{3}}] = [\mathrm{M}_{i+\frac{2}{3}}] \equiv \mu$, the coupled reaction then replicates the kinetics $\dot{n} = k\mu(\beta' - \alpha')$. For symmetry, we have introduced two intermediate mona with their inter-monon transitions used for coupling to the computational bias. In order to ensure that only a single token's worth of bias is consumed in the reaction, we also introduce two leak reactions such that the total free energy change is precisely that for a single token: $\Delta h = \log \frac{p+p}{q+p} + \log \frac{p+q}{q+q} = 2 \operatorname{arctanh} b$.

To verify that this system has the desired properties, we now solve for the steady state distribution. Employing detailed balance, we find for each n that $[\mathrm{X}_n^{(n)}]/[\mathrm{X}_n^{(n)}]$

$$\frac{[\mathrm{X}_n^{(n)}]}{[\mathrm{X}_0^{(n-1)}]} = \frac{\beta}{\alpha} \equiv \left(\frac{p}{q}\right)^{n_0} \qquad \Longrightarrow \qquad \frac{[\mathrm{X}_0^{(n)}]}{[\mathrm{X}_0^{(0)}]} = \left(\frac{p}{q}\right)^{-\frac{1}{2}(n+\frac{1}{2}-n_0)^2 + \frac{1}{2}(\frac{1}{2}-n_0)^2},$$

where we have used the intra-$\mathrm{X}_k^{(n)}$ steady state distribution, $[\mathrm{X}_k^{(n)}]/[\mathrm{X}_0^{(n)}] = (\frac{p}{q})^k$. This steady state distribution resembles a discrete Gaussian in n, and hence is relatively tightly centered about n_0; this is important to ensure good control over the distribution and that perturbing its mean value is practicable. As a sanity check, we can show that

this distribution indeed gives $\beta' = \alpha'$:

$$\begin{aligned}\frac{\beta'}{\alpha'} &= \left(\frac{p}{q}\right)^{n_0} \frac{\sum_{n\in\mathbb{Z}} \left(\frac{p}{q}\right)^{-\frac{1}{2}(n+\frac{1}{2}-n_0)^2+\frac{1}{2}(\frac{1}{2}-n_0)^2}}{\sum_{n\in\mathbb{Z}} \left(\frac{p}{q}\right)^{n}\left(\frac{p}{q}\right)^{-\frac{1}{2}(n+\frac{1}{2}-n_0)^2+\frac{1}{2}(\frac{1}{2}-n_0)^2}} \\ &= \frac{\sum_{n\in\mathbb{Z}} \left(\frac{p}{q}\right)^{-\frac{1}{2}(n+\frac{1}{2}-n_0)^2+\frac{1}{2}(\frac{1}{2}+n_0)^2}}{\sum_{n\in\mathbb{Z}} \left(\frac{p}{q}\right)^{-\frac{1}{2}(n-\frac{1}{2}-n_0)^2+\frac{1}{2}(\frac{1}{2}+n_0)^2}} \\ &= 1.\end{aligned}$$

Finally we seek to characterise the rates for the coupled reaction; first, we shall require the weight for each of the $\{\mathrm{X}_k^{(n)} : k\}$ subsystems:

$$\sum_{k=0}^{n} [\mathrm{X}_k^{(n)}] = \frac{[\mathrm{X}_0^{(n)}]}{t-1} \begin{cases} t^{n+1} - 1 & n \geq 0 \\ t - t^n & n \leq 0 \end{cases}$$

where $t = \frac{p}{q}$. Next, we use $t^n t^{-\frac{1}{2}(n+\frac{1}{2}-n_0)^2+\frac{1}{2}(\frac{1}{2}-n_0)^2} = t^{-\frac{1}{2}(n-\frac{1}{2}-n_0)^2+\frac{1}{2}(\frac{1}{2}+n_0)^2}$ to find the normalisation,

$$\begin{aligned}[\mathrm{X}_0^{(0)}]^{-1} = 1 &+ \tfrac{1}{t-1}[t\textstyle\sum_{n<0} - \sum_{n>0}] t^{-\frac{1}{2}(n+\frac{1}{2}-n_0)^2+\frac{1}{2}(\frac{1}{2}-n_0)^2} \\ &+ \tfrac{1}{t-1}[t\textstyle\sum_{n>0} - \sum_{n<0}] t^{-\frac{1}{2}(n-\frac{1}{2}-n_0)^2+\frac{1}{2}(\frac{1}{2}+n_0)^2}.\end{aligned}$$

This is non-trivial for general n_0, so we restrict to the two limiting cases of $n_0 = 0$ and $|n_0| \gg 1$. In the former case, the normalisation simplifies to $[\mathrm{X}_0^{(0)}] = \frac{b}{1+b}$ and so we have $\beta' = \alpha' = \frac{\alpha b}{1+b} \sum_{n\in\mathbb{Z}} t^{\frac{1}{8}-\frac{1}{2}(n-\frac{1}{2})^2}$. The sum is bounded from below by the integral $2\int_0^\infty t^{-\frac{1}{2}(s+\frac{1}{2})^2}\,\mathrm{d}s$, and from above by $2\int_0^\infty t^{-\frac{1}{2}(s-\frac{1}{2})^2}\,\mathrm{d}s$, from which we obtain $\alpha' = \alpha\sqrt{\pi b} + \mathcal{O}(b)$. In the latter case, suppose $n_0 \gg 1$ is positive (i.e. $\beta \gg \alpha$). The normalisation will then be dominated by the sums over $n > 0$ and so has asymptotically exact approximation

$$\begin{aligned}[\mathrm{X}_0^{(0)}]^{-1} &= \tfrac{t}{t-1} \textstyle\sum_{n\in\mathbb{Z}} t^{-\frac{1}{2}(n-\frac{1}{2}-n_0)^2+\frac{1}{2}(\frac{1}{2}+n_0)^2} - \tfrac{1}{t-1} \sum_{n\in\mathbb{Z}} t^{-\frac{1}{2}(n+\frac{1}{2}-n_0)^2+\frac{1}{2}(\frac{1}{2}-n_0)^2} \\ &= \tfrac{1}{b}[pt^{\frac{1}{2}(\frac{1}{2}+n_0)^2} - qt^{\frac{1}{2}(\frac{1}{2}-n_0)^2}] \textstyle\sum_{n\in\mathbb{Z}} t^{-\frac{1}{2}(n+\frac{1}{2}-n_0)^2},\end{aligned}$$

hence α' reduces to $2b\alpha + \mathcal{O}(b^2)$ in the case of $\beta \gg \alpha$, or $2b\beta + \mathcal{O}(b^2)$ in the case of $\alpha \gg \beta$. In the ideal case, in which only the excess B (when $\beta > \alpha$) is sequestered, we should have $\alpha' = \beta' = \alpha$. The consequence of the factor $(\sqrt{\pi b}, 2b)$ is to further suppress the current through the coupled transition. As the uncoupled monon transitions have current $\propto b$, this results in an overhead of $\sim b^{-3/2}$ in the best case and $\sim b^{-2}$ in the worst. This overhead is comparable to the time penalties we found for communication and synchronisation interactions in Chapter III, as for the coupled reaction to proceed both the monon and the complexed klonon must be in receptive states. Drawing further from the analogy, one may wonder whether reactions involving multiple klona are subject to even greater overheads; fortunately, this is not the case: provided only one sequestration

$$[\![\sigma \gg \circ\, \sigma']\!] \underset{q}{\overset{p}{\rightleftharpoons}} [\![\sigma \circ \gg \sigma']\!] \qquad \text{(INTRA–BIAS)}$$

$$\mathrm{M}_{i+\frac{1}{3}} + \mathrm{A}{:}[\![\sigma \circ \circ \gg]\!] \rightleftharpoons [\![\sigma \circ \ll]\!]{:}\mathrm{B} + \mathrm{M}_{i+\frac{2}{3}} \qquad \text{(POS–STEP)}$$

$$\mathrm{M}_{i+\frac{1}{3}} + \mathrm{A}{:}[\![\circ \gg]\!] \rightleftharpoons [\![\diamond]\!]{:}\mathrm{B} + \mathrm{M}_{i+\frac{2}{3}} \qquad \text{(POS–BASE)}$$

$$\mathrm{M}_{i+\frac{1}{3}} + \mathrm{A}{:}[\![\diamond]\!] \rightleftharpoons [\![\ll \circ]\!]{:}\mathrm{B} + \mathrm{M}_{i+\frac{2}{3}} \qquad \text{(NEG–BASE)}$$

$$\mathrm{M}_{i+\frac{1}{3}} + \mathrm{A}{:}[\![\gg \circ\, \sigma]\!] \rightleftharpoons [\![\ll \circ \circ\, \sigma]\!]{:}\mathrm{B} + \mathrm{M}_{i+\frac{2}{3}} \qquad \text{(NEG–STEP)}$$

Listing IV.3: The abstract molecular reaction scheme implementing $\mathrm{X}_k^{(n)} \rightleftharpoons \mathrm{X}_{k+1}^{(n)}$ and $\mathrm{X}_n^{(n)} \rightleftharpoons \mathrm{X}_0^{(n-1)}$ for the unary representation case. As described in Section IV.4, the species $\mathrm{X}_k^{(n)}$ is represented by a polymer of n copies of the $\circ$ monomer. These polymers are delimited by $[\![$ and $]\!]$, and are intercalated with a 'marker' monomer, $\gg$ or $\ll$. The position of the marker monomer corresponds to the value of k as explained in the text, and the (INTRA–BIAS) reaction tends to push it in the direction of its arrow. Note that $\gtrless$ represents either marker monomer and σ, σ' represent arbitrary strings of $\circ$ monomers.

klonon participates, by choosing a single representative from each of the reactant klona and product klona to engage in complexation with the sequestration klona, then the results stand.

IV.4. Molecular Counter Implementation

Here we present example implementations of the counter klona $\mathrm{X}_k^{(n)}$ used in Section IV.3. These implementations are abstract, but point towards a viable molecular realisation.

Unary Representation The simplest implementation of counters is given by representing numbers in unary, i.e. a Peano axiomatic approach. For a natural integer $n \in \mathbb{N}$, we propose a polymeric representation consisting of n repeated $\circ$ monomers and represented thus,

$$[\![\underbrace{\circ\, \circ\, \cdots\, \circ\, \circ}_{n}]\!].$$

We can then introduce a subcounter $0 \leq k \leq n$ by incorporating a marker monomer, $\gg$, placed between the monomers and which can migrate between the $n+1$ possible positions. Moreover, we can handle non-positive values of n and k via the alternative marker $\ll$. Whilst not strictly necessary, we also differentiate the case $n = k = 0$ with a

third marker monomer, $\diamond$. Example polymers are given for $n = -4, 0, 4$:

$$\begin{aligned}
\mathrm{X}_0^{(4)} &= [\![\gg \circ \circ \circ \circ]\!], & & & \mathrm{X}_0^{(-4)} &= [\![\circ \circ \circ \circ \gg]\!], \\
\mathrm{X}_1^{(4)} &= [\![\circ \gg \circ \circ \circ]\!], & & & \mathrm{X}_{-1}^{(-4)} &= [\![\circ \circ \circ \gg \circ]\!], \\
\mathrm{X}_2^{(4)} &= [\![\circ \circ \gg \circ \circ]\!], & \mathrm{X}_0^{(0)} &= [\![\diamond]\!], & \mathrm{X}_{-2}^{(-4)} &= [\![\circ \circ \gg \circ \circ]\!], \\
\mathrm{X}_3^{(4)} &= [\![\circ \circ \circ \gg \circ]\!], & & & \mathrm{X}_{-3}^{(-4)} &= [\![\circ \gg \circ \circ \circ]\!], \\
\mathrm{X}_4^{(4)} &= [\![\circ \circ \circ \circ \gg]\!], & & & \mathrm{X}_{-4}^{(-4)} &= [\![\gg \circ \circ \circ \circ]\!].
\end{aligned}$$

Notice the mirrored interpretation of marker positions as values of k for positive and negative values of n; this is intentional and serves to simplify the implementation of the intra-n biasing reactions. It should also be noted that these polymers are achiral and so their mirror images, e.g. $\mathrm{X}_1^{(4)} = [\![\circ \circ \circ \ll \circ]\!]$, are equivalent representations of the same polymer. This property greatly simplifies the implementation of the $\mathrm{X}_n^{(n)} \rightleftharpoons \mathrm{X}_0^{(n-1)}$ reactions. Altogether, this implementation can be achieved with five reactions as listed in Listing IV.3. In fact, this scheme can be implemented with just four reactions by replacing $\diamond$ with $\gg$ and combining reactions (POS-STEP) and (POS-BASE), but we prefer the given implementation for its symmetry. More concretely, these reactions engender the following bi-infinite Markov chain:

$$\begin{aligned}
\cdots \rightleftharpoons [\![\gg \circ \circ]\!] \rightleftharpoons [\![\circ \gg \circ]\!] \rightleftharpoons [\![\circ \circ \gg]\!] \rightleftharpoons [\![\gg \circ]\!] \rightleftharpoons [\![\circ \gg]\!] \rightleftharpoons [\![\diamond]\!] \\
[\![\diamond]\!] \rightleftharpoons [\![\ll \circ]\!] \rightleftharpoons [\![\circ \ll]\!] \rightleftharpoons [\![\ll \circ \circ]\!] \rightleftharpoons [\![\circ \ll \circ]\!] \rightleftharpoons [\![\circ \circ \ll]\!] \rightleftharpoons \cdots .
\end{aligned}$$

Binary Representation Whilst particularly simple, the unary representation is very space inefficient; worse, incrementing and decrementing n, for the purpose of modulating access to a sparse resource/disequilibrium reaction, will itself require interaction with a resource pool in order to acquire and release $\circ$ monomers. Fortunately an exponentially more compact representation exists in the form of positional notations such as binary, with spatial complexity logarithmic in $|n|$ and hence needing far fewer interactions with its respective monomer pool.

Unlike with the unary case, we cannot exploit the structure of the representation of n to represent k. Instead we shall need to store both explicitly. We will also require the ability to recognise the states $k = 0$ and $k = n$, both for determining whether the coupled reaction $\mathrm{A} \rightleftharpoons \mathrm{B}$ should proceed and for ensuring the reactions that generate the intra-n distribution do not produce illegal values of k. These two states correspond respectively to all the digits of k being 0 or being identical to those of n. Since $|k| < |n|$, its minimal representation will be no larger than that of n and hence a convenient structure for encoding the pair (n, k) with the ability to easily check the condition $n = k$ is given by a 'zipped' double-stranded polymer which we denote by $[\![{}^{n}_{k}]\!]$; for example, in base 2 the pair $(11, 6)$ would be given by

$$[\![{}^{1\,0\,1\,1}_{0\,1\,1\,0}]\!].$$

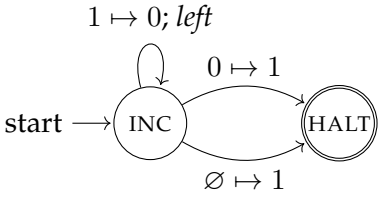


Figure IV.4: The state diagram for a Turing Machine [90] implementing incrementation of a natural number in binary positional notation. The alphabet is $\{\varnothing, 0, 1\}$, with $\varnothing$ indicating a blank square; numbers may be provided with any number of leading 0s, and the initial position should be the least significant digit.

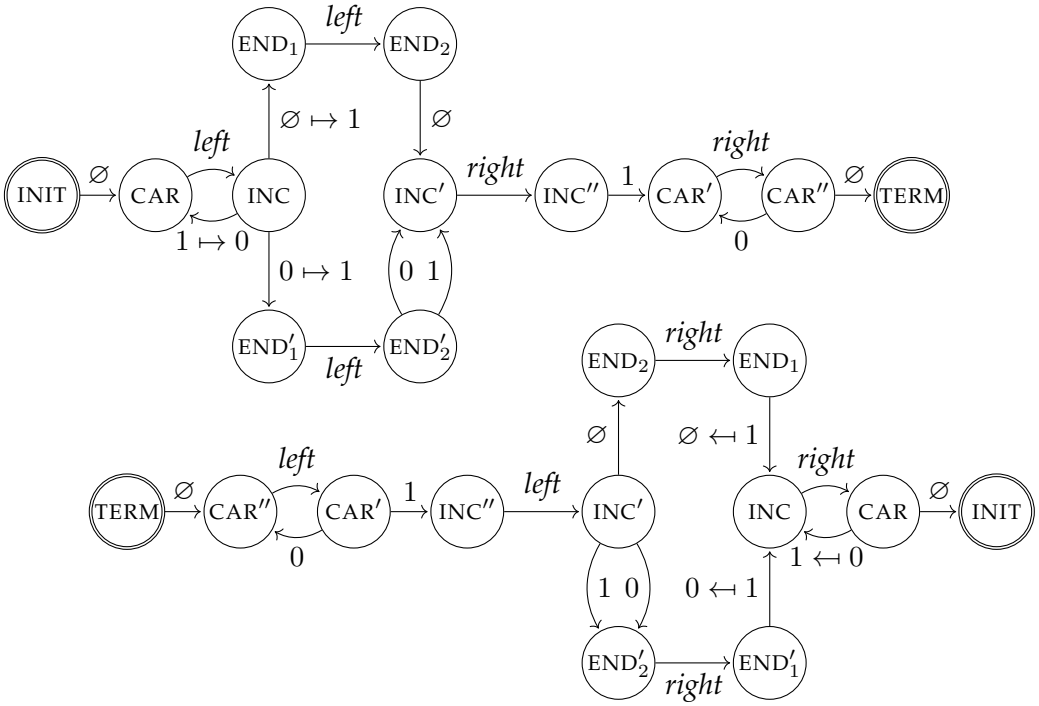


Figure IV.5: The state diagram for a Reversible Turing Machine [39] (and its reverse) implementing incrementation (resp. decrementation) of a natural number in binary positional representation. The alphabet is $\{\varnothing, 0, 1\}$, with $\varnothing$ indicating a blank square; numbers must be provided in the described prefix-free form, and the initial position should be the (empty) square to the immediate right of the least significant digit. The box-shaped 'subroutine' is used to handle the case when a new digit must be prepended; as this is a reversible branch, we must ensure that the converged state INC′ is able to uniquely determine the branch from whence it came.

In contrast to the unary case, this representation *does* have an intrinsic polarity, with the leftmost digits being the most significant. To ensure logical reversibility of the forthcoming reaction scheme, the polymer representation must be unique for each (n, k) pair, but positional representations admit a countably infinite equivalence class for each integer (e.g. 3 has base-10 representations $\{3, 03, 003, \ldots\}$). We resolve this ambiguity by trimming all leading zeros of n and trimming k to the same length. This has the consequence that $(0, 0)$ has the 'empty' representation $[\![]\!]$; it is certainly possible to introduce a special case for $(0, 0)$ of $[\![\begin{smallmatrix}0\\0\end{smallmatrix}]\!]$ for æsthetic reasons, but this would significantly complicate the reaction scheme for no other benefit.

Thus far, we have only considered the representation for non-negative $k, n \in \mathbb{N}$; there are a number of approaches one could take to extend the representation, from simply incorporating a *sign* trit, $\{-, 0, +\}$, similar to the approach for the unary case, to a two's complement approach. The two's complement approach is particularly attractive as it would require only minimal adjustments to the non-negative reaction scheme. The two's complement representation of negative numbers is realised by taking the equivalence class for the integer, e.g. $\{11, 011, 0011, \ldots\}$ for 3, and complementing each bit to obtain, e.g., $\{00, 100, 1100, \ldots\}$. That is, each negative integer has an infinite prefix of 1s instead of 0s as for the non-negative case. This approach is used in most conventional CPUs because no special-casing for negative numbers is necessary: adding a negative number m, with two's complement m', to a positive number n is equivalent to adding the positive numbers m' and n. As a simple justification for this, consider decrementing $1000\cdots000$. We will need to 'borrow' a 1 from each digit to the left, eventually obtaining $111\cdots111$. In the limit of infinitely many 0s, this results in infinitely many 1s. For brevity we shall restrict our attention to the non-negative case, but extending to the full domain is a straightforward—if tedious—exercise.

Whilst recognising the cases $k = 0$ and $k = n$ is not difficult in our representation, comparing $\sim \log n$ digits each time we wish to execute the coupled reaction is not ideal. By a slight increase in the complexity of the $\mathrm{X}_k^{(n)} \rightleftharpoons \mathrm{X}_{k+1}^{(n)}$ and $\mathrm{X}_n^{(n)} \rightleftharpoons \mathrm{X}_0^{(n-1)}$ algorithms, we can render these checks trivial. In particular, we augment the polymer with two bits of state for each monomer pair which we call Q, for eQuality, and Z, for Zero. As biochemical precedent, compare with phosphorylation sites. The function of these states is best understood by example; for $n = 11$, the Q-Z states for each of $k = 0, 1, 8, 11$ are given by

$$[\![\begin{smallmatrix}1\,0\,1\,1\\ \underline{0\,0\,0\,0}\end{smallmatrix}]\!], \qquad [\![\begin{smallmatrix}1\,0\,1\,1\\ \underline{0\,0\,0\,1}\end{smallmatrix}]\!], \qquad [\![\begin{smallmatrix}\underline{1\,0}\,1\,1\\ 1\,0\,0\,0\end{smallmatrix}]\!], \qquad [\![\begin{smallmatrix}\underline{1\,0}\,1\,1\\ 1\,0\,1\,1\end{smallmatrix}]\!].$$

Namely, a prefix of zeroes is marked by the Z state (bottom lines) and a prefix matching n is marked by the Q state (middle lines), and these are easily achieved by implementing two local invariants for each digit pair $\left[\begin{smallmatrix}x\\y\end{smallmatrix}\right]$:

(Q) The Q state is *on* if and only if $y = x$ and the Q state of the digit pair to its left (if it exists) is also *on*.

(Z) The Z state is *on* if and only if $y = 0$ and the Q state of the digit pair to its left (if it exists) is also *on*.

```
[]          `Inc` [‘1];
[‘0 x · bs] `Inc` [‘1 x · bs];
[‘1 · bs]   `Inc` [‘0 · bs’]:
    bs `Inc` bs’.
```

Listing IV.6: A recursive ℵ/`alethe` (see Chapter V) program implementing incrementation of a natural number in binary positional representation. Numbers are provided in the described prefix-free form as a list of digits $\{0, 1\}$ with the least-significant bit first. The second case pattern matches on two bits in order to ensure its output pattern is distinct from that of the first case, and this is where a program will stall should a number be provided which is not in prefix-free form.

It can be seen that these two states are orthogonal as the leading bit of n must be 1, with the possible exception of $(0, 0)$ where it makes sense to define Q and Z to be both *on* despite the lack of lack of monomers on which to mark said states.

Before implementing a reaction scheme, we first require an algorithmic understanding of reversibly incrementing/decrementing integers in positional notation. As it is possible (and indeed trivial) to do so for integers in unary notation as demonstrated by Listing IV.3 we should expect it to also be possible for positional notation, and indeed it is. The irreversible algorithm for incrementing a binary number is very simple (Figure IV.4): if there are any trailing 1 bits then we carry the 1 (flipping the intermediate 1 bits to 0) until we reach the first 0, which we flip to 1. If we reach the end of the number, we prepend a new 1 bit. Making this algorithm reversible is conceptually simple, but expressing it in the form of a Reversible Turing Machine (RTM, as introduced by Bennett [39]) is somewhat involved. An example implementation is provided in Figure IV.5, and can be broken down into three stages: (1) carrying; (2) incrementing the least significant 0 bit, or prepending a fresh 1 bit, followed by the logic to reversibly join these two branches; (3) ‘reverse’ carrying, in which we return to the starting position and use the fact that the suffix we are processing takes the form $100\cdots000$ to ensure we can logically reverse this stage. It can be readily checked that this scheme as presented is reversible, and indeed we provide the mechanical reversal of the RTM which may be seen to implement the operation of reversibly decrementing a positive natural number. To demonstrate that the algorithm is in fact conceptually simple, an equivalent recursive program written in a higher level reversible language is shown in Listing IV.6.

Finally, the reaction schemes for $\mathrm{X}_k^{(n)} \rightleftharpoons \mathrm{X}_{k+1}^{(n)}$ and $\mathrm{X}_n^{(n)} \rightleftharpoons \mathrm{X}_0^{(n-1)}$ are made manifest in Listings IV.7 and IV.8 respectively. In addition to implementing the reversible incrementation procedure in an abstract molecular sense, these schemes must also incorporate the relevant logic for ensuring k is appropriately bounded and that the Q and Z invariants are preserved (although these may be transiently broken within the intermediate states). Furthermore the logic for the second reaction (which we choose

to implement in reverse, i.e. $\mathrm{X}_0^{(n)} \rightleftharpoons \mathrm{X}_{n+1}^{(n+1)}$, per the freedom reversibility affords us) must reversibly copy the value of n into k, and thus the second scheme must read the entire length of the polymer. As for the case of $n \equiv 2^p - 1$ in $n \mapsto n+1$ (resp. $n+1 \mapsto n$), the scheme recruits (resp. releases) a pair of binary monomers ($\mathbb{B}_2$) which will be provided by some resource pool mediated by its own sequestration klona. An additional complication is the prevention of multiple initiation events: as the scheme operates via local rules, it is not possible to tell whether a molecule is in a stable state $\mathrm{X}_k^{(n)}$ or if it is currently undergoing a transition. To address this, a 'blocked' terminus is used to represent intermediate states. Symbolically, this takes the form $[\![\cdot \!]\!]$ instead of $[\![\cdot]\!]$.

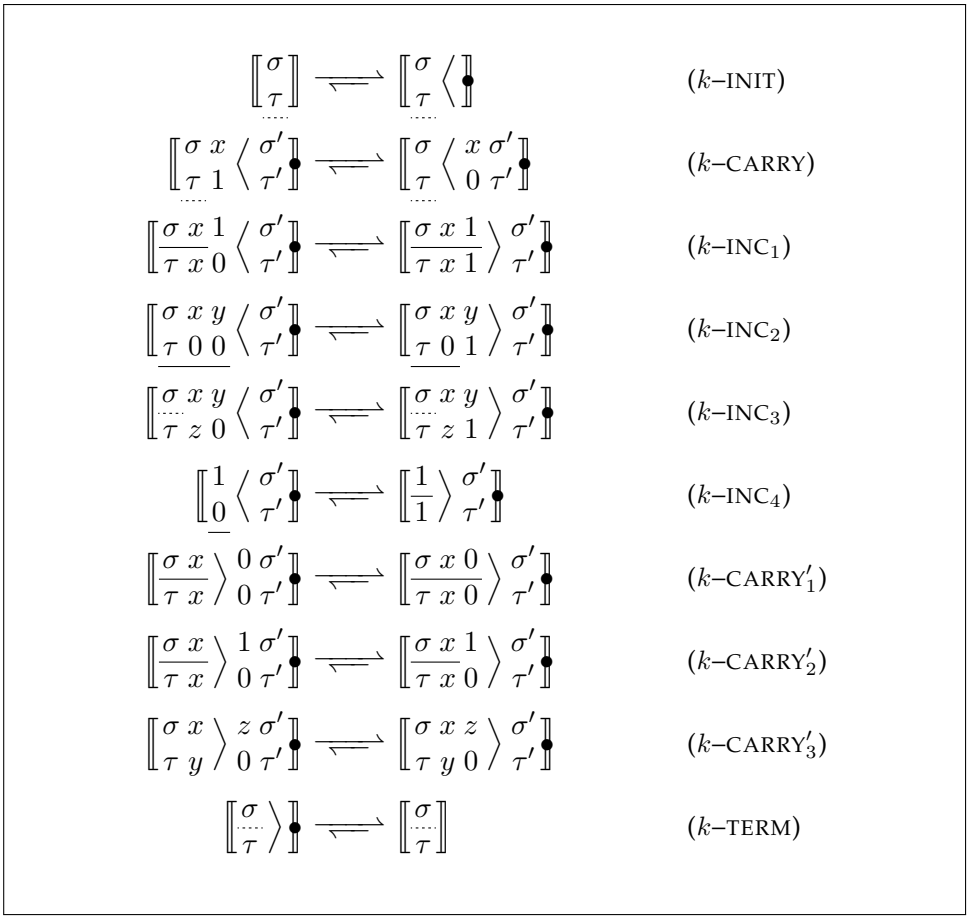
IV.5. Conclusion

In this chapter we studied the interaction of mona, such as molecular computers, with klona, shared resources, in a Brownian system with a limiting supply of free energy. As in Chapter III, we found this to be very expensive in comparison to independent computation. In general the cost, in terms of transition time, to couple mona to an arbitrary system of klona is of order $\mathcal{O}(b^{-2})$ (an overhead of $\mathcal{O}(b^{-1})$) but, in contrast to the mona-mona interactions of Chapter III, lighter overheads of order $\mathcal{O}(b^{-1/2})$ or better can be achieved under certain conditions. For example, whilst we found that all resource distribution schemes we could devise (some particular examples of which were illustrated in Section IV.2) did not admit a fractional resource availability above $\mathcal{O}(b)$ if time overheads were to be avoided, by allowing a time overhead of $\mathcal{O}(b^{-1/2})$ (thence a time penalty of $\sim \mathcal{O}(b^{-3/2})$) the full resource pool was rendered accessible. We conjectured that these results are in fact general bounds on the capabilities of any such system.

To drive a general out-of-equilibrium system of klona, one needs to supply sufficient free energy to overcome the entropy cost of the reaction. When this cost is known or bounded, such as in biological systems, the definition of 'sufficient' can be precomputed and built in to the specifications of the system. In general, however, the cost may be unknown or unbounded, or the favourable direction of the reaction may switch over time. We proposed a scheme in Section IV.3 to dynamically infer this cost, as well as automatically applying the correct amount of free energy tokens to pay for it, thereby exposing an equilibrated interface to the disequilibrium reaction. A more detailed specification of the (abstract) molecular realisation of such a scheme is presented in Section IV.4. This equilibrated interface can then be readily coupled to our monon transitions, however the cost must still be paid in terms of transition time overhead and indeed it is, as the concentration of the interface is commensurately reduced. Unfortunately our proposed scheme has a minimum overhead of $\mathcal{O}(b^{-1/2})$ even in the case where the underlying reaction is at or near equilibrium (where it is theoretically possible to achieve zero overhead). It is unknown whether a better scheme exists, whilst still retaining the ability to drive reactions that are arbitrarily far from equilibrium.

As any non-zero overhead to a class of interactions will lead to this class 'freezing out'

in very large reversible computers, their frequency must be minimised—the proposed scheme notwithstanding. Once again referencing the conclusions of Chapter III, these interactions thus minimised can make use of a subpopulation bias klona of greater free energy to drive these unfavourable interactions more effectively.



Listing IV.7: The abstract molecular reaction scheme implementing $\mathrm{X}_k^{(n)} \rightleftharpoons \mathrm{X}_{k+1}^{(n)}$ for the binary representation case. To only use a single bias token's worth of free energy, one can make use of a similar scheme to that used in Equation (IV.1), and perhaps making some of the reactions in this scheme unbiased. Alternatively one can accept the use of many tokens, which has the advantage of increasing the rate of the coupled reaction due to the effectively higher bias, but has the disadvantage of slower convergence of the sequestration klona to steady state—potentially interfering with correct operation of the coupled reaction during periods of high flux. In this scheme, the $\langle$ and $\rangle$ symbols represent klona that mark the current state and progress of the overall reaction. It is of note that, in the (k–CARRY) rule, there is no provision for Q to be set on the bit-pair to the left of $\binom{x}{1}$, as such a state would imply we are trying to increment $k \geq n$ and is thus illegal.

$$\mathrm{M}_{i+\frac{2}{3}} + \mathrm{B}{:}\left[\!\left[\begin{smallmatrix}\sigma\\ \underline{\tau}\end{smallmatrix}\right]\!\right] \rightleftharpoons \left[\!\left[\begin{smallmatrix}\sigma\\ \underline{\tau}\end{smallmatrix}\right\{\right]\!\bullet{:}[\text{Æ}\mathrm{B}{-}\mathrm{M}]^{\pm} \qquad (n\text{–INIT})$$

$$[\text{Æ}\mathrm{B}{-}\mathrm{M}]^{\pm}{:}\left[\!\left[\begin{smallmatrix}\sigma\ 1\\ \underline{\tau\ 0}\end{smallmatrix}\right\{\begin{smallmatrix}\sigma'\\ \tau'\end{smallmatrix}\right]\!\bullet \rightleftharpoons \left[\!\left[\begin{smallmatrix}\sigma\\ \underline{\tau}\end{smallmatrix}\right\{\begin{smallmatrix}0\ \sigma'\\ 0\ \tau'\end{smallmatrix}\right]\!\bullet{:}[\text{Æ}\mathrm{B}{-}\mathrm{M}]^{\pm} \qquad (n\text{–CARRY})$$

$$[\text{Æ}\mathrm{B}{-}\mathrm{M}]^{\pm}{:}\left[\!\left[\begin{smallmatrix}\sigma\ x\ 0\\ \underline{\tau\ 0\ 0}\end{smallmatrix}\right\{\begin{smallmatrix}\sigma'\\ \tau'\end{smallmatrix}\right]\!\bullet \rightleftharpoons \left[\!\left[\begin{smallmatrix}\sigma\ x\\ \underline{\tau\ 0}\end{smallmatrix}\right(\begin{smallmatrix}1\ \sigma'\\ 1\ \tau'\end{smallmatrix}\right]\!\bullet{:}[\text{Æ}\mathrm{B}{-}\mathrm{M}]^{\pm} \qquad (n\text{–INC}_1)$$

$$[\text{Æ}\mathrm{B}{-}\mathrm{M}]^{\pm}{:}\left[\!\left[\right\{\begin{smallmatrix}\sigma'\\ \tau'\end{smallmatrix}\right]\!\bullet \overset{\mathbb{B}_2}{\rightleftharpoons} \left[\!\left[\right(\begin{smallmatrix}1\ \sigma'\\ 1\ \tau'\end{smallmatrix}\right]\!\bullet{:}[\text{Æ}\mathrm{B}{-}\mathrm{M}]^{\pm} \qquad (n\text{–INC}_2)$$

$$[\text{Æ}\mathrm{B}{-}\mathrm{M}]^{\pm}{:}\left[\!\left[\begin{smallmatrix}\sigma\ x\\ \underline{\tau\ 0}\end{smallmatrix}\right(\begin{smallmatrix}y\ \sigma'\\ 0\ \tau'\end{smallmatrix}\right]\!\bullet \rightleftharpoons \left[\!\left[\begin{smallmatrix}\sigma\\ \underline{\tau}\end{smallmatrix}\right(\begin{smallmatrix}x\ y\ \sigma'\\ 0\ 0\ \tau'\end{smallmatrix}\right]\!\bullet{:}[\text{Æ}\mathrm{B}{-}\mathrm{M}]^{\pm} \qquad (n\text{–PREFIX}_1)$$

$$[\text{Æ}\mathrm{B}{-}\mathrm{M}]^{\pm}{:}\left[\!\left[\right(\begin{smallmatrix}\sigma'\\ \tau'\end{smallmatrix}\right]\!\bullet \rightleftharpoons \left[\!\left[\right)\begin{smallmatrix}\sigma'\\ \tau'\end{smallmatrix}\right]\!\bullet{:}[\text{Æ}\mathrm{B}{-}\mathrm{M}]^{\pm} \qquad (n\text{–PREFIX}_2)$$

$$[\text{Æ}\mathrm{B}{-}\mathrm{M}]^{\pm}{:}\left[\!\left[\begin{smallmatrix}\sigma\\ \tau\end{smallmatrix}\right)\begin{smallmatrix}x\ \sigma'\\ 0\ \tau'\end{smallmatrix}\right]\!\bullet \rightleftharpoons \left[\!\left[\begin{smallmatrix}\sigma\ x\\ \tau\ x\end{smallmatrix}\right)\begin{smallmatrix}\sigma'\\ \tau'\end{smallmatrix}\right]\!\bullet{:}[\text{Æ}\mathrm{B}{-}\mathrm{M}]^{\pm} \qquad (n\text{–PREFIX}_3)$$

$$[\text{Æ}\mathrm{B}{-}\mathrm{M}]^{\pm}{:}\left[\!\left[\begin{smallmatrix}\sigma\\ \tau\end{smallmatrix}\right)\begin{smallmatrix}1\ \sigma'\\ 1\ \tau'\end{smallmatrix}\right]\!\bullet \rightleftharpoons \left[\!\left[\begin{smallmatrix}\sigma\ 1\\ \tau\ 1\end{smallmatrix}\right\}\begin{smallmatrix}\sigma'\\ \tau'\end{smallmatrix}\right]\!\bullet{:}[\text{Æ}\mathrm{B}{-}\mathrm{M}]^{\pm} \qquad (n\text{–INC}')$$

$$[\text{Æ}\mathrm{B}{-}\mathrm{M}]^{\pm}{:}\left[\!\left[\begin{smallmatrix}\sigma\\ \tau\end{smallmatrix}\right\}\begin{smallmatrix}0\ \sigma'\\ 0\ \tau'\end{smallmatrix}\right]\!\bullet \rightleftharpoons \left[\!\left[\begin{smallmatrix}\sigma\ 0\\ \tau\ 0\end{smallmatrix}\right\}\begin{smallmatrix}\sigma'\\ \tau'\end{smallmatrix}\right]\!\bullet{:}[\text{Æ}\mathrm{B}{-}\mathrm{M}]^{\pm} \qquad (n\text{–CARRY}')$$

$$[\text{Æ}\mathrm{B}{-}\mathrm{M}]^{\pm}{:}\left[\!\left[\begin{smallmatrix}\sigma\\ \tau\end{smallmatrix}\right\}\right]\!\bullet \rightleftharpoons \left[\!\left[\begin{smallmatrix}\sigma\\ \tau\end{smallmatrix}\right]\!\right]{:}\mathrm{A} + \mathrm{M}_{i+\frac{1}{3}} \qquad (n\text{–TERM})$$

Listing IV.8: The abstract molecular reaction scheme implementing $\mathrm{X}_0^{(n)} \rightleftharpoons \mathrm{X}_{n+1}^{(n+1)}$ for the binary representation case. This overall reaction is unbiased, which raises the concern that the sequestration klona could spend an undesirable amount of time in the intermediate states above. To limit this possibility, the intermediate states should be made more unfavourable and this is achieved by the complementary biases in the rules (n–INIT) and (n–TERM). These biases need not be provided by the bias tokens, and in fact that is unideal as the bias is vanishingly small; instead it can be implemented by introducing an enthalpy change, such that there is a chance of thermal activation of the reaction, whilst the klona still prefer to exist in the non-intermediate states. As in the scheme for the intra-n reaction, klona are introduced to mark the state and progress of the overall reaction; in this case, {, }, (, and), with the { and } klona corresponding primarily to incrementation of n and the (and) klona to copying of the prefix of n into k.

Reversible computation is somewhat neglected in the field of molecular computation. Motivated by a need for a model of reversible computation appropriate for a Brownian molecular architecture, this chapter introduces the ℵ-calculus as a step towards this goal. This novel model is declarative, concurrent, and term-based—encapsulating all information about the program data and state within a single structure in order to obviate the need for a *von Neumann*-style discrete computational 'machine', a challenge in a molecular environment. The name is inspired by the Greek for 'not forgotten' (ἀλήθεια), due to the emphasis on (reversibly) learning and un-learning knowledge of different variables. To demonstrate its utility for this purpose, as well as its elegance as a programming language, a number of examples are presented; two of these examples, addition/subtraction and squaring/square-rooting, are furnished with designs for abstract molecular implementations, although more work needs to be done in order to realise something resembling ℵ experimentally. The example of addition/subtraction is reproduced in Figure V.1. A natural by-product of these examples and accompanying syntactic sugar is the design of a fully-fledged programming language, `alethe`, which is also presented along with an interpreter. Efficiently simulating ℵ on a deterministic computer necessitates some static analysis of programs within the `alethe` interpreter in order to render the declarative programs sequential. Finally, work towards a type system appropriate for such a reversible, declarative model of computation is presented.

V. The ℵ-Calculus

A declarative model of reversible programming with relevance to Brownian computers

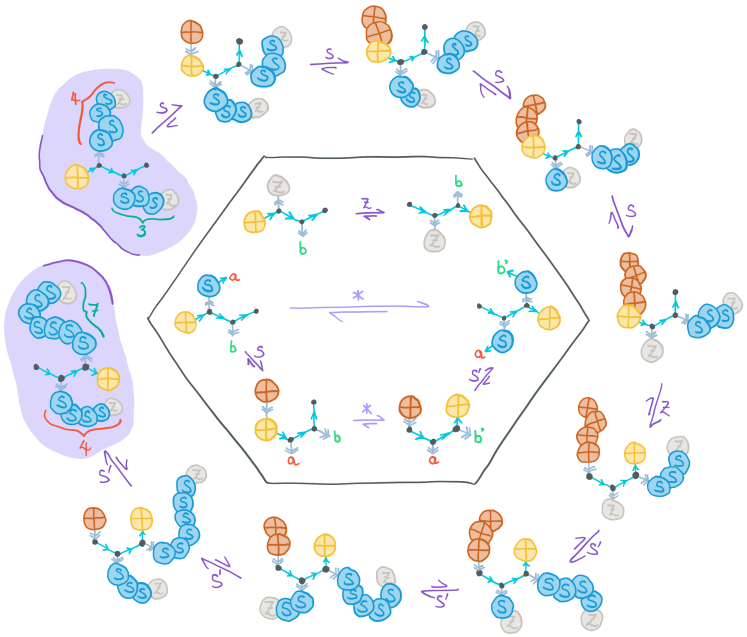


Figure V.1: An abstract molecular realisation of an ℵ definition of addition (resp. subtraction), here reversibly computing the sum of 4 and 3 (resp. the difference between 7 and 4).

V.1. Introduction

In this chapter we present a new model of reversible computing, the ℵ-Calculus, and an associated programming language `alethe` together with interpreter, whose properties we believe to be novel. The name is inspired by the Greek meaning 'not forgotten' (ἀλήθεια), as the semantics of ℵ revolve around the transformation of knowledge: 'unlearning' knowledge of one or more variables in order to 'learn' knowledge about one or more other variables, but in a reversible fashion such that nothing is ever truly forgotten. ℵ is declarative and concurrent and, whilst perhaps a little too abstract and high level for

this purpose, is motivated by a need for a reversible model for molecular programming and DNA computing. Crucial to such applications is that almost all information about not just the program's data, but the program state too, should be encoded within a computation 'term'. This is in contrast to the imperative and functional languages reviewed in the introduction, wherein state information such as the instruction counter (indicating where in the program the current execution context is) is implicitly stored in a special register or other hidden state of the processor. In א, the program itself is (reversibly) mutated during the course of its execution, such that the distinction between 'program' and 'processor' vanishes. The reason for this design is that a *von Neumann*-style architecture, in which there is a discrete processing unit that interacts with a memory unit to execute a program, is unsuitable for a molecular context as it is—in some sense—too 'bulky' and difficult to engineer. We assert that the proposed design is far more suitable to molecular implementation, or at least serves as a step towards this goal, and it is hoped the reader will be convinced of this by the examples and semantics illustrated herein.

V.2. א by Example

Before expositing the formal semantics of א, it is illuminating to introduce a number of examples of א in order to gain an intutition for its syntax and features. We begin with a reversible definition of addition. An immediate obstacle with reversible arithmetic is that binary operations such as addition are not invertible: it is not possible, given the answer 7, to infer which two numbers were originally added. As shown in Figure I.5, one possible resolution of this (due to Bennett [39, 40]) is to embed an irreversible computation $x \mapsto f(x)$ injectively as $x \mapsto (x, f(x))$. For addition, Bennett's algorithm would yield $+ : (x, y) \leftrightarrow (x, y, x + y)$. For example, adding 3 and 4 would give as output $(3, 4, 7)$. Nevertheless we are not satisfied with this embedding, and wish to better exploit reversible computational architectures by programming directly with reversible primitives. The benefits of doing so are that one often finds that far less temporary information need be generated than Bennett's algorithms might suggest, and also the injective embedding, $x \mapsto (x, f(x))$, retains the input which is excessive except in the trivial case of f being constant; for any other function f having any correlation at all between its outputs and inputs, only a partial image of x need be preserved. Moreover it is often the case that one can imagine a suitable and more preferable injective or bijective embedding. For example, a more useful embedding of addition might take the form $+ : (x, y) \leftrightarrow (x, x + y)$. In defining these injections more carefully, one then often finds that the residual information of the input can in fact be made use of; for example, a Peano arithmetic implementation of $+ : (x, y) \leftrightarrow (x, x + y)$ over the naturals readily begets the (domain-restricted) injection $\forall x > 0. \times : (x, y) \leftrightarrow (x, xy)$ without generating any temporary data whatsoever. Furthermore, when redundancy is eliminated as in these two cases one obtains the converse operation for free: that is, running $+ : (x, y) \leftrightarrow (x, x + y)$ in reverse yields subtraction, and $\forall x > 0. \times : (x, y) \leftrightarrow (x, xy)$ yields division, whilst their Bennett embeddings cannot do the same.

A little thought shows these must fail in certain cases. For the Bennett-style reversible embedding of addition, we saw that the forward direction of this program maps, e.g., $(3, 4)$ to $(3, 4, 7)$, and likewise the reverse direction maps $(3, 4, 7)$ to $(3, 4)$. Clearly the forward direction is an injection, but what about the reverse? Suppose we attempt to feed in $(3, 4, 5)$: as $3 + 4 \neq 5$, and as the forward direction is injective, there can be no pair (a, b) that maps to $(3, 4, 5)$. In fact, even our less redundant embedding $+ : (x, y) \leftrightarrow (x, x + y)$ suffers from non-injectivity of its inverse, for example there is no value $y \in \mathbb{N}$ satisfying $+ : (5, y) \leftrightarrow (5, 2)$. Yet another example occurs for the function $+' : (x, y) \leftrightarrow (x - y, x + y)$ defined over the integers, in which there is no pair (x, y) satisfying $+' : (x, y) \leftrightarrow (3, 6)$ (there is in $\mathbb{Q}$, however; namely, $(\frac{9}{2}, \frac{3}{2})$). Therefore, whilst we might expect reversible computers to compute bijective operations, this appears to be violated in even the simple example of addition. In fact, no violation of reversibility has occurred, and the 'primitive' steps of any reversible computer *are* bijective. What we have encountered here is simply a (co)domain error, the same as if we were to ask a conventional computer to evaluate $1/0$. That is, whereas conventional computers compute *partial* functions, reversible computers compute *partial* bijections (or partial isomorphisms). Exactly what happens when such an error is encountered depends on both the algorithm and architecture in question; for example, attempting to divide 1 by 0 may cause a computer to immediately complain, or it may enter an infinite loop if the algorithm is 'repeatedly subtract the divisor until the dividend vanishes'. We shall have more to say about how ℵ handles such errors in due course.

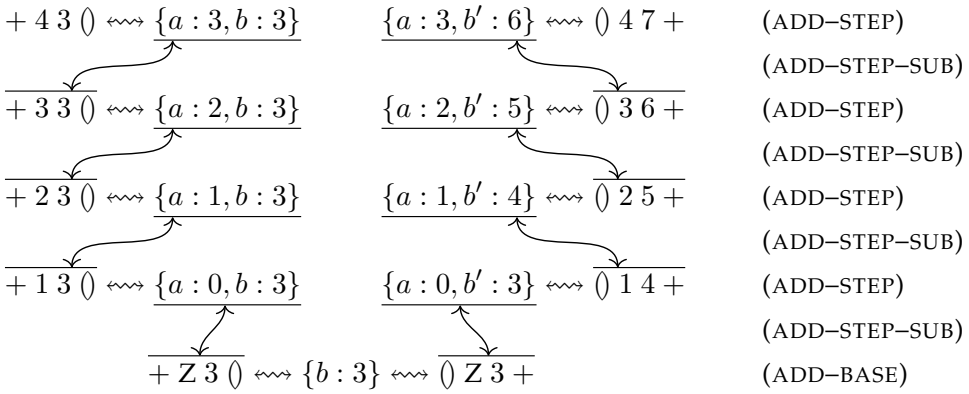
Addition Our examples will mostly concern natural numbers because they are particularly amenable to inductive and recursive definitions of both their structure and operations over them, such as addition and multiplication. The standard approach to this is the Peano axiomatic formulation, in which a natural number is defined to either be $\mathsf{Z} \equiv 0$, or $\mathsf{S}n \equiv n + 1$ whenever n is a natural number—i.e. the *successor* of n. For example, 4 is constructed as $\mathsf{S}(\mathsf{S}(\mathsf{S}(\mathsf{SZ})))$. In fact there are seven more axioms in order to clarify such subtle points as uniqueness of representation, conditions for equality, and non-negativity. Addition can then be defined recursively by the base case $\mathsf{Z} + b = b$ and the inductive step $\mathsf{S}a + b = \mathsf{S}(a + b)$, and multiplication by $a \cdot \mathsf{Z} = \mathsf{Z}$ and $a \cdot \mathsf{S}b = a + a \cdot b$. To render addition reversible, we write $+ : (\mathsf{Z}, b) \leftrightarrow (\mathsf{Z}, b)$ and $+ : (\mathsf{S}a, b) \leftrightarrow (\mathsf{S}a, \mathsf{S}(a + b))$, realising the proposed embedding $+ : (x, y) \leftrightarrow (x, x + y)$. Multiplication is embedded similarly, but there is a domain-restriction imposed in that a must not be zero (or else b cannot be uniquely recovered); b may, however, be zero. In ℵ, addition is written thus

$$
\begin{aligned}
&+\ \mathsf{Z}\ b\ ()\ =\ ()\ \mathsf{Z}\ b\ +; && \text{(ADD–BASE)}\\
&+\ (\mathsf{S}a)\ b\ ()\ =\ ()\ (\mathsf{S}a)\ (\mathsf{S}b')\ +: && \text{(ADD–STEP)}\\
&\quad +\ a\ b\ ()\ =\ ()\ a\ b'\ +. && \text{(ADD–STEP–SUB)}
\end{aligned}
$$

and is perhaps best understood by example. In Listing V.2, we perform the example addition of 4 and 3 and, as promised, an example failure mode in which we attempt to subtract 5 from 2.

$$
\begin{aligned}
&+\ \mathrm{Z}\ b\ ()\ =\ ()\ \mathrm{Z}\ b\ +; && \text{(ADD–BASE)}\\
&+\ (\mathrm{S}a)\ b\ ()\ =\ ()\ (\mathrm{S}a)\ (\mathrm{S}b')\ +: && \text{(ADD–STEP)}\\
&\quad +\ a\ b\ ()\ =\ ()\ a\ b'\ +. && \text{(ADD–STEP–SUB)}
\end{aligned}
$$

(a) The definition of reversible natural addition in ℵ.



(b) The ℵ execution path when reversibly adding 4 to 3 or, in reverse, subtracting 4 from 7. The ⇜⇝ arrows refer to pattern matching/substitution, whilst the solid arrows refer to instantiation/consumption of 'sub-terms'.

(c) The ℵ execution path when attempting to (erroneously) subtract 5 from 2. The recursive algorithm identifies that $2 - 5 \equiv 0 - 3$, but there is no matching definition for this and therefore computation 'stalls' on this sub-term. This is usually addressed in the natural numbers by employing the saturating option of 'monus', i.e. $2 \dot{-} 5 = 0$, but it is not reversible.

Listing V.2: The definition of, and example applications of, reversible addition in ℵ.

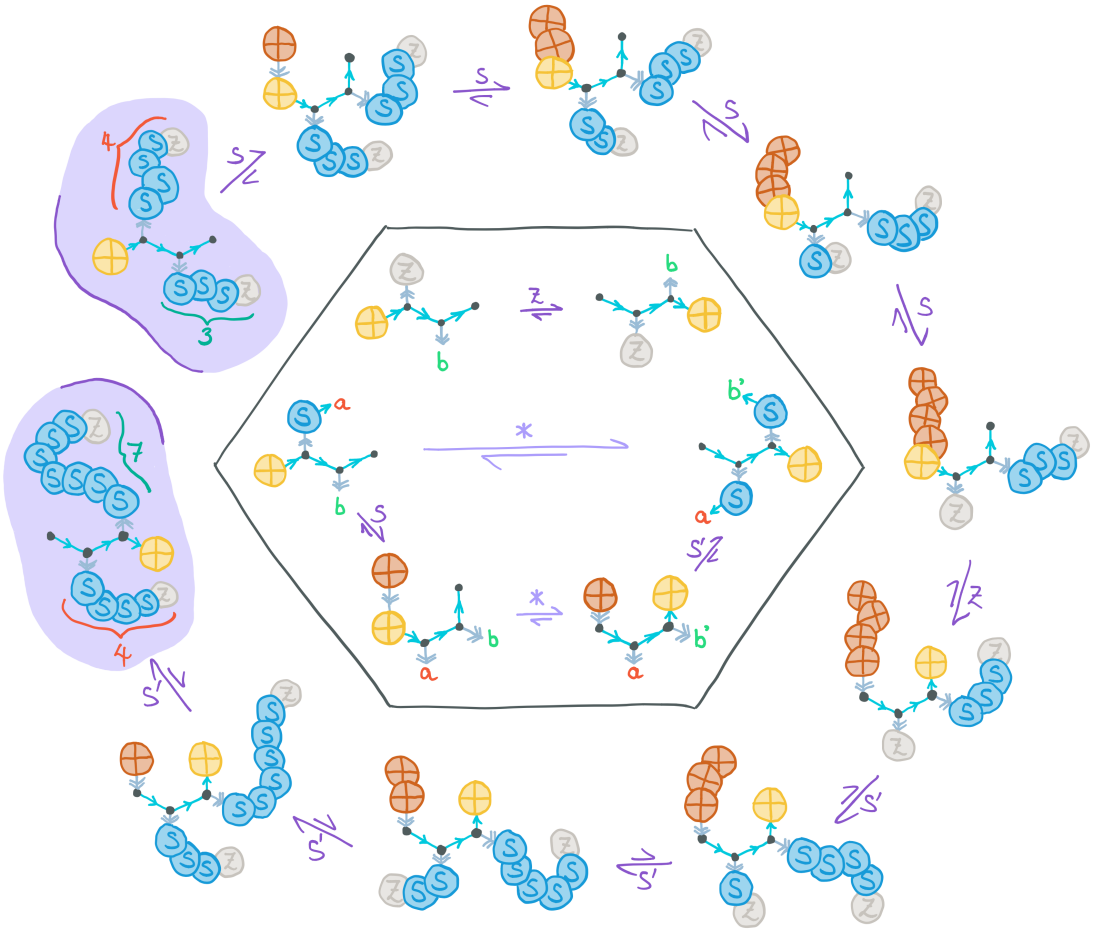


Figure V.3: An abstract molecular translation of the reversible addition definition given in Listing V.2a, as well as the example addition of 4 and 3 following Listing V.2b. The abstract molecular model espoused here is defined by a fixed set of atoms (e.g. $\{+, \mathrm{S}, \mathrm{Z}, \bullet\}$) connected by two kinds of bond. Atoms joined by single-headed bonds are analogous to ℵ terms, whilst double-headed bonds corresponds to nesting of composite terms. The bonds are represented by arrows because there is an intrinsic polarity/directionality to the molecules; this is not necessary, and can be replaced by auxiliary atoms such as L and R, but it does simplify our representation. Atoms are rendered by circles, whilst the reaction definitions also use variables written as uncircled letters. The • 'atom' is special in that it is a placeholder for a nested composite term. Reaction arrows for 'elementary' reactions are labelled by rule names, and starred reaction arrows indicate an effective reaction composed of multiple elementary steps. The arrows are drawn biased in anticipation of a biasing mechanism to be discussed later on within this section.

א can thus be seen, in a loose sense, to be a term-rewriting system. It is 'loose' in the sense that its 'sub-terms' exist 'separate' from their parent term. In the example of Listing V.2, an addition term is written $+\ a\ b\ ()$ and is mapped by the transition rules to $()\ a\ c\ +$ where $c \equiv a + b$; here $+$ is an 'atom', a, b and c are terms representing natural numbers (as composite terms formed from nested applications of the atoms S and Z), and $()$, or 'unit', is the empty term and is used by convention in א to avoid certain ambiguities. A program corresponds to a series of definitions of transition rules which pattern match on terms, and then substitute the variables into an output pattern. This matching process is subject to certain constraints that ensure reversibility. Moreover, a rule may specify sub-rules that indicate how to transform knowledge of some variable, e.g. b, to knowledge of another, e.g. b', and is the primary mechanism of composition in א. Inherent to the semantics of the calculus is a secondary composition mechanism, which was the only mechanism available in an earlier iteration of this calculus (see Section V.8): composite terms at any level are all subject to the same transition rules. This behaviour is more reminiscent of functional programming languages, but is somewhat clumsy in practice; nevertheless it is not without utility in א—in particular, it is well suited for when a continuation-passing style approach is favoured. The sub-rule mechanism, on the other hand, is more reminiscent of declarative programming languages.

When there is no matched rule, such as in Listing V.2c, this simply means that there is no successor state and so computation cannot continue. It is in fact very similar in nature to the case when computation succeeds, as then we obtain a term which has a predecessor state via some rule, but lacks a rule to generate a successor state. To properly distinguish these two conditions, we must explicitly mark 'true' halting states; for addition, this is written as

$$\mathbf{!} + \_\ \_\ ();\quad \mathbf{!}\ ()\ \_\ \_\ +;$$

where **!** ('bang') indicates that any term of the specified forms is a valid computational output. The former corresponds to the output of a subtraction, and the latter to the output of an addition. This subtlety is further contextualised by Figure III.2.

We conclude this first example by alluding to a possible molecular implementation, as depicted in Figure V.3. Here we see the importance of א being interpretable as a term-rewriting system, as the entirety of the state of the computation must be encoded within a single macromolecule (or molecular complex). Strictly speaking, this is not a requirement as DNA Strand Displacement systems [19] achieve computation without this requirement; in payment for this, though, the entire reaction volume is dedicated to the same (typically analogue) program. The precise mechanism of the reactions is omitted from the figure, instead expressing the model as an abstract chemical reaction network. Finding possible reaction mechanisms to implement א, or similar calculi, in real chemical systems will be the subject of future work. Whilst the abstract molecular formalism is attractive form an explanatory perspective, א provides a more concise formalism that is also easier to manipulate and to explicate its semantics.

Squaring Eliminating redundancy in the definition of addition yielded its inverse for free, but one may still object that the additional output is 'garbage' and not of any utility. What will happen if one uses this addition subroutine many times in a larger program? Naively we may expect this garbage to accumulate, requiring either active dissipation of the additional entropy or the application of Bennett's algorithm to clean it up. In fact, by retraining one's thought process from the irreversible programming paradigm to the reversible paradigm, it is often possible to make use of this garbage data. We demonstrate this with the example of finding the square of a natural number. The candidate function, $\square : n \leftrightarrow n^2$, is an injection and so clearly meets our requirement of partial bijection. Therefore, we have good reason to believe that it is possible to implement it. The obvious approach of using multiplication will not work because its reversible embedding will take a form not dissimilar to $\times : (m, n) \leftrightarrow (m, m \cdot n)$, and so would yield $\square : n \leftrightarrow (n, n^2)$. Whilst this suffices for realising the square of a number, it retains too much redundancy in its outputs.

Often a helpful tactic is to consider an inductive approach. For the square numbers this is encapsulated by $(n+1)^2 - n^2 = 2n + 1$, from which can be obtained the identity $n^2 \equiv \sum_{k=0}^{n-1} 2k + 1$. Again, we need to be clever: in order to achieve our desired (partial) bijection, we need to completely consume our input value of n. This can be done by evaluating the sum in reverse. Instantiate a new variable, $m = 0$; as m is set to a known value, this is reversible. Then, perform the following loop until n reaches 0: decrement n, add $2n + 1$ to m, repeat. At the end of this loop, n will have reached a unique value (and can thus be reversibly destroyed) and m will have been set to the square of the original value of n. In addition, this can be implemented with our addition subroutine in two steps, by adding n twice to m (retaining the value of n) and finally incrementing m. This is implemented in ℵ in Listing V.4, and the example of squaring 3 (equivalently, taking the square root of 9) is presented in Listing V.5.

V.3. ℵ by Example 2: Parallelism & Concurrency

Having introduced the essence of ℵ in the previous section, we now dive deeper into some more advanced features and examples of ℵ.

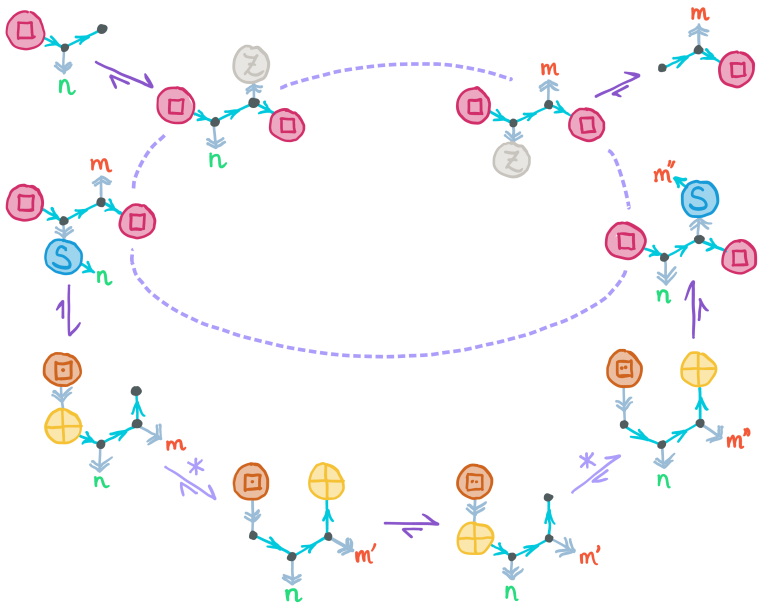
Sugar To reduce boilerplate and increase clarity in longer programs, it is helpful to introduce some syntactic sugar (shorthands). More sugar will be introduced later for the definition of the programming language `alethe` (which is really just sugared ℵ), but for now only a light sprinkling is required.

Many rules take the rote form

$$\begin{aligned} &\mathbf{!}\, f\ x_1\ x_2\ \cdots\ x_m\ ();\quad \mathbf{!}\, ()\ y_1\ y_2\ \cdots\ y_n\ f; \\ &f\ x_1\ x_2\ \cdots\ x_m\ () = ()\ y_1\ y_2\ \cdots\ y_n\ f: \\ &\qquad \cdots \end{aligned}$$

$$
\begin{aligned}
&!\,\square\ \_\ ();\quad !\,()\ \_\ \square;\\
&\square\ n\ () = \square\ n\ \mathrm{Z}\ \square; && \text{(SQUARE–INIT)}\\
&\square\ \mathrm{Z}\ m\ \square = ()\ m\ \square; && \text{(SQUARE–TERM)}\\
&\square\ (\mathrm{S}n)\ m\ \square = \square\ n\ (\mathrm{S}m'')\ \square: && \text{(SQUARE–STEP)}\\
&\qquad +\ n\ m\ () = ()\ n\ m'\ +. && \text{(SQUARE–STEP–SUB}_1\text{)}\\
&\qquad +\ n\ m'\ () = ()\ n\ m''\ +. && \text{(SQUARE–STEP–SUB}_2\text{)}
\end{aligned}
$$

(a) The definition of □ in ℵ.



(b) The definition of □ in the abstract molecular formalism introduced in Figure V.3. The dashed lines indicate that there exists a term matching both patterns, and this is characteristic of conditionals and looping in ℵ. Note that computation remains unambiguous and deterministic: by Figure III.2, each intermediate state has both a predecessor and a successor, and of the two matching patterns one will correspond to the reverse direction and one to the forward direction. For example, in the middle of a loop one may encounter the intermediate species □ 2 5 □ and this term matches both the patterns □ (Sn) m □ and □ n (Sm'') □, but if it matches the former the computation will proceed forward whilst if the latter the computation will proceed backward. It will be shown later how the semantics keep track of this, and how we ensure the program really is unambiguous.

Listing V.4: A reversible definition of squaring of natural numbers, □ : $n \leftrightarrow n^2$, both in ℵ and in an abstract molecular formalism.

$$! \quad \square\ 3\ () \leftsquigarrow \{n : 3\} \leftsquigarrow \square\ 3\ \mathrm{Z}\ \square \quad \cdots \qquad \text{(SQUARE–INIT)}$$

$$\begin{aligned}
\cdots \quad & \square\ 3\ \mathrm{Z}\ \square \leftsquigarrow \underline{\{n : 2, m : \mathrm{Z}\}} && \text{(SQUARE–STEP)}\\
& \quad \overleftarrow{\underline{+\ 2\ \mathrm{Z}\ ()}} \leftrightarrow \underline{()\ 2\ 2\ +} && \text{(SQUARE–STEP–SUB}_1\text{)}\\
& \quad\ \overleftarrow{\underline{\{n : 2, m' : 2\}}} && \textit{checkpoint}\\
& \quad \overleftarrow{\underline{+\ 2\ 2\ ()}} \leftrightarrow \underline{()\ 2\ 4\ +} && \text{(SQUARE–STEP–SUB}_2\text{)}\\
& \quad\ \overleftarrow{\underline{\{n : 2, m'' : 4\}}} \leftsquigarrow \square\ 2\ 5\ \square \quad \cdots && \text{(SQUARE–STEP)}
\end{aligned}$$

$$\begin{aligned}
\cdots \quad & \square\ 2\ 5\ \square \leftsquigarrow \underline{\{n : 1, m : 5\}} && \text{(SQUARE–STEP)}\\
& \quad \overleftarrow{\underline{+\ 1\ 5\ ()}} \leftrightarrow \underline{()\ 1\ 6\ +} && \text{(SQUARE–STEP–SUB}_1\text{)}\\
& \quad\ \overleftarrow{\underline{\{n : 1, m' : 6\}}} && \textit{checkpoint}\\
& \quad \overleftarrow{\underline{+\ 1\ 6\ ()}} \leftrightarrow \underline{()\ 1\ 7\ +} && \text{(SQUARE–STEP–SUB}_2\text{)}\\
& \quad\ \overleftarrow{\underline{\{n : 1, m'' : 7\}}} \leftsquigarrow \square\ 1\ 8\ \square \quad \cdots && \text{(SQUARE–STEP)}
\end{aligned}$$

$$\begin{aligned}
\cdots \quad & \square\ 1\ 8\ \square \leftsquigarrow \underline{\{n : \mathrm{Z}, m : 8\}} && \text{(SQUARE–STEP)}\\
& \quad \overleftarrow{\underline{+\ \mathrm{Z}\ 8\ ()}} \leftrightarrow \underline{()\ \mathrm{Z}\ 8\ +} && \text{(SQUARE–STEP–SUB}_1\text{)}\\
& \quad\ \overleftarrow{\underline{\{n : \mathrm{Z}, m' : 8\}}} && \textit{checkpoint}\\
& \quad \overleftarrow{\underline{+\ \mathrm{Z}\ 8\ ()}} \leftrightarrow \underline{()\ \mathrm{Z}\ 8\ +} && \text{(SQUARE–STEP–SUB}_2\text{)}\\
& \quad\ \overleftarrow{\underline{\{n : \mathrm{Z}, m'' : 8\}}} \leftsquigarrow \square\ \mathrm{Z}\ 9\ \square \quad \cdots && \text{(SQUARE–STEP)}
\end{aligned}$$

$$\cdots \quad \square\ \mathrm{Z}\ 9\ \square \leftsquigarrow \{m : 9\} \leftsquigarrow ()\ 9\ \square \quad ! \qquad \text{(SQUARE–TERM)}$$

$$! \quad \square\ 3\ () \leftrightarrow \square\ 3\ \mathrm{Z}\ \square \leftrightarrow \square\ 2\ 5\ \square \leftrightarrow \square\ 1\ 8\ \square \leftrightarrow \square\ \mathrm{Z}\ 9\ \square \leftrightarrow ()\ 9\ \square \quad !$$

Listing V.5: An example application of the □ definition from Listing V.4.

which we can abbreviate with an infix form as

$$x_1\ x_2\ \cdots\ x_m\ \text{`}f\text{`}\ y_1\ y_2\ \cdots\ y_n\text{:}$$
$$\cdots$$

where the halting patterns are implied. In the special case of f a single symbol, such as $+$, $\times$ or $\Box$, we omit the backticks and write, e.g., $4\ 3 + 4\ 7$. Note that, if f is a composite term, then we additionally assert $!\ f$.

We have already seen sugar for numeric data, e.g. $4 \equiv \mathrm{S}(\mathrm{S}(\mathrm{S}(\mathrm{SZ})))$. Another common data type is that of lists. As is standard in the functional world, we opt for singly linked lists implemented as composite pairs. If $\textsc{Cons}\ x\ y$ represents the pair (x, y) and $\textsc{Nil}$ the empty list, then the list $[5\ 2\ 4\ 3]$ has corresponding representation $\textsc{Cons}\ 5\ (\textsc{Cons}\ 2\ (\textsc{Cons}\ 4\ (\textsc{Cons}\ 3\ \textsc{Nil})))$. We also introduce sugar for matching on a partial prefix of a list, i.e. $[x\ y \cdot z\!\!\!/]$ corresponds to $\textsc{Cons}\ x\ (\textsc{Cons}\ y\ z\!\!\!/)$.

Parallelism With this sugar thus defined, we can rewrite the somewhat clumsy definition of recursively mapping a function f over a list,

$$\begin{aligned}
&!\ \textsc{Map}\ _;\quad !\ (\textsc{Map}\ _)\ _\ ();\quad !\ ()\ _\ (\textsc{Map}\ _);\\
&(\textsc{Map}\ f)\ \textsc{Nil}\ () = ()\ \textsc{Nil}\ (\textsc{Map}\ f);\\
&(\textsc{Map}\ f)\ (\textsc{Cons}\ x\ x\!\!\!/)\ () = ()\ (\textsc{Cons}\ y\ y\!\!\!/)\ (\textsc{Map}\ f)\text{:}\\
&\qquad f\ x\ () = ()\ y\ f.\\
&\qquad (\textsc{Map}\ f)\ x\!\!\!/\ () = ()\ y\!\!\!/\ (\textsc{Map}\ f).
\end{aligned}$$

more concisely and clearly as

$$\begin{aligned}
&[]\ \text{`}\textsc{Map}\ f\text{`}\ [];\\
&[x \cdot x\!\!\!/]\ \text{`}\textsc{Map}\ f\text{`}\ [y \cdot y\!\!\!/]\text{:}\\
&\qquad x\ \text{`}f\text{`}\ y.\\
&\qquad x\!\!\!/\ \text{`}\textsc{Map}\ f\text{`}\ y\!\!\!/.
\end{aligned}$$

Notice that the order in which the sub-rules are executed does not make a difference to the final result, due to referential transparency; in fact ℵ, being declarative, does not ascribe any importance to the ordering of the statements. Moreover, should a rule not be necessary for the final computation (or if there are multiple routes to the answer) then that rule will not necessarily be executed. A rule may even be evaluated more than once; for example, Bennett's algorithm may be implemented as

$$\begin{aligned}
&x\ \text{`}\textsc{Bennett}\ f\text{`}\ x\ y\text{:}\\
&\qquad x\ \text{`}f\text{`}\ \textit{garbage}\ y.
\end{aligned}$$

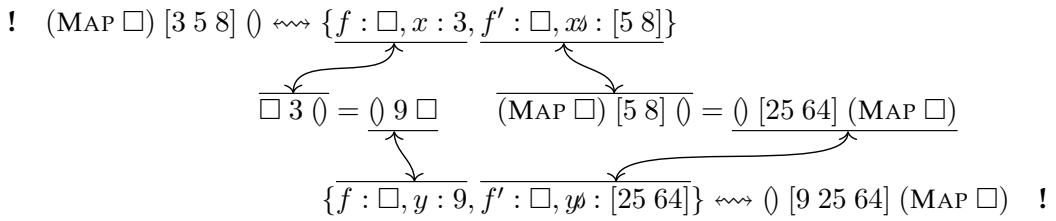
wherein the function f will be evaluated once in the forward direction, its output y will be copied, and then f will be evaluated again in the reverse direction to consume

the garbage. This arbitrarity in rule ordering and execution is important to its ability to operate in a stochastic system such as a molecular context, although in practice a compilation pass that chooses and enforces an optimal execution plan is important for efficiency.

The implicit duplication of variables that may occur means that one possible interpretation of MAP is

$$
\begin{aligned}
&[]\ \text{`MAP}\ f\text{`}\ [];\\
&[x \cdot x\emptyset]\ \text{`MAP}\ f\text{`}\ [y \cdot y\emptyset]:\\
&\qquad \text{`DUP}\ f\text{`}\ f'.\\
&\qquad x\ \text{`}f\text{`}\ y.\\
&\qquad x\emptyset\ \text{`MAP}\ f'\text{`}\ y\emptyset.
\end{aligned}
$$

from which we can see that not only is the order of the sub-rules arbitrary, but that they can be evaluated in parallel as the following example makes clear:



There remains a subtle point to be made: variables can be implicitly duplicated if they are used by multiple rules, but there is then a contract made with these rules that they really do return the variable unchanged. It is not possible to ensure this statically, however, and so it is entirely possible that the copies of some variable may diverge. In this case, running DUP in reverse will fail, and hence computation will stall. If duplication is not used, then computation will occur linearly and the changed value may be fed into subsequent rules unnoticed. In this case, computation may run to completion but yield an incorrect result due to the logic error.

Sorting A larger example program, which is capable of sorting a list using an arbitrary comparison function, is presented in Listing V.6. Whilst of poor algorithmic complexity, insertion sort is employed for simplicity. A more efficient implementation using merge sort is available in the accompanying standard library[1]. Clearly sorting is an irreversible process, as its purpose is to discard information regarding the original ordering of the list. The presented sorting algorithm puts a little effort towards increasing the utility of the garbage, in that the garbage data is a list saying where the original list item was placed into the sorted list *at the moment of insertion*. The insertion sort included in the standard library (Listing V.7) applies some additional processing to make this garbage data correspond to the permutation that maps the original list to the sorted list.

[1] https://github.com/hannah-earley/alethe-examples.

```
FALSE `NOT` TRUE;
TRUE `NOT` FALSE;

`< m Z` FALSE;
`< Z (S n)` TRUE;
`< (S m) (S n)` b:
    `< m n` b.

`≤ m n` b':
  `< n m` b.   b `NOT` b'.
`> m n` b:
  `< n m` b.
`≥ m n` b':
  `< m n` b.   b `NOT` b'.
```

```
! _ `INSERTIONSORT p` _ _;
(INSERTIONSORT p) xs () = IS p xs [] [] IS;
IS p [x · xs] ns ys IS = IS p xs [n · ns] zs IS:
    x ys `INSERT p` n zs.
IS p [] ns ys IS = () ns ys (INSERTIONSORT p);

x [] `INSERT p` Z [x];
x [y · ys] `INSERT p` n [z z' · zs]:
    `p x y` b.
    x [y · ys] b `INSERT' p` n [z z' · zs].

x [y · ys] TRUE `INSERT' p` Z [x y · ys];
x [y · ys] FALSE `INSERT' p` (S n) [y · zs]:
    x ys `INSERT p` n zs.
```

```
[3 2 0 7 6 4 5 1] `INSERTIONSORT <` [1 4 3 3 3 0 0 0] [0 1 2 3 4 5 6 7].
[3 2 0 7 6 4 5 1] `INSERTIONSORT ≥` [6 2 2 1 0 2 1 0] [7 6 5 4 3 2 1 0].
                                    └──garbage──┘
```

Listing V.6: An ℵ implementation of the comparison operators $<$, $\leq$, $>$ and $\geq$, and of insertion sort that can make use of these comparison operators. Lastly, the list [3 2 0 7 6 4 5 1] is sorted into both ascending and descending order.

```
xs `INSERTIONSORT p` ns' ys:
    ns `REVERSE` ns'. -- list reversal
  ! ~GO p xs [] [] = ~GO p xs [n · ns] ys'.
    ~GO p [x · xs] ns ys = ~GO p xs [n · ns'] ys':
        x ys `INSERT p` n ys'.
        ns `MAP (IS1 n)` ns'.
```

```
m `IS1 n` m':
    `< m n` b.
    `< m' n` b.
    m `IS2 b` m'.
m `IS2 TRUE` m;
m `IS2 FALSE` (Sm);
```

```
[77 2 42 68 41 36 8 36] `INSERTIONSORT <` [7 0 5 6 4 2 1 3] [2 8 36 36 41 42 68 77].
```

Listing V.7: The `alethe` implementation of insertion sort from the standard library. In contrast to the implementation in Listing V.6, this definition yields more useful 'garbage' data in that ns' contains the permutation which maps xs to ys, as shown in the boxed example. Specifically, this corresponds to the following permutation (in standard notation):

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 7 & 0 & 5 & 6 & 4 & 2 & 1 & 3 \end{pmatrix}.$$

Concurrency In addition to automatic parallelisation of independent sub-rules, א also supports defining transitions between separate terms. The motivation for this is for א to be able to fully exploit molecular architectures, but its utility is general and it is intended that a future version of the `alethe` interpreter will support concurrency.

Biasing Computation In Figure V.3 and Listing V.4b, abstract molecular implementations of addition and squaring were introduced respectively. Notably, the reaction arrows—whilst reversible—were biased in the forward direction. From a thermodynamic perspective, the direction in which a reaction occurs cannot be specified in an absolute sense, and depends on the conditions of the reaction volume at a given point in time. Moreover, at equilibrium the net direction of each reaction is null: they make no net progress. These concerns were discussed in more detail in Chapter II, but suffice it to say that trying to arrange the conditions of the system such that the computational terms themselves are inherently biased is impracticable, and will also limit the number of transition steps that can be executed. Moreover, it makes it difficult to run sub-rules in a reverse direction (such as is needed for Bennett's algorithm).

The solution is to separate the concerns of computation and biasing said computation. In biochemical systems, this is (mostly) achieved by using a common free energy carrier in the form of ATP and ADP + P_i, which are held in disequilibrium such that the favourable reaction direction is $ATP + H_2O \rightleftharpoons ADP + P_i + H^+$. Other biochemical reactions are then coupled to one or more copies of this hydrolysis reaction. For our purposes, we generalise this to assume that free energy is available in the form of two terms $\oplus$ and $\ominus$, where the concentration of $\oplus$ in the reaction volume is greater than that of $\ominus$. The 'computational bias' quantifying the average net proportion of computational transitions that are successful is found to be $b = ([\oplus] - [\ominus])/([\oplus] + [\ominus])$.

With a bias system thus defined, we can couple any computational transition to it by simply having the input side consume a $\oplus$ term and the output side release a $\ominus$ term. The more transitions that couple to the bias source, the faster and more robustly the computation proceeds. For example, the square definition can be amended like so:

$$!\,\square\ \_\ ();\quad !\,()\ \_\ \square;$$

$$\left\{\begin{matrix}\oplus \\ \square\ n\ ()\end{matrix}\right\} = \left\{\begin{matrix}\ominus \\ \square\ n\ \mathrm{Z}\ \square\end{matrix}\right\}; \qquad \text{(SQUARE–INIT)}$$

$$\left\{\begin{matrix}\oplus \\ \square\ \mathrm{Z}\ m\ \square\end{matrix}\right\} = \left\{\begin{matrix}\ominus \\ ()\ m\ \square\end{matrix}\right\}; \qquad \text{(SQUARE–TERM)}$$

$$\left\{\begin{matrix}\oplus \\ \square\ (\mathrm{S}n)\ m\ \square\end{matrix}\right\} = \left\{\begin{matrix}\ominus \\ \square\ n\ (\mathrm{S}m'')\ \square\end{matrix}\right\}: \qquad \text{(SQUARE–STEP)}$$

$$+\ n\ m\ () = ()\ n\ m'\ +. \qquad \text{(SQUARE–STEP–SUB}_1\text{)}$$

$$+\ n\ m'\ () = ()\ n\ m''\ +. \qquad \text{(SQUARE–STEP–SUB}_2\text{)}$$

The braces indicate that these definitions are concurrent, in that the transition will draw (release) two terms from (into) the reaction volume. As written above, the scheme is

fairly basic in that it only drives computation in one direction. An improved scheme may make use of a hidden variable in each term to indicate its preferred direction of computation. If this direction is reversed, then the transition will instead draw a $\ominus$ term and release a $\oplus$ term. If a sub-rule then needs to be run in the opposite direction to its parent, then the hidden variable need simply be inverted. This scheme can be made even more sophisticated by allowing the hidden variable to refer to alternate bias sources in case there are multiple to which the transition could couple (as was recommended in Chapters III and IV).

Communication A more canonical application of concurrency is that of communication between different computers. There are many possible communication schemes, and an overview and analysis of schemes appropriate for Brownian computers was presented in Chapter III, but we shall consider the simple example of an open communication channel between two computers, ALICE and BOB. ALICE has a list, the contents of which she wishes to send to BOB. A naive approach to this may resemble the ℵ definition

$$\textsc{Alice}\ [x \cdot xs] = \begin{Bmatrix}\textsc{Alice}\ xs \\ \textsc{Courier}\ x\end{Bmatrix}; \qquad \begin{Bmatrix}\textsc{Bob}\ ys \\ \textsc{Courier}\ y\end{Bmatrix} = \textsc{Bob}\ [y \cdot ys];$$

where the COURIER terms are used to convey list items between the two computers. Unfortunately, this does not quite realise the desired behaviour in a Brownian context: Suppose ALICE starts with the list $[1\ 2\ 3\ 4\ 5]$. She will generate the courier terms $(\textsc{Courier}\ 1)$, $(\textsc{Courier}\ 2)$, …, $(\textsc{Courier}\ 5)$, as expected, but these terms will be delievered to BOB via a diffusive approach. Except for the case of a one-dimensional system, BOB will almost certainly receive these couriers in a random order, completely uncorrelated from the original order. This may be avoided if delivery over the channel is substantially faster than the rates of term fission/fusion reactions, but this is not likely in a chemical system. Moreover, this being a reversible Brownian system, ALICE will from time to time re-absorb a courier that she previously dispatched, thereby increasing the opportunity for the list order to be shuffled.

The take-home message, here, is that whilst the local dynamics of a concurrent reversible computation system—such as ℵ—may well be reversible, the global dynamics are not necessarily reversible. In fact, this is a common property of microscopically reversibly systems, and is the basis of thermodynamics: the manifestation of macroscopically irreversible dynamics from microscopically reversible physics. It is doubtful that a model of concurrent reversible programming could preclude the possibility of such increases in entropy without severely restricting the system: likely any attempt to do so would effectively prevent the use of concurrency. Nevertheless, the programmer has a high degree of control here and so can, in principle, avoid entropy generation except where desired. In the above case of sending an ordered list, one could for example explicitly sequence the couriers. This suggests that a system of reversible molecular computers could have exceptionally low entropy generation and could realise very strange behaviours by getting as close to the implementation of a Maxwell Dæmon as physics allows. Conversely, one is also free to exploit the thermodynamic properties of

the system; for example, one could realise (to the extent the laws of physics permit) a true random number generator.

Resources Our last example of concurrency concerns the distribution of conserved resources amongst computers. If our computers are to respect the reversibility of the laws of physics, they should also respect mass conservation and so will need to contend with their finity. Amorphous computing presents one approach to achieving powerful computation from limited computational subunits, but it does so by making extensive use of communication which precludes the ability to perform meaningfully reversible computation. We instead suppose that the computers can exchange resources, such as memory units and structural building blocks, with the environment. Whilst this also has thermodynamic consequences, as shown in Chapter IV, it at least separates the concerns of computation from resource access and thus preserves the ability to perform reversible computation.

A host of resource distribution schemes, and thermodynamic analyses thereof, were presented in Chapter IV, but we content ourselves here with demonstrating how computers may interact with resources free in solution. In particular, we amend the definition of addition to be mass-conserving:

$$+\ \mathrm{Z}\ b\ ()\ =\ ()\ \mathrm{Z}\ b\ +; \qquad \text{(ADD–BASE)}$$

$$\left\{\begin{matrix}\mathrm{S} \\ +\ (\mathrm{S}a)\ b\ ()\end{matrix}\right\} = ()\ (\mathrm{S}a)\ (\mathrm{S}b')\ +: \qquad \text{(ADD–STEP)}$$

$$+\ a\ b\ ()\ =\ ()\ a\ b'\ +. \qquad \text{(ADD–STEP–SUB)}$$

Effects and Contexts As briefly mentioned, composite terms are subject to the same transition rules. Suppose LENA is learning ℵ and experiments with this, creating the following trivial wrapper around □:

$$\begin{aligned}&!\ \_\ \text{`MySquare`}\ \_;\\ &\textsc{MySquare}\ n\ ()\ =\ \textsc{MySquare}\ (\square\ n\ ())\ \textsc{MySquare};\\ &\textsc{MySquare}\ (()\ m\ \square)\ \textsc{MySquare}\ =\ \textsc{MySquare}\ m\ ();\end{aligned}$$

This definition, if a little contrived, will function as intended. Suppose, now, that she wants to inspect what happens within the loop of □, and so writes the following:

$$\begin{aligned}&!\ \_\ \text{`MySquare'`}\ \_;\\ &\textsc{MySquare'}\ n\ m\ ()\ =\ \textsc{MySquare'}\ (\square\ n\ m\ \square)\ \textsc{MySquare'};\\ &\textsc{MySquare'}\ (\square\ n'\ m'\ \square)\ \textsc{MySquare'}\ =\ \textsc{MySquare'}\ n'\ m'\ ();\end{aligned}$$

Now, what happens if we instantiate the term MYSQUARE′ 3 Z ()? Well, the final term could be any of () 3 Z MYSQUARE′, () 2 5 MYSQUARE′, () 1 8 MYSQUARE′ or () Z 9 MYSQUARE′. Whilst this arguably achieves LENA's aim of inspecting the execution of □, there is a problem in that ℵ claims all of these terms are halting states yet Figure III.2 asserts there

should be a maximum of two halting states. Despite being contrived, this example shows that we need to introduce another constraint into ℵ to prevent unexpected entropy generation: Any composite term or sub-term can only be created in a halting state, and can only be consumed in a halting state. Moreover, the creation and consumption patterns must be unambiguous as to which of the term's two halting states they refer (if, indeed, there are two). Otherwise, each instance of this ambiguity would multiply the state space and thus there would be an exponential increase in the size of the state space over time.

With the constraint on composite terms clarified, one can then ask whether all terms must follow the same set of rules. It turns out there is an important case where distinguishing between composite terms and top-level terms is useful; specifically, some cases of effectful computation. Suppose that the reaction volume has been endowed with a spatial lattice along which terms can travel, and imagine a term (CONS CHARLIE DAN) with such a wanderlust. If the composite terms CHARLIE and DAN want to travel to different locations, then where will the term end up? It is likely that a tug-of-war will occur, and so it makes sense to restrict the effecting of translocation to top-level terms.

This distinction is achieved by introducing term contexts: a top-level term has a top-level context, which is some label (itself a term) that may contain information, whilst composite terms have 'one-hole contexts'. For example, a term attached to a lattice may have context LATTICE while a term free in solution might have context FREE. Concretely, the aforementioned tug-of-war may be resolved by giving CHARLIE control by making him the top-level term, rendered as LATTICE : CHARLIE (DAN). Meanwhile, DAN is rendered as (LATTICE : (CHARLIE •)) : DAN where • indicates that this is a one-hole context. One-hole contexts can be defined for many data structures, but for a tree they correspond to removing some sub-tree of interest, replacing it with a 'hole'; see McBride [91] for some interesting properties of one-hole contexts. Rules may pattern match on top-level contexts and thus consume the information held within, or even create and destroy top-level terms, but they cannot match on one-hole contexts as this would risk altering their structure; instead, one-hole contexts can only be matched by 'opaque variables'. An opaque variable is a special variable found only in context patterns, which can match against a top-level context or a one-hole context, whilst regular variables in a context pattern can only match against top-level contexts. That is, the following are allowed

$$\{\text{LATTICE} : \text{CHARLIE}\ x\} = \left\{\begin{matrix}\text{LATTICE} : \text{CHARLIE}\ \langle\rangle \\ \text{FREE} : x\end{matrix}\right\};$$

$$\{\gamma : \text{DAN}\ [\text{DAN'S STUFF}]\} = \left\{\begin{matrix}\gamma : \text{DAN} \\ \text{LATTICE} : [\text{DAN'S STUFF}]\end{matrix}\right\};$$

whilst this is not

$$\{(\text{LATTICE} : (c\ \bullet)) : \text{DAN}\} = \{(\text{LATTICE} : (\bullet\ \text{DAN})) : c\};$$

This is not too onerous a restriction, as if one wishes to manipulate the structure of the one-hole context one can simply match against a higher level context.

In the earlier ℵ definitions, contexts were missing from the rule patterns. This can be seen as another example of syntactic sugar, with a missing context implying an opaque variable context (i.e. γ :) such that the rule can match against any term at any level. For concurrent definitions, however, all participating terms must be contextualised as otherwise it is unclear how to assign the results to the bound one-hole contexts.

It is as yet unclear how translocation along a lattice, or other effects, is actually achieved. Typically effects will be introduced as additional computational primitives, as by definition their actions are not 'computational'. As these computational primitives must be instantiated as top-level terms, we require a way to interface between computational terms and effector terms and this warrants a continuation-passing-style approach. For example, walking along a lattice from coordinate α to coordinate β may be realised thus:

$$
\begin{aligned}
& ! \textsc{Charlie}' \_; \quad ! \textsc{Charlie}'' \_; \\
& \{\textsc{Latt} : \textsc{Charlie}\ \beta\ x\} = \{\textsc{Latt} : \textsc{Walk}\ \beta\ (\textsc{Charlie}'\ x)\}; \\
\textit{(primitive)} \quad & \{\textsc{Latt} : \textsc{Walk}\ \beta\ c\} \leftrightarrow \{\textsc{Latt} : \textsc{Walk}'\ \alpha\ c\}; \\
& \{\textsc{Latt} : \textsc{Walk}'\ \alpha\ (\textsc{Charlie}''\ x)\} = \{\textsc{Latt} : \textsc{Charlie}'''\ \alpha\ x\};
\end{aligned}
$$

That is, WALK is given a coordinate to travel to as well as a continuation (which is free to perform additional computation during the walk, if desired). WALK then replaces the destination coordinate with the origin coordinate to ensure reversibility. Finally, we define a transition from WALK′ in order to return control to the continuation. This reveals a subtle point, that the effectful primitives are intentionally not marked as halting so that control can be transferred to and fro' them.

V.4. The Calculus

The ℵ-calculus thus introduced has a very simple definition. In BNF notation, it is

$$
\begin{aligned}
(\textsc{pattern term}) \quad & \tau ::= \textsc{atom} \mid \textsc{var} \mid (\ \tau^*\ ) \\
(\textsc{party}) \quad & \pi ::= \tau : \tau^* \mid \textsc{var}' : \tau^* \\
(\textsc{definition}) \quad & \delta ::= \{\pi^*\} = \{\pi^*\} : \pi^*_. \mid ! \tau
\end{aligned}
$$

where ATOM is an infinite set of atomic symbols (conventionally starting with an uppercase letter or a symbol), VAR an infinite set of variables (conventionally starting with a lowercase letter), and VAR′ is an orthogonal infinite set of variables (conventionally rendered in Greek) used for opaquely matching one-hole-contexts. A program is a series of definitions, δ^*, and a physical term is simply a pattern term without variables. Notice that the form of the sub-rules differs from the examples: a sub-rule can be separated into the instantiation of a term according to an input pattern, the evolution of that term, followed finally by the consumption of its final halting state according to the output pattern. In this view, these sub-terms are identified by one-hole-contexts—specifically, opaque variables. This formulation not only allows the semantics to represent automatic

parallelisation and a non-deterministic sub-rule ordering, but also begets an additional feature whereby sub-rules can instantiate top-level terms. The sub-rule forms in the examples can then be seen as sugar, i.e. $s = t.$ is equivalent to $\lambda : s.\ \lambda : t.$ where $\lambda \in \text{VAR}'$ is a fresh opaque variable. To illustrate, recall the definition of natural addition from Listing V.2a; desugared, this takes the form

1. $\mathbf{!} + a\ b\ ();$
2. $\mathbf{!}\ ()\ a\ b\ +;$
3. $\{\alpha : +\ \mathrm{Z}\ b\ ()\} = \{\alpha : ()\ \mathrm{Z}\ b\ +\};$ (ADD–BASE)
4. $\{\alpha : +\ (\mathrm{S}a)\ b\ ()\} = \{\alpha : ()\ (\mathrm{S}a)\ (\mathrm{S}b')\ +\}:$ (ADD–STEP)

4a. $\beta : +\ a\ b\ ().$

4b. $\beta : ()\ a\ b'\ +.$ (ADD–STEP–SUB)

That is, there are four definitions: two 'halting', and two 'computational'. On the left side of definition 4, we have a bag of one party. This party has as context pattern the opaque variable α, and its pattern term is a composite pattern term of length 4, consisting of the atom $+$, the composite pattern term of length 2 consisting of the atom S and the variable a, the variable b, and the empty composite pattern term ('unit'). There is also an opaque variable β, confined to the inner scope of definition 4. During execution of the sub-rule in the forward direction, the variables a and b will first be consumed in order to generate the sub-term $+\ a\ b\ ()$, which will be bound to the opaque variable β. This sub-term will then evolve to its other halting state, whereupon it will match sub-rule $4b$ and thus β will be consumed and the variables a and b' obtained.

Semantics To formalise the semantics embodied in the preceding examples, we define a transition relation $\leftrightarrow$ that maps a bag of terms to another bag of terms according to the rules defined for the current program. By a bag of terms, we aim to evoke the notion of a concoction of computational molecules in solution; a bag is a multiset, which can contain multiple copies of elements, but unlike a physical solution it lacks a concept of space. If desired, spatial locations can be readily simulated by appropriately chosen top-level contexts. Recalling that the global dynamics of the system are irreversible, we should expect that $\leftrightarrow$ is a non-deterministic relation. That is, application of the relation to a set of possible bags of terms may increase the size of this set, and the logarithm of the set size is identified with the entropy of the system. Concretely, the set of bags is the macrostate of the system, and each bag within the set is a microstate.

The calculus being reversible, the relation should share properties with equivalence relations. Namely, if S, T, U are bags of terms, then we have

$$S \leftrightarrow S \quad \text{(REFL)}$$

$$S \leftrightarrow T \implies T \leftrightarrow S \quad \text{(SYMM)}$$

$$S \leftrightarrow T \wedge T \leftrightarrow U \implies S \leftrightarrow U \quad \text{(TRANS)}$$

but we also make clear that these transitions can occur within an environment of other

non-participating terms, Γ,

$$S \leftrightarrow T \implies \forall \Gamma.\, \Gamma \cup S \leftrightarrow \Gamma \cup T \quad \text{(EXT)}$$
$$s \leftrightarrow t \implies \forall \Gamma.\, \Gamma \cup \{s\} \leftrightarrow \Gamma \cup \{t\} \quad \text{(EXT}')$$

where the second rule, in which s and t are single terms rather than bags of terms, is introduced as a convenient abuse of notation for later definitions.

In order to physically effect rule transitions, we introduce 'mediator terms' delimited by angle brackets, which interact with computational terms and can represent each of the intermediate states. If the program is given by $\mathcal{P}$, then the mechanism by which these mediator terms come into and out of existence is as follows,

$$\mathcal{P} \vdash (I = O : R) \implies \{\} \leftrightarrow \{\langle I \varnothing \varnothing R \varnothing O \rangle\} \quad \text{(INST–COMP)}$$
$$\mathcal{P} \vdash (!\, \tau) \implies \{\} \leftrightarrow \{\langle \tau \rangle\} \quad \text{(INST–HALT)}$$

from which we see that the environment can contain an arbitrary number of copies of each[2]. The mediator terms for computational rules are sextuples $\langle II'BR\Gamma O \rangle$ consisting of, respectively, the bag of unmatched input patterns I, the bag of matched input patterns I', the bag of resultant bindings B, the bag of sub-rules R, the local environment for internal sub-rules Γ, and the bag of output patterns O. The mediator terms for halting rules are trivial singletons $\langle \tau \rangle$ containing the relevant pattern, τ.

To assist rules in binding composite terms, we permit the current focus of a term to vary over time,

$$c : (\vec{l}\ t\ \vec{r}) \leftrightarrow (c : (\vec{l} \bullet \vec{r})) : t \quad (\text{OHC}_1)$$
$$c : [\vec{l}\ t\ \vec{r}] \leftrightarrow (c : [\vec{l} \bullet \vec{r}]) : t \quad (\text{OHC}_2)$$

where $\vec{l}$ and $\vec{r}$ are (possibly empty) strings of terms and t is the term focus. The bracketed terms in the second rule will be explained shortly. Note that these one-hole-context rules may not be needed for all architectures, being implicitly true in a molecular architecture for example.

The action of halting mediators is simply to mark terms which are in a known halting state,

$$\exists b'.\, t \overset{\tau}{\sim} b' \implies \{x : t, \langle \tau \rangle\} \leftrightarrow \{x : [t], \langle \tau \rangle\} \quad \text{(TERM)}$$

with $[t]$ serving as an indicator of a halting state and where $t \overset{\tau}{\sim} b'$ means that t unifies against τ with bindings b' (see Listing V.9).

Computational mediators are somewhat more complicated. We first render their temporal symmetry manifest by two reversibility rules,

$$\langle I \varnothing \varnothing R \varnothing O \rangle \leftrightarrow \langle O \varnothing \varnothing R \varnothing I \rangle \quad (\text{REV}_1)$$
$$\langle \varnothing I B R \Gamma O \rangle \leftrightarrow \langle \varnothing O B R \Gamma I \rangle \quad (\text{REV}_2)$$

[2] This complicates the aforementioned entropic interpretation of sets of term-bags, as the entropy will tend to diverge as the number of mediator terms tends to infinity. A more realistic implementation would leave the mean number of extant mediator terms finite and bounded, and perhaps even fix the number. This would have a further consequence on the kinetics and thermodynamics of the system, with the well-characterised Michælis-Menten kinetics a good starting point to the analysis thereof.

where these rules will be seen to be necessary even for computation in a single direction. Applying a computational rule consists first of binding against a matching term for each input pattern, followed by substituting the bindings into sub-terms per the sub-rules. The sub-terms may either be top-level or local. These transitions may all occur in parallel, i.e. a sub-term may be instantiated if all of its requisite variables are bound, even if not all the input patterns have matched a term.

$$\begin{aligned}
&x : t \overset{\pi}{\sim} b \implies && \text{(INP)}\\
&\{x : t, \langle (I \cup \{\pi\}) I' B R \Gamma O\rangle \leftrightarrow \{\langle I(I' \cup \{\pi\})(B \cup b) R \Gamma O\rangle\}\\
\pi \in R \wedge x : t \overset{\pi}{\sim} b &\implies && \text{(SUB}_1\text{)}\\
&\langle I I'(B \cup b) R \Gamma O\rangle \leftrightarrow \langle I I' B R(\Gamma \cup \{x : [t]\}) O\rangle\\
x \notin \text{VAR}' &\implies && \text{(SUB}_2\text{)}\\
&\{\langle I I' B R(\Gamma \cup \{x : [t]\}) O\rangle\} \leftrightarrow \{x : [t], \langle I I' B S \Gamma O\rangle\}
\end{aligned}$$

where x is a context. The (SUB$_2$) transition enables top-level terms to escape the locally scoped environment. Completion of computation occurs by application of (REV$_2$) followed by the (INP) and (SUB$_{1,2}$) in reverse; clearly if there is no route from the set of input variables to the set of output variables then the computation will stall. It may be desirable to augment the transition rules with implicit variable duplication,

$$\langle I I'(B \cup \{b\}) R \Gamma O\rangle \leftrightarrow \langle I I'(B \cup \{b, b\}) R \Gamma O\rangle \quad \text{(DUP)}$$

otherwise rules which wish to increase their parallelisability should explicitly duplicate variables as needed. We shall also need to enable the sub-environment to evolve,

$$\Gamma \leftrightarrow \Gamma' \implies \langle I I' B R \Gamma O\rangle \leftrightarrow \langle I I' B R \Gamma' O\rangle \quad \text{(SUB}_3\text{)}$$

These semantics, encapsulated by the $\leftrightarrow$ transition rule, are summarised in Listing V.8. An example of their application is provided in Listing V.10. It remains to describe the operation of binding/unification. This operation is defined as one would expect: a pattern matches if it is a variable, if it is an atom and the term is the same atom, or if it is a composite pattern and the term is a *halting* composite term of the same length and if the pattern and term match pair-wise. This is summarised in Listing V.9.

Reversibility The (SYMM) transition renders the semantics trivially reversible, but this is fairly weak. We conclude the discussion of the semantics of א by proving that the semantics are *microscopically* reversible.

Definition V.1 (Microscopic Reversibility)**.** A transition is primitively microscopically reversible if it is a uniquely invertible structural rearrangement. A transition is microscopically reversible if it is decomposable into a series of primitively microscopically reversible transitions.

Theorem V.2. *The semantics of א are microscopically reversible.*

Proof. The rules (REFL,SYMM,TRANS,EXT,EXT′,TERM,REV$_1$,REV$_2$,SUB$_3$) are microscopically reversible either trivially or inductively.

The instantiation rules (INST–COMP,INST–HALT) are less obvious, but can be realised in a microscopically reversible fashion in much the same way that cells translate an mRNA template to a polypeptide product. We shall first need a microscopically reversible way to duplicate a term, for which we assume that the environment contains an excess of structural building blocks (i.e. free atoms, variables, units $()$, etc). A term t to be duplicated is first marked as such i.e. $t \mapsto \bar{t}$. These modified terms deviate from the definitions introduced at the beginning of this section, and are instead intermediate transitional structures used for the small-step microscopically reversible semantics described here. This marking is then propagated throughout the structure to all atoms, variables, and units by the following two microscopically reversible transitions:

$$\overline{\gamma : t} \leftrightarrow \overline{\gamma} : \bar{t} \qquad \text{(DUP–PROP}_1\text{)}$$

$$\overline{(x\ \vec{x})} \leftrightarrow (\hat{x}\ \vec{x}) \qquad \text{(DUP–PROP}_2\text{)}$$

$$(\vec{x}\ \hat{x}\ y\ \vec{y}) \leftrightarrow (\vec{x}\ \overline{x}\ \hat{y}\ \vec{y}) \qquad \text{(DUP–PROP}_3\text{)}$$

where rules (DUP–PROP$_{2,3}$) are used to distribute the marking throughout composite terms. Note the use of the auxiliary 'hat' marker $\hat{\cdot}$ to sequence this propagation in a microscopically reversible manner. Elementary terms thus marked recruit fresh copies of themselves from the free structural building blocks,

$$\frac{}{a} \leftrightarrow \frac{a}{a} \qquad \frac{}{u} \leftrightarrow \frac{u}{u} \qquad \frac{}{v} \leftrightarrow \frac{v}{v} \qquad \frac{}{()} \leftrightarrow \frac{()}{()} \qquad \text{(DUP–FRESH)}$$

where $a \in$ ATOM, $u \in$ VAR and $v \in$ VAR′, and where the drawing of a fresh copy from the environment is implicit. These are then ligated together (in a prescribed right-to-left order to ensure microscopic reversibility), and the composite structure disentangled,

$$\left(\vec{s}\ \frac{t}{t}\ \frac{\vec{u}}{\vec{u}}\right) \leftrightarrow \left(\vec{s}\ \frac{t\ \vec{u}}{t\ \vec{u}}\right) \qquad \text{(DUP–LIG}_1\text{)} \qquad \left(\frac{\vec{t}}{\vec{t}}\right) \leftrightarrow \frac{(\vec{t})}{(\vec{t})} \qquad \text{(DUP–TOPO}_1\text{)}$$

$$\left\{\vec{s}\ \frac{t}{t}\ \frac{\vec{u}}{\vec{u}}\right\} \leftrightarrow \left\{\vec{s}\ \frac{t\ \vec{u}}{t\ \vec{u}}\right\} \qquad \text{(DUP–LIG}_2\text{)} \qquad \left\{\frac{\vec{\pi}}{\vec{\pi}}\right\} \leftrightarrow \frac{\{\vec{\pi}\}}{\{\vec{\pi}\}} \qquad \text{(DUP–TOPO}_2\text{)}$$

$$\frac{\gamma}{\gamma} : \frac{t}{t} \leftrightarrow \frac{\gamma : t}{\gamma : t} \qquad \text{(DUP–TOPO}_3\text{)}$$

finally resulting in a fully duplicated and disentangled structure $\frac{t}{t}$ from $\bar{t}$ for t any term, party, or party-bag. Finally, the instantiation rules can be realised microscopically reversibly thus,

$$(I = O : S) \leftrightarrow (\overline{I} = \overline{O} : \overline{S}) \qquad \text{(INST–COMP}_1\text{)}$$

$$\left\{\left(\frac{I}{I} = \frac{O}{O} : \frac{S}{S}\right)\right\} \leftrightarrow \{(I = O : S), \langle I \varnothing\varnothing S \varnothing O\rangle\} \qquad \text{(INST–COMP}_2\text{)}$$

$$(!\ \tau) \leftrightarrow (!\ \overline{\tau}) \qquad \text{(INST–HALT}_1\text{)}$$

$$\left\{\left(!\ \frac{\tau}{\tau}\right)\right\} \leftrightarrow \{(!\ \tau), \langle\tau\rangle\} \qquad \text{(INST–HALT}_2\text{)}$$

$$
\begin{array}{rcll}
 & & S \leftrightarrow S & (\textsc{refl}) \\
S \leftrightarrow T & \Longrightarrow & T \leftrightarrow S & (\textsc{symm}) \\
S \leftrightarrow T \wedge T \leftrightarrow U & \Longrightarrow & S \leftrightarrow U & (\textsc{trans}) \\
S \leftrightarrow T & \Longrightarrow & \forall \Gamma.\, \Gamma \cup S \leftrightarrow \Gamma \cup T & (\textsc{ext}) \\
s \leftrightarrow t & \Longrightarrow & \forall \Gamma.\, \Gamma \cup \{s\} \leftrightarrow \Gamma \cup \{t\} & (\textsc{ext}') \\
\mathcal{P} \vdash (I = O : R) & \Longrightarrow & \{\} \leftrightarrow \{\langle I \varnothing \varnothing R \varnothing O \rangle\} & (\textsc{inst–comp}) \\
\mathcal{P} \vdash (!\, \tau) & \Longrightarrow & \{\} \leftrightarrow \{\langle \tau \rangle\} & (\textsc{inst–halt}) \\
 & & c : (\vec{l}\ t\ \vec{r}) \leftrightarrow (c : (\vec{l} \bullet \vec{r})) : t & (\textsc{ohc}_1) \\
 & & c : [\vec{l}\ t\ \vec{r}] \leftrightarrow (c : [\vec{l} \bullet \vec{r}]) : t & (\textsc{ohc}_2) \\
\exists b'.\, t \overset{\tau}{\sim} b' & \Longrightarrow & \{x : t, \langle \tau \rangle\} \leftrightarrow \{x : [t], \langle \tau \rangle\} & (\textsc{term}) \\
 & & \langle I \varnothing \varnothing R \varnothing O \rangle \leftrightarrow \langle O \varnothing \varnothing R \varnothing I \rangle & (\textsc{rev}_1) \\
 & & \langle \varnothing I B R \Gamma O \rangle \leftrightarrow \langle \varnothing O B R \Gamma I \rangle & (\textsc{rev}_2) \\
x : t \overset{\pi}{\sim} b & \Longrightarrow & & (\textsc{inp}) \\
 & & \{x : t, \langle (I \cup \{\pi\}) I' B R \Gamma O \rangle\} \leftrightarrow \{\langle I (I' \cup \{\pi\}) (B \cup b) R \Gamma O \rangle\} & \\
\pi \in R \wedge x : t \overset{\pi}{\sim} b & \Longrightarrow & & (\textsc{sub}_1) \\
 & & \langle I I' (B \cup b) R \Gamma O \rangle \leftrightarrow \langle I I' B R (\Gamma \cup \{x : [t]\}) O \rangle & \\
x \notin \textsc{var}' & \Longrightarrow & & (\textsc{sub}_2) \\
 & & \{\langle I I' B R (\Gamma \cup \{x : [t]\}) O \rangle \leftrightarrow \{x : [t], \langle I I' B R \Gamma O \rangle\} & \\
\Gamma \leftrightarrow \Gamma' & \Longrightarrow & \langle I I' B R \Gamma O \rangle \leftrightarrow \langle I I' B R \Gamma' O \rangle & (\textsc{sub}_3)
\end{array}
$$

Listing V.8: Summary of ℵ semantics.

$$
\frac{\alpha \in \textsc{atom}}{\alpha \overset{\alpha}{\sim} \varnothing}\ (\textsc{unif–atom}) \qquad \frac{v \in \textsc{var}}{t \overset{v}{\sim} \{v \mapsto t\}}\ (\textsc{unif–var})
$$

$$
\frac{\bigwedge_i t_i \overset{\pi_i}{\sim} b_i}{[t_1 \cdots t_n] \overset{\pi_1 \cdots \pi_n}{\sim} \bigcup_i b_i}\ (\textsc{unif–sub}) \qquad \frac{\lambda \in \textsc{var}' \quad t \overset{\pi}{\sim} b}{\gamma : t \overset{\lambda : \pi}{\sim} \{\lambda \mapsto \gamma\} \cup b}\ (\textsc{unif–ctxt}_1)
$$

$$
\frac{\gamma \overset{\pi_1}{\sim} b_1 \quad t \overset{\pi_2}{\sim} b_2}{\gamma : t \overset{\pi_1 : \pi_2}{\sim} b_1 \cup b_2}\ (\textsc{unif–ctxt}_2)
$$

Listing V.9: Summary of unification semantics for ℵ.

1. $\mathbf{!} + a\ b\ ();$
2. $\mathbf{!}\ ()\ a\ b\ +;$
3. $\{\alpha : +\ \mathrm{Z}\ b\ ()\} = \{\alpha : ()\ \mathrm{Z}\ b\ +\};$ (ADD–BASE)
4. $\{\alpha : +\ (\mathrm{S}a)\ b\ ()\} = \{\alpha : ()\ (\mathrm{S}a)\ (\mathrm{S}b')\ +\}:$ (ADD–STEP)

4a. $\quad \beta : +\ a\ b\ ().$

4b. $\quad \beta : ()\ a\ b'\ +.$ (ADD–STEP–SUB)

(a) Desugared definition of reversible natural addition in ℵ. It will be convenient to make the definitions $I_4 = \{\alpha : +\ (\mathrm{S}a)\ b\ ()\}$, $O_4 = \{\alpha : ()\ (\mathrm{S}a)\ (\mathrm{S}b')\ +\}$ and $R_4 = \{\beta : +\ a\ b\ (), \beta : ()\ a\ b'\ +\}$.

$$
\begin{array}{rcll}
 & & \{():[+34()]\} & \\
\mathcal{P} \vdash \mathbf{!}{+}ab() & \Longrightarrow & \leftrightarrow \{\langle {+}ab()\rangle, ():[+34()]\} & (\text{INST–HALT}) \\
 & & \leftrightarrow \{\langle {+}ab()\rangle, ():(+34())\} & (\text{TERM}) \\
\mathcal{P} \vdash \mathbf{!}{+}ab() & \Longrightarrow & \leftrightarrow \{():(+34())\} & (\text{INST–HALT}) \\
\mathcal{P} \vdash (I_4 = O_4 : R_4) & \Longrightarrow & \leftrightarrow \{\langle I_4 \varnothing\varnothing R_4 \varnothing O_4\rangle, ():(+34())\} & (\text{INST–COMP}) \\
():(+34()) \overset{\alpha:+(\mathrm{S}a)b()}{\sim} \{\cdots\} & \Longrightarrow & \leftrightarrow \{\langle \varnothing I_4\{\alpha{\mapsto}(), a{\mapsto}2, b{\mapsto}4\} R_4 \varnothing O_4\rangle\} & (\text{INP}) \\
\cdots & \Longrightarrow & \leftrightarrow \{\varnothing I_4\{\alpha{\mapsto}()\} R_4\{\beta{:}[+24()]\} O_4\} & (\text{SUB}_1) \\
\{\beta{:}[+24()]\} \leftrightarrow \{\beta{:}[()26+]\} & \Longrightarrow & \leftrightarrow \{\varnothing I_4\{\alpha{\mapsto}()\} R_4\{\beta{:}[()26+]\} O_4\} & (\text{SUB}_3) \\
\cdots & \Longrightarrow & \leftrightarrow \{\langle \varnothing I_4\{\alpha{\mapsto}(), a{\mapsto}2, b'{\mapsto}6\} R_4 \varnothing O_4\rangle\} & (\text{SUB}_1) \\
 & & \leftrightarrow \{\langle \varnothing O_4\{\alpha{\mapsto}(), a{\mapsto}2, b'{\mapsto}6\} R_4 \varnothing I_4\rangle\} & (\text{REV}_2) \\
():(()37+) \overset{\alpha:()(\mathrm{S}a)(\mathrm{S}b')+}{\sim} \{\cdots\} & \Longrightarrow & \leftrightarrow \{\langle O_4 \varnothing\varnothing R_4 \varnothing I_4\rangle, ():(()37+)\} & (\text{INP}) \\
 & & \leftrightarrow \{\langle I_4 \varnothing\varnothing R_4 \varnothing O_4\rangle, ():(()37+)\} & (\text{REV}_1) \\
\mathcal{P} \vdash (I_4 = O_4 : R_4) & \Longrightarrow & \leftrightarrow \{():(()37+)\} & (\text{INST–COMP}) \\
\mathcal{P} \vdash \mathbf{!}()ab+ & \Longrightarrow & \leftrightarrow \{\langle()ab+\rangle, ():(()37+)\} & (\text{INST–HALT}) \\
 & & \leftrightarrow \{\langle()ab+\rangle, ():[()37+]\} & (\text{TERM}) \\
\mathcal{P} \vdash \mathbf{!}()ab+ & \Longrightarrow & \leftrightarrow \{():[()37+]\} & (\text{INST–HALT})
\end{array}
$$

(b) One possible derivation of $() : [+\ 3\ 4\ ()] \leftrightarrow () : [()\ 3\ 7\ +]$ using the semantics for ℵ.

Listing V.10: ℵ semantics as applied to the example of natural addition.

The one-hole-context rules are nearly trivially microscopically reversible, except for the choice of partition. This can be achieved by a movable marker like so,

$$c : (x\ \vec{x}) \leftrightarrow c : (\!|\hat{x}\ \vec{x}|\!) \qquad (\text{OHC}_{11}) \qquad c : [x\ \vec{x}] \leftrightarrow c : [\![\hat{x}\ \vec{x}]\!] \qquad (\text{OHC}_{21})$$

$$c : (\!|\vec{x}\ \hat{x}\ y\ \vec{y}|\!) \leftrightarrow c : (\!|\vec{x}\ x\ \hat{y}\ \vec{y}|\!) \qquad (\text{OHC}_{12}) \qquad c : [\![\vec{x}\ \hat{x}\ y\ \vec{y}]\!] \leftrightarrow c : [\![\vec{x}\ x\ \hat{y}\ \vec{y}]\!] \qquad (\text{OHC}_{22})$$

$$c : (\!|\vec{\ell}\ \hat{t}\ \vec{r}|\!) \leftrightarrow (c : (\vec{\ell} \bullet \vec{r})) : t \qquad (\text{OHC}_{13}) \qquad c : [\![\vec{\ell}\ \hat{t}\ \vec{r}]\!] \leftrightarrow (c : [\vec{\ell} \bullet \vec{r}]) : t \qquad (\text{OHC}_{23})$$

The rules (INP,SUB$_1$,SUB$_2$) require a microscopically reversible realisation of a bag with random draws. In order to be microscopically reversible, the draw needs to be deterministic at any given time, even if draws at different times are uncorrelated. This is achieved by representing a bag as a string that can be shuffled via swap operations, i.e.

$$\{\vec{x} \cdot x\ \vec{y}\} \leftrightarrow \{\vec{x}\ x \cdot \vec{y}\} \qquad (\text{BAG–SHUFF}_1)$$

$$\{\vec{x}\ x \cdot y\ \vec{y}\} \leftrightarrow \{\vec{x}\ y \cdot x\ \vec{y}\} \qquad (\text{BAG–SHUFF}_2)$$

where $\cdot$ is the focus of the swap operation, required for microscopic reversibility. Random draws are then implemented by simply picking the first element of the string.

The last rules to demonstrate microscopic reversibility for are the unification semantics. We leave this as an exercise for the reader, with the hint that its realisation is similar to that of the instantiation rules. □

Computability The earlier examples give reasonable assurance that ℵ is Turing complete and that programming in it is 'easy' in the sense that programs can be readily composed. For avoidance of doubt, however, we prove that ℵ is Turing complete in two ways: the first proves a stronger claim, that ℵ is Reversible-Turing complete, meaning that it can efficiently and faithfully simulate a Reversible Turing Machine without generating garbage; the second proves Turing completeness in a more conventional fashion in order to demonstrate its high level of composability.

Theorem V.3. *The ℵ-calculus is Reversible-Turing Complete, i.e. it can simulate any Reversible Turing Machine (RTM) as defined by Bennett [39].*

Proof. An RTM is a collection of one or more bi-infinite tapes divided into squares, each of which can be blank or can contain a symbol, as well as a machine head with an internal state. Symbols are drawn from a finite alphabet, and internal states from a distinct finite alphabet. At any time, the RTM head looks at a single square on each tape and executes one of a finite set of reversible rules. The rule chosen is uniquely determined by the current state of the system. Each rule depends on the current internal state of the machine head, which it may alter, and for each tape it can either ignore its value, possibly moving the tape one square to the left or right, or it can depend on the square taking a certain value u, which it can replace with a certain other value v.

For example, a 6-tape RTM with symbol alphabet $\{A, B, C, D, E, F\}$ and state alphabet $\{\text{START}, S_1, S_2, \text{STOP}\}$ might have a rule $S_1\ C\ \varnothing\ /\ D\ /\ /\ \rightarrow\ B\ A\ -\ F\ 0\ +\ S_2$. This rule

```
[BLANK x · xs] `POP` BLANK [x · xs];          ! TAPE ℓ x r;
[(SYM x) · xs] `POP` BLANK xs;                ! SYM x;
[] `POP` BLANK [];                            ! BLANK;

(TAPE ℓ x r) `LEFT` (TAPE ℓ' x' r'):          t `RIGHT` t':
    ℓ `POP` x' ℓ'.                                t' `LEFT` t.
    r' `POP` x r.
```

Listing V.11: This ℵ program defines all the ingredients necessary to simulate any Reversible Turing Machine. We represent a bi-infinite tape as its one-hole-context; that is, a tape is given by the square in the current position, x, as well as 'all' the squares to the left of it, ℓ, and 'all' the squares to the right of it, r. Obviously we can't actually represent *all* the squares to the left and the right. Instead we make use of the fact that, at any one time, only a finite bounded region of the tape is non-blank. As such, ℓ contains all the squares to the left up to the last symbol, and similarly for r. If we keep going left, past the final symbol, then ℓ will be the empty list, and BLANK squares will be created as needed. The rules LEFT and RIGHT are convenience functions for changing the focus of a tape.

applies if and only if the current internal state is S_1, the values of tapes 1 and 4 are the symbols C and D respectively, and tape 2 is blank. If so, it will change the internal state to S_2, replace the values of tapes 1, 2 and 4 with the symbols B, A and F respectively, and move tapes 3 and 6 one square to the left and right respectively, leaving tape 5 alone. The reverse of the rule is given by $S_2\ B\ A\ /\ F\ /\ /\ \rightarrow\ C\ \varnothing\ +\ D\ 0\ -\ S_1$. The necessary conditions for the ruleset to be deterministic and reversible are that all the domains of the forward rules are mutually exclusive, as are all the domains of the reversed rules. Of course, the domains of the forward and reverse rules will intersect in any useful program.

It is easy to translate any description of an RTM into ℵ. Bi-infinite tapes can be easily represented and manipulated, per the program in Listing V.11, and any rule (such as the above) can be mechanically translated as so,

$$
\begin{aligned}
&S_1\ (\textsc{Tape}\ \ell_1\ (\textsc{Sym}\ C)\ r_1)\ (\textsc{Tape}\ \ell_2\ \textsc{Blank}\ r_2)\ t_3\ (\textsc{Tape}\ \ell_4\ (\textsc{Sym}\ D)\ r_4)\ t_5\ t_6 \\
=\ &S_2\ (\textsc{Tape}\ \ell_1\ (\textsc{Sym}\ B)\ r_1)\ (\textsc{Tape}\ \ell_2\ (\textsc{Sym}\ A)\ r_2)\ t'_3\ (\textsc{Tape}\ \ell_4\ (\textsc{Sym}\ F)\ r_4)\ t_5\ t'_6\text{:} \\
&\quad t_3\ \text{`}\textsc{Left}\text{`}\ t'_3. \\
&\quad t_6\ \text{`}\textsc{Right}\text{`}\ t'_6.
\end{aligned}
$$

whilst the special START and STOP states are marked thus,

$$
\begin{aligned}
&!\ \textsc{Start}\ t_1\ t_2\ t_3\ t_4\ t_5\ t_6; \\
&!\ \textsc{Stop}\ t_1\ t_2\ t_3\ t_4\ t_5\ t_6;
\end{aligned}
$$

That this translation faithfully reproduces the operation of the RTM is obvious from the high level semantics of ℵ. □

```
xs `Mu (Const n)` n' (Garbage xs):
    `Dup n` n'.

[x] `Mu Succ` (Sx) (Garbage);

xs `Mu (Proj i)` y (Garbage ys zs):
    `Dup i` i'.
  ! ~Go i' [] xs = ~Go Z [y · ys] zs.
    ~Go (Sn) ys [x · xs] = ~Go n [y · ys] xs;

xs `Mu (Sub g fs)` z (Garbage xs h hs):
    ys `Mu g` z h.
    `~Go fs xs` ys hs.
    `~Go [] xs` [] [];
    `~Go [f · fs xs]` [y · ys] [h · hs]:
        `Dup xs` xs'.
        xs' `Mu f` y h.
        `~~Go fs xs` ys hs.

[n · xs] `Mu (Rec f g)` y (Garbage n' xs hs):
    `Dup xs` xs'.
    xs' `Mu f` x h.
  ! ~Go n Z g x xs [h] = ~Go Z n' g y xs hs.
    ~Go (Sn) m g x xs hs = ~Go n (Sm) g y xs [h · hs]:
        `Dup m` m'. `Dup xs` xs'.
        [m' x · xs'] `Mu g` y h.

xs `Mu (Minim f)` i (Garbage xs hs):
  ! ~Go f Z xs (SZ) [] = ~Go f (Si) xs Z hs.
    ~Go f i xs (Sn) hs = ~Go f (Si) xs y [n h · hs]:
        `Dup i` i'. `Dup xs` xs'.
        [i' · xs'] `Mu f` y h.
```

```
data Const n;
-- C_n(xs) = n

data Succ;
-- S(x) = x + 1

data Proj i;
-- P_i(xs) = x_i
-- N.B: xs is 1-indexed.

data Sub g fs;
-- (g ∘ fs)(xs) =
--     g(f_1(xs) ··· f_n(xs))

data Rec f g;
-- ρ(f,g)(0,xs) = f(xs)
-- ρ(f,g)(n+1,xs) =
--     g(n, ρ(f,g)(n,xs), xs)

data Minim f;
-- μ(f)(xs) =
--     min{n : f(n,xs) = 0}
```

Listing V.12: An ℵ program* implementing the μ-recursive functions. The μ-recursive functions are generated by the three functions, C_n, S and P_i, and the three operators, $\circ$, ρ and μ, whose definitions are given in the comments above. The ℵ values corresponding to these functions can be directly composed using the operators (e.g. (Rec (Const 0) (Proj 1))), and then evaluated with the Mu atom.

*Some additional sugar has been used in this program, mainly in the form of nested definitions and the more concise expression of looping constructs. See Section V.6 for an in-depth definition of these sugared forms.

Corollary V.4. *The ℵ-calculus is Turing Complete.*

Proof. Bennett [39] proved that a Reversible Turing Machine can simulate any Turing Machine (and vice-versa), and so the corollary follows immediately from ℵ's Reversible-Turing completeness. To drive the point home, however, we also implement the μ-Recursive Functions (Listing V.12)—three functions, and three composition operators, which together are capable of representing any computable function over the naturals and hence are Turing complete. For example, addition, multiplication and factorials can be defined in terms of each other (in our ℵ realisation) as

$$\begin{aligned} \mathit{add} &\mapsto (\textsc{Rec}\ (\textsc{Proj}\ 1)\ (\textsc{Sub}\ \textsc{Succ}\ [(\textsc{Proj}\ 2)])) \\ \mathit{mul} &\mapsto (\textsc{Rec}\ (\textsc{Const}\ 0)\ (\textsc{Sub}\ \mathit{add}\ [(\textsc{Proj}\ 2)\ (\textsc{Proj}\ 3)])) \\ \mathit{fac} &\mapsto (\textsc{Rec}\ (\textsc{Const}\ 1)\ (\textsc{Sub}\ \mathit{mul}\ [(\textsc{Proj}\ 2)\ (\textsc{Sub}\ \textsc{Succ}\ [(\textsc{Proj}\ 1)])])) \end{aligned}$$

and then one can evaluate, e.g., $(\textsc{Mu}\ \mathit{fac})\ [7]\ ()$, obtaining $()\ 5040\ \mathit{garbage}\ (\textsc{Mu}\ \mathit{fac})$. □

V.5. Implementation Concerns

Although the ℵ-calculus is microscopically reversible by Theorem V.2, ℵ has many degrees of freedom wherein macroscopic reversibility can be violated. There are two sources of entropy generation; the first is ambiguity in rule application, and the second is intrinsic to any useful implementation of concurrency. We already saw in Section V.3 how concurrency leads to entropy generation, but in this section we will expand on the other mechanism of entropy generation and how it can be avoided at the compiler-level. This is important because the primary motivation for reversible programming is to avoid unexpected entropy leaks, and so generally the appearance of ambiguity is a bug rather than intentional. In addition to ambiguity, we will also discuss other aspects of a realistic implementation to avoid or minimise the presence of random walks: optimising the evaluation order of sub-rules ('serialisation') and inferring the direction of computation. A reference implementation of these algorithms (as well as additional sugar) is made available as a language and interpreter, `alethe` (see Section V.6).

Nevertheless, in some cases the appearance of ambiguity/non-determinism may be intentional and so we should provide the programmer a mechanism to *explicitly* weaken the ambiguity checker when desired. An example of this might be in implementing a fair coin toss for generating randomness,

$$\begin{aligned} &\text{`}\textsc{Coin}\text{`}\ \textsc{Tails}; \\ &\text{`}\textsc{Coin}\text{`}\ \textsc{Heads}; \end{aligned}$$

If the computational architecture is Brownian, such as a molecular computer, then this can be used to exploit the thermal noise of the environment to get as close as classical physics allows to a true random number generator (RNG). The way this RNG is used is by constructing a term $\textsc{Coin}\ ()$. This term then matches both rules, and so depending on which is chosen the term may evolve to $()\ \textsc{Tails}\ \textsc{Coin}$ or to $()\ \textsc{Heads}\ \textsc{Coin}$. Moreover,

this process will depend on the probability distribution realised by the implementation; if we assume this is uniform[3], then the coin toss will be fair with each term observed with probability $\frac{1}{2}$. If the probability distribution is non-uniform, the situation is less clear and depends on the exact rates of the forward and reverse transitions for each rule. The analysis is beyond the scope of this chapter, but if the dynamics takes the form

$$() \text{ TAILS COIN} \underset{\lambda_t}{\overset{\lambda'_t}{\rightleftharpoons}} \text{COIN } () \underset{\lambda'_h}{\overset{\lambda_h}{\rightleftharpoons}} () \text{ HEADS COIN},$$

then the steady-state ratio of heads to tails will be $\frac{\lambda_h/\lambda'_h}{\lambda_t/\lambda'_t}$. By microscopic reversibility, we expect $\lambda_h = \lambda'_h$ and similarly for λ_t (unless the system is biased away from equilibrium), and therefore even a non-uniform distribution will have a uniformly distributed steady-state. Note the interesting property that, in reverse, an ambiguous/non-deterministic reversible rule is indistinguishable from irreversibility. Here, for example, the inverse of a fair coin toss is a process that consumes[4] a bit.

Ambiguity Checker To exclude these sorts of process, we thus introduce an ambiguity checker. Not only does the introduction of an ambiguity checker reassure the programmer that they are writing information-preserving code, but the algorithm also serves as an invaluable debugging tool. In a large codebase, it can be hard to keep track of all the different patterns employed (the standard library has nearly two thousand), let alone ensure that there are no ambiguities. The ambiguity checker in `alethe` performs this check for the programmer and, most importantly, is able to tell the programmer where the ambiguities lie.

In an unambiguous program, there are only a few valid scenarios for each term. It can match two computational rules (where we think of the forward and reverse directions of each rule as distinct rules for the current purpose), in which case it is an intermediate state of a computation. It can match one computational rule and one or more halting rules, in which case it is a terminal state of the computation. It can match one computational rule, in which case the program has entered an erroneous state and stalled. It can match one or more halting rules, in which case the term has no computational capacity, and most likely represents a data value. Lastly, it can match no rules, in which case it is an invalid term that cannot be constructed under normal conditions.

[3] A more sophisticated implementation might reify and expose control over the distribution to the programmer, allowing arbitrary probabilities to be assigned. Going further, an interesting research direction would be to what extent ℵ can be augmented quantum mechanically.

[4] There is a subtle point to make here: if the bit being consumed has no computational content and is uniformly distributed, then this consumption does not generate any entropy. That is, drawing a random bit from the environment and then replacing it is a completely isentropic process. For non-uniform distributions, the process can also be isentropic provided that the producer/consumer has a matching distribution. The problem arises when the distribution of the bit does not fit that of the producer/consumer, with deterministic computation being an extreme case wherein the bit can only take on a prescribed value (this is true even if the bit is pseudorandom).

The remaining possibilities can be summarised two-fold: a term can match two computational rules and one or more halting rules, in which case the computational path from the terminus is ambiguous, or a term can match more than two computational rules (and zero or more halting rules), which is the more obvious ambiguity scenario.

The above cases can be simplified: for a given term, consider all the rules it matches but coalesce all halting rules, if any, into a single rule. Then, a term is unambiguous if it matches at most two such rules, and ambiguous if it matches more than two.

Clearly it is impractical to test this for all possible terms, there being a countable infinity of them; instead, one can determine whether it is possible to construct an ambiguous term. To do so, build a graph G whose nodes are all the patterns occurring on either side of a computational rule—which we will colour white for reasons that will become apparent—and all the patterns occurring in a halting rule—which we will colour black. An edge is drawn between two nodes if it is possible to construct a term satisfying both patterns. This condition can be determined inductively: if either pattern is variable, then draw an edge. If neither pattern is variable, then draw an edge only if they are both the same atom, or they are both composite terms of the same length and their respective sub-patterns matches pairwise. In order to proceed, we first prove a lemma about this graph.

Lemma V.5. *There exists a term that simultaneously satisfies each of a set of patterns* Π *if and only if the subgraph of* G *formed from the nodes* Π *is complete.*

Proof. The ($\implies$) case is obvious. We prove the converse by structural induction. As base cases, we note that the statement is trivially true if $|\Pi| = 0, 1, 2$. Now consider the possible forms of some $\pi \in \Pi$:

(ATOM) If π is an atom, a, then the only term satisfying it is the same atom, a. As π has edges to every other pattern in Π, a must satisfy each of these other patterns too.

(VAR) Suppose there is a term t that satisfies all the patterns $\Pi \setminus \{\pi\}$. As t satisfies the variable pattern π vacuously, t must satisfy all of Π. Without loss of generality then, we can ignore all variable patterns.

($\tau_1 \cdots \tau_n$) If π is a composite term, then there can only be an edge to the other patterns if these are composite terms of the same length or are the variable pattern. By the previous case, we can ignore these variable patterns. Now, index the patterns by $i = 1 \ldots m$ and write $\pi_i = (\tau_1^{(i)} \cdots \tau_n^{(i)})$. Then construct graphs $H_1 \cdots H_n$ such that the nodes of H_j are the patterns $\{\tau_j^{(i)} : i = 1 \ldots m\}$. If we draw edges according to the same rules as for G, then by the inductive definition of the edge condition for G, each of the graphs H_i will be complete. By inductive assumption, a term t_i exists satisfying all patterns in H_i for each i, and therefore the term $(t_1 \cdots t_n)$ must satisfy all the patterns Π.

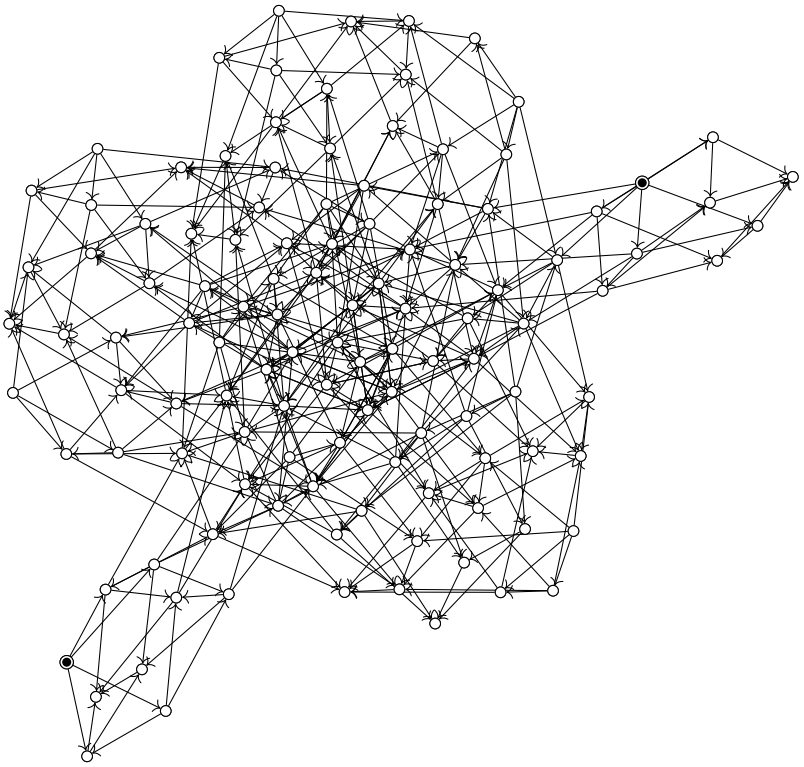
The lemma follows. □

Using Lemma V.5, we see that the different cases of unambiguous and ambiguous terms reduce to considering the structures of the complete subgraphs of G. Moreover, it suffices to only consider subgraphs formed from three nodes, i.e. *triangles*: the first ambiguous case, of two white nodes and one or more black nodes, implies the existence of a triangle with two white nodes and one black node; the second case, of three or more white nodes and zero or more black nodes, implies the existence of a triangle with three white nodes. Conversely, a triangle with two or more white nodes implies one of these ambiguous cases. The only triangles present for the unambiguous cases contain two or more black nodes. Therefore we can enumerate all the triangles of G containing two or more white nodes; if there are any, then there is an ambiguity and the patterns present in the triangles should be reported to the programmer. If there are none, then the program is unambiguous and deterministic.

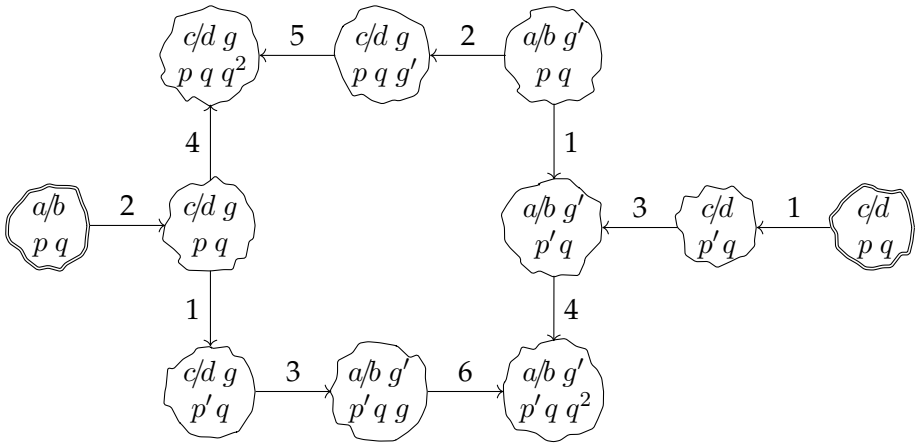
In fact, there is one more potential source of ambiguity. Suppose that the two patterns of a sub-rule are not orthogonal (a common occurrence), then it is in principle possible that a halting term might match either side, and therefore lead to a 2-way ambiguity. The evaluation rules of our reference interpreter for `alethe` do not suffer from this ambiguity, as it determines ahead of time the order and directionality of all sub-rules, but it is possible for a faithful implementation of א to suffer so. There are two ways to avoid this: the first is to require sub-rule patterns be orthogonal, though this also forbids many legal unambiguous programs—requiring additional levels of tedious and unnecessary indirection to circumvent the restriction—and so is undesirable; the second is to apply type inference (work towards which is presented in Section V.7) to gain the extra knowledge needed to distinguish between legal and ambiguous programs. Namely, type inference allows the compiler to infer what sequences of patterns and terms occur in a program, and hence to determine which halting patterns are computationally connected. With this knowledge, and knowledge of the types of variables in a rule, it can determine what forms the halting terms of a sub-rule will take and therefore whether there is a unique mapping between terms and patterns or not.

Serialisation Heuristics The Brownian semantics of א, particularly in the absence of coupling to a bias source, are not very performant. Specifically, sub-rule transitions execute a random walk in a phase space whose size—in the worst case—scales exponentially with the number of variables present in the current rule. Whilst explicit bias coupling circumvents this by essentially serving to annotate the preferred order and directionality of the sub-rules, it undermines the declarative nature of א. Instead, it is possible to algorithmically infer an appropriate execution path. Moreover, it can sometimes be the case (such as in the example of fractional addition) that the execution path is non-trivial and so granting the compiler the power to perform this routing automatically makes the job of the programmer easier: the programmer need only specify how all the variables relate to one another, and may even specify this excessively, and the compiler can figure out which relations are sensible to use.

To illustrate the problem, consider the following program excerpt which adds two



(a) The full transition graph. Due to the size of the graph, labels have been suppressed. Halting states are marked black.



(b) A more practical excerpt of the full transition graph.

Figure V.13: Two views of the transition graph for the fraction-addition routine. Nodes correspond to bags of variables in a 'known' state. An edge with label ℓ is drawn from node α to node β if sub-rule ℓ can map the knowledge state α to the knowledge state β (with possible variable duplication/elision). It can be seen that there are two paths from $\{a/b, p, q\}$ to $\{c/d, p, q\}$ worth considering: $\vec{2}$–$\vec{4}$–$\overleftarrow{5}$–$\overleftarrow{2}$–$\vec{1}$–$\overleftarrow{3}$–$\overleftarrow{1}$, and $\vec{2}$–$\vec{1}$–$\vec{3}$–$\vec{6}$–$\overleftarrow{4}$–$\overleftarrow{3}$–$\overleftarrow{1}$. These are of roughly equal computational complexity, and so either is a viable choice.

simplified fractions:

$$a/b \,\backslash{+}\, (\textsc{Frac}\ p\ q)\backslash\ c/d\text{:}$$

1. $p - p'$.
2. $a/b \,\backslash{\sim}\, (\textsc{Frac}\ p\ q)\backslash\ c/d\ g$.
3. $c/d \,\backslash{\sim}\, (\textsc{Frac}\ p'\ q)\backslash\ a/b\ g'$.
4. $(\mathrm{S}q) \,\square\, q^2$.
5. $g' \,\backslash{\times}\, g\backslash\ q^2$.
6. $g \,\backslash{\times}\, g'\backslash\ q^2$.

where a fraction is represented by $\frac{p}{q+1} \equiv (\textsc{Frac}\ p\ q)$ with $p \in \mathbb{Z}$ and $q \in \mathbb{N}$. Maintaining the invariant that fractions are in their simplest form is non-trivial, but involves finding the greatest common divisor of the numerators and denominators of two intermediate results, g and g', and then showing that their product $gg' = (q+1)^2$. It turns out that this fact can then be used to eliminate the intermediate garbage. In the unassisted semantics of ℵ, the graph of all intermediate states generated by applying these rules transitively is given by Figure V.13a. This graph has 117 nodes. It can therefore be seen that a computation is very likely to get stuck in one of the 115 intermediates, and will take a long time to find its way from the input variables, $\{a/b, p, q\}$, to the output variables, $\{p, q, c/d\}$. Many of these states are not interesting computational, and indeed only a small subgraph of 11 (or even 8) nodes is needed to perform the desired computation as shown in Figure V.13b.

From this transition subgraph, we can see that there are two possible paths to get from the input variables to the output variables, and the goal of the serialisation algorithm is to find these paths and to pick the most optimal. Here both of these paths are of equal complexity, and so either is valid. Having made a choice, the compiler can then direct the computer to perform the prescribed series of transitions, rather than executing a random walk. In a Brownian computational system, for example, this would be achieved by adding 'fake' dependencies to the sub-rules to force a linear ordering (or possibly partially parallel, where this makes sense) and automatically coupling the sub-rules to the bias source in the correct direction. As we shall see, automatically determining the cost of a path is non-trivial and sometimes even impossible, and so a perfect solution to the serialisation problem is not generally possible. Nonetheless, for many programs it is possible to find the appropriate path, or to give the compiler enough information to make a better choice, and so further discussion of serialisation is warranted.

The simplest and least efficient serialisation strategy is as follows: suppose you start with the input variables $\{x, y, z\}$. Enumerate all the sub-rules that can be evaluated given this current 'knowledge state', e.g. $x \,\backslash\textsc{Foo}\backslash\ y\ w$ and $z \sim x$ would qualify but $w \,\backslash\textsc{Map Bar}\backslash\ v$ would not because neither the variable w nor the variable v are currently available. Evaluate all of these sub-rules in parallel, making sure to duplicate variables as necessary to avoid unlearning any. Here, we would end up with the knowledge state $\{w, x, y, z\}$. Now ignore these sub-rules going forward and repeat this process until all sub-rules have been used once (and only once), and all variables are known. If

some sub-rule has not been used or a variable remains unknown, then there is a logic error and the programmer should be appropriately chastised. Now, perform this entire process again, but starting from the output variables. If all is well, then we will have obtained two routes: one from the input variables to all variables, and one from the output variables to all variables. As these routes are reversible, we are immediately rewarded with a route from the input to the output variables, using each sub-rule twice. Whilst clearly inefficient, this strategy demonstrates that each sub-rule need be used at most twice (once in each direction).

A better strategy is to construct a 'transition graph' as follows: take as nodes the powerset of all variables, i.e. $\varnothing$, $\{x\}$, $\{y\}$, $\{x, y\}$, etc. Mark the input and output nodes specially for later. Draw an edge between two nodes if a sub-rule (with possible variable duplication/elision) can be used to map between the two knowledge states, and label the edge with this sub-rule and in which direction it is to be applied. With the transition graph thus constructed, all possible routes between the input and output nodes can be enumerated[5]. To proceed, a cost heuristic is needed in order to rank routes by preference. For example, if the goal is simply to minimise the number of sub-rules used then Dijkstra's algorithm will suffice. Note that, in the reference interpreter, transition graphs are constructed more restrictively. Namely, variable duplication/elision is not implemented as this always[6] leads to an exponentially large transition graph and, in practice, it is rarely needed. Where it is needed, the programmer must instead explicitly use DUP (and create a new variable). This provides dramatically better performance at the expense of a slight inconvenience.

We saw an example of this algorithm earlier in Figure V.13. Another example, deserving of further comment, is given in Listing V.14. There are two routes of apparent equal cost, $\vec{1}\text{–}\vec{2}\text{–}\overleftarrow{2}\text{–}\overleftarrow{3}$ and $\vec{1}\text{–}\vec{2}\text{–}\vec{3}\text{–}\overleftarrow{3}$, but this is misleading. If the tree has n nodes across $\ell \sim \log n$ levels, a single (non-recursive) step of POLISH has time complexity α, and a single (non-recursive) step of POLISHREADS has time complexity β, then the first route can be shown to have time complexity $\mathcal{O}(2^\ell n(\alpha + \beta))$ whilst the second has complexity $\mathcal{O}(n(\alpha + 2\ell\beta))$. That is, if the serialisation algorithm picks a route which repeats a recursive step then consequently there will be an exponential overhead. Moreover, this becomes less obvious in the cases of corecursion, or higher order functions where a function may be passed into itself (recursively or corecursively)—compare, for example, the Y combinator. As such it is not generally possible to algorithmically discriminate this scenario (although the extra knowledge afforded by the type inference algorithm developed in Section V.7 may help). In fact, the reference interpreter for `alethe` does not even try to do so. Instead, it exposes a method by which the programmer can adjust

[5] In fact there are infinitely many routes. It is therefore appropriate to restrict the routes under consideration to a sane (and finite) subset; namely, we avoid revisiting any node, and we also exclude routes that make use of a sub-rule more than twice.

[6] The serialisation algorithm in `alethe` is still subject to an exponential worst case complexity. The size of the transition graph constructed depends on the inter-dependency of the variables: the more dependency, the closer to linear in the number of variables the graph size is. If there is minimal dependency, such as in the automatically generated implemention of DUP for **data** , $a\ b\ c\ d\ e$, then the worst case will be realised.

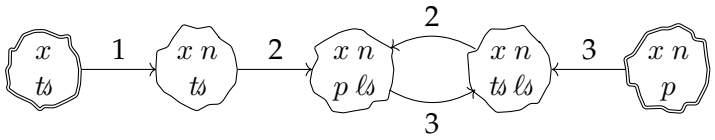
```
data Tree α = Tree α [Tree α]
polish :: Tree α → [(Int, α)]
polish (Tree x xs) = (length xs, x) : concatMap polish xs
```

(a) A snippet of `Haskell` code for converting an arbitrary tree into Polish notation. Polish notation is an isomorphic linear representation of tree-like structures that is well known for its use for arithmetic expressions. For example, the expression $(3 + 4) \times 7 - 2^5$ can be represented as $- \times + 3\ 4\ 7\ \wedge\ 2\ 5$.

```
(Tree x ts) `Polish` [(, x n) · p]:
1.    `Length ts` n.
2.    ts `ConcatMap Polish` p ls..
3.    p `PolishReads n` ts ls.
```

```
data Tree x ts;
data , a b;
  -- 2-tuples
```

(b) An excerpt of the program in (e). This is a rough translation of the `Haskell` implementation into `alethe`. Unfortunately, the implementation of ConcatMap in `alethe` has additional garbage in the form of a list of the lengths of the intermediate lists (*ls*). To eliminate this garbage, we write a function PolishReads which takes a string of trees in Polish notation and converts them back into trees, also generating the same garbage value.



(c) The motif employed above—generating the garbage in two different ways—can be used to eliminate it and achieve the desired (partial) bijection, as shown by its transition graph. There are two possible execution paths, $\vec{1}$–$\vec{2}$–$\overleftarrow{2}$–$\overleftarrow{3}$ and $\vec{1}$–$\vec{2}$–$\vec{3}$–$\overleftarrow{3}$. As explained in the text, the second route is *strongly* preferred—hence the cost annotation on (2).

```
t `Polish` p:
    p `PolishRead` [] t.

[(, x n) · xs] `PolishRead` xs' (Tree x ts):
    xs n `PolishReads` xs' ts.
```

```
xs Z `PolishReads` xs [];
xs (Sn) `PolishReads` xs'' [t · ts]:
    xs `PolishRead` xs' t.
    xs' n `PolishReads` xs' ts.
```

(d) In fact, programming in a 'reversible-first' manner permits us to find the far more efficient and naturally bijective program above. This program was written by considering how to read a tree from its Polish representation: In our experience, picking the 'harder' direction of a bijective computation to implement helps to suppress one's learned intuition for irreversible programming, and hence makes it less likely to fall into the traps of 'shortcuts' with their accompanying garbage.

Listing V.14: An example of serialisation as applied to the interconversion between trees and Polish notation... (continued on next page)

```
`Length []` Z;
`Length [x · xs]` (Sℓ):
    `Length xs` ℓ.

`Sum []` Z;
`Sum [Z · ns]` σ:
    `Sum ns` σ.
`Sum [(Sn) · ns]` (Sσ):
    `Sum [n · ns]` σ.

[] `Map f` [];
[x · xs] `Map f` [y · ys];
    x `f` y.
    xs `Map f` ys.

`Map f []` [];
`Map f [x · xs]` [y · ys];
    `f x` y.
    `Map f xs` ys.

[] `Concat` [] [];
[[] · xss] `Concat` ys [Z · ℓs]:
    xss `Concat` ys ℓs.
[[x · xs] · xss] `Concat` x · ys [(Sℓ) · ℓs]:
    [xs · xss] `Concat` ys [ℓ · ℓs].

xs `ConcatMap f` ys ℓs:
    xs `Map f` yss.
    yss `Concat` ys ℓs.
```

```
data Tree x ts;
data , a b;

n `TreeSize'` (Tree x ts)` (Sn'):
    n `~Go ts` n'.
    n `~Go []` n;
    n `~Go [t · ts]` n'':
        n `TreeSize' t` n'.
        n' `~~Go ts` n''.

`TreeSize t` n:
    Z `TreeSize' t` n.

(Tree x ts) `Polish` [(, x n) · p]:
    `Length ts` n.
    ts `ConcatMap Polish` p ℓs.
    p `PolishReads n` ts ℓs.

[(, x n) · xs] `PolishRead` xs' (Tree x ts) (Sℓ):
    xs `PolishReads n` xs' ts ℓs.
    `Length ℓs` n.
    `Sum ℓs` ℓ.
    `Map TreeSize ts` ℓs.

xs `PolishReads Z` xs [] [];
xs `PolishReads (Sn)` xs'' [t · ts] [ℓ · ℓs]:
    xs `PolishRead` xs' t ℓ.
    xs' `PolishReads n` xs' ts ℓs.
```

(e) The full `alethe` program implementing interconversion between trees and Polish notation, following the `Haskell` snippet in (a). In fact, a little thought and optimisation will reveal that the ℓ outputs of the read functions can be eliminated, which then simplifies the definition of POLISH to that given in (d) and obviates the need for the auxiliary functions LENGTH, SUM, MAP, etc. Nevertheless, the point of this is to show that it is not unreasonable to arrive at the program listed here, and it may not be entirely obvious that a simpler implementation with less garbage is possible.

Listing V.14: (continued) ...and the full `alethe` program being considered here.

the heuristic cost function used in serialisation: each edge in the transition graph has a weight, given by the number of full-stops following the sub-rule in the source; therefore, in this case rule (2) should be annotated with a cost of 2 (via two full-stops) in order to penalise its repeated use. To inspect the route chosen, the `:p` directive may be issued to the interpreter.

Directional Evaluation Consider an intermediate term along a reversible computation path (Figure III.2). In isolation, one cannot know in which direction computation should proceed. Bias coupling helps with this, but `alethe` does not make use of this. Moreover, it would be nice to avoid explicit bias-coupling where practical. As such, we need some concept of computational 'momentum' to maintain a consistent direction. This requires that the programs are deterministic such that phase space is branchless, which fortunately we have guaranteed through the ambiguity checker. Recall that, treating the forward and reverse directions of a rule as distinct, every term matches at most two rules; if we maintain a consistent direction of computation, then one of these rules will be the converse of the rule that was employed to reach the current state. More concretely, let the terms of the computation be labelled as some contiguous subset of $\{|n\rangle : n \in \mathbb{Z}\}$. By determinism, there is a unique rule $r(n \mapsto n+1)$ enacting each transition $|n\rangle \mapsto |n+1\rangle$, and by reversibility the unique rule enacting the transition $|n+1\rangle \mapsto |n\rangle$ is $\bar{r}$, the converse of r. Therefore, given a term $|n+1\rangle$, the rule $\bar{r}(n \mapsto n+1)$ can *only* take us to $|n\rangle$; it can never also be the rule taking us to $|n+2\rangle$. It is, however, possible that the rule taking us to $|n+2\rangle$ is the same as $r(n \mapsto n+1)$. To summarise, all we need do is keep track of which rule we just applied, r; then, when considering the new term, we find any rules it matches and exclude $\bar{r}$ from this set. If the set is empty, we have entered an error state and should report it to the user. Otherwise, our ambiguity check has ensured that it either contains a single computational rule, which we duly apply, or contains one or more halting rules, whence we would halt the computation and report the result. This evaluation logic applies just as well to sub-rules. The one edge case is when evaluating a new term; as only halting terms can be constructed, it will have at most one computational rule to choose from and thus there is no ambiguity.

V.6. A Spoonful of Sugar: `alethe`

For clarity and convenience, we have introduced a number of syntactic shorthands in Sections V.2 and V.3. We now summarise and extend this sugar to construct a programming language, `alethe`, the definition of which is given in Listing V.15. Implementing the additional measures discussed in Section V.5, a reference interpreter for `alethe` is also made available[7] together with a 'standard library' and select examples[8]. We proceed with the definition of `alethe` by desugaring each form in turn.

[7] https://github.com/hannah-earley/alethe-repl
[8] https://github.com/hannah-earley/alethe-examples

$$
\begin{aligned}
&\text{(PATTERN TERM)} && \tau ::= \alpha \mid \text{VAR} \mid (\ \tau^*\ ) \mid \sigma \\
&\text{(ATOM)} && \alpha ::= \text{ATOM} \mid\ \sim \alpha \mid \#\text{``}\,\text{CHAR}^*\,\text{''} \mid \#\alpha \\
&\text{(VALUE)} && \sigma ::= \mathbb{N} \mid \text{``}\,\text{CHAR}^*\,\text{''} \mid [\,\tau^*\,] \mid [\,\tau^+ \,.\, \tau\,] \mid \_ \\
&\text{(PARTY)} && \pi ::= \tau : \tau^* \mid \text{VAR}' : \tau^* \\
&\text{(PARTIES)} && \Pi ::= \pi \mid \Pi\,;\,\pi \\
&\text{(RELATION)} && \rho ::= \tau^* = \tau^* \mid \tau^*\ \text{`}\tau^*\text{'}\ \tau^* \\
&\text{(DEFINITION HEAD)} && \delta_h ::= \rho \mid \{\,\Pi\,\} = \{\,\Pi\,\} \\
&\text{(DEFINITION RULE)} && \delta_r ::= \delta_h\,;\mid \delta_h : \Delta^+ \\
&\text{(DEFINITION HALT)} && \delta_t ::= \mathbf{!}\,\tau^*;\mid \mathbf{!}\,\rho\,; \\
&\text{(DEFINITION)} && \delta ::= \delta_r \mid \delta_t \\
&\text{(COST ANNOTATION)} && \xi ::= .^* \\
&\text{(DECLARATION)} && \Delta ::= \delta \mid \pi\,.\xi \mid \rho\,.\xi \mid \mathbf{!}\,\rho\,.\xi \\
&\text{(STATEMENT)} && \Sigma ::= \delta \mid \mathbf{data}\ \tau^*; \mid \mathbf{import}\ \text{``}\text{MODULE PATH}\text{''}; \\
&\text{(PROGRAM)} && \mathcal{P} ::= \Sigma^*
\end{aligned}
$$

Listing V.15: Definition of `alethe` syntax.

(PATT. TERM) $\tau ::= \alpha \mid \text{VAR} \mid (\ \tau^*\ ) \mid \sigma$
The definition of a pattern term only differs from its definition in ℵ by the inclusion of sugared values, σ.

(ATOM) $\alpha ::= \text{ATOM} \mid\ \sim \alpha \mid \#\text{``}\,\text{CHAR}^*\,\text{''} \mid \#\alpha$
Atoms are fundamentally the same as in ℵ, but there are four ways of inputting an atom. If the name of an atom begins with an uppercase letter or a symbol, then it can be typed directly. In fact, any string of non-reserved and non-whitespace characters that does not begin with a lowercase letter (as defined by Unicode) qualifies as an atom; if it does begin with a lowercase letter, it qualifies as a variable. If one wishes to use an atom name that does not follow this rule, one can use the form #``CHAR*'', where CHAR* is any string (using `Haskell`-style character escapes if needed). Additionally you can prefix a symbol with # to suppress its interpretation as a relation (e.g. #+). Finally, if an atom is prefixed with some number of tildes then it is locally scoped: that is, you can nest rule definitions and make these unavailable outside their scope, and you can refer to locally scoped atoms in outer scopes by using more tildes (there is no scope inheritance). You can even use an empty atom name if preceded by a tilde, which can be useful for introducing anonymous definitions. Example code using this sugar is given in Listing V.7.

(VALUE) $\sigma ::= \mathbb{N} \mid \text{``}\,\text{CHAR}^*\,\text{''} \mid [\,\tau^*\,] \mid [\,\tau^+ \,.\, \tau\,] \mid \_$
The sugar here for natural numbers ($\mathbb{N}$) and lists is familiar from earlier, but there are two additional sugared value types: strings[9], which are lists of character atoms where the character atom for, e.g., the letter 'x' is 'x, and units, which can be inserted with _ instead of ().

(PARTY) $\pi ::= \tau : \tau^* \mid \text{VAR}' : \tau^*$
Parties are the same as in raw ℵ. Note, however, that the reference interpreter of `alethe` has no separate representation for opaque variables: if the context is a variable, then it is assumed to be opaque rather than the variable pattern. When concurrency and contextual evaluation is supported in a future version, this deficiency will need to be addressed.

(PARTIES) $\Pi ::= \pi \mid \Pi\,;\,\pi$
Bags of parties are delimited by semicolons.

(RELATION) $\rho ::= \tau^* = \tau^* \mid \tau^* \,\text{`}\tau^*\text{'}\, \tau^*$
Relations are shorthand for where there is a single, variable context. These are familiar from the previous examples.

(DEF. HEAD) $\delta_h ::= \rho \mid \{\,\Pi\,\} = \{\,\Pi\,\}$
A rule is either a relation, or a mapping between party-bags which are enclosed in braces.

(DEF. RULE) $\delta_r ::= \delta_h\,; \mid \delta_h : \Delta^+$
If a rule has no sub-rules, then it is followed by a semicolon, otherwise it is followed by a colon and then its sub-rules/any locally scoped definitions. These declarations must either be in the same line, or should be indented more than the rule head similar to `Haskell`'s off-side rule or `python`'s block syntax.

(DEF. HALT) $\delta_t ::= \,!\,\tau^*; \mid !\,\rho\,;$
As well as the canonical halting pattern form, there is also sugar for a relation wherein both sides of the relation each beget a halting pattern, as does the infix term (if any).

(DEF.) $\delta ::= \delta_r \mid \delta_t$
This is the same as for ℵ.

(COST ANN.) $\xi ::= .^*$
Sub-rules can be followed by more than one full-stop, with the number of full-stops used quantifying the cost of the sub-rule for use in the serialisation algorithm heuristics as explained towards the end of Section V.5.

[9] Once again, using `Haskell`-style character escapes if needed

(DECL.) $\Delta ::= \delta \mid \pi\ .\xi \mid \rho\ .\xi \mid \mathbf{!}\ \rho\ .\xi$
In addition to the party sub-rules present in ℵ, there are three sugared forms. If a declaration is a definition then, after introducing and resolving fresh anonymous names for any locally scoped atoms, the behaviour is the same as if the definition was written in the global scope. If a declaration takes the form of a relation, then it is the same as if we introduced a fresh context variable and bound each side of the relation to it. If the relation is preceded by **!**, then it both declares a sub-rule and defines a pair of halting patterns, e.g.

$$\mathbf{!}\sim\textsc{Go}\ p\ xs\ []\ [] = \sim\textsc{Go}\ p\ xs\ [n \cdot ns]\ ys'.$$

becomes

$$\begin{aligned}&\mathbf{!}\sim\textsc{Go}\ p\ xs\ []\ [];\\ &\mathbf{!}\sim\textsc{Go}\ p\ xs\ [n \cdot ns]\ ys';\\ &\ \ \sim\textsc{Go}\ p\ xs\ []\ [] = \sim\textsc{Go}\ p\ xs\ [n \cdot ns]\ ys'.\end{aligned}$$

(STATEMENT) $\Sigma ::= \delta \mid \mathbf{data}\ \tau^*; \mid \mathbf{import}\ ``\textsc{module path}''$;
A statement is a definition, a data definition, or an import statement. A data definition **data** S n; is simply sugar for

$$\begin{aligned}&\mathbf{!}\ \textsc{S}\ n;\\ &\text{`}\textsc{Dup}\ (\textsc{S}\ n)\text{`}\ (\textsc{S}\ n'):\\ &\qquad\text{`}\textsc{Dup}\ n\text{`}\ n'.\end{aligned}$$

i.e. it marks the pattern as halting and automatically writes a definition of DUP. Future versions may automatically write other definitions, such as comparison functions, or include a derivation syntax akin to `Haskell`'s. The **import** statement imports all the definitions of the referenced file, as if they were one file, and supports mutual dependency. Future versions may support partial, qualified and renaming import variants.

(PROGRAM) $\mathcal{P} ::= \Sigma^*$
A program is a series of statements. Additionally, comments can be added in `Haskell`-style:

```
-- this is a comment.
! S {- so is this... -} n; -- ...and this.
{- and this is
   a multiline comment -}
```

We also note that the interpreter for `alethe` treats the atoms Z, S, CONS, NIL, GARBAGE, and character atoms specially when printing to the terminal. Specifically, they are

automatically recognised and printed according to the sugared representations defined above. That is, except for the GARBAGE atom which, when in the first position of a term, hides its contents—rendering as {~GARBAGE~}. This is useful when working with reversible simulations of irreversible functions that generate copious amounts of garbage data. If one assigns the garbage to a variable, then the special `:g` directive can be used to inspect its contents. Other directives include `:q` to quit, `:l file1 ...` to load the specified files as the current program, `:r` to reload the current program, `:v` to list the currently assigned variables after the course of the interpreter session, and `:p` to show all the loaded rules of the current program (and the derived serialisation strategy for each, see later). Computations can be performed in one of two ways; the first, `| (+ 3) 4 _`, takes as input a halting term, attempts to evaluate it to completion, and returns the output if successful, i.e. `() 7 (+ 3)`. The second, `> 4 '+ 3' y`, takes a relation as input and attempts to get from one side to the other. For example, here we evaluate `(+ 3) 4 ()`, obtaining `() 7 (+ 3)` which we then unify against `() y (+ 3)`, finally resulting in the variable assignment $y \mapsto 7$. To run in the opposite direction, use `<` instead. Note that, whilst the interpreter does understand concurrent rules, it is not yet able to evaluate them.

Unification Efficiency Beyond the implementation concerns expressed in Section V.5, another issue of practical importance is the efficiency of identifying patterns matching a given term: to date, the standard library alone contains just shy of two thousand patterns, and so searching the known patterns linearly is impractical. Our interpreter constructs a trie-like structure for this purpose such that the time complexity for unification scales as $\mathcal{O}(m + \log n)$ where m is the number of patterns which match and n the total number of patterns. Though asymptotically this is the best complexity possible, there are certainly improvements that could be made to the unification algorithm. In particular, the data structures used are primarily binary trees; hash tables with fixed lookup time for reasonable values of n would yield significant improvements in distinguishing atoms, and random access arrays would yield significant improvements for distinguishing between composite terms of differing length. Notwithstanding these possible improvements, the reference interpreter has proven sufficiently fast for demonstrating non-trivial computations in `alethe`. For example, computing $8! = 40\,320$ takes only a few seconds despite the unary representation of natural numbers.

V.7. Towards a Type System

Type systems are very useful, in that they allow a compiler to statically analyse a program before it is run and identify whether it is ever possible for a value to be passed to a function that does not understand such an input. This is extremely powerful, and leads to the `Haskell` maxim 'if it compiles, it runs': that is, although there may be logical errors, the program should not crash.

We may be tempted to try to start with a simple polymorphic type system, such as Hindley-Milner, but unfortunately Hindley-Milner is unsuitable for ℵ being that it

does not have objects that can be uniquely assigned a computational role. Consider attempting to type the symbol $\square$ from Listing V.4, for example, which implements squaring/square rooting for natural numbers. One is tempted to say something like $\square :: \mathbb{N}\ () \leftrightarrow ()\ \mathbb{N} :: \square$, but this does not work. Inspecting the definition more closely, we see that $\square$ appears in a number of contexts: $\square\ n\ ()$, $\square\ n\ m\ \square$, and $()\ m\ \square$. It is the second context where our attempt to type $\square$ breaks down: not only does its apparent arity change, but it appears twice in the same expression! Similar problems are encountered when trying to type other declarative languages, such as `Prolog`. To proceed, we shall require that any candidate type system treats terms *holistically*, not ascribing any special importance to any particular part of a term. Such a holistic type system will most likely also be declarative in nature.

A fruitful way of thinking of types of terms is by considering their endpoints—their halting states. A well-formed term will be able to eventually transition to one or more halting terms, and the properties of these halting terms are what we are most interested in. For example, we would like to know that if we construct a term from $+$, two natural numbers, and $()$, then we will be able to extract two natural numbers—rather than, say, a boolean and a string—once computation finishes. But, the 'end' state is free to wander back to the 'beginning', and so it is more appropriate to consider the type of the term as an unordered pair, consisting of the two possible halting states. Checking this type will then consist of enumerating which rules may be encountered along a computational path, verifying that all these rules are consistent with the initial type and that the other state is reachable in principle. If the ambiguity checker is turned off, then we see that it is possible to have a term with more than two halting states and so its type should be a set of types corresponding to each possible halting state. We then notice that conventional data—such as a natural number—is of the same ilk, save that it only has one halting state. We call these type-sets *isotypes*, and their inhabitants *isovalues*, as each isovalue has one or more isomorphically equivalent representations.

How might these isotypes look? We begin with conventional data; let us call the isotype of a natural number $\textsc{Nat}$, with inhabitants Z and $\mathsf{S}n$ where n inhabits $\textsc{Nat}$. Let us now attempt to type addition. As a first approximation, we write $:: + \textsc{Nat}\ \textsc{Nat}\ () = ()\ \textsc{Nat}\ \textsc{Nat}\ +$; where $::$ at the beginning marks this as an isotype. This is in the right spirit, but doesn't readily admit a consistent type theory. In particular, how would we represent the isotype of a continuation in the term $\textsc{Walk}\ \beta\ (\textsc{Charlie}'\ x)$, as introduced towards the end of Section V.3? Being more careful, we can say that an addition isovalue is the unordered pair consisting of $+\ a\ b\ ()$ and $()\ c\ d\ +$ where a, b, c and d are all isovalues of isotype $\textsc{Nat}$. We need this explication, because the isotypes of the arguments could be far more complicated. Where it is unambiguous, however, it will be reasonable to write this isotype as $:: + \textsc{Nat}\ \textsc{Nat}\ () = ()\ \textsc{Nat}\ \textsc{Nat}\ +;$. Notice that, as this isotype is holistic, it is completely orthogonal to the isotype $:: + \textsc{Rat}\ \textsc{Rat}\ () = ()\ \textsc{Rat}\ \textsc{Rat}\ +$; where $\textsc{Rat}$ is the isotype of rationals. This reveals that we get overloading for free in this type system. It is nevertheless desirable to introduce a notion similar to `Haskell`'s typeclasses, however, or else (partial) isomorphisms employing addition, but polymorphic in the isotype they are adding, will have much too generic an isotype (in particular, the type inference algorithm

would not be able to conclude that an isotype $:: + a\ b\ () = ()\ c\ d\ +;$ does not exist, and so we would have four isotype variables rather than the 1 expected). Typeclasses permit the programmer to specify that an isotype of the form $:: + a\ b\ () = ()\ c\ d\ +;$ is only legal if it is of the more restrictive form $:: + a\ a\ () = ()\ a\ a\ +;$ for some isotype a. Furthermore, the introduction of typeclasses permits the abbreviation of complex (ad-hoc) polymorphic isotypes.

What, then, might a polymorphic isotype look like? A good example is given by the map function, which would have as isotype the unordered pair of $(\textsc{Map}\ f)\ x\!\!\not\partial\ ()$ and $()\ y\!\!\not\partial\ (\textsc{Map}\ f)$ where $x\!\!\not\partial$ is an inhabitant of the isotype $[a]$, i.e. the isotype of lists of elements of isotype a, and where $y\!\!\not\partial$ is an inhabitant of $[b]$. f is required to be some term which, when used to construct the halting states $f\ x\ ()$ and $()\ y\ f$, yields the isotype $:: f\ a\ () = ()\ b\ f$. More concisely, we write this as

$$
\begin{aligned}
&:: [a]\ \text{`}\textsc{Map}\ f\text{`}\ [b]:\\
&::\qquad a\ \text{`}f\text{`}\ b.\\
&\quad [\,]\ \text{`}\textsc{Map}\ f\text{`}\ [\,];\\
&\quad [x \cdot x\!\!\not\partial]\ \text{`}\textsc{Map}\ f\text{`}\ [y \cdot y\!\!\not\partial]:\\
&\qquad\quad x\ \text{`}f\text{`}\ y.\\
&\qquad\quad x\!\!\not\partial\ \text{`}\textsc{Map}\ f\text{`}\ y\!\!\not\partial.
\end{aligned}
$$

and we note the similarity of the isotype to the implementation of MAP at the isovalue level. The composite term $\textsc{Map}\ f$ deserves further attention; this is itself halting, and so can be considered to be an implicitly declared anonymous isotype with one representation.

As briefly mentioned earlier, a type checker (and a type inference algorithm, if it exists) will perform a reachability analysis—starting from one halting pattern, it will enumerate all the accessible rules, finding the most general type consistent with these. The information gleaned this way is very valuable for all sorts of static analyses. We can determine what other halting states are accessible, as well as whether there are any missing cases along the way that could lead to a halting state. We can also resolve the issue of sub-rule ambiguity raised earlier, helping strengthen the phase space restrictions and avoid unintentional generation of entropy. It may also be possible to use this information to refine the cost heuristic used for serialisation, improving runtimes. Furthermore, knowledge of the execution path gives opportunities for identifying code motifs and performing structural transformations, a necessary ingredient for performing the kind of code optimisations a compiler is wont to do.

There is, however, a potential hiccup in the reachability analysis that occurs when a continuation-passing-style is used. In this case, there are rules which are shared between different computations. For example, in $\textsc{Walk}\ \beta\ c \leftrightarrow \textsc{Walk'}\ \alpha\ c$, c is a continuation. There will be a number of computations which each pass control to such a WALK term, and later accept control back from such a WALK' term. Consequently, no definitive isotype can be assigned to these intermediate terms. Moreover, consider these two

potential computations:

$$\textsc{Foo} = \textsc{Walk}\ \varphi\ \textsc{Foo}';$$
$$\textsc{Bar} = \textsc{Walk}\ \beta\ \textsc{Bar}';$$

It is conceivable that the reachability analyser would, starting from FOO, (correctly) determine that the pattern WALK β c is reachable, and then (incorrectly) presume the term BAR is reachable from FOO. If so, the type checker would then (rightly) complain of a type error, perhaps that the isotypes of FOO′ and BAR′ do not unify. The remedy to the first issue, of indefinite isotyping of intermediate terms, is to only assign definitive isotypes to sets of halting terms; intermediate terms are assigned definite isotypes only in the course of inferring isotypes for particular halting sets, and these intermediate isotypes are appropriately scoped such that the parallel process of isotype inference may proceed in spite of conflicting intermediate term isotypes. The second issue is remedied by specialising types as early as possible: in order to pass the ambiguity checker, any computation making use of a shared rule as above must pass control to the shared rule in a way orthogonal to any other computations passing control. In the above example, this holds because FOO′ and BAR′ are orthogonal patterns such that no term can be constructed satisfying them both simultaneously. Therefore, by immediately specialising the type of the continuation c in WALK, there is no risk of reachability 'leaking' into inaccessible rules. One may be concerned that perhaps the reachability graph is much more complicated, engendering other reachability leaks, but each such confluence point between distinct computational paths *must* have this same explicit orthogonality property in order to satisfy the ambiguity checker, and so there is in fact no risk. However, this remedy may be somewhat overzealous; suppose that the example above is part of a larger program,

$$\textsc{Qux}\ \textsc{False} = \textsc{Foo};$$
$$\textsc{Qux}\ \textsc{True} = \textsc{Bar};$$

then the eager specialisation of the continuation isotype will yet result in a type-checking error, similar to `Haskell`'s (optional) monomorphism restriction. To circumvent such monomorphism, it would be advantageous to allow the type checker to introduce additional scopes for intermediate types where such branches occur, providing that these alternate paths eventually converge to a single type. If this is not possible, then the solution may simply be to expect the programmer to annotate a 'partial isotype' of any shared polymorphic rules in order to give the type checker sufficient information.

We are optimistic that, in programs without shared rules, it is possible to perform automatic type inference without any type annotations as in the Hindley-Milner type system. As for programs with shared rules, we are less certain; nonetheless, the power and generality of type inference algorithms in type systems as sophisticated as `Haskell`'s most recent iterations suggests that it is attainable. We also note that the above is just a sketch of a possible type system, and will require further refinement and formalisation. Finally, we have neglected as yet to consider concurrency. Concurrency severely

undermines the power of the type system, as there is no longer any continuity between terms: at any point, a term could be consumed, produced, or merged with another. It may be possible to extend the notion of isotype to include all the terms that might interact with one another, but it is unclear how useful this would be. Alternatively, one could simply ascribe types to single halting patterns, instead of seeking isotypes and an enumeration of all possible isomorphic representations of an isovalue. Again, this severely weakens the utility of the type system. A desirable compromise would be to identify the non-concurrent parts of a program, and apply type checking and inference to just these parts; a more rudimentary level of type checking could then be applied to the concurrent parts to at least verify the consistency of any non-concurrent sub-rules employed.

V.8. The Σ-Calculus

An earlier prototype, Σ was more functional in nature. Its terms were nested applications of 'general permutations', which could be named for convenience (and for achieving recursion without a fixed point combinator). Its definition was

$$\text{(TERM)} \quad \tau ::= \langle \pi\tau^* : \tau^*\pi \rangle \mid \text{VAR} \mid \text{REF} \mid \tau*$$

where REF is a reference to a named pattern. A generalised permutation is given by $\langle \pi\tau^* : \tau^*\pi \rangle$, where π is a special variable that matches the permutation itself. That is, the term $(\langle \pi\ x\ y : y\ x\ \pi \rangle\ 1\ 2)$ would transition to $(2\ 1\ \langle \pi\ x\ y : y\ x\ \pi \rangle)$. Notice that this is the convention taken in ℵ, that the atom or term in the first position tends to take the 'active' role, and tends to place itself at the end of the term after rule execution. In Σ, however, this convention is enshrined in the language itself because all the information concerning the transition rules is held within the permutation terms themselves, and so it is required that there be two special locations within a term corresponding to the current and previous permutation. The first position is used for the current permutation in analogy to the λ-calculus, and the last position is used for the previous permutation for symmetry. Permutations are 'general' in the sense that they can alter tree structure, and can copy/elide variables.

The Σ-calculus can also be proven to be microscopically reversible and Reverse-Turing complete, but the absence of sub-rules renders composition more unwieldy. Additionally, discriminating different cases is tricky. As with the λ-calculus, where data can be represented with functions via Church encoding, we can represent data in Σ with permutations and these permutations will perform case statements. Taking lists as an example, we have

$$\begin{aligned} \text{CONS} &\equiv \langle \pi\{nc\}zf : c\{nf\}z\pi \rangle \\ \text{NIL} &\equiv \langle \pi\{nc\}zf : n\{fc\}z\pi \rangle \end{aligned}$$

where $\{xyz\}$ is sugar for $(\bot xyz\top)$ used to represent an inert expression ($\top \equiv \bot \equiv \varepsilon \equiv ()$, not being a permutation, is an inert object that does nothing and is used to represent

halting states in Σ). Using these definitions, list reversal can be implemented thus,

$$\begin{aligned}
\textsc{rev} &\equiv \langle \pi l \top : \textsc{nil}\{\lambda\nu\}\{\{\varepsilon\varepsilon\}l\}\pi\rangle \\
\lambda &\equiv \langle \pi\{\textsc{rev}\,\nu\}\{\{r'r''\}\{\tilde{l}l'l''\}\}\tilde{r} : \tilde{l}\{\textsc{rev}'\,\nu\}\{\{\tilde{r}r'r''\}\{l'l''\}\}\pi\rangle \\
\nu &\equiv \langle \pi\{\textsc{rev}'\,\lambda\}\{r\{l'l''\}\}\textsc{cons} : \textsc{cons}\{\textsc{rev}\,\lambda\}\{\{l'r\}l''\}\pi\rangle \\
\textsc{rev}' &\equiv \langle \pi\{\lambda\nu\}\{r\{\varepsilon\varepsilon\}\}\textsc{nil} : \bot r\pi\rangle
\end{aligned}$$

which is certainly not the most edifying program in the world! It would be used as so:

$$(\textsc{rev}[1\,4\,6\,2]\top) \xleftrightarrow{*} (\bot[2\,6\,4\,1]\textsc{rev}')$$

μ-Recursive Functions To give a deeper flavour of Σ, we implement the μ-recursive functions (recall their definition and `alethe` implementation from Listing V.12). In order to simulate a notion of function composition, we first adopt a 'Σ-function' motif: a function is represented by the triple $F = \{f\ z\ f'\}$ where f and f' are the initial and terminal permutations, and z is any data we wish to bind to F. The permutations f and f' must be such that $(f\ z\ f'\ x\ \top) \xleftrightarrow{*} (\bot\ y\ g\ f\ z\ f')$, where x is the input, y the output and g any garbage data.

The base μ-recursive functions can then be implemented thus:

$$\begin{aligned}
Z &\equiv \{z\varepsilon z\} & z &\equiv \langle z\varepsilon z\ n\ \top : \bot\ 0\ n\ z\varepsilon z\rangle \\
S &\equiv \{s\varepsilon s\} & s &\equiv \langle s\varepsilon s\ n\ \top : \bot\ \{\textsc{succ}n\}\ \varepsilon\ s\varepsilon s\rangle \\
\Pi^k_i &\equiv \{\pi^k_i\varepsilon\pi^k_i\} & \pi^k_i &\equiv \langle \pi^k_i\varepsilon\pi^k_i\ \{n_1,\ldots,n_k\}\ \top : \\
& & & \qquad \bot\ n_i\ \{n_1,\ldots,n_{i-1},n_{i+1},\ldots,n_k\}\ \pi^k_i\varepsilon\pi^k_i\rangle
\end{aligned}$$

Note that we have written $\langle z\varepsilon z\cdots$ instead of $\langle \pi\varepsilon z\cdots$ or $\langle \pi\varepsilon\pi\cdots$ for clarity.

The composition operator for F of arity k over functions G_i is given by $\{c_k\{F\vec{G}\}c'_k\}$, where

$$\begin{aligned}
c_k &\equiv \langle c_k\{F\vec{G}\}c'_k\ \vec{n}\ \top : c''_k\ F\ (G_1 \wedge \vec{n})\ \cdots\ (G_k \wedge \vec{n})\ c_k\rangle \\
c''_k &\equiv \langle c''_k\ F\ (m_1\ h_1 \vee G_1)\ \cdots\ (m_k\ h_k \vee G_k)\ c_k : c'_k\ \vec{h}\ \vec{G}\ (F \wedge \vec{m})\ c''_k\rangle \\
c'_k &\equiv \langle c'_k\ \vec{h}\ \vec{G}\ (y\ h_f \vee F)\ c''_k : \bot\ y\ \{h_f\ \vec{h}\}\ c_k\{F\vec{G}\}c'_k\rangle
\end{aligned}$$

where we have introduced sugared forms $(F \wedge x) \equiv (fzf'\ x\ \top)$ and $(y\ h \vee F) \equiv (\bot\ y\ h\ fzf')$.

The primitive recursion operator is a little more complicated. To understand the implementation below, it is important to know that the number 0 is represented as $\{\textsc{zero}\ \varepsilon\}$ and the number $m+1$ is given by $\{\textsc{succ}\ m\}$. When the form is unknown, we can write a unified representation as $\{\tilde{m}m\}$ where $\tilde{m}$ is the constructor and m is either ε or the predecessor. Furthermore, the constructors are defined as follows:

$$\begin{aligned}
\textsc{zero} &\equiv \langle \pi\{pq\}zr : p\{rq\}z\pi\rangle \\
\textsc{succ} &\equiv \langle \pi\{pq\}zr : q\{pr\}z\pi\rangle
\end{aligned}$$

We write the primitive recursion of F and G (arities k and $k+2$) as $R_{FG} = \{\rho_k\{FG\}\rho'_k\}$, defined by:

$$\begin{aligned}
\rho_k &= \langle \rho_k\{FG\}\rho'_k\ \{\vec{n}\{\tilde{m}m\}\} \top : \tilde{m}\ \{\zeta_k\,\sigma_k\}\ \{FG\vec{n}m\}\ \rho_k\rangle \\
\rho'_k &= \langle \rho'_k\ \{\zeta'_k\,\sigma''_k\}\ \{yhFG\}\ \tilde{m} : \bot\ y\ \{\tilde{m}h\}\ \rho_k\{FG\}\rho'_k\rangle \\
\\
\zeta_k &= \langle \zeta_k\ \{\rho_k\,\sigma_k\}\ \{FG\vec{n}\varepsilon\}\ \textsc{zero} : \zeta'_k\ (F \wedge \vec{n})\ G\ \zeta_k\rangle \\
\zeta'_k &= \langle \zeta'_k\ (y\ h \vee F)\ G\ \zeta_k : \textsc{zero}\ \{\rho'_k\,\sigma''_k\}\ \{yhFG\}\ \zeta'_k\rangle \\
\\
\sigma_k &= \langle \sigma_k\ \{\zeta_k\,\rho_k\}\ \{FG\vec{n}m\}\ \textsc{succ} : \sigma'_k\ (R_{FG} \wedge \vec{n}m)\ \vec{n}m\ \sigma_k\rangle \\
\sigma'_k &= \langle \sigma'_k\ (y\ h \vee R_{FG})\ \vec{n}m\ \sigma_k : \sigma''_k\ (G \wedge \vec{n}my)\ F\ h\ \sigma'_k\rangle \\
\sigma''_k &= \langle \sigma''_k\ (z\ h' \vee G)\ F\ h\ \sigma'_k : \textsc{succ}\ \{\zeta'_k\,\rho'_k\}\ \{z\{hh'\}FG\}\ \sigma''_k\rangle
\end{aligned}$$

In the above, the ρ permutation switches between the ζ and σ permutations depending on if $\{\tilde{m}m\} = 0$ or > 0; the ζ route computes $y = F(\vec{n})$, whilst the σ route recurses down to compute $y = R_{FG}(\vec{n}, m)$ and then $z = G(\vec{n}, m, y)$. Both these routes then put their results into a common representation and merge the control flow via $\tilde{m}$.

Finally, we implement the minimalisation operator as $M^m_F = \{\mu_k\{Fm\}\mu'_k\}$; M^m_F is a generalisation of the more canonical minimalisation operator (which is equivalent to M^0_F),

$$M^m_F(\vec{n}) = \min\{\ell \geq m : f(\vec{n};\ell) = 0\}$$

Interestingly, this is much simpler to implement than primitive recursion—only requiring four permutations—yet is the necessary ingredient for universality:

$$\begin{aligned}
\mu_k &= \langle \mu_k\{Fm\}\mu'_k\ \vec{n}\ \top : \mu''_k\ m\ (F \wedge \vec{n}m)\ (M_F^{\{\textsc{succ}\,m\}} \wedge \vec{n})\ \mu_k\rangle \\
\mu''_k &= \langle \mu''_k\ m\ (\{\tilde{y}y\}h \vee F)\ q\ \mu_k : \tilde{y}\ \{\mu'_k\mu'''_k\}\ m\ (\{\tilde{y}y\}h \vee F)\ q\ \mu''_k\rangle \\
\mu'''_k &= \langle \mu'''_k\ \{\mu'_k\mu'''_k\}\ m\ p\ (m'h \vee M_F^{\{\textsc{succ}\,m\}})\ \textsc{succ} : \\
&\qquad \textsc{succ}\ \{\mu''_k\mu'_k\}\ m'\ p\ (m'h \vee M_F^{\{\textsc{succ}\,m\}})\ \mu'''_k\rangle \\
\mu'_k &= \langle \mu'_k\ \{\mu''_k\mu'''_k\}\ m'\ (F \wedge \vec{n}m)\ (M_F^{\{\textsc{succ}\,m\}} \wedge \vec{n})\ \tilde{\mu} : \bot\ m'\ \{\tilde{\mu}\vec{n}\}\ \mu_k\{Fm\}\mu'_k\rangle
\end{aligned}$$

In the above, we compute $F(\vec{n}; m)$ and check if it's 0, if so we return m as m', otherwise we compute $M^{m+1}_F(\vec{n})$ in a recursive manner and return its result as m'. Clearly, if there is no m such that $F(\vec{n}; m) = 0$ then $M^m_F(\vec{n}) = \bot\ \forall m$. For concision, we compute both functions simultaneously, making use of Σ's parallelism to approximate laziness. We then conveniently reverse both computations in a Bennett-like manner to produce only a single bit of garbage (the history data we return is $\tilde{\mu}$, representing whether or not $m = m'$, and the input vector $\vec{n}$).

The Interpreter Whilst the implementation of the μ-recursive functions demonstrates that realising meaningful programs in Σ is certainly possible, it is inelegant and clunky. Rather than programming in Σ feeling like the λ-calculus, as was intended, it feels much

more like programming in assembly. Nevertheless, we release the source code of the interpreter[10], example code[11], and syntax highlighters[12] for completeness.

V.9. Conclusion

The examples furnishing this chapter demonstrate the utility of the א-calculus, as well as its appropriateness as a candidate model targeting reversible molecular and Brownian computational architectures. Though a molecular implementation is as yet unrealised, an interpreter for the programming language `alethe` serves as a useful testbed for א. Going forward, we hope to extend the interpreter to support the full concurrent language. Additionally, development of a type system appropriate for a language that is both reversible and declarative would be helpful for improving code analysis and reducing programmer errors, and work towards this is underway. Lastly, it is hoped that א can be realised experimentally in a molecular context.

Towards a Molecular Implementation Figure V.3 and Listing V.4b are suggestive of a high level molecular implementation of the א-calculus. Nevertheless, much more work is needed to develop a viable molecular implementation. Translation from א terms and programs to the abstract molecular schemes is fairly routine: one should first eliminate sub-terms by introducing intermediate terms, an example of which can be seen in the orange plus 'atoms' in Figure V.3. One should also be careful to conserve mass, as discussed in Section V.3. The resulting reaction schemes will in general still be difficult to embed in a realistic chemical system. The reason for this is that arbitrary structural rearrangements may occur in a single א reaction, but realistic chemical reactions tend to operate in small, local steps. Therefore, additional compilation steps besides mass conservation and sub-term promotion will be needed. A further concern is that the arbitrary nesting of terms in א means they may not be easily embeddable in flat three-dimensional space. Methods to address this issue may consist of extensible 'tethers', or terms could be split in two and linked 'virtually' via a unique address (akin to pointers in conventional computational architectures).

Before this, however, it is necessary to identify the compilation target. This will require construction of a lower level reversible molecular computational model that is Turing Complete, dynamic, and term-based. As discussed in Chapter I, the current predominant methods for molecular computation are unsuitable. To our knowledge, there are not any existing molecular architectures that meet the combined requirements of א: reversibility, ability to couple to a common free energy currency, arbitrary structural assemblies, and dynamic reassembly of said structures. Whilst the absence of an existing architecture satisfying these properties might suggest that none exist, we believe that it is possible to construct one. Indeed, biological enzymes routinely perform all of these

[10] https://github.com/hannah-earley/sigma-repl
[11] https://github.com/hannah-earley/sigma-examples
[12] https://github.com/hannah-earley/sigma-syntax

tasks. The challenge is in designing a simple programmable system to perform particular transitions. Ideally such a system will be constructed from nucleic acids, as they are much easier to work with and design than polypetides are.

VI. Conclusion

In this thesis we explore the intersection between reversible computation and molecular computation. The benefits of reversible computation are primarily from the perspective of energy efficiency, and it has been known for some time (see, e.g., Frank [9]) that the gains in energy efficiency are asymptotic. That is, reversible computers can exhibit a rate of computation that scales much better with system size than an equivalent irreversible computer. In Chapter II we refined these results, proving that it is not possible to do better than adiabatic scaling in any system, classical or quantum. This adiabatic scaling corresponds to a computation rate $\sim \sqrt{PM}$ where P is the rate of supply of free energy and M the computational mass-energy. In contrast, the scaling for irreversible computers is $\sim P$. Asymptotically, P can scale no better than the convex bounding surface area of the system, A, and M no better than the volume of the system, V, leading to respective scaling laws $\sqrt{AV}$ and A. Strictly speaking these apply to the mesoscopic regime; for the microscopic and cosmic scales we further refined these results.

This performance metric of net computation rate ignores the importance of interaction between computer subunits. In Chapters III and IV we therefore extended this analysis to consider, respectively, computer-computer communication/synchronisation interactions and interactions between computers and shared resources. These turned out to be expensive in the case of Brownian architectures, and we conjectured about the applicability of these results to any architecture in which individual computational units evolve independently. If the same aggregate rate of interactions as the rate of computation were to be maintained, then we showed that the net computation rate would have to fall to that of irreversible computation, A. From the point of view of molecular or Brownian computers, which we showed in Section II.4 were easily able to saturate the adiabatic scaling bounds, this was because the timescale to perform an interaction was on the order of $\sim 1/b^2$ where b quantifies the *computational bias*, a measure of the free energy density of the system. As $b \ll 1$ is vanishing for large systems, this is untenable: ideally, all computational transitions in a Brownian computer should operate at a characteristic timescale $\lesssim 1/b$. Whilst concerning from an engineering perspective, we presented various mitigation steps in their respective conclusions. Primarily these centred on reducing the instantaneous proportion of computational subunits permitted to participate in such interactions. Moreover, for current computational scales up to the order of metres it is likely that reasonable values of $b \gtrsim 0.1$ could be achieved and thus no mitigation may be necessary at present.

Finally we proposed a model of computation that might be appropriate for programming such a reversible Brownian computer. The motivation for this was the limited attention which this intersectional computation paradigm has received, and thus this model fills a vital gap. It is also novel in being the first (to our knowledge) declarative

model of reversible programming. Whilst the א-calculus is fully functional as a programming language, more work needs to be done to develop it into a viable specification for a reversible molecular computer. The examples in Sections IV.4 and V.2 serve as an initial blueprint for this work, and we hope to further develop this over the coming years.

References

[1] Hannah Earley. 'Engines of Parsimony: Part I. Limits on Computational Rates in Physical Systems'. In: (2020). arXiv: 2007.03605 [cond-mat.stat-mech].

[2] Hannah Earley. 'Engines of Parsimony: Part II. Performance Trade-offs for Communicating Reversible Computers'. In: (2020). arXiv: 2011.04054 [cond-mat.stat-mech].

[3] Hannah Earley. 'Engines of Parsimony: Part III. Performance Trade-offs for Reversible Computers Sharing Resources'. 2020. arXiv: 2012.05655 [cond-mat.stat-mech].

[4] Hannah Earley. 'The ℵ Calculus. A declarative model of reversible programming with relevance to Brownian computers'. In: (2020). arXiv: 2011.14989 [cs.PL].

[5] Hannah Earley. *Modelling approaches to molecular computation*. https://gtr.ukri.org/projects?ref=studentship-1781682. 2016.

[6] K Eric Drexler. *Engines of Creation. The Coming Era of Nanotechnology*. Anchor, 1986.

[7] Rolf Landauer. 'Irreversibility and heat generation in the computing process'. In: *IBM J. Res. Dev.* 5.3 (1961), pp. 183–191.

[8] Leo Szilard. 'Über die Entropieverminderung in einem thermodynamischen System bei Eingriffen intelligenter Wesen'. In: *Zeitschrift für Physik* 53.11-12 (1929), pp. 840–856. English translation: Leo Szilard. 'On the decrease of entropy in a thermodynamic system by the intervention of intelligent beings'. In: *Behavioral Science* 9.4 (1964), pp. 301–310.

[9] Michael Patrick Frank. 'Reversibility for efficient computing'. PhD thesis. Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science, 1999.

[10] Leonard M Adleman. 'Molecular computation of solutions to combinatorial problems'. In: *Science* 266.5187 (1994), pp. 1021–1024.

[11] James D Watson and Francis HC Crick. 'Molecular structure of nucleic acids: a structure for deoxyribose nucleic acid'. In: *Nature* 171.4356 (1953), pp. 737–738.

[12] Hao Wang. 'Proving theorems by pattern recognition—II'. In: *Bell Labs Tech. J.* 40.1 (1961), pp. 1–41. ISSN: 1089-7089.

[13] Erik Winfree. *Simulations of Computing by Self-Assembly*. Tech. rep. California Institute of Technology, Department of Computer Science, 1998.

[14] David Doty et al. 'The tile assembly model is intrinsically universal'. In: *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*. IEEE. 2012, pp. 302–310.

[15] David Soloveichik, Georg Seelig and Erik Winfree. 'DNA as a universal substrate for chemical kinetics'. en. In: *Proc. Natl. Acad. Sci. U. S. A.* 107.12 (Mar. 2010), pp. 5393–5398.

[16] Paul Wilhelm Karl Rothemund. 'A DNA and restriction enzyme implementation of Turing Machines'. In: *DNA based computers* 27 (1995), pp. 75–119.

[17] Kevin Montagne et al. 'Programming an in vitro DNA oscillator using a molecular networking strategy'. In: *Molecular systems biology* 7.1 (2011), p. 466.

[18] Jocelyn Y Kishi et al. 'Programmable autonomous synthesis of single-stranded DNA'. In: *Nature Chemistry* 11 (Nov. 2017), p. 1112.

[19] Georg Seelig et al. 'Enzyme-free nucleic acid logic circuits'. en. In: *Science* 314.5805 (Dec. 2006), pp. 1585–1588. ISSN: 0036-8075, 1095-9203.

[20] Lulu Qian, Erik Winfree and Jehoshua Bruck. 'Neural network computation with DNA strand displacement cascades'. en. In: *Nature* 475.7356 (July 2011), pp. 368–372. ISSN: 0028-0836, 1476-4687.

[21] Kevin M Cherry and Lulu Qian. 'Scaling up molecular pattern recognition with DNA-based winner-take-all neural networks'. In: *Nature* 559.7714 (2018), pp. 370–376.

[22] Anupama J Thubagere et al. 'A cargo-sorting DNA robot'. In: *Science* 357.6356 (2017).

[23] Damien Woods et al. 'Diverse and robust molecular algorithms using reprogrammable DNA self-assembly'. In: *Nature* 567.7748 (2019), pp. 366–372.

[24] Chris Thachuk, Erik Winfree and David Soloveichik. 'Leakless DNA Strand Displacement Systems'. en. In: *DNA Computing and Molecular Programming*. Springer, Cham, Aug. 2015, pp. 133–153.

[25] Niranjan Srinivas et al. 'On the biophysics and kinetics of toehold-mediated DNA strand displacement'. In: *Nucleic acids research* 41.22 (2013), pp. 10641–10658.

[26] Robert RF Machinek et al. 'Programmable energy landscapes for kinetic control of DNA strand displacement'. In: *Nature communications* 5.1 (2014), pp. 1–9.

[27] Patrick Irmisch, Thomas E Ouldridge and Ralf Seidel. 'Modelling DNA-strand displacement reactions in the presence of base-pair mismatches'. In: *Journal of the American Chemical Society* (2020).

[28] David Yu Zhang and Erik Winfree. 'Control of DNA strand displacement kinetics using toehold exchange'. en. In: *J. Am. Chem. Soc.* 131.47 (Dec. 2009), pp. 17303–17314.

[29] Matthew R Lakin et al. 'Visual DSD: a design and analysis tool for DNA strand displacement systems'. In: *Bioinformatics* 27.22 (2011), pp. 3211–3213.

[30] Rachel Cummings, David Doty and David Soloveichik. 'Probability 1 computation with chemical reaction networks'. In: *Natural Computing* 15.2 (2016), pp. 245–261.

[31] Lulu Qian and Erik Winfree. 'Scaling up digital circuit computation with DNA strand displacement cascades'. en. In: *Science* 332.6034 (June 2011), pp. 1196–1201. ISSN: 0036-8075, 1095-9203.

[32] Gourab Chatterjee et al. 'A spatially localized architecture for fast and modular DNA computing'. en. In: *Nat. Nanotechnol.* 12.9 (Sept. 2017), pp. 920–927.

[33] Kenichi Fujibayashi et al. 'Error suppression mechanisms for DNA tile self-assembly and their simulation'. en. In: *Nat. Comput.* 8.3 (Sept. 2009), pp. 589–612.

[34] Hugh Everett III et al. *The Many-Worlds Interpretation of Quantum Mechanics*. Princeton University Press, 1973.

[35] *Cortex-A78 – Microarchitectures – ARM – WikiChip*. https://en.wikichip.org/wiki/arm_holdings/microarchitectures/cortex-a78. (Visited on 15/11/2020).

[36] Saed G Younis and Thomas F Knight Jr. 'Practical implementation of charge recovering asymptotically zero power CMOS'. In: *Proceedings of the 1993 symposium on Research on integrated systems*. 1993, pp. 234–250.

[37] Saed G Younis. *Asymptotically Zero Energy Computing Using Split-Level Charge Recovery Logic*. Tech. rep. Massachusetts Inst of Tech Cambridge Artificial Intelligence Lab, 1994.

[38] Michael P Frank et al. 'Reversible Computing with Fast, Fully Static, Fully Adiabatic CMOS'. In: (2020). arXiv: 2009.00448 [cs.AR].

[39] Charles H Bennett. 'Logical Reversibility of Computation'. In: *IBM J. Res. Dev.* 17.6 (Nov. 1973), pp. 525–532.

[40] Charles H Bennett. 'Time/space trade-offs for reversible computation'. In: *SIAM Journal on Computing* 18.4 (1989), pp. 766–776.

[41] Andrew Lewis Ressler. 'The design of a conservative logic computer and a graphical editor simulator'. PhD thesis. Massachusetts Institute of Technology, 1981.

[42] Edward Fredkin and Tommaso Toffoli. 'Conservative Logic'. en. In: *Collision-Based Computing*. Ed. by Andrew Adamatzky. Springer London, 1981, pp. 47–81.

[43] Charles H Bennett. 'The thermodynamics of computation—a review'. en. In: *Int. J. Theor. Phys.* 21.12 (Dec. 1982), pp. 905–940.

[44] Carlin James Vieri. 'Pendulum–a reversible computer architecture'. MA thesis. Massachusetts Institute of Technology, 1995.

[45] Carlin James Vieri. 'Reversible computer engineering and architecture'. PhD thesis. Massachusetts Institute of Technology, 1999.

[46] Norman Margolus. 'Physics-like models of computation'. In: *Physica D: Nonlinear Phenomena* 10.1-2 (1984), pp. 81–95.

[47] Christopher Lutz and Howard Derby. 'Janus: a time-reversible language'. In: *Caltech class project* (1982).

[48] Tetsuo Yokoyama and Robert Glück. 'A reversible programming language and its invertible self-interpreter'. In: *Proceedings of the 2007 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation*. 2007, pp. 144–153.

[49] Henry G Baker. 'Lively linear Lisp: "look ma, no garbage!"' In: *ACM Sigplan notices* 27.8 (1992), pp. 89–98.

[50] Henry G Baker. 'NREVERSAL of fortune—the thermodynamics of garbage collection'. In: *Memory management*. Springer, 1992, pp. 507–524.

[51] Jean-Yves Girard. 'Linear logic'. In: *Theoretical computer science* 50.1 (1987), pp. 1–101.

[52] Jean-Philippe Bernardy et al. 'Linear Haskell: practical linearity in a higher-order polymorphic language'. In: *Proceedings of the ACM on Programming Languages* 2.POPL (2017), pp. 1–29.

[53] Michael P Frank. *The R programming language and compiler*. 1997.

[54] Ed Fredkin. 'Feynman, Barton and the reversible Schrödinger difference equation'. In: *Feynman and computation*. CRC Press, 2018, pp. 337–348.

[55] Ben Rudiak-Gould. *Kayak*. http://esoteric.sange.fi/essie2/download/kayak/kayak.html. 2002. (Visited on 05/04/2017).

[56] Urban Müller. *The `Brainfuck` language*. http://aminet.net/package/dev/lang/brainfuck-2 (mirrored at https://github.com/hannah-earley/aminet-bf). 9th June 1993. (Visited on 20/12/2020).

[57] Ben Olmstead. *The `Malbolge` language*. http://www.lscheffer.com/malbolge_spec.html (mirrored at https://gist.github.com/hannah-earley/c55ede14e5185f5a143fc613b9fa8577). 1998. (Visited on 20/12/2020).

[58] Tetsuo Yokoyama, Holger Bock Axelsen and Robert Glück. 'Towards a reversible functional language'. In: *International Workshop on Reversible Computation*. Springer. 2011, pp. 14–29.

[59] Shin-Cheng Mu, Zhenjiang Hu and Masato Takeichi. 'An injective language for reversible computation'. In: *International Conference on Mathematics of Program Construction*. Springer. 2004, pp. 289–313.

[60] Roshan P James. 'The computational content of isomorphisms'. PhD thesis. Indiana University, 2013.

[61] R P James and A Sabry. *Theseus: a high level language for reversible computing, work-in-progress report at RC (2014)*. 2014.

[62] S Chandrasekhar. 'Dynamical instability of gaseous masses approaching the Schwarzschild limit in general relativity'. In: *Physical Review Letters* 12.4 (1964), p. 114.

[63] G E Moore. 'Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp.114 ff'. In: *IEEE Solid-State Circuits Society Newsletter* 11.3 (2006), pp. 33–35. ISSN: 1098-4232.

[64] Seth Lloyd. 'Ultimate physical limits to computation'. In: *Nature* 406.6799 (2000), pp. 1047–1054.

[65] Hans J Bremermann. 'Optimization through evolution and recombination'. In: *Self-organizing systems* 93 (1962), p. 106.

[66] Norman Margolus and Lev B Levitin. 'The maximum speed of dynamical evolution'. In: *Physica D* 120.1 (Sept. 1998), pp. 188–195.

[67] Joan A Vaccaro and Stephen M Barnett. 'Information erasure without an energy cost'. In: *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 467.2130 (2011), pp. 1770–1778.

[68] Baidyanath Misra and EC George Sudarshan. 'The Zeno's paradox in quantum theory'. In: *Journal of Mathematical Physics* 18.4 (1977), pp. 756–763.

[69] Lev B Levitin and Tommaso Toffoli. 'Thermodynamic cost of reversible computing'. In: *Physical review letters* 99.11 (2007), p. 110502.

[70] Subhash S Pidaparthi and Craig S Lent. 'Energy dissipation during two-state switching for quantum-dot cellular automata'. In: *Journal of Applied Physics* (2020).

[71] L Landau. ''Zur Theorie der Energieübertragung bei Stössen II'. In: *Phys. Z. Sowjetunion* 2 (1932), pp. 46–51.

[72] Clarence Zener. 'Non-adiabatic crossing of energy levels'. In: *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character* 137.833 (1932), pp. 696–702.

[73] J Robert Oppenheimer and George M Volkoff. 'On massive neutron cores'. In: *Physical Review* 55.4 (1939), p. 374.

[74] Hans A Buchdahl. 'General relativistic fluid spheres'. In: *Physical Review* 116.4 (1959), p. 1027.

[75] MP Fewell. 'The atomic nuclide with the highest mean binding energy'. In: *American Journal of Physics* 63.7 (1995), pp. 653–658.

[76] Dany Page. 'Strange stars: Which is the ground stage of QCD at finite baryon number'. In: *arXiv preprint astro-ph/9602043* (1996).

[77] Hannes Risken. *Fokker-planck equation*. Springer, 1996.

[78] Luca Cardelli. 'Two-Domain DNA Strand Displacement'. In: (June 2010). arXiv: 1006.2993 [cs.LO].

[79] P. T. Johnstone. *Notes on Logic and Set Theory*. USA: Cambridge University Press, 1987. ISBN: 0521335027.

[80] Herbert S Wilf. *generatingfunctionology*. CRC Press, 1989.

[81] Richard P Stanley. *Enumerative Combinatorics Volume 1 second edition*. 1986.

[82] Mireille Bousquet-Mélou and Marni Mishna. 'Walks with small steps in the quarter plane'. In: *Contemp. Math* 520 (2010), pp. 1–40.

[83] Helmut Prodinger. 'The kernel method: a collection of examples'. In: *Sém. Lothar. Combin* 50 (2004), B50f.

[84] Melissa E. O'Neill. *PCG: A Family of Simple Fast Space-Efficient Statistically Good Algorithms for Random Number Generation*. Tech. rep. HMC-CS-2014-0905. Claremont, CA: Harvey Mudd College, Sept. 2014.

[85] Mireille Bousquet-Mélou. 'An elementary solution of Gessel's walks in the quadrant'. In: *Advances in Mathematics* 303 (2016), pp. 1171–1189.

[86] Michael P Frank. Personal Communication. Nov. 2021.

[87] Michael P Frank. 'Asynchronous ballistic reversible computing'. In: *2017 IEEE International Conference on Rebooting Computing (ICRC)*. IEEE. 2017, pp. 1–8.

[88] Michael P Frank et al. 'Asynchronous ballistic reversible fluxon logic'. In: *IEEE Transactions on Applied Superconductivity* 29.5 (2019), pp. 1–7.

[89] Tadao Murata. 'Petri nets: Properties, analysis and applications'. In: *Proceedings of the IEEE* 77.4 (1989), pp. 541–580.

[90] Alan Mathison Turing. 'On computable numbers, with an application to the Entscheidungsproblem'. In: *J. of Math* 58.345-363 (1936), p. 5.

[91] Conor McBride. 'The derivative of a regular type is its type of one-hole contexts'. In: *Unpublished manuscript* (2001), pp. 74–88.