Computation by Origami-Templated DNA Walkers

Michael Austin Boemo

Christ Church

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Trinity 2016

To Nancy, Michael, Mary, and Aldo Boemo.

Acknowledgements

First and foremost, I wish to thank my supervisors **Professor Andrew J. Turberfield** and **Professor Luca Cardelli** for their constant support. Their investment in this project, as well as in my own personal development, is the reason I was able to make it this far. Without them, I would not be the scientist or person that I am today.

There are many other individuals that helped me reach this point, and I regret that I only have space to thank a few of them here. At Rutgers University, my decision to pursue a PhD was due to the fantastic experience of working with my undergraduate supervisors, **Professor Shridar Ganesan** and **Professor Gyan Bhanot**. I am very grateful for the friendship of **Dr. Shabnam Beheshti** and **Dr. Costis Papageorgakis**, both of whom I have aspired to and drawn inspiration from. Finally, I could never mention Rutgers without mentioning **Professor Barry V. Qualls**, my biggest supporter over the years.

At the University of Oxford, I benefited greatly from the knowledge of all members of the Turberfield Group. In particular, **Dr. Jonathan Bath** provided practical advice and guidance that was critical for the success of this work. **Alexandra E. Lucas** provided guidance in designing the experiments in this thesis, and her advice shaped the direction of this project. I also benefited from the work on DNA walkers and fluorescence spectroscopy done by **Natalie E.C. Haley**, and her work on single molecule imagining of DNA walkers represents an important next step in the field.

At Microsoft Research Cambridge, many good ideas emerged from conversations with **Dr. Andrew Phillips** and **Dr. Rasmus Petersen**. Much of the work in this thesis was inspired by their work on Microsoft Visual DSD, particularly the compiler for DNA walker circuits and the formalisation of blocking logic.

I gratefully acknowledge the European School of DNA Nanotechnology, created through a Marie Curie Action by the European Commission, for funding this project and for organising many thought-provoking meetings. In particular, **Dr. Lise Pedersen** devoted a lot of time and effort to organising these meetings and creating a great experience for the fellows.

To those named above (and many more), I am immensely thankful and lucky to have known and worked with all of you. This thesis is as much your achievement as it is my own.

Abstract

Interactions between DNA molecules can be used to perform computation. These DNA computing systems often use DNA molecules as freely diffusing reactants in a well-mixed solution. We demonstrate how DNA walkers tethered to an origami-templated track can perform computation. A DNA walker can block a track that intersects with its own, preventing another walker from stepping down this blocked track. These blockages are primitive operations that can be used to perform computation. This thesis demonstrates how blocking interactions between DNA walkers can evaluate formulae posed in propositional logic.

When anchorages in the track are viewed as networked machines and the DNA walker is viewed as a coordinated message passed between them, DNA walker circuits can be modelled as a distributed system. Techniques from formal verification can be used to check this system for errors, determining the probability with which the system will end up in a certain state. This forms the basis of a compiler that can automatically design a DNA walker circuit that evaluates a given propositional formula within a specified error tolerance. To show how DNA walker circuits can be simplified, we create a propositional logic system called blocking logic that is proven to be both sound and complete.

DNA walker circuits can be implemented and measured experimentally by using fluorescence spectrophotometry to track the position of a walker on the track. To demonstrate proof of principle, circuits were built that implement NOT and NOR operators. To make these circuits operate with minimal error, different sources of possible error were investigated and quantified.

Cumulatively, the novel contributions that this thesis makes to the field are:

- the experimental design and implementation of a DNA computing system that uses DNA walkers,
- probabilistic model checking software that automatically designs these DNA walker circuits,
- a propositional logic system that can simplify a DNA walker circuit to an equivalent circuit that uses fewer tracks.

Contents

Chapter 1

Introduction

1.1 Thesis Organisation

This thesis shows how DNA walkers can interact with one another to perform computation. Each walker steps down a track templated by a DNA origami tile, and if two tracks intersect, one walker can block the track of another walker at the junction. The particular computation performed by these blocking interactions is the evaluation of propositional formulae. These formulae are composed of logical propositions that can be true or false, joined together by logical connectives to create a compound proposition. For example, $x \wedge y$ (read: "x and y") is a compound proposition that is true if both propositions x and y are true. The following chapters describe both the theoretical properties and experimental implementation of DNA walker systems that can perform computation.

The thesis is organised as follows. Chapter 1 introduces the fundamentals of DNA nanotechnology and DNA computing in order to place this work in context. DNA is an attractive material for building nanodevices due to the specificity of base pair bonding, which makes building structures via self-assembly straightforward. The field has created DNA-based nanodevices that are both diverse in their applications and

imaginative in design. Many of these devices can perform computation by using different approaches than the one presented here.

Chapter 2 describes the theoretical properties of DNA walker circuits that evaluate propositional formulae, showing how a circuit design can be deduced from the formula that it should evaluate. To assist in the design process, a compiler is presented that will compile a DNA circuit design from a propositional formula supplied by the user. One of the critical steps in the design process is identifying how a circuit can be simplified. The rules for simplification are found by proving that blocking logic is a formal system that is both sound and complete, which provides the tools necessary for simplification.

Chapter 3 explains the experimental materials and methods used in this thesis. Many of the methods are similar to those used previously for DNA walkers, but several modifications are needed to account for the blocking mechanism. All of the DNA sequences and experimental protocols needed to repeat the experiments in this thesis are given in the appendices.

Chapter 4 shows how to implement DNA walker circuits experimentally. Two logic gates, NOT and NOR, are built and measured. The NOT gate is one of the simplest nontrivial circuits because it only has one blocking interaction. A NOR gate is an important example because it is functionally complete, meaning that any other gate can be constructed out of NOR gates. While constructing other logic gates is possible, the NOT and NOR gate provide proof of principle.

Chapter 5 describes different sources of error that may occur in DNA walker circuits and quantifies them to determine how often they occur. Identifying and quantifying as many sources of error as possible enables the construction of circuits that operate with high reliability, and marks a major step towards making DNA walker circuits an effective and reliable form of DNA computation. Finding and measuring all possible sources of error would be a considerable challenge and remains

an important topic for future work.

Chapter 6 discusses the conclusions that can be drawn from this thesis and ideas for future work. While the experimental methods and theoretical tools presented here represent a great deal of progress for this technology, further improvements are needed to make DNA walker circuits reliable and scalable. When such improvements are made, DNA walker circuits will be a valuable tool for applications such as nanofabrication, controllable synthesis of chemical reactants, and nanomedicine.

This thesis makes experimental, computational, and theoretical contributions to the field of DNA computing. The novel contributions this thesis makes are as follows.

- The experimental contribution is the design and implementation of a new type of DNA computation system that operates by using the blocking interactions between DNA walkers.
- The computational contribution is software that compiles a propositional formula into a design for a DNA walker circuit that can evaluate the formula.
- The theoretical contribution is a sound and complete propositional logic system that can be used to derive rules for finding equivalent DNA walker circuits.

1.2 Why DNA?

There are many building materials that can be used to create nanodevices. Why use DNA? One of the most attractive properties of DNA is its biocompatibility. Graphene, another commonly used nanomaterial, is a sheet of graphite one atom thick that can can be used to create two-dimensional nanocrystals [39]. It can also form carbon nanotubes [28] which have a number of applications [35] including a remarkable stiffness [92], high thermal conductivity [11], and a potential use as quantum wires [89]. Despite these applications, recent work suggests that graphene may be toxic to

cells which places limits on biological applications. As a biomolecule, DNA has no such limitation [12].

Watson-Crick base pairing in DNA is highly specific: adenine (A) forms two hydrogen bonds with thymine (T) and cytosine (C) forms three hydrogen bonds with guanine (G) in the characteristic right-handed double helix [86, 98]. While other base pairings are possible, such as G-T ("wobble" base pairing) and G-A bonding [48], these base pairings are less stable than traditional Watson-Crick base pairing [6, 66]. This specificity allows for construction of DNA nanodevices by self-assembly (see, for example, [51, 55]) which makes the nanofabrication of DNA-based devices and structures straightforward.

In addition to biocompatibility and base pairing specificity, DNA is easy to work with. It is a very robust molecule, and it is stable for up to 6 weeks at physiological temperature (37°C) when suspended in water [1]. DNA is stable for much longer periods of time at lower temperatures and in specialised storage conditions, which makes it an attractive material for archiving data [16, 24]. In addition, the cost of DNA synthesis continues to decrease, making it a cost-effective material [18].

1.3 Fundamental Operations

While DNA can be used to build static structures, it can also be used in the construction of dynamic nanodevices that can participate in a number of fundamental operations. These operations can reversibly or irreversibly change the state of the device, allowing it to respond to environmental stimuli or perform a coordinated function. The two operations used in this thesis are strand displacement and cleavage by enzymes.

1.3.1 DNA Strand Displacement

A DNA double helix is composed of two strands that are Watson-Crick reverse complements. Strand displacement occurs when another strand in solution takes the place of one of the strands hybridised in a double helix, displacing it. If the invading strand is identical to the strand that it displaces, there is no difference in free energy between the reactant and the product so the reaction will proceed slowly. This is known as blunt strand exchange.

Equilibrium of a strand displacement reaction can be shifted towards the product by use of a short "toehold" domain (Figure 1.1). The invading strand hybridises to a toehold and proceeds to displace the incumbent strand. There are more base pairs bound in the products than in the reactants, so this "toehold-mediated" strand displacement reaction has reduced the free energy of the system [77, 78]. Toehold-mediated strand displacement is one of the main operations used to create dynamic DNA nanodevices. For example, Yurke and colleagues created DNA nanotweezers that can switch between open and closed configurations by toehold-mediated strand displacement [106]. Turberfield and colleagues showed how toehold-mediated strand displacement allows DNA molecules to be used as fuel for nanomachines [93].

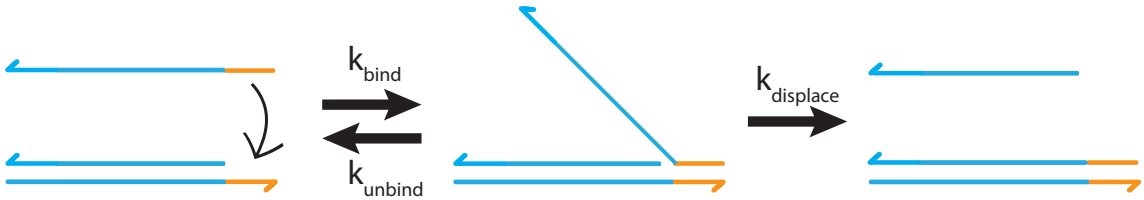


Figure 1.1: Toehold-mediated strand displacement. Individual strands of DNA are represented as half-arrows with the arrowhead at the 3' end. The invading strand hybridises to a short toehold domain, but can unbind again because hybridisation to a short toehold is a weak interaction. When the invading strand displaces the incumbent strand, the forward rate constant is much higher than the reverse rate constant because the reverse reaction cannot proceed via toehold hybridisation. The products of toehold-mediated strand displacement have more bases bound than the reactants, so the products have a lower free energy.

1.3.2 Restriction Enzymes

Restriction enzymes are one of the main tools of modern molecular biology, and catalyse the hydrolysis of phosphodiester bonds to cleave DNA near certain recognised nucleotide sequences [14, 60]. These recognised sequences, known as the enzyme's restriction site, allow for the targeted sequence-specific cleavage of a DNA molecule. The energy for this cleavage is supplied by the buffer in the form of a cofactor, which is generally a cation or ATP [13]. The specific cofactor required depends on the type of restriction enzyme.

Restriction enzymes have certain temperature and buffer conditions in which their cutting activity is optimal. Many enzymes have an optimal cutting activity near 37°C, but the temperature will vary with the type of enzyme [4]. It is important for the enzyme to have enough DNA to clamp onto, so some restriction enzymes have reduced activity when cutting near the end of a DNA molecule [2].

Certain restriction enzymes cut both DNA strands to produce blunt or sticky ends while others, known as nicking enzymes, only cut one strand. The experiments in this thesis use a DNA walker mechanism whereby a nicking enzyme creates a toehold so that a DNA walker can step via toehold-mediated strand displacement (Section 1.5).

1.4 DNA Self-Assembly

Base pairing specificity makes self-assembly a viable method for building DNA nanostructures [99]. The various DNA nanostructures that have been built can be broadly partitioned into origami and non-origami.

1.4.1 Non-Origami DNA Nanostructures: Lattices, Crystals, and Cages

Holliday junctions are "cross-shaped" four-way junctions formed by two DNA duplexes that occur naturally during homologous recombination [43]. A major landmark in the field of DNA nanotechnology was Nadrian Seeman's work in 1982 on immobile Holliday junctions, where the position of the junction stays fixed relative to the DNA strands [80]. Typical Holliday junctions have symmetry in the sequences that cause migration of the junction or unbinding into two double helices (Figure 1.2). Seeman created four rules for designing sequences whereby the junction stays fixed. The tessellation of immobile Holliday junctions creates the repeating structure of a DNA lattice [64, 73, 74, 103]. Applications of these DNA-based lattices have included using a DNA binding protein to create a switchable lattice configuration [59], the incorporation of a nanoactuator [38], and templating nanoelectronic components for potential uses in nanocircuitry [54].

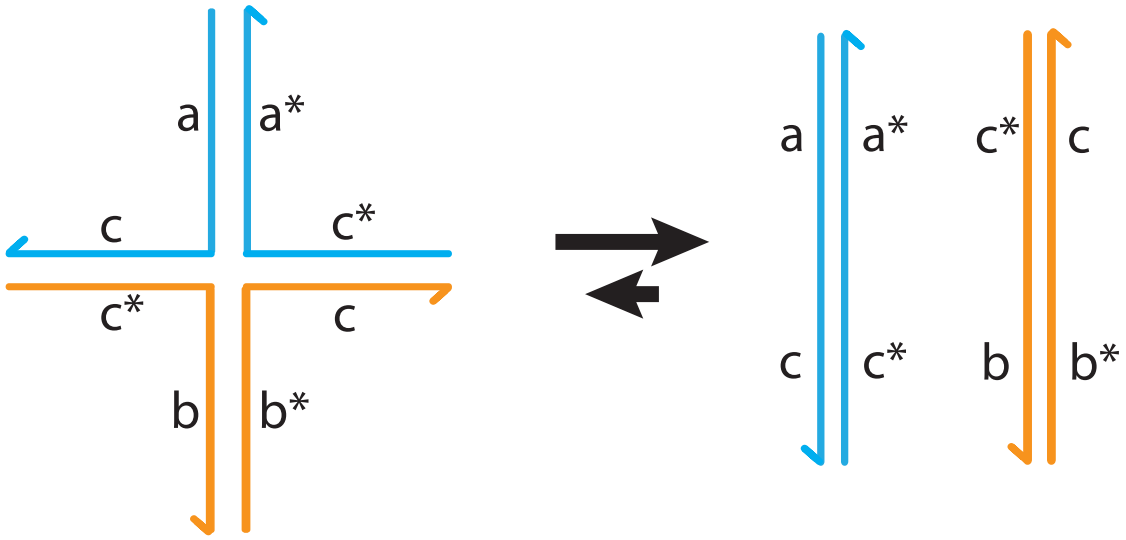


Figure 1.2: A typical Holliday junction. The junction motif has a higher free energy than two double helices (blue and orange), which shifts the equilibrium to the right. In addition to separating into two double helices, the junction between strands can also migrate. Seeman's work in [80] pioneered a method to design sequences for an immobile Holliday junction.

In addition to lattices, a number of three-dimensional shapes have been made such as a tetrahedron [40], octahedron [82, 109], and bipyramid [37]. One of the main applications of these DNA three-dimensional shapes is use as DNA cages. These cages can encapsulate proteins [36] and transcription factors [25], as well as perform targeted delivery of payloads to mammalian cells [96].

1.4.2 Origami-based Nanostructures

Work done in 2006 by Paul Rothemund demonstrated how a long, single-stranded, circular molecule of DNA (the "scaffold") could be folded into various shapes by hybridising to hundreds of short oligonucleotides (the "staples") as shown in Figure 1.3 [75]. These structures were constructed by self-assembly and were shown to form with high yield and stability, making DNA origami a useful method for building DNA nanostructures. Rothemund identified that DNA origami could form a variety of shapes in his 2006 paper (Figure 1.4) and the field has since used DNA origami to

create a wide variety of machines and devices.

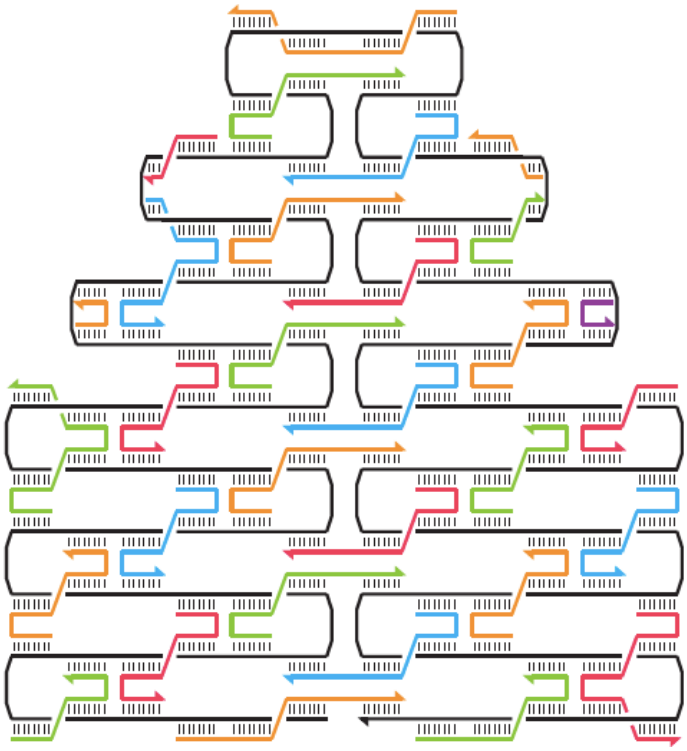


Figure 1.3: Figure from [75] showing a DNA origami tile formed from the hybridisation of a scaffold (black) to short staples (orange, green, blue, and red).

While Rothemund's original origami structures were two-dimensional, many groups have subsequently created three-dimensional origami structures (see, for example, [41, 47]) as well as the software tools needed to design them [33]. Origami-based DNA cages were made by Andersen and colleagues who designed a DNA origami box with a controllable lid [7] as well as by Douglas and colleagues who created a nanorobot out of DNA origami that can deliver payloads [32]. Origami has also been critical in templating the assembly of nanophotonic and nanoelectric devices [21, 31]. Origami has a multitude of uses; the applications above are only a sample. In the scope of this thesis, origami is used as a rigid substrate that templates the assembly of DNA walker circuits.

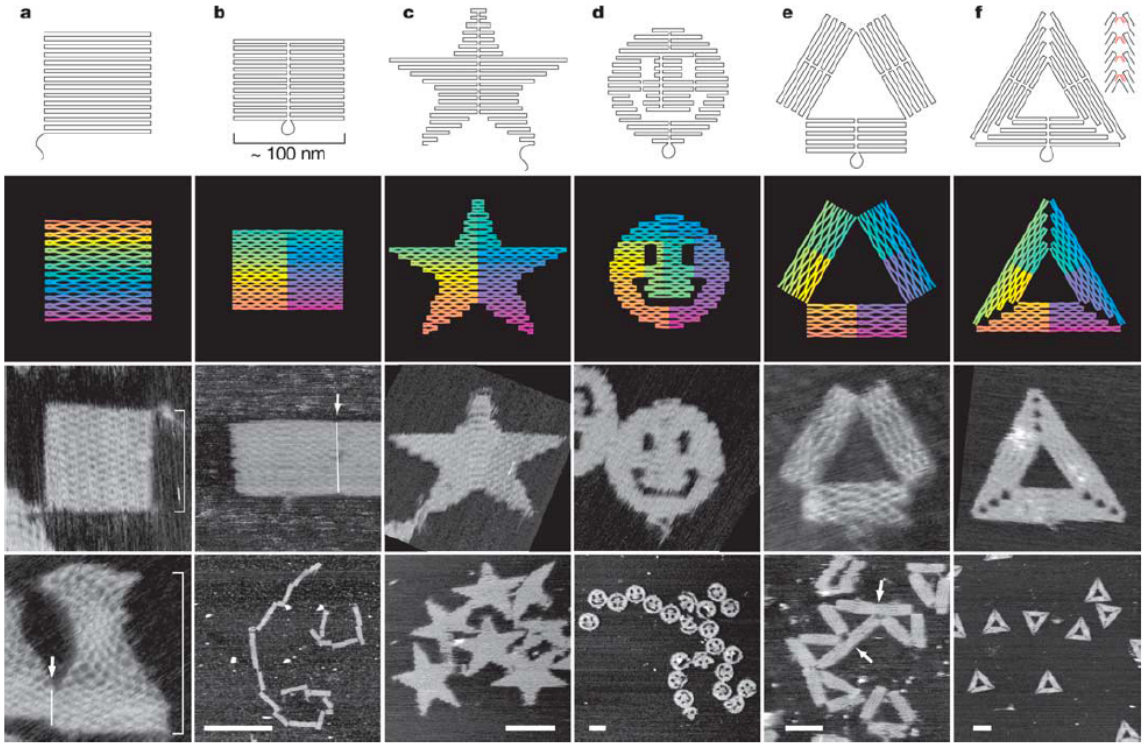


Figure 1.4: Figure from [75] that shows the design (top) and atomic force microscopy (bottom) images for (a) a square, (b) a tile, (c) a star, (d) a smiley face, and (e)-(f) two triangles.

1.4.3 The Minimum Twist DNA Origami Tile

The crossover positions on Rothemund's original DNA origami tiles from [75] have DNA helices with 10.7 base pairs per helical turn, while unconstrained B-DNA is generally considered to have 10.1-10.4 base pairs per helical turn [104]. This strain at the crossover positions introduces a global right-handed twist in the origami tile [30]. Wickham and colleagues created a new tile design that has 10.4 base pairs per helical turn at the crossover positions, reducing the twist in the tile (Figure 1.5) [100].

In this thesis, the minimum twist tile design shown in Figure 1.5 is used as a molecular pegboard to template the assembly of a track of anchorages on which a DNA walker will step (Section 1.5). If each anchorage is a 5' extension of a staple strand in the origami, then the minimum twist tile can accomodate a linear array of

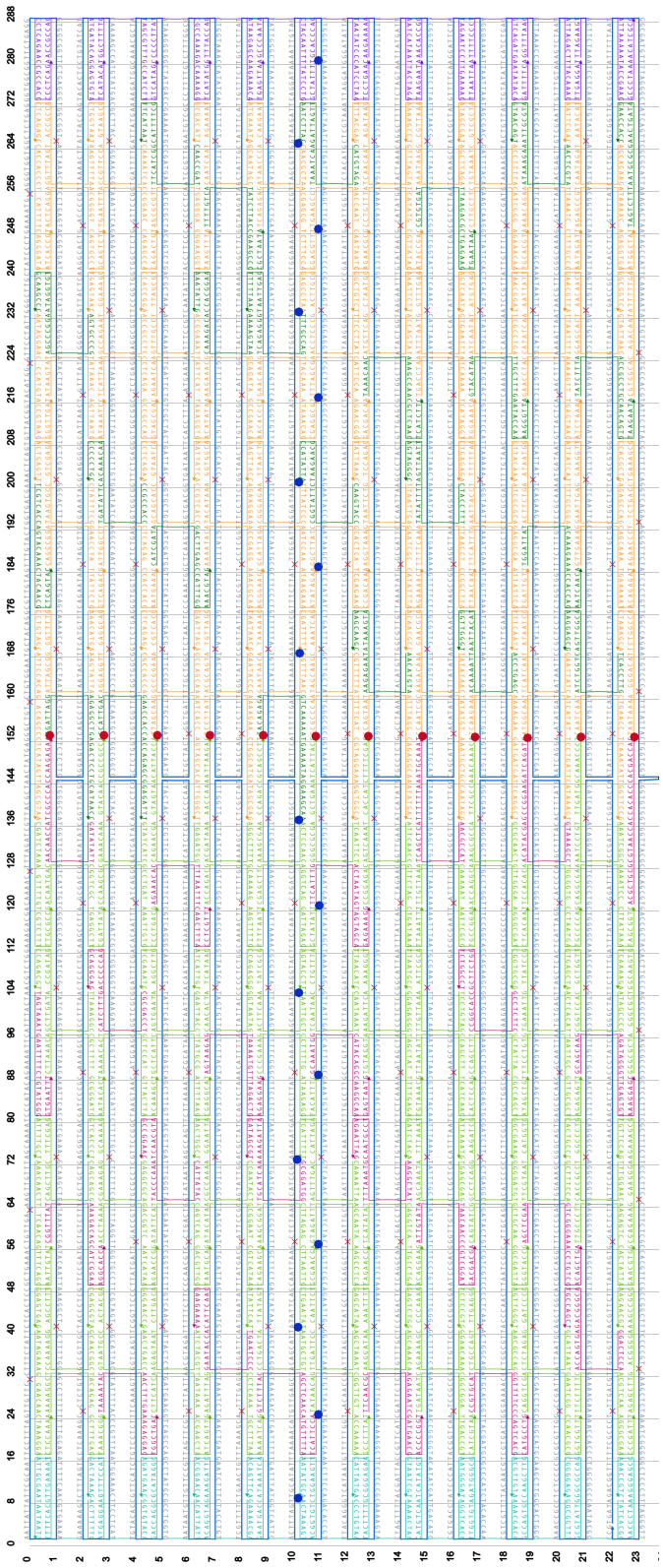


Figure 1.5: caDNAno [33] schematic of the minimum twist DNA origami tile from [100]. The tile can act as a molecular pegboard, accomodating twelve 5' staple modifications along its shortest dimension (red dots) and eighteen 5' staple modifications along its widest dimension (blue dots).

12 anchorages across its shortest dimension (Figure 1.5, red dots) and 18 anchorages across its widest dimension (Figure 1.5, blue dots). Therefore, the space on each individual origami tile is limited, so it is important to make the number of anchorages needed in the track as small as possible.

While the minimum twist origami design reduces the global twist present in Rothemund's original design, it is important to note that tile origami structures are still very flexible [30]. Short oligonucleotide linkers are enough to bind an origami tile into a cylindrical tube, and interhelix crossover sites are shown to have similar flexibility to single-stranded DNA [20]. This means that a tile is sufficiently flexible to fold over on itself while suspended in solution, bringing otherwise distant regions of the origami into close proximity. The effect that this can have on the DNA systems presented in this thesis is investigated in Section 5.2.

1.5 DNA Walkers

Moving structures from one location to another in a controlled, programmable fashion is of central concern in biology, materials science, and other fields. In biology, kinesin is a motor protein that hydrolyses ATP to step along microtubules [95]. It can transport cargo from one location in the cell to another through its characteristic walking movement. For the nanofabrication of materials, controlled and reliable assembly at the nanoscale is a major challenge. A nanodevice that is capable of programmable, directed movement could simplify this process by transporting materials between locations.

One of the most exciting features of DNA nanotechnology is the ability to create dynamic nanostructures that can move, change configuration, and interact with other nanostructures or the surrounding environment. A DNA walker is a dynamic DNA nanostructure that is capable of moving in discrete steps on a substrate. While

all DNA walkers adhere to this broad definition, the design of the walker and the substrate on which it steps can vary widely. In addition, different DNA walker designs often come with advantages and disadvantages; the design choice should depend on the intended application. The review written in 2015 by Pan and colleagues describes many designs created by the field [63].

1.5.1 Non-autonomous DNA Walkers

Non-autonomous walkers require the addition of external reagents or signals to complete each step. Their movement is "clocked" into discrete timesteps, whereby each walker in the system takes a step forward when the stepping signal is added. A key advantage is that the walkers' stepping can be precisely controlled, as the experimenter knows where the walkers are positioned on the track at each timestep.

While clocked stepping can reduce errors in some applications, it comes with the disadvantage of being largely incompatible with biological systems. The stepping signals typically come in the form of oligonucleotides that are added to solution [81, 83]. This is possible *in vitro* by manually adding the strands to the reaction solution or by using a microfluidics device to wash strands in and out. In a cell, however, this degree of external control would not be possible.

The walker developed by Shin and Pierce in 2004 shows a strategy for implementing a non-autonomous walker (Figure 1.6) [83]. The authors intended to mimic the stepping of kinesin by creating a walker with two "feet". Oligonucleotides are added to the solution that tether the walker's feet to anchorages on a track. By iteratively adding a strand that tethers the walker's foot to the next anchorage along with a strand that displaces the strand that tethered the walker to the previous anchorage, the walker can proceed down the track in a stepwise fashion.

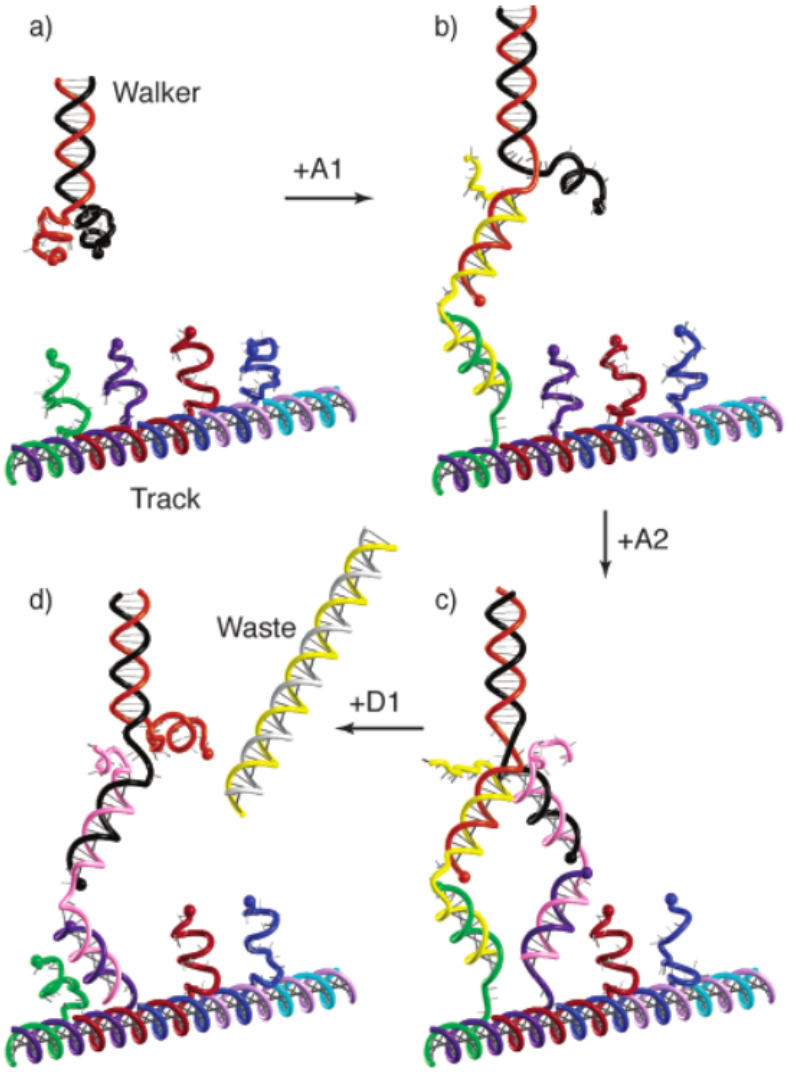


Figure 1.6: Figure from [83], which shows a non-autonomous walker stepping. Each step requires the addition of a tethering strand to bind the walker's foot to the next anchorage in the track, along with a remover strand that displaces the strand tethering the walker to the previous anchorage. By iteratively adding these two strands to the solution, the walker can step down the track.

1.5.2 Autonomous DNA Walkers

Autonomous walkers are able to step continuously without the addition of any external reagents or signals. All materials are added to the solution at the beginning of the experiment, and the walker is allowed to step until the experiment is concluded. Autonomous stepping can make the DNA walker's movement highly stochastic and difficult to predict. As a result, autonomous walkers can be more prone to errors in applications where precise and controlled movement is required. In particular, coordinating the timing of autonomous walkers can be difficult, especially when an application requires one walker to reach a certain location before another walker. Despite these drawbacks, autonomous walker systems have the advantage of being self-contained. These systems do not need any externally added reagents or a user to monitor and control their state. If further improvements were made to autonomous DNA walker systems, a long term goal in the field is creating walker systems that operate within cells to perform biomedical functions.

Examples of autonomous DNA walkers include the DNAzyme walker from Tian and colleagues, which is capable of directed movement along a track [91]. This walker uses a deoxyribozyme to catalyse the cleavage of an anchorage in the track so that the walker can step to the next anchorage via toehold-mediated branch migration. Pei and colleagues used a similar mechanism to create a DNA spider that has a DNAzyme on each of its four legs [65]. The DNA spider moves by biased diffusion as the DNAzymes on its legs cleave a field of anchorages. This design was improved upon by Lund and colleagues who created robot-like behaviour from a DNA spider [57]. While DNAzymes catalyse the hydrolysis of the phosphodiester backbone, this function can also be performed by normal restriction endonucleases. The "burnt-bridges" walker from Bath and colleagues uses a similar mechanism to the walker by Tian *et al.*, but the energy comes from a nicking enzyme rather than a DNAzyme [10]. Nicking enzymes can also power a two-footed walker that can step to the end of

a track and then reverse direction [9].

The DNA walker system used in this thesis is the autonomous "burnt-bridges" walker designed by Bath and colleagues (see the mechanism in Figure 1.7) [10]. It was chosen for its simplicity, reliability, and the system's ability to create effectively irreversible state changes by enzymatic cleavage. The burnt-bridges walker is autonomous, so it does not require any external control or clocking. Therefore, if the design were further improved upon, the systems presented in this thesis may be useable in biomedical applications.

The burnt-bridges walker is a single strand of DNA that hybridises to single-stranded DNA anchorages (Figure 1.7). These anchorages are tethered to an origami tile, and can be arranged in a linear array to form a track. The energy input for the walker's directed movement comes from the action of a nicking enzyme, which binds to the restriction site that is completed when the walker hybridises to an anchorage. Cutting the anchorage exposes the toehold on the walker, which allows it to step onto the next anchorage in the track via toehold-mediated branch migration.

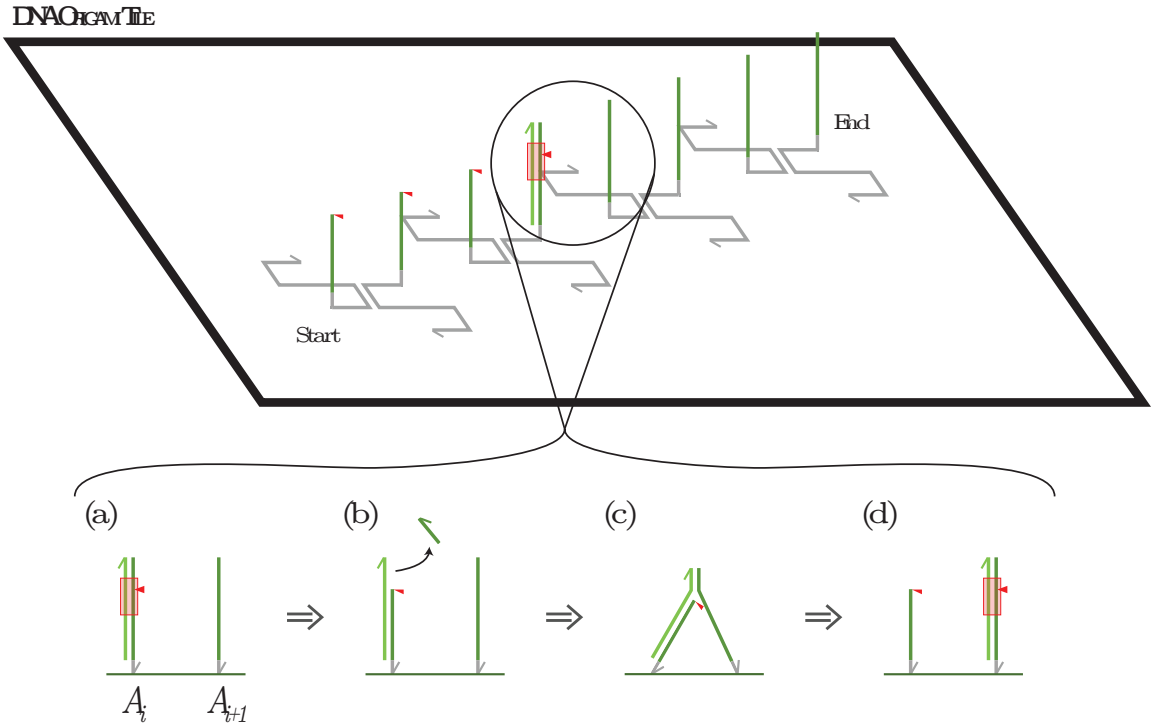


Figure 1.7: The burnt-bridges DNA walker mechanism [10]. Single-stranded oligonucleotides are shown as half-arrows, with the arrowhead indicating the 3' end. (a) A single-stranded DNA walker is bound to a complementary single-stranded anchorage (A_i), completing the restriction site (red box) for a nicking enzyme. (b) The nicking enzyme cuts the 5' end of anchorage A_i at the location shown by the red triangle, exposing a short toehold on the 3' end of the DNA walker. (c) The toehold on the walker binds to anchorage A_{i+1}. (d) The DNA walker steps from anchorage A_i to A_{i+1} via a toehold-mediated branch migration reaction. By continuously repeating these steps, a DNA walker can navigate down a track of single-stranded anchorages. The cut toeholds inhibit the walker from stepping backwards, creating unidirectional motion. (This figure was prepared by Alexandra E. Lucas for a joint publication in [15].)

1.6 Principles of Computation

Before highlighting some examples of previous work done in DNA computation and discussing the advantages of performing computation with DNA walkers, this section describes topics and terms from computer science that are commonly used in molecular computing. Unfamiliar readers are referred to any standard text on the theory

of computation, such as [17] or [84].

There is no one definition of "computation," and the definition is the subject of an ongoing philosophical debate (see [68] for the mechanistic side of the argument and a broad discussion on the debate). For the purpose of this thesis, a suitable definition for computation is the process of a machine accepting (or possibly rejecting) an input written in a formal language. There are several different types of machines that can accept or reject inputs: finite state automata, pushdown automata, and Turing machines [94].

Different machines have different levels of computational potential: some computations can be performed by one type of machine, but not by another. The Church-Turing Thesis states that there is no computation, physical or theoretical, that cannot be performed by a Turing machine [23, 94]. A Turing machine consists of a tape of infinite length, divided into cells that each contain one symbol from a finite alphabet. It also has a head that can read and write symbols on a cell of the tape, as well as move the tape one cell to the right or one cell to the left. The state register stores which state the Turing machine is in. A table matches the current state and the current symbol under the head to a set of instructions that tells the machine whether to erase or write a symbol, move the head, or assume a new state. A function is called computable if there is a Turing machine that can perform the above manipulations to reach an end state.

There is another theoretical machine called a universal Turing machine that can emulate any Turing machine by reading a description of that machine as well as the input that the Turing machine would have been given. Any system that can perform the function of a universal Turing machine is called Turing complete. A simple example of a Turing complete system is the lambda calculus (see [42] for a modern description and [23] for the original work). Traditional silicon-based computing systems are Turing complete up to a memory limit, as they are not able to replicate the

infinite tape feature of a Turing machine.

1.7 DNA Computing

Fundamental operations on DNA, such as enzymatic cleavage and toehold-mediated strand displacement, change the state of a DNA system. These fundamental operations can be used as the primitive operations for performing computation with DNA. This was first demonstrated by Adleman in 1994 by using techniques of molecular biology to solve the Hamiltonian path problem [5]. A graph has a Hamiltonian path if there is a path that visits each of the graph's vertices exactly once. Adleman combinatorially encoded a large number of paths in DNA sequences, and then used a series of gel purifications and sequence-specific exclusions to determine which DNA sequences (if any) encoded a Hamiltonian path. Adleman's method required seven days of laboratory work to solve a very specific computational problem, but a wide variety of mechanisms have been designed since then to perform more general computation. This section categorises several examples into DNA computing mechanisms that operate as freely diffusing strands interacting in a well-mixed solution, as well as other computing mechanisms that use DNA molecules tethered to rigid structures.

1.7.1 DNA Circuits in a Well-mixed Solution

DNA molecules can perform computation as freely diffusing reactants in solution, where the interactions between molecules are determined by base pairing specificity rather than geometric or topological constraints. Sakamoto and colleagues used a DNA hairpin mechanism to evaluate the satisfiability of a Boolean formula [76]. Seelig *et al.* created AND, OR, and NOT gates using single stranded DNA molecules as inputs and outputs [79]. In addition to using DNA to solve problems in propositional logic, work done by Qian and colleagues in 2011 created a stack machine that could

perform Turing universal computation by using DNA strand displacement [70]. To make DNA circuits scalable and more efficient, Zhang *et al.* designed a catalytic gate whereby an input DNA molecule can catalyse the release of an output DNA molecule to amplify a signal [108].

DNA mechanisms have a computational potential ranging from that of propositional logic to Turing universal. However, the interactions between DNA molecules can also be viewed as a chemical reaction network. Work by Soloveichik *et al.* showed that interaction between DNA molecules via strand displacement can emulate any chemical reaction network [85].

Some of the most ambitious DNA computing work to date was done by Lulu Qian and Erik Winfree in 2011, where they created a 4-bit square root calculator by using cascading logic gates [71]. There were over 130 individual DNA molecules used in the mechanism. The success of such a complicated system is an important demonstration of the scalability of DNA circuits.

1.7.2 DNA Circuits Tethered to Rigid Structures

There are also DNA computing mechanisms that operate by using reactants tethered to rigid structures. The rigid structure introduces a geometric constraint where two reactants can only interact if they are close to one another. DNA origami is typically used as the rigid structure because it can template assembly.

One of the first examples of a DNA computer that used a rigid structure was work by Liu *et al.* in 2000, where all possible solutions of a combinatorial problem were tethered to a surface [56]. As solutions were shown to be incorrect, they were eliminated and the remaining solution was amplified by PCR. This mechanism, however, could only solve combinatorial problems.

In 2005, Yin and colleagues created a DNA mechanism that uses a system of rigid tracks to mimic the mechanical operation of a Turing machine, demonstrating

universal computation [105]. Wickham and colleagues developed a DNA mechanism that enabled the burnt-bridges walker from [10] to navigate a network of tracks on an origami tile, computing via a binary decision tree [101]. The mechanism was modelled and computational potential further studied in [26, 27].

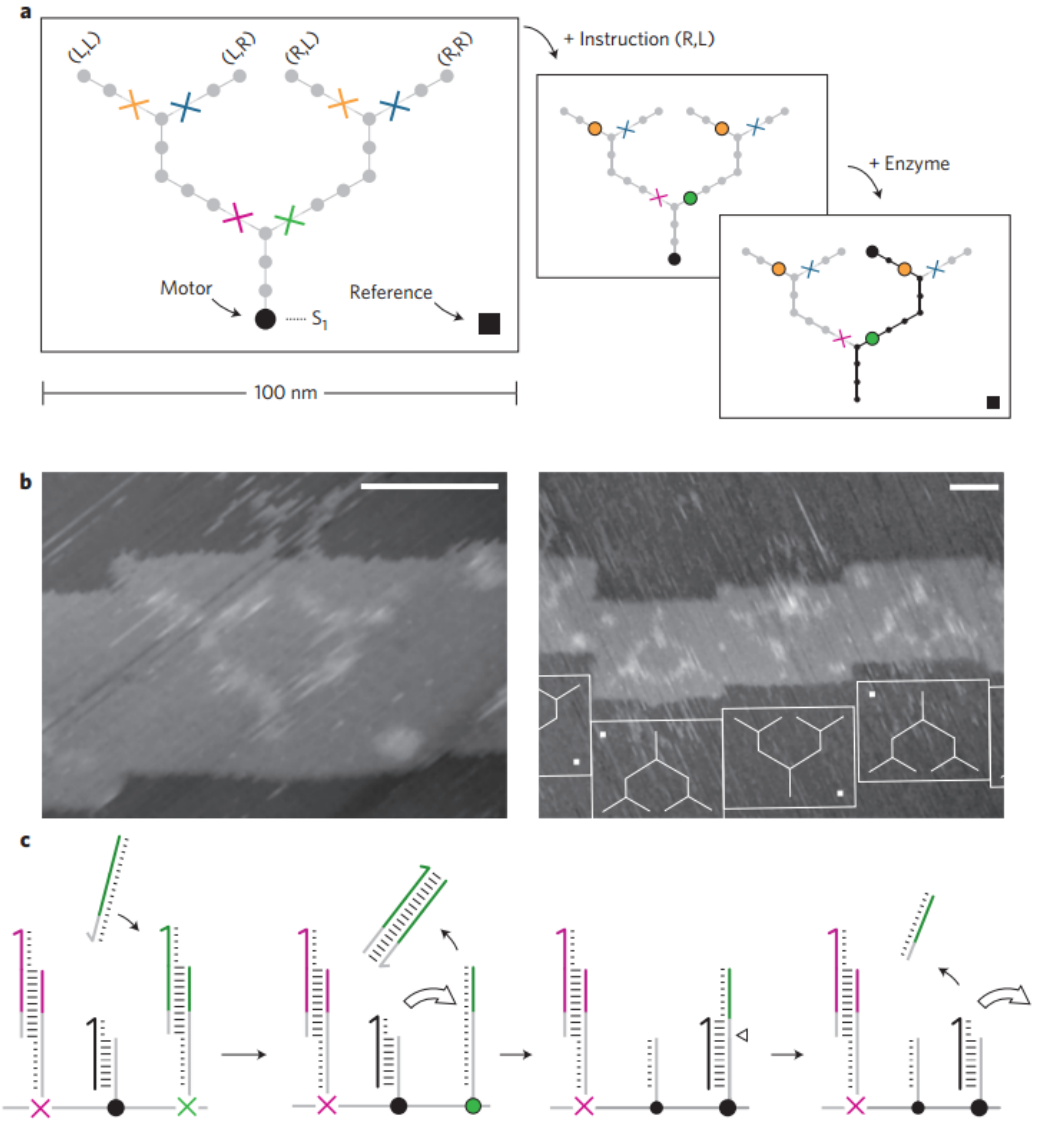


Figure 1.8: Image from [101] showing the design and mechanism of the binary decision tree that uses the burnt-bridges walker mechanism. (a) Schematic of the binary decision tree, showing each branched track guarded by blocking strands. The route that the walker takes down the track can be programmed by adding the appropriate release strand that removes a blocking strand. (b) Atomic force microscopy images of the track on the origami tile. (c) Mechanism for unblocking a path down the track.

1.7.3 Why Compute with DNA Walkers?

The examples above show a wide variety of methods for computing with DNA, both with reactants that diffuse in a well-mixed solution and with DNA molecules tethered to a rigid structure; why compute with DNA walkers? The main advantage of computing with DNA walkers is gated transportation. As discussed in Section 1.6, a computation is the acceptance (or possible rejection) of an input written in a formal language. For DNA computing systems that use walkers, the system can incorporate a walker that moves to a certain location if and only if the input is accepted.

Figure 1.9 shows a simplified schematic of the binary decision tree from [101]. Each fork in the track represents a choice between a propositional variable and its negation, and two or more cascading forks provide a natural implementation of an AND ($\wedge$) operator. Beginning at the start of the branching track, a DNA walker and any cargo attached to it will only reach the rightmost branch if $\neg X \wedge \neg Y$ is true. If $\neg X \wedge \neg Y$ is false, then the walker is kept away from this location. This feature is difficult to achieve with other methods of DNA computation, making computing with DNA walkers a useful technology in applications where this property is required.

1.7.4 Errors and Stochasticity in DNA Computing

A DNA computation is a chemical reaction, and chemical reactions rarely occur with 100% yield. There are side reactions, unwanted products, and unexpected intermediates that can all lead to errors in the computation. Some quantity of error is inevitable in DNA computing, and effectively managing these errors is an important aspect of any DNA computer's design.

An important part of designing a DNA computation system is controlling the Watson-Crick base pairing specificity of the oligonucleotides in the system so that only the intended strands hybridise to each other. One common error is a leak reaction whereby two DNA molecules in the system can hybridise unexpectedly. This

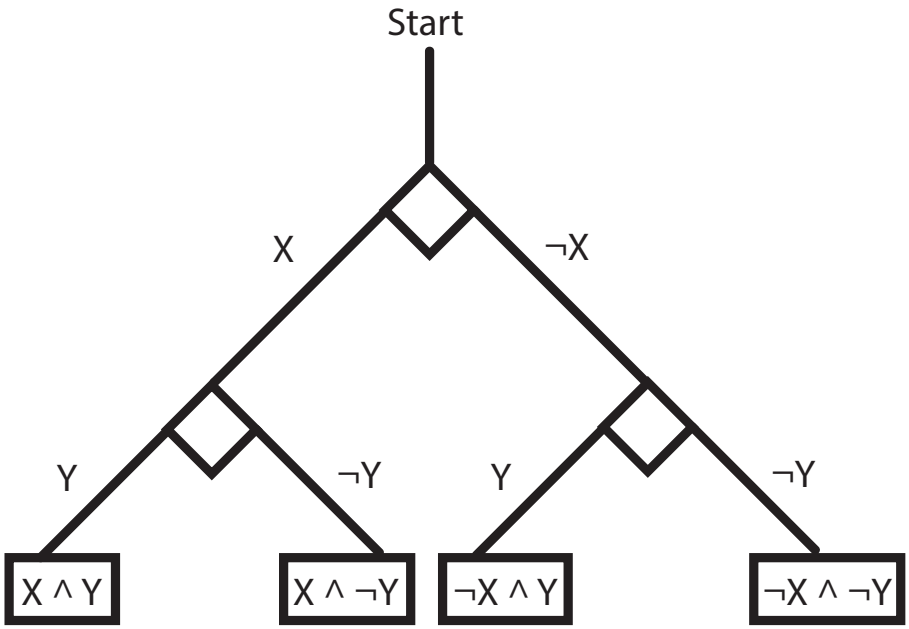


Figure 1.9: Simplified schematic of the binary decision tree system from [101]. A walker begins stepping at the start of the branching track, and each diamond is a fork in the track. The path that the walker takes at a fork depends on which path has been unblocked, and this path represents a choice between a propositional variable and its negation. Multiple forks implement an AND operator. The location that the walker steps to depends on which of $\{X \wedge Y,\ X \wedge \neg Y,\ \neg X \wedge Y,\ \neg X \wedge \neg Y\}$ is true. If cargo is attached to the walker, the cargo's destination is under logic-gated control which is difficult to achieve with other methods of DNA computation.

can occur due to poor design of the system or because the design requires many of the same sequence motifs (such as a restriction site on multiple strands). There are also inevitable, unpredictable leaks that are the result of weak binding between sequences that are not perfect Watson-Crick reverse complements. As systems scale up, these leak reactions can be very difficult to detect by hand.

One advantage of Qian and Winfree's design in [71] was that the catalytic and thresholding activity of the seesaw gates provided error control. A signal had to be above a certain threshold in order to propogate. This allowed the authors to create a large number of cascading DNA logic gates. More recent work by Thachuk and colleagues provided a sequence motif that makes a step towards leakless DNA strand displacement reactions [90]. The authors also propose making the ends of hybridised strands G-C rich to prevent the creation of a toehold due to fraying.

While there are limited steps that can be taken to control error in DNA computing systems that use freely diffusing reactants, improving methods of error control is a major goal for the field. The error control methods described here are more difficult to implement in computation systems that use DNA walkers, but creating thresholding by requiring a certain number of walkers to reach a location would be an interesting topic for future work. In this thesis, errors are minimised by careful design. Chapter 2 discusses computational and theoretical tools that can reduce the error in designs, and DNA walker-specific errors are quantified in Chapter 5.

1.8 Modelling in DNA Nanotechnology

Observing what happens at the nanoscale can be difficult, and leak reactions become increasingly difficult to detect by hand as systems scale up. In biology, mathematical and computational models can "connect the dots" between experimental measurements by proposing a mechanism for how the system behaves between these time

points. Computational methods can also assist by automatically designing DNA systems in such a way that the number of potential leak reactions is minimised. In doing so, modelling can both lead to insight while saving time and money by improving experimental design.

The existing models used in DNA nanotechnology vary in resolution. Some models use molecular dynamics to predict how individual bases in a DNA sequence can form and break bonds over time. This type of model can be ideal for measuring hybridisation rates and observing the process of strand displacement, but it has the drawbacks of being computationally expensive and only capturing a short period of simulation time. Other models investigate the DNA system at the domain level, abstracting away the fine detail to focus on the "bigger picture" of a DNA system's various strands and how their domains interact. The choice of model will depend on the scientific question to be answered.

One fine resolution model is oxDNA, created by Ard Louis and Jonathan Doye in 2013, which is a course-grained model for DNA molecular dynamics [34, 88]. oxDNA has been used to study many of the nanostructures produced by the field, including DNA nanotweezers [62] and the burnt-bridges walker [87]. CanDo, a modelling platform created by Kim and colleagues, can give a three-dimensional model of an origami structure and predict its flexibility [19, 49]. While both oxDNA and CanDo are useful for estimating physical properties of a system, NUPACK is a software package specifically geared towards system design [107]. By calculating the partition function of a system, NUPACK can determine the minimum free energy configuration and provide the probability that each base is bound.

Lower resolution models abstract DNA systems to domain-level interactions. In 2009, Andrew Phillips and Luca Cardelli developed a DNA strand displacement calculus, along with a syntax and semantics for describing strand displacement primitives [67]. This strand displacement calculus formed the basis for Microsoft Visual DSD,

which allows users to compile a DNA strand displacement system by inputting reactants in the language's syntax [53]. Once compiled, the software uses the strand displacement calculus to detect leaks and predict the reaction's products. Subsequent improvements have been made to Visual DSD in order to incorporate tethering, which allows the tethering of reactants to rigid structures [52]. With this modification, Visual DSD can be used for DNA systems that are tethered to rigid structures or systems that operate by using freely diffusing reactants. The model presented in Chapter 2 will abstract DNA systems even further to model a system of DNA walkers at the track level.

Chapter 2

DNA Walker Circuits

This chapter describes how interactions between DNA walkers can perform computation. It is assumed that each DNA walker can only step down its own track, and will continue stepping until it reaches the end of its track or until it is blocked. When a DNA walker steps down its track, it can block another track that intersects with its own. Once blocked, a walker will not be able to step down the track to reach the track's end. The blocking walker has therefore interacted with another walker by blocking it, changing the state of the system. Whether or not a blocking walker begins stepping down its track is a logical proposition that is true or false.

Track blocking by a DNA walker is a primitive operation that can be used to perform computation (Section 2.1). These primitive blocking operations can be used to create logic gates. However, correct operation relies on walkers in the system reaching the ends of their respective tracks in a particular order. Otherwise, errors can occur (Section 2.2). Therefore, it is possible to describe a DNA walker circuit using the propositional formula that it evaluates (Section 2.3). The axioms and theorems from propositional logic can be used to search for a simpler, equivalent circuit that uses fewer logic gates. Some DNA walker circuits, such as the FANOUT gate, cannot be described using propositional logic and are instead abstracted as a

directed graph (Section 2.4). All DNA walker circuits can be viewed as distributed systems (Section 2.5), and techniques from formal verification can be applied to find the probability of error in the circuit. Sections 2.3 - 2.5 all work towards Section 2.6, where these techniques can be used to create a compiler that automatically designs a DNA walker circuit to evaluate a given propositional formula within a specified error tolerance. The compiler takes an input formula written in propositional logic and works by,

1. using the axioms and theorems from propositional logic to search for an equivalent propositional formula that uses fewer logical connectives,
2. parsing the propositional formula into a tree,
3. using the parsing tree and a choice of gate design to create a directed graph that describes the topology of the circuit,
4. using the directed graph to compile a probabilistic model for the circuit,
5. searching through the possible track lengths for the track design that has the shortest tracks while still operating within a specified error tolerance.

To determine whether a circuit can be simplified, blocking is defined as a formal logic system that is proven both sound and complete (Section 2.7). Finally, this formal system is used to simplify an example circuit to demonstrate how it can be used (Section 2.8). The descriptions and figures are similar to those in [15].

The notation in this chapter is as follows. The truth value of a propositional variable, formula, or circuit is either True or False. Propositional variables are written in italics and the corresponding tracks and walkers are written in boldface. For example, walker **x** begins stepping on the **x** track if the propositional variable x holds the value True. Equivalence between two formulae or circuits is denoted by $\equiv$, a binary equivalence relation which indicates that the two operands have the same

truth table. For clarity, logical connectives are sometimes written using the symbols in Table 2.1.

NOT	$\neg$
AND	$\wedge$
OR	$\vee$
IMPLIES	$\Rightarrow$
NOR	$\downarrow$
NAND	$\uparrow$

Table 2.1: The logical operators in the left column can be represented by using their logical symbols shown in the right column.

2.1 Blocking as a Primitive Operation for Computation

The function $f : \mathcal{B}^k \to \mathcal{B}$ where $\mathcal{B} = \{\text{False}, \text{True}\}$ is called a Boolean function. It takes k input arguments that each hold a value of True or False and produces a single logical output. Simple examples include NOT, AND, OR, and NOR (Table 2.2). There are sets of Boolean functions that are functionally complete; any Boolean function is the composition of functions from a functionally complete set. There are several sets with two elements, such as $\{\mathsf{NOT}, \mathsf{IMPLIES}\}$, and also two singleton sets, $\{\mathsf{NOR}\}$ and $\{\mathsf{NAND}\}$. Any Boolean function can be written as the composition of either NOR or NAND functions. For example, if x and y are propositional variables,

$$\mathsf{AND}(x, y) \equiv \mathsf{NOR}(\mathsf{NOR}(x, x), \mathsf{NOR}(y, y)),$$

$$\mathsf{NOR}(x, y) \equiv \mathsf{NAND}(\mathsf{NAND}(\mathsf{NAND}(x, x), \mathsf{NAND}(y, y)), \mathsf{NAND}(\mathsf{NAND}(x, x), \mathsf{NAND}(y, y))).$$

A logic gate is a physical device that implements a Boolean function. In traditional silicon-based computing, logic gates operate by using transistors as electronic switches [8]. In this thesis, switches are blocking interactions between DNA walkers. Logic gates with feedback (Figure 2.1) are Turing complete up to a memory limit, and hence

are an important way of implementing the set of computable functions.

NOT	
x	Output
False	True
True	False

AND		
x	y	Output
False	False	False
True	False	False
False	True	False
True	True	True

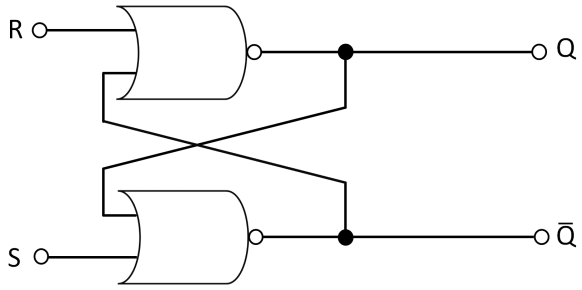
OR		
x	y	Output
False	False	False
True	False	True
False	True	True
True	True	True

NOR		
x	y	Output
False	False	True
True	False	False
False	True	False
True	True	False

IMPLIES		
x	y	Output
False	False	True
True	False	False
False	True	True
True	True	True

NAND		
x	y	Output
False	False	True
True	False	True
False	True	True
True	True	False

Table 2.2: Truth tables for the Boolean functions used in this thesis. The NOT function takes one input (it has arity 1) while the other functions take two inputs (they have arity 2). The output of the function is shown for every possible combination of inputs.



SR Latch		
S	R	Response
False	False	No Change
True	False	Q=True,$\overline{Q}$=False
False	True	Q=False,$\overline{Q}$=True
True	True	Not Allowed

Figure 2.1: An SR latch (or "flip flop") is a simple example of logic gates with feedback. The latch consists of two NOR gates where the output of one gate is also the input to the other. By changing the S and R signals, the latch can flip between states Q and $\overline{Q}$.

Blocking operations are attractive because implementing a NOR gate is straightforward (Figure 2.2, top left). Each propositional variable is associated to its own DNA walker and track. If a propositional variable holds the value True, the corresponding walker will begin stepping down its track. For each gate, there is a designated output track. If the output walker makes it to the end of this track, the gate outputs True. Otherwise, the gate outputs False. A NOR gate requires three tracks: an output track

that corresponds to a proposition always equal to True (denoted by **1**), and two additional tracks corresponding to propositional variables x and y. The **x** and **y** tracks are each able to block the **1** track at an intersection. If one or both of propositions x and y hold the value True, the corresponding walker or walkers will block the **1** track. The **1** walker then cannot reach the end of its track and the gate outputs False. If both propositions x and y are False, the **1** walker can reach the end of its track and the gate outputs True.

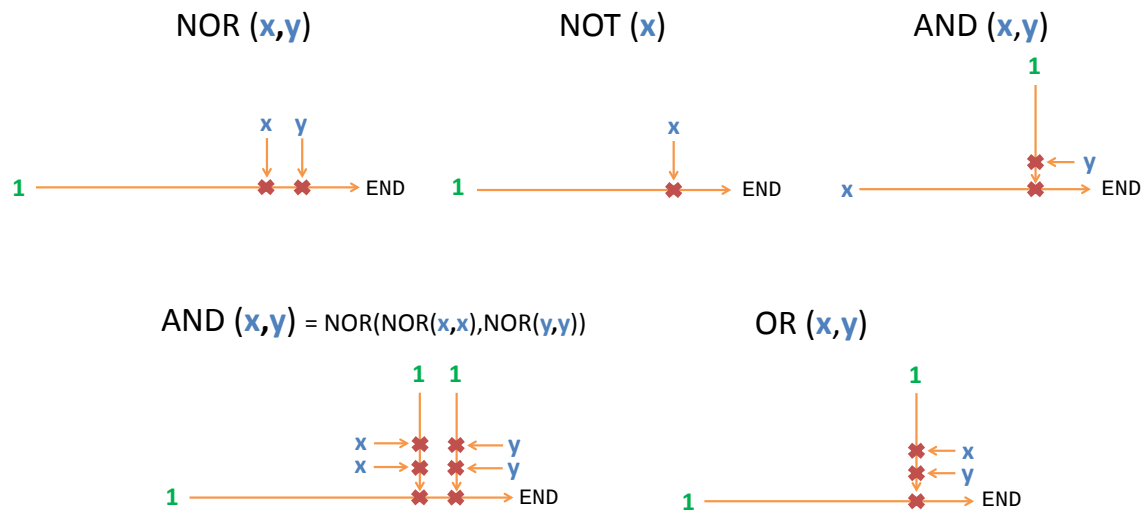


Figure 2.2: Track diagrams showing possible choices of design for NOR, NOT, OR, and AND logic gates that use the interaction (blockage) between DNA walkers. Each walker steps only on its own track, and all walkers begin stepping at the same time. Walkers denoted by **1** will always walk while **x** and **y** are walkers that will only start walking if their respective propositions hold the value True. Walkers can block a track at the junctions marked with red crosses. The gate evaluates to True if the output walker whose track has END at the end of it is indeed able to step to the end without being blocked.

A NOR operator is functionally complete, meaning any other gate can be constructed using the composition of NOR gates (see the track digram for AND in Figure 2.2, bottom left). However, there are other equivalent gate designs that are simpler because they require fewer tracks and junctions. As shown in Figure 2.2, it is possible

to reduce the number of tracks in the AND gate from seven to three. These simplifications are important for reducing error and fitting a circuit onto an origami tile. While the track design can be simplified, there are other simplifications that can be computed using axioms and theorems from propositional logic.

2.2 Order of Execution

For a DNA walker circuit, a track has executed when its walker reaches the end of the track. The order in which these processes execute is critical for DNA walker circuits to produce the correct output. An incorrect order of execution is an important source of error, but this error can be controlled by carefully designing the circuit. For example, the NOT gate in Figure 2.2 requires the **x** walker to step to the end of the **x** track and block the **1** track before the **1** walker arrives at the junction. If both walkers step at the same rate and begin stepping at the same time, then the **x** track must be shorter than the **1** track so that the **x** walker arrives at the junction first. Therefore, the **x** track must execute before the **1** track. This has a direct implication for DNA walker circuit design: there is limited space on a minimum twist DNA origami tile (see Section 1.4.3). The rate at which a walker steps is sequence-specific, so the stepping rate can also be controlled by changing the sequence of individual walkers [58].

Controlling the order of execution for processes that use autonomous walkers is more difficult than controlling non-autonomous walkers. A signal must be given each time a non-autonomous walker takes a step forward, so controlling the timing with which each walker reaches the end of its track would be straightforward in this case. Timing for autonomous walkers is highly stochastic. To understand why, consider a very simple model for the NOT gate in Figure 2.2 where a track **x** consisting of two anchorages blocks track **1** on its sixth anchorage. In this model, each track is associated with a fair coin that is flipped at each time step, and the walker on the

track takes a step forward if its coin shows heads. The correct order of execution in this circuit is the **x** walker stepping to the end of its track and blocking the **1** track before the **1** walker arrives at the sixth anchorage. Otherwise, the circuit will output an error because the **x** walker will not have blocked the **1** walker. This error corresponds to flipping six heads on the **1** track's coin before flipping two heads on the **x** coin, which is unlikely but not impossible. This motivated the design of the probabilistic modelling described in this chapter, and is further explored in Section 2.6.

2.3 Abstraction of Localised Circuits in Propositional Logic

Propositional logic (PL) is a branch of mathematical logic that studies the truth value of logical propositions and propositional formulae [69]. Its uses include determining the truth value of new, compound formulae made up of propositions that are joined by logical connectives, as well as identifying how formulae can be simplified to an equivalent form. The truth tables in Table 2.2 are simple examples of using logical connectives to create new, compound formulae by connecting the atomic propositions x and y.

PL is not Turing complete, as it has no feedback with which to create loops, memory, or recursion. Despite having a weak computational potential, PL provides a useful framework for the manipulation and study of propositional formulae. DNA walker circuits are designed to evaluate such formulae, and the action of many circuits can be described using PL. The axioms and theorems from PL can identify whether the formula that a circuit evaluates can be simplified to an equivalent propositional formula that uses fewer logic gates.

There is a PL that is defined by the 4-tuple $\mathcal{P} = (\Sigma^{\mathcal{P}}, \Omega^{\mathcal{P}}, \mathcal{A}^{\mathcal{P}}, \mathcal{T}^{\mathcal{P}})$ where:

- $\Sigma^{\mathcal{P}}$ is the finite set of all propositional variables, sometimes referred to as atomic formulas.

- $\Omega^{\mathcal{P}}$ is a set of nullary, unary, and binary logical connectives. There is a partition by arity, or the number of arguments a logical connective takes, so that,

$$\Omega^{\mathcal{P}} = \Omega_0^{\mathcal{P}} \cup \Omega_1^{\mathcal{P}} \cup \Omega_2^{\mathcal{P}},$$

  where,

$$\begin{aligned}\Omega_0^{\mathcal{P}} &= \{\text{False}, \text{True}\}, \\ \Omega_1^{\mathcal{P}} &= \{\neg\}, \\ \Omega_2^{\mathcal{P}} &= \{\vee, \wedge, \Rightarrow, \Leftrightarrow, \uparrow, \downarrow\}.\end{aligned}$$

  For example, $\Omega_2^{\mathcal{P}}$ is the set of logical connectives that take two arguments. If $x, y \in \Sigma^{\mathcal{P}}$, then the AND operator in the equation $x \wedge y$ is a member of $\Omega_2^{\mathcal{P}}$. There is a set $\overline{\mathcal{P}}$ which is the smallest set of well-formed formulae that is defined inductively as follows:

  1. Any $a \in \Sigma^{\mathcal{P}}$ is a formula in $\overline{\mathcal{P}}$.
  2. If $a_1, ..., a_n \in \overline{\mathcal{P}}$ and $f \in \Omega_n^{\mathcal{P}}$, then $f(a_1, ..., a_n) \in \overline{\mathcal{P}}$.

- Let $\mathcal{A}^{\mathcal{P}} \subset \overline{\mathcal{P}} \times \overline{\mathcal{P}}$. If $(a_1, a_2) \in \mathcal{A}^{\mathcal{P}}$, then $a_1 \equiv a_2$ where $\equiv$ is a binary equivalence realtion that indicates a_1 and a_2 have the same truth table. The set $\mathcal{A}^{\mathcal{P}}$ is the set of axioms for $\mathcal{P}$, which includes commutativity, associativity, and distribution. A complete list is given in Appendix A.

- $\mathcal{T}^{\mathcal{P}}$ is a set of inference rules that return a conclusion from a set of premises. Examples include substitution and Modus Ponens, and a list is given in Appendix A.

The set $\overline{\mathcal{P}}$ is a formal language, and a formula in $\overline{\mathcal{P}}$ together with a chosen design

for each gate is enough to completely determine the topology of a corresponding DNA walker circuit. In the context of DNA walker circuits, topology refers to the "connectedness" of the tracks. Two tracks are said to be topologically connected if they intersect such that the walkers on both tracks can interact with each other. While the PL abstraction of a DNA walker circuit is useful because it provides tools for simplification, this framework is insufficient to represent all DNA walker circuits. A more powerful representation of DNA walker circuits is abstracting the topology of the circuit as a directed graph.

2.4 Abstraction of Localised Circuits as a Directed Graph

While PL can only represent DNA walker circuits that evaluate the formulae in $\overline{\mathcal{P}}$, a directed graph can represent more general circuits by using the circuit's topology. When one track intersects with another track, the walkers on the two tracks can interact with one another via blocking. Therefore, these two walkers, as well as the tracks on which they step, are said to be topologically connected.

A directed graph $G_D = (V, E)$ is a tuple consisting of a set of vertices V and a set of edges $E = \{(\mathbf{i}, \mathbf{j}) : \mathbf{i}, \mathbf{j} \in V\}$. Each of the vertices represents a track in the circuit, and an edge $(\mathbf{i}, \mathbf{j})$ indicates that track $\mathbf{i}$ blocks track $\mathbf{j}$. The edge also implies that the walker on track $\mathbf{i}$ should step to the end of its track before the walker on track $\mathbf{j}$ for correct operation. For example, the NOR gate from Figure 2.2 can be abstracted by the directed graph,

$$V = \{\mathbf{1}, \mathbf{x}, \mathbf{y}\}, \ E = \{(\mathbf{x}, \mathbf{1}), (\mathbf{y}, \mathbf{1})\}.$$

The set V shows that the gate has three tracks, one for each walker. The set of ordered pairs E indicates that the $\mathbf{x}$ walker blocks the $\mathbf{1}$ walker and the $\mathbf{y}$ walker

blocks the **1** walker. The vertices and edges for the other gates in Figure 2.2 are shown in Table 2.3.

	V	E
NOR	$\{\mathbf{1}, \mathbf{x}, \mathbf{y}\}$	$\{(\mathbf{x}, \mathbf{1}), (\mathbf{y}, \mathbf{1})\}$
NOT	$\{\mathbf{1}, \mathbf{x}\}$	$\{(\mathbf{x}, \mathbf{1})\}$
OR	$\{\mathbf{1_1}, \mathbf{1_2}, \mathbf{x}, \mathbf{y}\}$	$\{(\mathbf{1_2}, \mathbf{1_1}), (\mathbf{x}, \mathbf{1_2}), (\mathbf{y}, \mathbf{1_2})\}$
AND	$\{\mathbf{1}, \mathbf{x}, \mathbf{y}\}$	$\{(\mathbf{1}, \mathbf{x}), (\mathbf{y}, \mathbf{1})\}$

Table 2.3: The sets of vertices V and edges E in the directed graph that correspond to each logic gate in Figure 2.2. Subscripts are used to differentiate between unique tracks.

A key use of the directed graph structure is that it can represent circuits that cannot be posed in PL. The FANOUT of a gate is the number of other gates connected to the output. However, in the context of DNA walker circuits, the output of a gate is whether the output walker steps to the end of its track. A FANOUT gate for DNA walkers initiates the stepping of n new walkers when the input signal is True. When $n = 2$, the FANOUT gate can be written as the following directed graph, using subscripts to distinguish between different **1** walkers:

$$V = \{\mathbf{x}, \mathbf{1_1}, \mathbf{1_2}, \mathbf{1_3}, \mathbf{1_4}\},$$
$$E = \{(\mathbf{x}, \mathbf{1_1}), (\mathbf{x}, \mathbf{1_2}), (\mathbf{1_1}, \mathbf{1_3}), (\mathbf{1_2}, \mathbf{1_4})\}.$$

As shown in Figure 2.3, FANOUT requires an additional property of the walker-track system: the walker must be able to block a track and keep walking, blocking additional tracks thereafter. This property, as well as the FANOUT gate itself, is not necessary to perform propositional logic. It can, however, be useful in simplifying track designs by using fewer walkers overall.

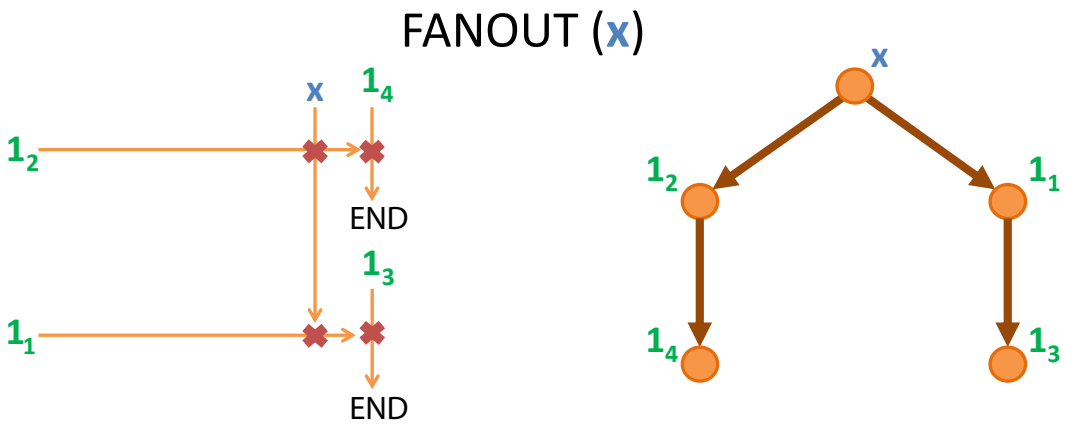


Figure 2.3: A track diagram of a possible design choice for a FANOUT gate (left) along with its directed graph abstraction (right). This design requires a blocking mechanism where a walker can block a track and keep walking, blocking other tracks thereafter.

A directed graph abstraction can describe the topology of any DNA walker circuit, including complex designs such as the SR latch from Figure 2.1. Figure 2.4 shows the directed graph for the SR latch, which uses two NOR gates and two FANOUT gates. This graph, unlike the graphs of previous circuits shown above, contains a cycle of vertices:

$$(\mathbf{1_1}, \mathbf{1_4}, \mathbf{1_5}, \mathbf{1_6}, \mathbf{1_9}, \mathbf{1_{10}}).$$

The cycle makes the order of execution difficult to control. The order (from first to last) in which the walkers should reach the end of their tracks is: $\mathbf{1_1}, \mathbf{1_4}, \mathbf{1_5}, \mathbf{1_6}, \mathbf{1_9}, \mathbf{1_{10}}$. However, the cycle means that walker $\mathbf{1_{10}}$ must block Walker $\mathbf{1_1}$, and therefore $\mathbf{1_{10}}$ should execute before $\mathbf{1_1}$. It also means that the logic gates would have to be executed several times to making a functional SR latch, which requires walkers that can step on a renewable track. Both of these reasons would make this design very difficult to implement experimentally; it illustrates that while an SR latch can be abstracted as a directed graph, DNA walker circuits are not well-suited to this type of computation. DNA walker circuits are most useful for applications where feedback is not required and the circuit must be executed only once.

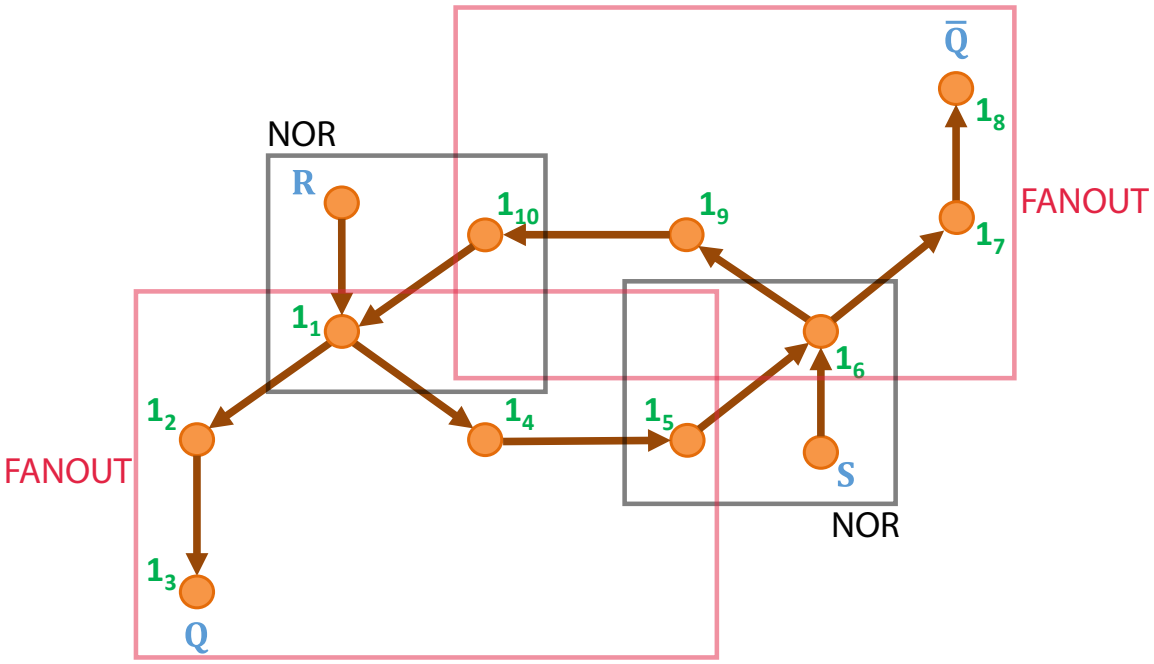


Figure 2.4: A directed graph abstraction of a DNA walker circuit that can implement the SR latch from Figure 2.1. The circuit incorporates two NOR gates (boxed in black) and two FANOUT gates (boxed in red). If the latch is in state Q, then walker $\mathbf{1_3}$ has reached the end of its track; if the latch is in state $\overline{Q}$, then walker $\mathbf{1_8}$ has reached the end of its track. This graph contains a cycle, which would make this design difficult to implement experimentally.

2.5 Localised DNA Circuits as Distributed Systems

The previous two sections have shown how to determine the topology of a DNA walker circuit. The topology can be determined when a propositional formula is given together with a design choice for each logic gate. While this method comes with the advantage of using axioms and inference rules from PL to simplify the formula, it is limited to circuits that can be posed in PL. These shortcomings can be circumvented by using a directed graph abstraction, which can represent any DNA walker circuit by describing its topology directly. While PL and directed graphs give

useful information about the circuit's topology, they provide no information about the circuit's geometry, or how the tracks can be arranged on a two dimensional lattice. To use the circuit's topology to infer its geometry, the circuit can be modelled as a distributed system. This justifies the use of techniques from formal verification to search for a track geometry that allows the circuit to operate within a specified error tolerance.

A distributed system is a network of autonomous computers that can perform a coordinated action by passing messages between different computers in the network. In contrast with centralised systems, distributed systems have multiple autonomous components and multiple points of possible error. When the anchorages on an origami tile are viewed as networked computers and the walker is viewed as the the message, a DNA walker circuit becomes a distributed system. The advantage of viewing the DNA circuit in this light is that it can be readily represented and analysed as a Petri net.

2.5.1 Petri Nets

Petri nets provide a mathematical way to describe distributed systems [29]. They consist of places, transitions, and arcs. The arcs link places to transitions or transitions to places, but never link two transitions or two places. Hence, the arcs show all of the possible transitions that can occur between places. The places in the Petri net may contain a discrete number of tokens, and the the distribution of tokens on places is called a marking. These tokens may move between places via a transition when the transition fires, which occurs when the input places all have the required number of tokens to enable the transition.

The burnt-bridges walker stepping down a track of anchorages can be modelled as shown in Figure 2.5. The initial marking shows the DNA walker, represented by a token, on the first anchorage G1. The stochastic Petri net model gives each walker

a rate at which it steps forward onto the next anchorage. The transition from the first anchorage G1 to the second anchorage G2 fires at the rate at which the walker steps from one anchorage to the next. These transitions require two tokens to fire. The tokens in the bottom row of nodes are used up as the walker steps forward, so another walker will not be able to step down the track after the current walker has finished. Physically, this bottom row of nodes represents the 5' end of the anchorages that are irreversibly cut by the nicking enzyme. Walkers are assumed to only step forward and to remain on the last anchorage once they reach the end of their track.

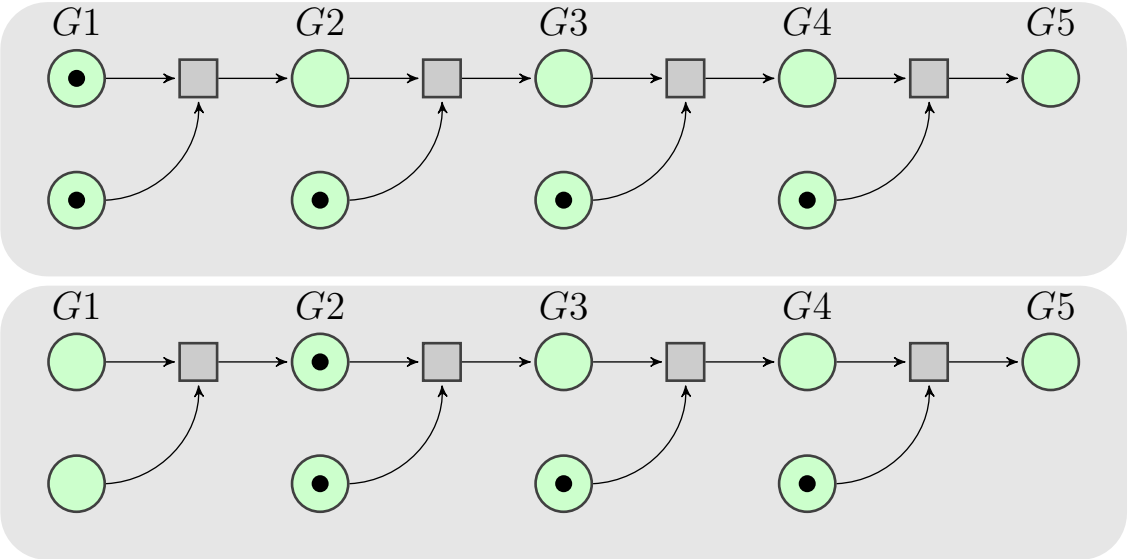


Figure 2.5: Petri net representation of the burnt-bridges walker [10] showing the initial marking (top) and the marking after the walker has taken one step (bottom). In the initial marking, the token that represents the DNA walker is on anchorage G1. The transition where the walker moves to G2 requires two tokens to fire: the token that represents the walker, and the token that represents the end of the anchorage that is irreversibly cut by the nicking enzyme. When both tokens are present, the walker token transitions to G2 and the token representing the anchorage is used up. Because the anchorage tokens are used up as the DNA walker steps, the track cannot be reused by another walker.

A reusable track can be represented in a similar fashion (Figure 2.6). In contrast to the burnt-bridges track, there are no "helper" tokens that allow the DNA walker to step. Instead, each transition requires only one token (the DNA walker token) to

fire. While this simple model treats the DNA mechanism that replenishes the track as a black box, there are other Petri net models that can incoporate the mechanism. For example, if the mechanism for the reusable track works by replacing the cut anchorages, a new transition can be created that replaces the anchorage tokens at a specified rate.

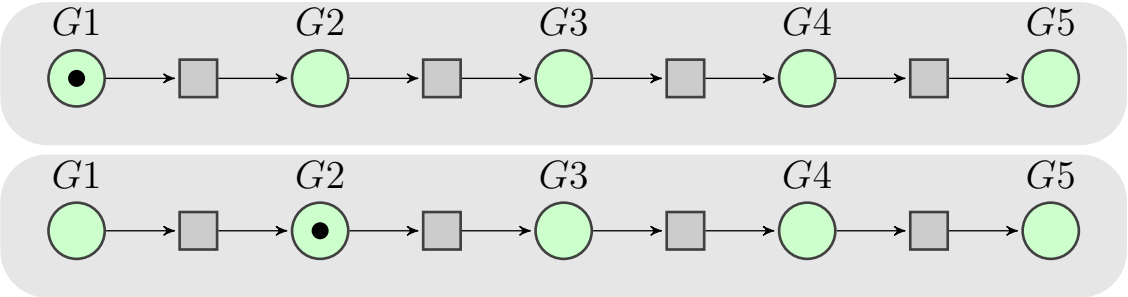


Figure 2.6: Petri net representation of a reusable track showing both the initial marking (top) and the marking after the walker has taken one step (bottom). As in Figure 2.5, the DNA walker is represented by a token. In this case, the transitions only require one token to fire and no tokens are used up in the process. After the walker has reached the end of its track, another walker can use the track again.

To model non-trivial DNA walker circuits as a stochastic Petri net, the above models for stepping must be modified to include blocking primitives. This requires adding an additional track that is topologically connected to the first via a junction that allows track blockage. For the purposes of this example, the two tracks will be called "green" and "blue" to correspond to the colours in Figure 2.7.

There are two ways to implement a blocking primitive. The first is a symmetric blocking junction (Figure 2.7, top) whereby whichever walker arrives at the junction first can block the track of the other walker. If the green walker arrives first, the green walker can block the blue track, and vice versa. In the Petri net model, the green and blue walkers will compete for a shared token that is required to step from the third to the fourth anchorage. The walker that arrives at the junction first will use up the shared token, leaving the other walker unable to step past the junction. A simple example of an application for which a symmetric junction is well-suited is

performing Bernoulli trials with probability of heads equal to p and probability of tails equal to $1 - p$. The green walker reaching the end of its track indicates heads, and the blue walker reaching the end of its track indicates tails. Adjusting the relative track lengths can give the desired value of p, and the outcome of the trial is either the green or the blue walker reaching the end of its track.

The second method for implementing a blocking primitive uses an asymmetric junction, where there is a designated blocking walker that can block the track for the other walker (Figure 2.7, bottom). In the Petri net model, the blue walker can block the green track, but the green walker is unable to block the blue track. This type of junction is best suited to the logic gate designs in Figure 2.2 where there is always a designated blocking walker at each junction. In general, asymmetric blocking junctions also tend to require simpler DNA mechanisms and are therefore less prone to error.

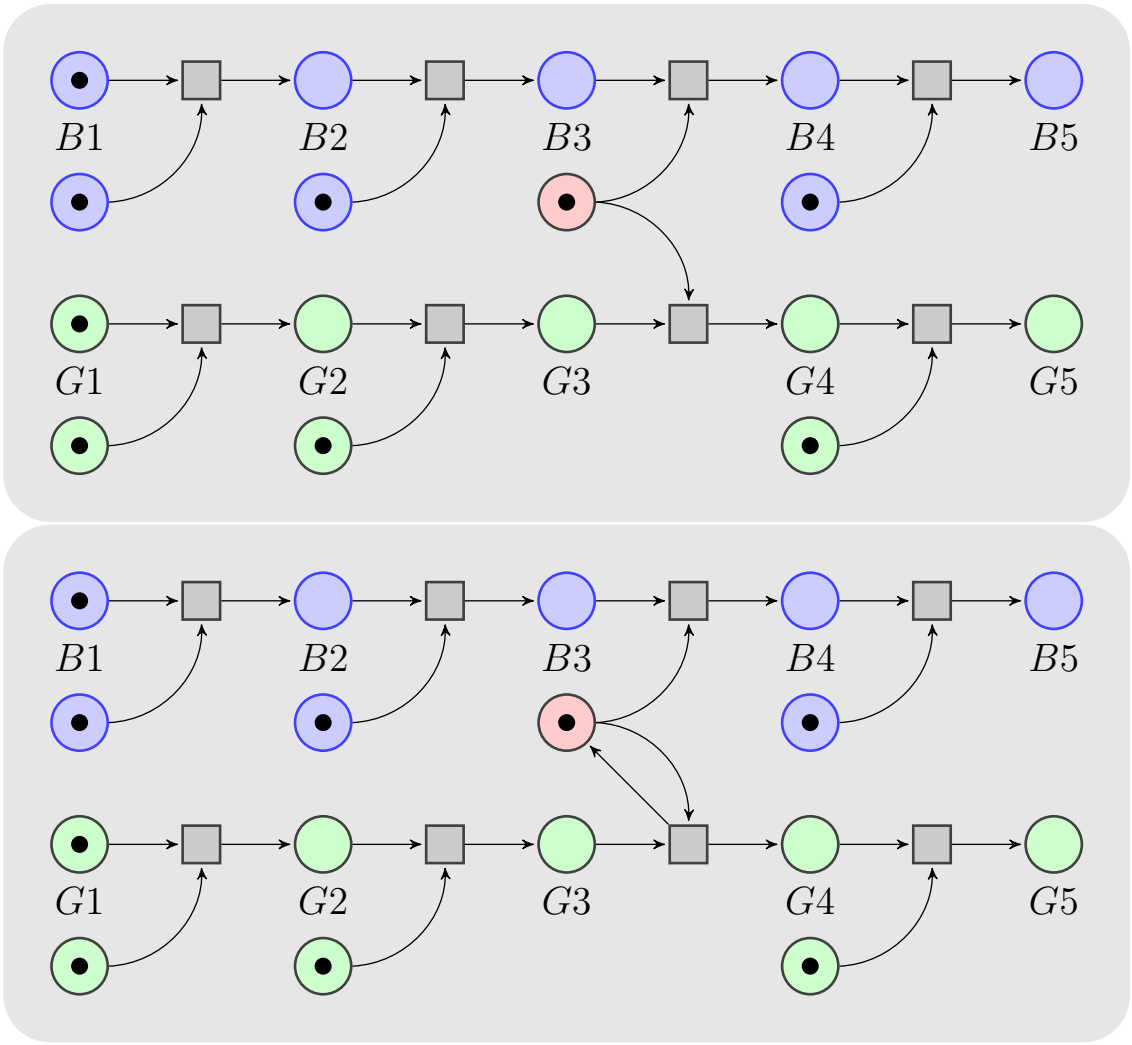


Figure 2.7: (top) A Petri net model of a blocking primitive that uses a symmetric junction. The walker that arrives at the junction first can block the track for the other: the green walker can block the blue track, or the blue walker can block the green track. The walker that arrives at the third anchorage first will take the token from the shared place (shown in red), leaving the other walker unable to step past the third anchorage in its track. (bottom) A Petri net model of an asymmetric junction. In this case, there is a designated blocking walker. The blue walker is able to block the green track, but the green walker is unable to block the blue track. If the blue walker arrives at the junction first, it uses up the token from the shared place, leaving the green walker unable to step from its third anchorage to its fourth anchorage. If the green walker arrives at the junction first, it uses the token from the shared place but then replaces it afterwards so that the blue walker can still reach the end of its track.

2.5.2 Designing a System Using Formal Verification Techniques

A stochastic Petri net maps onto a continuous time Markov chain (CTMC) in a well-understood way: there is a one-to-one correspondence between states in the CTMC and possible markings in the stochastic Petri net [61]. The firing rates for transitions in the Petri net are the rates for changing states in the CTMC. While the Petri net provides a convenient model for capturing computational behaviour, the corresponding CTMC gives a natural framework for computing the probability that a system will end up in a certain state. Some of the markings in the Petri net correspond to errors in the circuit, where a walker that should have been blocked reached the end of its track before the blocking walker arrived at the junction. The probability of ending up in those error states can be found by analysing the CTMC.

PRISM is a probabilistic model checker that can analyse CTMC models [50]. A CTMC can be encoded in the state-based PRISM language, which allows a user to easily define a (sometimes complex) Markov chain by describing the states and transitions in the high level language. PRISM compiles the input to a CTMC and analyses it to determine the probability of taking a certain path between states, or the probability of ending up in a certain state after the system is allowed to run for a long time.

An example of a model written in the PRISM language for a blue track that is three anchorages in length intersecting a green track that is nine anchorages in length is shown in Figure 2.8. A important type of error that can arise in DNA walker circuits is "missed-chance" error (MCE), which occurs when the walker that should be blocked has already stepped past the junction by the time the blocking walker arrives at the junction. Hence, the blocking walker has missed its chance to block the track. To minimise the probability of MCE, the arrival time of walkers at junctions must be carefully controlled. When two walkers have similar stepping rates,

the probability of MCE can be lowered by lengthening the track of the walker that should be blocked. PRISM can evalute the probability of MCE for the example in Figure 2.8 by checking the following property:

```
P = ?  [F (b_loc = bt_max) & (g_loc >= gt_bt_jun)].
```

This is the probability that the blue walker has reached the final anchorage in its track (the junction anchorage) after the green walker has already stepped past the junction. PRISM evaluates the probability of MCE as 1.95% for tracks with these lengths.

Model checking can be used to determine the circuit design with the shortest tracks that still has a MCE below a given tolerance. Finding this balance is critical: a compact system is easier to design and fit onto an origami tile, but tracks that are too short will cause MCEs. By representing the DNA system as a Markov chain and analysing each possible combination of track lengths, one can search for a design that is optimal in the sense that it is compact and minimises the probability of MCEs. This assigns a natural number to each track corresponding to the smallest number of anchorages needed to stay within the specified tolerance for the MCE. The track lengths, together with the assumption that a track is always blocked on its penultimate anchorage, or as close as possible if the track is blocked twice, are sufficient to determine the track geometry of the system.

```
ctmc
const double step_rate = 1; // Assumption: all motors have the same rate of stepping

//-------------------------------------------------------
// GREEN TRACK (GT)

const int gt_max = 9; // Maximum green track length

const int gt_bt_jun = 8; // Junction location is the 7th anchorage on the green track

module GT

      // Possible locations (states) of the green walker.
      // The green walker is initialised on the first anchorage.
      g_loc : [1..gt_max] init 1;

      // Is the track blocked?
      [unblocked] g_loc < gt_max -> step_rate : (g_loc'=min(g_loc+1,gt_max));
      [blocked] g_loc < gt_bt_jun -> step_rate : (g_loc'=min(g_loc+1,gt_bt_jun));
      [blocked] g_loc > gt_bt_jun -> step_rate : (g_loc'=min(g_loc+1,gt_max));


endmodule

/-------------------------------------------------------
// BLUE TRACK (BT)

const int bt_max = 3; // Maximum blue track length

const int bt_gt_jun = bt_max; // Junction location is at the end of the blue track

module BT

      // Possible location (states) of the blue walker.
      // The blue walker is initialised on the first anchorage.
      b_loc : [1..bt_max] init 1;

      // Step forward
      [] true -> step_rate : (b_loc'=min(b_loc+1,bt_max));

      // Block the green track?
      [blocked] b_loc = bt_max -> true;
      [unblocked] b_loc < bt_max -> true;

endmodule
```

Figure 2.8: Example of a script in the PRISM language that models a blue walker blocking a green track. Comments in the code are preceded by a double front slash (//). Both walkers are assumed to have the same rate of stepping. The module GT controls the green walker stepping down its track of nine anchorages. The junction is located on the eigth anchorage of the green track. If the green track is blocked by the blue walker, then the green walker cannot step beyond the junction. The blue track consists of three anchorages, and the third anchorage intersects the green track at the junction. If the blue walker reaches the junction, it switches the GT module into a blocked state. Once in the blocked state, the green walker can only reach the end of its track if it has passed the junction before the blue walker arrives. It is assumed that if the two walkers bind to the junction at the same time, the green walker will still be able to reach the end of its track.

2.6 Compiler Design

To find a design that minimises errors caused by incorrect order of execution (see Section 2.2) a compiler was written that automatically designs a DNA walker circuit by abstracting a DNA walker circuit as a directed graph, using the directed graph to create a Petri net model and CTMC, and searching for a track design that operates with minimal error. The only inputs required are a propositional formula (or directed graph if the circuit cannot be posed in PL) and a tolerance for MCE. These inputs are compiled into a track design for a DNA walker circuit that has the shortest possible track lengths while still operating within the specified MCE tolerance.

Given two inputs, a propositional formula and a MCE tolerance, the compiler designs a track by performing the following five steps:

1. using the axioms and theorems from PL to search for an equivalent propositional formula that uses fewer logical connectives,

2. parsing the propositional formula into a tree,

3. using the parsing tree and a choice of gate design to create a directed graph that describes the topology of the circuit,

4. using the directed graph to compile PRISM code that models the circuit,

5. searching through the possible track lengths for the track design that has the shortest tracks while still operating within the MCE tolerance.

The output is a diagram for a compact track system that meets the MCE tolerance. For simplicity, attention is restricted to circuits that can be posed in PL. It is assumed that the blocking mechanism only allows each walker to block one track; the walkers cannot block a track and continue stepping to block other tracks thereafter. The remainder of this section provides the details behind each of the above steps. Equation

2.1 is used as an example to illustrate each step in the compiler's algorithm, starting from the propositional formula and finishing with the track design.

DNA walker circuits that have fewer gates also tend to have fewer tracks. The first step in the compiler is using the theorems and axioms from PL to search for an equivalent formula that uses fewer logical connectives. If one is available, using it can reduce MCE by reducing the number of blocking interactions needed in the circuit. This lowers the probability of MCE while also helping the circuit fit onto an origami tile. SymPy [45] searches for a simplest form by searching for propositional formulae with equivalent truth tables and returning the Sum of Products (SOP) form or Product of Sums (POS) form, whichever has the fewest logical connectives. It uses the distributive rule from PL to make the following simplification:

$$(\mathsf{OR}(\mathsf{AND}(x, y), \mathsf{AND}(x, z) \equiv \mathsf{AND}(x, \mathsf{OR}(y, z)). \tag{2.1}$$

The parsing tree for the unsimplified propositional formula (left) and the simplified propositional formula (right) are shown below. The leaves of the tree are the propositional variables and the remaining vertices are logical connectives.

After making a choice for the design of each gate, the parsing tree can be abstracted into a directed graph to show the topology of the circuit. For the working example of Equation 2.1, the gates are designed as shown in Figure 2.2. The directed graph abstraction for each gate is threaded onto the parsing tree by replacing each vertex in the parsing tree with the directed graph structure of the corresponding gate. The result is a directed graph that shows the topology of the whole circuit. For the rightmost parsing tree above, the directed graph abstraction is shown in Figure 2.9.

It was constructed by threading together the directed graphs for one AND and one OR gate.

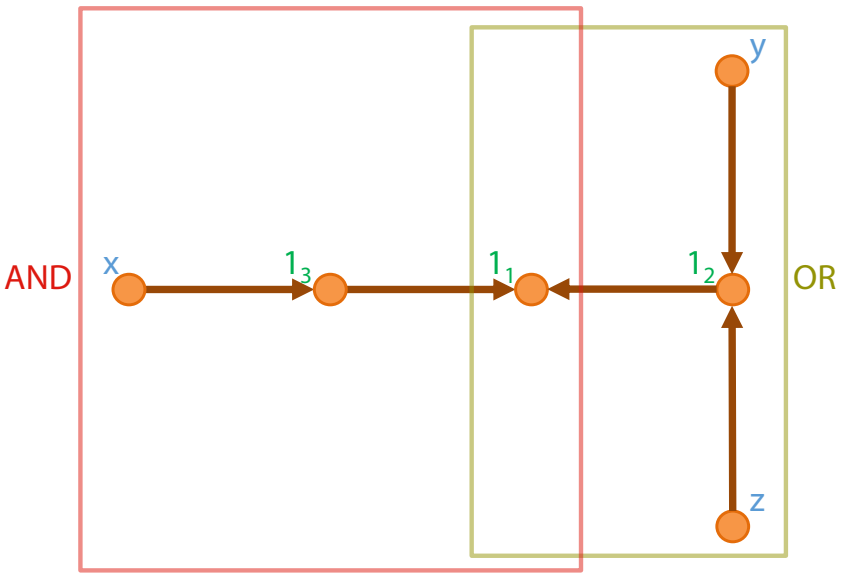


Figure 2.9: Directed graph abstraction of the propositional formula on the right hand side of Equation 2.1. This graph uses the gate designs shown in Figure 2.2. The directed graph is made by threading the directed graph structure for each gate onto the parsing tree for the propositional formula. In this example, the output of an OR gate (boxed in gold) is the input for an AND gate (boxed in red).

As shown by the example code in Figure 2.8, the modular nature of the state-based PRISM language makes it straightforward to compile PRISM code that models the DNA walker circuit. Every track in the directed graph corresponds to one module in the PRISM script. However, there are challenges associated with using PRISM to search through all of the possible lengths that each track can take. The number of cases to check increases rapidly with the number of tracks in the system: if tracks are allowed to vary between 1 and 20 anchorages in length, then a brute force search for a system with n tracks has complexity is $O(20^n)$. This means that in the worst case, there would be 20^n cases to check; a better method is needed.

While it is possible to design better and more efficient searches than the brute force

method[1], Algorithm 1 presents an alternative approach that works by decomposing any circuit into a series of individual blocking interactions. If a track in the circuit is blocked, the length of the blocking track (or tracks) is held constant while the length of the blocked track is increased until the probability of MCE falls within the tolerance.

The algorithm can be illustrated by using the directed graph in Figure 2.9. All tracks are assigned a trivial track length of 2 anchorages. The **x** track blocks the $\mathbf{1_3}$ track, so PRISM code is compiled for this blocking interaction. Starting with the lengths of **x** and $\mathbf{1_3}$ both equal to 2 anchorages, PRISM calculates the probability of MCE. If the probability lies outside of the tolerance, the length of $\mathbf{1_3}$ is increased by one anchorage to lower the probability of MCE and checked by PRISM again. The process is repeated until $\mathbf{1_3}$ is sufficiently long that the probability of MCE lies within the tolerance. The length of $\mathbf{1_2}$ is found in a similar fashion, using PRISM code that models **y** and **z** blocking $\mathbf{1_2}$. Once the lengths for $\mathbf{1_2}$ and $\mathbf{1_3}$ are found, PRISM code is compiled that models $\mathbf{1_2}$ and $\mathbf{1_3}$ both blocking $\mathbf{1_1}$. Holding the lengths for $\mathbf{1_2}$ and $\mathbf{1_3}$ constant, the length of $\mathbf{1_1}$ is incrementally increased until the MCE falls within the tolerance.

[1]The number of cases to check can be reduced by requiring that each track is at least twice as long as any track that blocks it. It is also possible to use Monte Carlo methods to more efficiently navigate the search space.

```
Data: directed graph G and MCE error tolerance P(MCE)
Result: a dictionary anchorageLengths that assigns a length to each track
anchorageLengths{all nodes} = 2;
leafNodes := nodes in G with indegree 0;
while leafNodes not empty do
    for blockerTrack in leafNodes do
        blockedTrack = track blocked by blockerTrack;
        compile PRISM model;
        check error with PRISM;
        while error > P(MCE) do
            lengthBlockedTrack ++;
            check error with PRISM;
        end
        if lengthBlockedTrack > anchorageLengths{blockedTrack} then
            anchorageLengths{blockedTrack} = lengthBlockedTrack;
        end
    end
    Delete leafNodes from G;
    leafNodes := nodes in G with indegree 0;
end
```

Algorithm 1: Algorithm that decomposes a circuit into individual blocking interactions to determine the shortest length of each track that meets the MCE tolerance.

Running Algorithm 1 on the directed graph in Figure 2.9 with a 15% MCE tolerance results in the following track lengths:

- $length(\mathbf{1_1}) = 14$ anchorages,
- $length(\mathbf{1_2}) = length(\mathbf{1_3}) = 7$ anchorages,

- $length(\mathbf{x}) = length(\mathbf{y}) = length(\mathbf{z}) = 2$ anchorages.

Using the lengths of each individual track shown above and the blocking topology from the directed graph, the compiler generates a track design (Figure 2.10) that evaluates the propositional formula in Equation 2.1.

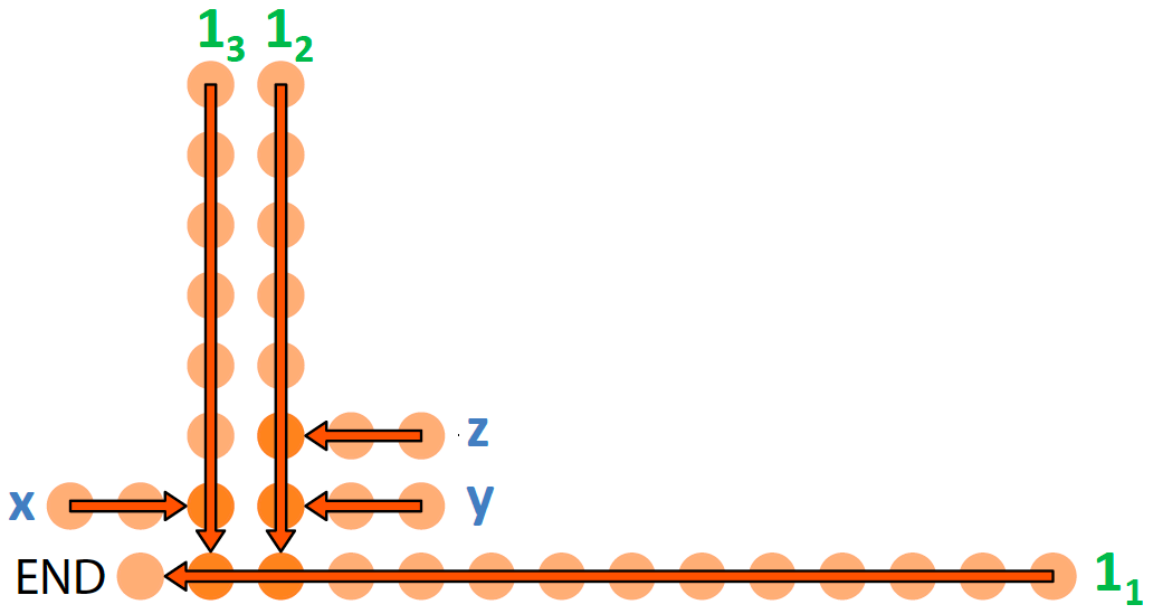


Figure 2.10: Compiler-generated track design of the DNA walker circuit that can evaluate Equation 2.1 within a 15% MCE tolerance. Each anchorage is represented by a light orange circle. The individual tracks, along with their directionality, are indicated by orange arrows. Blocking junctions occur where one track intersects another, and the propositional variable associated with each track is labelled at the track's starting anchorage. The output, or truth value, of the propositional formula is given by whether or not the $\mathbf{1_1}$ walker is able to reach the end of its track.

The length of the tracks in the circuit depends on the MCE tolerance that is chosen. Lower tolerances result in less compact circuits with longer tracks. For example, if the MCE tolerance in the above example were adjusted to 10% instead of 15%, the new track lengths needed to meet this tolerance would be:

- $length(\mathbf{1_1}) = 18$ anchorages,
- $length(\mathbf{1_2}) = length(\mathbf{1_3}) = 8$ anchorages,

- $length(\mathbf{x}) = length(\mathbf{y}) = length(\mathbf{z}) = 2$ anchorages.

The track $\mathbf{1_1}$ must now be long enough that it would be difficult to implement this circuit on an origami tile. If a lower MCE tolerance is required, it would be necessary to slow down the $\mathbf{1_1}$ walker so that the length of the track stays reasonable. This can be achieved by using a walker with a different DNA sequence that has a slower stepping rate or by adding a mismatch between the DNA walker and its track anchorages [58].

2.7 Blocking as a Logic System

The circuit shown in Figure 2.10 evaluates the propositional formula $x \wedge (y \vee z)$, but it prompts a number of questions. The gate designs that were threaded together to evaluate the propositional formula are not unique; Figure 2.2 shows two designs for an AND gate. What other gate designs exist, and is there a simplest design for each gate? If the gate designs are not unique, it follows that any circuit design that evaluates a given propositional formula is also not unique. Of all the different circuit designs that can evaluate a propositional formula, is one of the designs simpler than the rest? If so, how can it be found?

A logical system based on blocking provides a way to answer these questions. This formal system is denoted by $\mathcal{L}$, and it establishes rules for proving that two circuits are equivalent. These rules are shown to be both sound and complete: all rules can be derived from a set of axioms, and all derivations will produce rules that are always true. The full proofs for soundness and completeness are lengthy, and are therefore left to Appendix B. This section introduces only what is needed to see the usefulness of the result.

When each walker can only block one track, a circuit C is a rooted tree where each vertex is a track and the root of the tree is the output track. A track can be blocked

by its children (if they exist) and can block its parent (if it exists). This definition of a circuit means that any circuit has propositional variables $b_1, b_2, ..., b_n$ and the truth value of the circuit depends on the truth value of each of these propositional variables. Excluded from the list of propositional variables is the track that always holds the value True, which is denoted by 1. In blocking logic, 1 tracks are logical connectives, as shown below. For example, the NOT gate in Figure 2.2 has only one propositional variable x. Furthermore, note that an individual track is a circuit, albeit a trivial one, as it is a tree with one vertex.

The system $\mathcal{L} = (\Sigma^{\mathcal{L}}, \Omega^{\mathcal{L}}, \mathcal{A}^{\mathcal{L}}, \mathcal{T}^{\mathcal{L}})$ is a formal system of blocking logic that is defined as follows. Let $\Sigma^{\mathcal{L}}$ be a finite set of propositional variables. There is a surjective mapping between the propositional variables associated with a circuit and that circuit's tracks. Through an abuse of notation, tracks will be denoted by the propositional variables they are associated with, and this convention is adopted for the remainder of this section. There are two logical connectives in $\mathcal{L}$:

- 1 is a constant representing a circuit that is always True,
- $\otimes$ is a binary operator such that $X \otimes Y$ indicates that the circuit X blocks the circuit Y.

Therefore, the set of logical connectives is $\Omega^{\mathcal{L}} = \{1, \otimes\}$. Circuit $C = X \otimes Y$ is composed of two circuits X and Y, so C is only True if Y is True and Y not blocked by X . Therefore, the operator $\otimes$ has the truth table shown in Table 2.4, which is the same truth table as the negation of the Implies operator from PL.

The set $\{\neg, \Rightarrow\}$ is functionally complete. It follows that $\Omega^{\mathcal{L}}$ is also functionally complete, as $X \Rightarrow Y$ can be expressed by $Y \otimes X \otimes 1$ and $\neg X$ can be expressed as $X \otimes 1$. By convention, the action of the $\otimes$ operator is left associative so that,

$$X \otimes Y \otimes Z = (X \otimes Y) \otimes Z.$$

The syntax for a circuit can be easily defined using the notation of Backus-Naur Form, where nonterminals are enclosed in angled brackets:

$$\langle C \rangle ::= 1 \mid b_i \mid \langle C \otimes C \rangle. \tag{2.2}$$

The $\otimes$ symbol is generally omitted to simplify notation so that XY means $X \otimes Y$.

X	Y	$X \otimes Y$
False	False	False
True	False	False
False	True	True
True	True	False

Table 2.4: Truth table for the $\otimes$ operator. The expression $X \otimes Y$ signifies that X blocks Y, and evaluates to True if Y reaches the end of the track.

$\overline{\mathcal{L}}$ is the language of well-formed formulae in $\mathcal{L}$, and can be defined inductively as the smallest set such that:

- $C \in \Sigma^{\mathcal{L}} \cup \{1\}$ is a well-formed formula,
- if C_1 and C_2 are well-formed formulae, then $C_1 \otimes C_2$ is a well-formed formula.

There is a set $\mathcal{A}^{\mathcal{L}}$ which consists of the following three axioms for $\mathcal{L}$:

A1. $((ZX1)(YX1)1)((ZY1)X1)1$,

A2. $(XY1)X1$,

A3. $(Y(X(Y1)1)1)(X1(Y1)1)1$.

These axioms are tautologies, which means that **A1** - **A3** will always evaluate to True any truth value of X, Y, and Z. The set $\mathcal{T}^{\mathcal{L}}$ is the set of inference rules for the system, which only contains one inference rule:

$$\text{MP} \coloneqq \frac{X \mid YX1}{Y}.$$

This inference rule, known as Modus Ponens, deduces Y if both X and $YX1$ hold.

The axioms, together with Modus Ponens, can be used to derive a set of rules for simplifying circuits. Here, the equivalence of two circuits X and Y is taken to mean that both circuits $XY1$ and $YX1$ are tautologies and can be derived from the axioms. If two circuits are equivalent, then they have the same truth table[2]. The following circuit equivalencies can be proven in $\mathcal{L}$:

R1. $XX \equiv 1Y$,

R2. $Y(XZ) \equiv X(YZ)$,

R3. $X11 \equiv X$,

R4. $ZX1Y \equiv ZY1X$,

R5. $X(1Y) \equiv 1Y$,

R6. $1YX \equiv X$,

R7. $Y(YX) \equiv YX$,

R8. $((X(Y1))1)Z \equiv X(YZ)$,

R9. $XYY \equiv X1Y$,

R10. $Y(X1)(Y1) \equiv X1(Y1)$.

This is not a comprehensive list of all rules that can be derived in $\mathcal{L}$, but it gives a selection of useful properties. Rule **R3**, for example, gives the blocking logic analogue of cancelling double-negatives. Rule **R7** shows that there is no logical benefit to blocking a track with multiple copies of the same circuit. While these two rules are

[2]Equivalence, as it is presented in this section, means that a rule has been derived from the axioms that shows two circuits have the same truth table. It is presented this way so that a selection of simplification rules can be presented with minimal preliminaries. Appendix B takes a more formal approach, where equivalence comes from semantic and syntactic entailment. The definitions of syntactic and semantic entailment can confuse the unfamiliar reader, so a more intuitive notion of equivalence is used here.

quite intuitive, there are more convoluted rules such as **R8** - **R10** that would be difficult to find by hand. The soundness and completeness of $\mathcal{L}$ guarantees that all rules can be derived, no matter how complex they are.

The main use of blocking logic is the simplification of circuits: it can be used to find an equivalent circuit that uses fewer tracks (if such a simplified circuit exists). Completeness of $\mathcal{L}$ means that if there are two equivalent circuits, there is a proof showing this equivalence. The following section provides a simple example of how $\mathcal{L}$ can be used this way.

2.8 Simplifying a Circuit

The example in Figure 2.10 shows a DNA walker circuit that can evaluate the propositional formula $x \wedge (y \vee z)$. It is natural to ask whether this is the simplest track design that evaluates the propositional formula, and the system $\mathcal{L}$ provides the tools to answer this question. Representing the circuit from Figure 2.10 using the syntax from $\mathcal{L}$ gives,

$$x1(z(y1)1).$$

Applying rules **R2** and **R3** from the previous section gives:

$$\begin{aligned} x1(z(y1)1) &\overset{R2}{\equiv} z(y1)(x11), \\ &\overset{R3}{\equiv} z(y1)x. \end{aligned}$$

This circuit is equivalent to the original but has two fewer tracks. The original circuit and its simplified version are both shown in Figure 2.11.

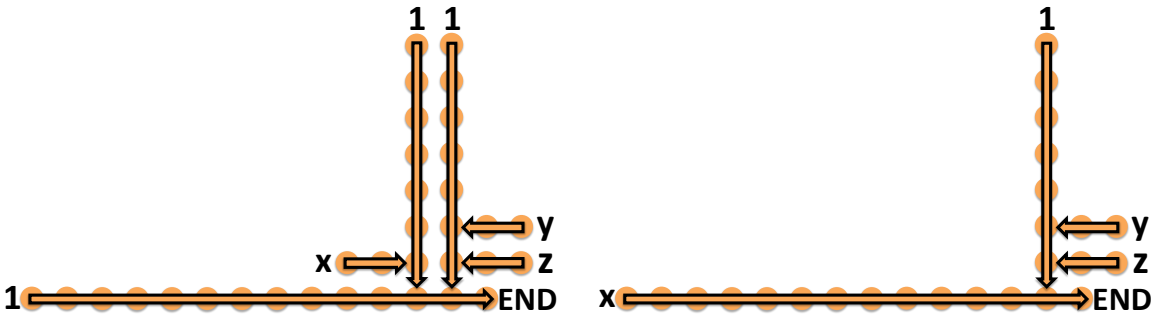


Figure 2.11: Example of how a DNA walker circuit can be simplified by using the formal system $\mathcal{L}$ defined in Section 2.7. In the syntax from $\mathcal{L}$, the blocking circuit $x1(z(y1)1)$ (shown on the left) can be simplified into the circuit $z(y1)x$ (shown on the right) which uses two fewer tracks.

2.9 Discussion

This chapter demonstrates how primitive blocking operations can be used to design DNA walker circuits that can evaluate formulae written in PL. As these circuits scale up, designing them by hand becomes impractical; order of execution must be carefully controlled, so the design problem is better suited to computer-aided design. A propositional formula written in the syntax and grammar of PL can be compiled into a track design by viewing DNA walker circuits as a distributed systems. This provides the justification for modelling the circuits as a stochastic Petri net, which can be mapped onto a CTMC. PRISM is a software tool for formal verification that provides a state-based scripting language for encoding CTMCs and evaluating the probability of certain paths through the state space after the system is allowed to run for a long time. This approach can be used to design a DNA walker circuit that evaluates a propositional formula within a specified error tolerance, solving the design problem and helping to control the order of execution. Blocking circuits can be written as a formal system, which provides rules for finding equivalent circuits. These rules can be used to search for a simpler design of the circuit, which can help

reduce error.

In this chapter, all walkers are presumed to have similar rates of stepping so that the timing of the walkers' arrival at the junction can be controlled by adjusting the lengths of each track. If a very low tolerance of MCE is imposed, the length required for some tracks may be large. To circumvent this difficulty, a walker's rate of stepping can be reduced by introducing a mismatch between the walker and its anchorages [58]. The amount that the rate is reduced will depend on where the mismatch is introduced: mismatches near the 5' end of the anchorage will cause a large reduction in stepping rate, while mismatches closer to the tile will only reduce the rate by a small amount. There is a disadvantage, however, in that using this strategy for large circuits can make their operation prohibitively slow.

Limited space on the origami tile, together with the need to lengthen tracks and avoid MCE, makes it important to simplify the circuit design where possible. As shown in Section 2.8, a circuit is simpler than one of its equivalent circuits if it uses fewer tracks. Minimising the number of tracks also minimises the number of junctions, lowering the probability for missed-chance error in the circuit. Many of the equivalence rules from Section 2.7 are reduction rules that find an equivalent circuit while reducing the number of tracks.

There are many different possible designs for each logic gate, which appears to complicate the problem of finding a simplest circuit. Figure 2.2 shows that the AND gate can be constructed out of three NOR gates, but there is also a simpler design that only uses three tracks. This introduces a question regarding gate design: which gate design should be chosen to create the simplest circuit that operates with minimal error? Completeness means that all equivalence rules can be derived in $\mathcal{L}$. Therefore, an important consequence of the completeness of $\mathcal{L}$ (see Appendix B) is that the choice of gate design does not matter. Regardless of which gate design is chosen to implement each Boolean function, simplification in $\mathcal{L}$ will still reduce the circuit

design down to a form that uses a minimum number of tracks. This makes the problem of gate design much easier: any gate design that implements the Boolean function is suitable, as the circuit can later be simplified.

This chapter, and indeed this whole thesis, focuses on implementing blocking circuits by using DNA walkers. It is important to note that blocking logic is independent of any method of implementation. Blocking circuits can be implemented in a number of ways, using a wide variety of devices and architectures that can operate on different scales of both time and size. One can imagine a mechanical machine whereby each time a crank is turned, every walker advances one step on its track. There may be more abstract implementations: suppose there is a biomolecule or gene that is always switched on (playing the role of the **1** walker) that can inhibit the action of another biological agent which may or may not be present (playing the role of the walker that corresponds to a propositional variable). In this more general context, the system $\mathcal{L}$ represents a regulatory network where inhibition, if it exists, is total. However, the network structure is restricted to a tree. Extending this to general networks that have regulation and feedback would be a possible topic for future work.

Chapter 3

Experimental Methods

This chapter outlines the techniques and materials used in the experimental implementation of DNA walker circuits, where the tiles from Section 1.4.3 are used to template the circuit's assembly. The main experimental contribution of this thesis is the design and implementation of circuits that compute NOT and NOR. Many of the techniques outlined below are similar to those used in [100–102]. However, additional techniques were required for the blocking action of the DNA walkers. The DNA sequences used in all experiments described in this thesis are listed in Appendix C. A step-by-step protocol for a typical NOR gate experiment is given in Appendix D.

3.1 Materials and Tile Self-Assembly

The minimal twist DNA origami tile [100] was assembled using the 231 staple oligonucleotides listed in Appendix C. Staple oligonucleotides were supplied by Integrated DNA Technologies (IDT) at 150μM suspended in H_2O. The DNA origami scaffold used was the M13mp18 *lac* vector, a circular single-stranded DNA molecule of 7,249 nucleotides. The M13mp18 vector used was supplied by Affymetrix, and arrived suspended in 1× TE (10mM Tris-HCL pH 7.5, 1mM EDTA).

All oligonucleotides for the DNA walker mechanism, such as track anchorages and

fluorophore labelled strands, were supplied by IDT and resuspended in H_2O upon arrival. The type of purification used for each type of mechanism strand is shown in Table 3.1. All purification was carried out by IDT prior to shipping. A full list of oligonucleotide sequences for the DNA mechanism used in the NOT and NOR gates are listed in Appendix C.

Purification for Blocking Mechanism Oligonucleotides	
Oligo Type	**Purification**
Walkers	High Performance Liquid Chromatography (HPLC)
Fluorophore Strands	HPLC
Track Anchorages	Polyacrylamide Gel Electrophoresis (PAGE)
Junction Anchorages	PAGE
Release Strands	Standard Desalting
Origami Staples	Standard Desalting

Table 3.1: Type of purification used for each category of oligonucleotide used in the DNA walker circuit. All purification was carried out by IDT prior to shipping.

The nicking enzyme used for the DNA walker circuit was Nt.BbvCI, supplied by New England Biolabs (NEB) in 1× CutSmart® Buffer (50mM Potassium Acetate, 20mM Tris-acetate, 10mM Magnesium Acetate, 100 ug/ml BSA, pH 7.9). The restriction site is shown below with the cutting location indicated by an asterisk.

5′ – C C*T C A G C – 3′

3′ – G G A G T C G – 5′

Nt.BbvCI has optimal cutting activity at 37°C. Therefore, all fluorescent spectrophotometry experiments were done at this temperature. To ensure temperature control of the samples, fluorimetry measurements were taken using a Cary Eclipse Fluorescence Spectrophotometer equipped with a Peltier device and water bath for temperature regulation.

The origami tiles that templated the tracks for the DNA walker circuits self-assembled in a "one pot" reaction. A thermocycler was used to cool the sample

from 95°C to 20°C at a rate of 1°C per minute. The concentration of each type of oligonucleotide in the self-assembly solution is shown in Table 3.2. Staples with 5' anchorage extensions ("track" staples) are all in excess to the M13mp18 scaffold to ensure that each tile has a track. Staples with 5' junction anchorage extensions are in greater excess to track staples, as the blocking mechanism will not operate correctly if a tile assembles without the junction anchorages.

Concentrations for Self-Assembly		
Type	**Excess**	**Conc.** (μM)
M13mp18 Scaffold	1	0.06
Origami Staples	5	0.29
Track Staples	3	0.17
Junction Staples	5	0.29
Fluorophore Staples	3	0.17
Fluorophore Strands	5	0.29
Blocking Walker Tether(s)	3	0.17
Covering Strands (see Section 3.5)	200	11

Table 3.2: Concentrations of all DNA molecules in the tile self-assembly reaction for a DNA walker circuit. All excesses are relative to the M13mp18 scaffold.

3.2 Measuring a Walker's Movement with Fluorescence Spectrophotometry

The NOT and NOR gates are designed to look similar to the schematic in Figure 2.2. Fluorophores are tethered at the first and last anchorage of the **1** track to report the walker's movement. A TEX615 fluorophore is tethered near the first anchorage in the **1** track, and a Cy5 fluorophore is tethered near the last anchorage in the **1** track. The fluorophores were added as 3' modifications onto strands 20 nucleotides in length. The nucleotide sequence was used as an "address" whereby the modified oligonucleotide would hybridise to its reverse complement. The reverse complement was a 3' staple extension adjacent to a 5' anchorage staple. The fluorophore was tethered to the tile

near the desired anchorage when the fluorophore-modified oligonucleotide hybridised to its reverse complement. This method makes it simple and inexpensive to change where a fluorophore is tethered on the tile.

Photobleaching is an effectively irreversible change in a fluorophore molecule due to the generation of a reactive oxygen species when the fluorophore is in an excited state [3]. It leads to the destruction of fluorophore molecules when they are exposed to high intensity illumination for an extended period of time. For the experimental timescales and illumination levels used in this work, no significant amount of photobleaching was observed for TEX615 and Cy5. For example, the Cy5 signal is stable over 700 minutes in Figure 5.4.

A dark quencher is a molecule that can dissipate the excitation energy from a fluorophore as heat, reducing a fluorophore's emission when it is in close proximity to the quencher. The movement of the output walker was measured by tethering a TEX615 fluorophore near the start of the track and a Cy5 fluorophore near the end of the track. An Iowa Black® RQ broad spectrum dark quencher (supplied by Integrated DNA Technologies) was attached to the 5' end of the walker. When a laser is shone through the sample at the fluorophores' excitation wavelengths, the emission signal decreases when the walker is in close proximity to the fluorophore. Taking measurements of the fluorescence signal every minute indicates the movement of the output walker over time.

When the walker is bound to the start of the track, the signal of the nearby TEX615 fluorophore decreases. After the enzyme is added and the walker steps away, the TEX615 signal increases once the quencher is no longer in close proximity. The decrease in TEX615 signal before the enzyme is added provides evidence that the walker has loaded correctly onto the tile.

As the walker reaches the end of its track, the quencher on the walker decreases the signal of the Cy5 fluorophore tethered there. When the walker binds to the last

anchorage in the track, the mismatch in the enzyme's restriction site on the last anchorage prevents the walker from stepping away. As the walker remains stationary at the end of its track, the Cy5 fluorophore remains quenched. For logic gates, the Cy5 signal is the output of the gate as it indicates whether or not the output walker has reached the end of its track.

3.3 Loading the Walkers

DNA walkers are hybridised to the first anchorage in the track and are left out of the tile self-assembly reaction. This helps ensure that the walkers do not bind to track anchorages when the tile is only partially formed, avoiding errors. After the tile has assembled, the walker-anchorage duplex is loaded onto the tile by adding the duplex to the solution and incubating for two hours at 37°C. The staple domain on the track anchorage anneals into the tile, loading the walker onto the tile at the starting position at the beginning of the track. In the NOT and NOR gate, the **1** walker is the output walker: the output of the gate is whether or not the **1** walker reaches the end of the track. Output walkers are loaded onto the tile using this method.

This method, while simple, does not always lead to high rates of walker loading. Typically, 80-90% of assembled tiles are successfully loaded when this method is used (see Table 4.2 and Table 4.3 in the next chapter). For the reporter walker, this does not pose any difficulty as it is straightforward to normalise the fluorescence signals (see Section 3.6) for imperfect loading. Blocking walkers, however, need to have near 100% loading or some tiles may not have a walker present to block the track.

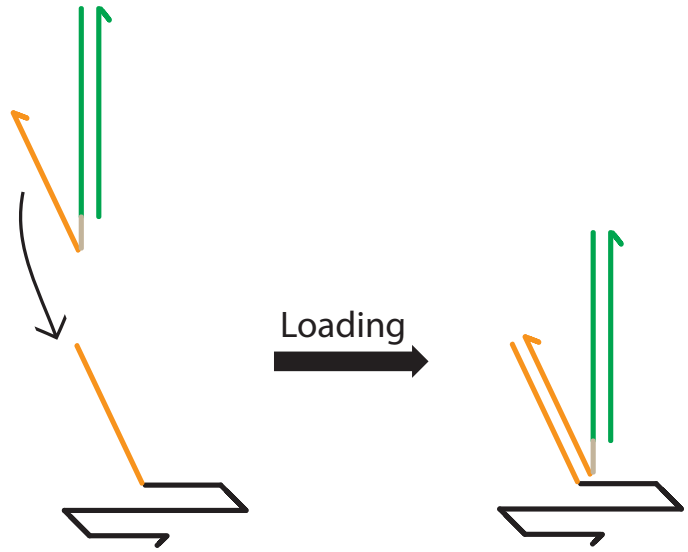


Figure 3.1: Loading the blocking walkers into position on the tile. A staple strand in the origami tile has a 5' extension domain that acts as a binding site for an address strand (orange). The blocking walker (green) is hybridised to a strand that contains an anchorage domain (green) and an address domain (orange). The blocking walker is loaded into position when the address hybridses to its reverse complement.

The blocking walkers can be loaded using a different method (Figure 3.1) that is designed to load walkers on nearly 100% of the assembled tiles, as shown by the data in Figure 3.2. The walker is annealed onto a strand that has two domains: a domain at its 5' end that is the reverse complement of the walker, and a domain at its 3' end that acts as an address. A separate tether strand, which has a staple domain and a 5' extension complementary to the address domain, is included in the origami self-assembly reaction. After self-assembly, the walker-anchorage duplex is added to the solution and incubated at 37°C for two hours. As the address hybridises to its complementary tether, the walker is loaded into place on the tile.

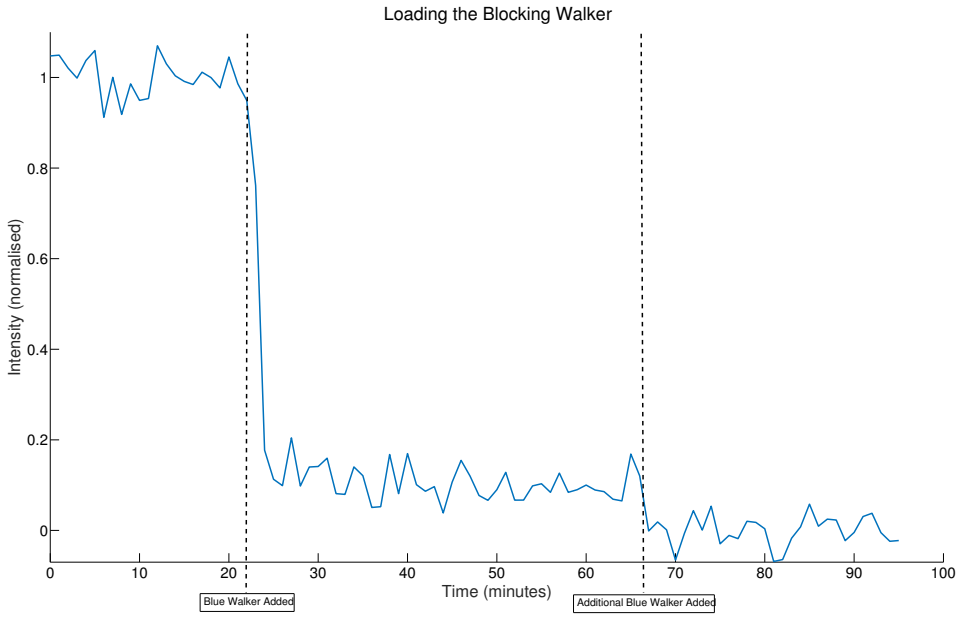


Figure 3.2: Measuring the loading of a blocking walker. The walker has a 5' Iowa Black dark quencher modification, and the walker is hybridised to the addressable first track anchorage. The address domain hybridises to its reverse complement on the tile, loading the walker into position at the start of the track. A fluorophore is tethered near the starting position of the track, and the quencher causes the fluorophore's signal to decrease as the walker loads into position. When additional walker was titrated into the solution, there was very little decrease in the fluorescence signal, indicating a high percentage of walker loading.

3.4 Purification by Size Exclusion

During origami self-assembly, an excess of staples and track anchorages is used to increase the probability of correct assembly. Once the walkers are loaded onto the tile, however, these excess staples and anchorages can interfere with the walkers' stepping and cause reaction leaks. These leaks can be prevented by removing the excess strands before the stepping reaction is started.

The origami solution is purified by size exclusion after the walkers are loaded onto the tile. Sephacryl S-300 high resolution chromatography resin (GE Healthcare) was resuspended in 1× origami buffer and handpacked into chromatography columns.

The origami solution was added to the column and filtered through the resin by centrifugation three times in a series to ensure that all excess staples were removed.

Sephacryl S-300 excludes single-stranded DNA of 118 base pairs or less, as claimed by the manufacturer. All staples, fluorescent labelled strands, and anchorage strands are shorter than 118 base pairs; they are filtered out of the solution by the resin. The large DNA origami tile is able to pass through the resin unimpeded. Therefore, the only strands present in the final origami solution are those that are annealed in the origami tile.

3.5 Covering and Releasing the Anchorages

Following purification by size exclusion, there is a time delay before the stepping reaction can be started by adding the enzyme. The origami solution must be diluted to the correct volume for the cuvettes, and then added to the fluorescence spectrophotometer. During this delay, the walker can undergo blunt strand exchange with anchorages near the starting position due to their close proximity. This can cause the walker to step several anchorages in the absence of the enzyme, which can cause an error in the blocking mechanism.

To prevent early stepping by blunt strand exchange, all anchorages except for the starting anchorages have a covering mechanism that keeps the walker from binding to an anchorage's toehold until the nicking enzyme is added (Figure 3.3) [100]. The anchorage has an extra 5' domain that binds to a covering strand. The covering strand covers the toehold of the anchorage to prevent premature walker binding, and has a long 3' toehold. The covering strand is removed by its reverse complement uncovering strand just before the enzyme is added. The uncovering strand hybridises to the long toehold on the covering strand to remove it from the anchorage. This technique is used in all experiments in Chapter 4-5 to prevent premature walker stepping. For

simplicity, the cover/uncover mechanism is omitted in the track design illustrations shown in these chapters.

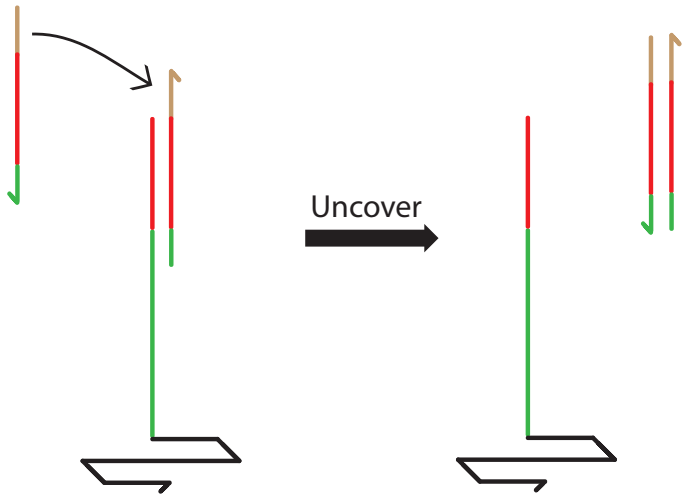


Figure 3.3: The covering and uncovering mechanism used to prevent the walker from stepping down the track via blunt strand exchange before the enzyme is added to solution. The anchorage domain (green) has a 5' extension (red) that binds to a covering strand. The covering strand also extends to bind to the toehold in the anchorage domain. The anchorage is uncovered before the enzyme is added by adding an uncovering strand, which binds to the cover strand's toehold (gold) and displaces it from the anchorage via toehold-mediated strand displacement.

3.6 Normalising the Signal

The magnitude of a fluorophore's signal depends on a number of different factors. Different fluorophores, such as TEX615 and Cy5, will emit at different intensities even when they are at the same concentration. The signal can vary depending on the nucleotides that are near the fluorophore, and the signal decrease by the quencher on the walker will depend on how close the fluorophore is tethered to the anchorage. This section describes a method for normalising for all of these factors so that the fluorescence signals for a DNA walker experiment are given as a percentage of maximum fluorophore signal under those experimental conditions.

When the origami solution is placed in the fluorescence spectrophotometer for data

collection, the walker with its attached quencher is loaded in its starting position on the track. The quencher on the walker decreases the signal from the nearby TEX615 fluorophore, but the Cy5 fluorophore at the end of the end of the track is unquenched. A baseline for the unquenched signal intensity for Cy5 is taken during this time, and this unquenched baseline is denoted $\mathbf{U}_{Cy5}$.

To find the unquenched baseline $\mathbf{U}_{TEX}$ for TEX615, all walkers must be removed from the first anchorage of the track. To ensure that all walkers are removed, a strand complementary to the walker is added to the solution after the experiment has finished. These strands hybridise to the walker and displace the walker from the track via toehold mediated strand displacement. The TEX615 signal after all of the walkers have been displaced from the tile is the unquenched baseline, and is denoted $\mathbf{U}_{TEX}$.

After all walkers have been displaced from the tile, the quenched baselines $\mathbf{Q}_{Cy5}$ and $\mathbf{Q}_{TEX}$ can be found by adding a large excess of walker to the solution. The excess is sufficiently large to ensure that every anchorage in the solution has a walker bound to it, including the first and last anchorages in the track. When walkers are bound to these anchorages, the quencher on the walker will decrease the signal from the TEX615 and Cy5 fluorophores to give the fully quenched baselines.

Let $S_{TEX}(t)$ and $S_{Cy5}(t)$ be the intensities of TEX615 and Cy5, respectively, at time t. The normalised signals can be found using the baselines:

$$|S_{TEX}(t)| = \frac{S_{TEX}(t) - \mathbf{Q}_{TEX}}{\mathbf{U}_{TEX} - \mathbf{Q}_{TEX}},$$
$$|S_{Cy5}(t)| = \frac{S_{Cy5}(t) - \mathbf{Q}_{Cy5}}{\mathbf{U}_{Cy5} - \mathbf{Q}_{Cy5}}.$$

These normalised signals are fractions between 0 and 1, indicating the fraction of the fluorophore's maximum intensity under the experimental conditions. In addition to normalising the signals for experimental conditions, the normalised signals provide two pieces of additional information. First, the intensity of $|S_{TEX}(t)|$ before the

enzyme is added gives a measure of how much walker was correctly loaded into its starting position on the tile. Second, the $|S_{Cy5}(t)|$ signal near the end of the run provides a measure of how much walker has reached the end of the track. This type of normalisation also allows for comparison between different experiments when the signals are normalised for the amount of walker loaded.

Chapter 4

Experimental Implementation of DNA Walker Circuits

This chapter shows the mechanisms whereby the blocking primitives from Section 2.1 are implemented by DNA walkers tethered to a DNA origami substrate. Experimental work was done in collaboration with Alexandra E. Lucas in the Oxford Department of Physics, and the figures and descriptions for DNA mechanisms are similar to the joint publication in [15]. Different design strategies can be used to block a walker (Section 4.1), and step-by-step mechanisms are given to demonstrate how they can be implemented. As shown in Section 4.1.1, the example mechanism given for destroying the track has a number of advantages over the others: it is simple, the junction anchorage does not require any additional secondary structure, and no additional domains are required on the walkers. For these reasons, this mechanism was chosen to implement a NOT gate (Section 4.3) and a NOR gate (Section 4.4). A NOR gate is functionally complete; any other logic gate can be constructed out of NOR gates. This demonstrates proof of principle that these circuits are capable of emulating propositional logic.

4.1 Strategies for Implementing Blocking Primitives with DNA Walkers

There are different strategies for blocking a walker: destroying anchorages in the track that a walker was meant to step onto (Section 4.1.1), trapping a walker so it cannot move away from the junction between tracks (Section 4.1.2), and removing a walker from the track (Section 4.1.3). Each strategy comes with advantages and disadvantages.

In each of the strategies below, a blocking "blue" walker blocks a "green" walker. Single-stranded molecules of DNA are represented by half-arrows with the arrowhead designating the 3' end. Domains that are reverse complements of one another are given the same colour. A completed restriction site is indicated by a red box, and the enzyme cuts at the location shown by the small red arrow.

4.1.1 Destroying the Track

To block by destroying the track, a blocking walker must irreversibly destroy the track of another walker. When the track that a walker would step down has been destroyed, the walker can no longer reach the end of its track and has been blocked. A mechanism based on destruction of the track is shown in Figure 4.1. There is a long anchorage at the junction of the two tracks that has a green binding site at its "top" (farthest from the origami tile) and a blue binding site at its "bottom" (nearest the origami tile). When this anchorage is intact, then the junction acts as a normal green anchorage and the green walker steps onto the green binding site at the top of the junction anchorage.

If the blue walker arrives at the junction first, it hybridises to its binding site at the bottom of the junction. When the enzyme cuts the junction anchorage, the top of the junction anchorage (containing the green binding site) diffuses away from the

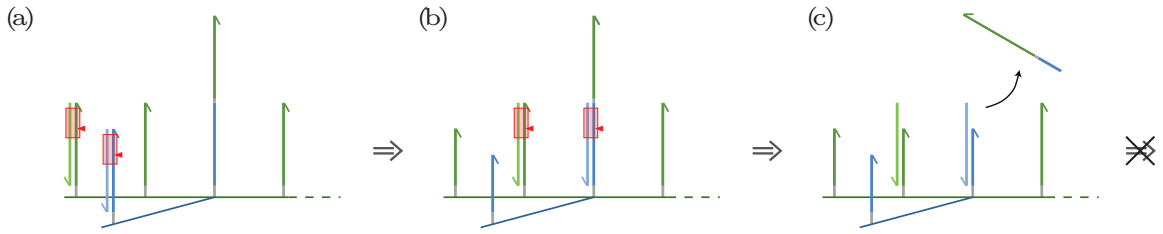


Figure 4.1: An example of a DNA mechanism that blocks by destroying the track. (a) The blue walker is likely to arrive at the junction first to block the track for the green walker. (b) By hybridising to its binding site near the bottom of the junction anchorage, the blue walker completes a restriction site for the nicking enzyme on the junction. (c) The enzyme cuts the junction, causing the top portion of the junction (which contains the green binding site) to diffuse away from the track. Without the junction binding site, the green walker is unable to reach the end of its track and is blocked.

tile. If the green walker subsequently arrives at the junction, the absence of a binding site on the junction anchorage prevents it from reaching the end of the track.

A key advantage is that the blue walker can continue stepping down its track after blocking the green track, potentially blocking subsequent tracks. This type of mechanism would be ideal for constructing a FANOUT gate. However, the long junction anchorage can introduce a possible error into the system: the extended length of the junction anchorage may allow it to hybridise to the green walker before it has reached the adjacent anchorage, increasing the chance of a missed-chance error. This error is quantified in the following chapter, and can be alleviated by shortening the length of the green toehold domain on the junction anchorage to lower the rate at which it hybridises to the green walker.

Different origami structures have different degrees of flexibility. In some cases, the tile may bend, allowing the blocked green walker and the final anchorage of the green track to come close together. This may cause a slow leak whereby a blocked walker can still reach the end of its track. When this system is implemented using a flexible tile, a gap can be introduced in the green track after the junction. If the green walker is unblocked, the long junction anchorage can help it cross this gap and

reach the end of the track. When the green walker is blocked, the gap prevents the blocked walker from reaching the end of the track when the tile flexes.

Of the three mechanisms shown in this section, this mechanism has a number of advantages over the other two. It is the simplest: the junction anchorage consists of only one oligonucleotide, mismatches in the enzyme's restriction site are not incorporated into the junction, and the walkers do not need any extra "tail" domains. Eliminating these extra features can help reduce the chance of error and ensure that the walkers step at a consistent, predictable rate. Hence, this mechanism was chosen to implement a NOT gate and a NOR gate. However, there are two additional mechanisms that may be useful if the application requires trapping or removal of the walker.

4.1.2 Trapping the Walker

To block using the strategy of trapping the walker, the blocking walker must immobilise the walker that it blocks so that it is unable to move from that location. If the immobilised walker is unable to move, then it cannot reach the end of its track and is blocked. The example mechanism that implements this strategy is shown in Figure 4.2. The blue walker enables the green walker to bind more stably at the junction than it can with the next anchorage in the track. Therefore, the green walker remains bound to the junction because it is the most energetically favourable state.

Both walkers used in the mechanism have complementary single-stranded tail domains: the blue walker has a 3' tail domain while the green walker has a 5' tail domain. If the green walker arrives first, it can hybridise to its binding site on the junction anchorage, the enzyme cuts the junction, and the green walker steps on as normal. However, if the blue walker arrives at the junction first, it steps onto its binding site at the top of the junction anchorage. When the green walker arrives, its 5' tail domain hybridises to the complementary tail domain at the 3' end of the blue

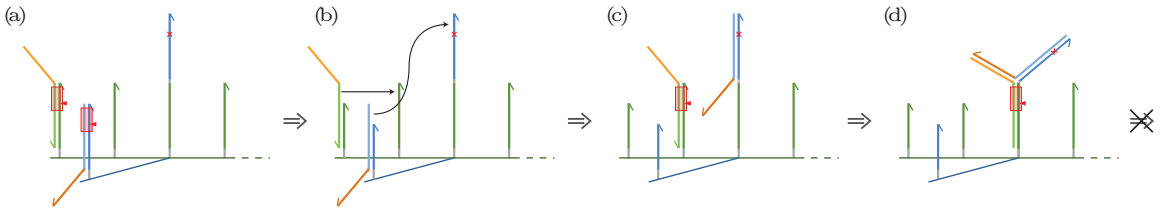


Figure 4.2: A mechanism that traps the green walker at the junction by stabilizing the walker-junction complex with extra base pairing. (a) The blue walker is likely to arrive at the junction first to block the green walker. (b) The blue walker steps onto to its binding site at the top of the junction anchorage. (c) The green walker arrives at the junction, ready to step onto its binding site at the bottom of the junction anchorage. (d) When the green walker hybridises to the junction, the 5' tail on the green walker and the 3' tail on the blue walker hybridise to each other. The extra base pairing from the bound tail domains makes it more energetically favourable for the green walker to remain bound to the junction instead of stepping onto the next anchorage, even after the junction anchorage is cut.

walker. Even when the enzyme cuts the junction anchorage, the green walker will not step onto the next anchorage because it is stabilized by the extra base pairing provided by the bound tail domain.

The mechanism relies on the blue walker remaining at the junction to hybridise to the green walker, so this mechanism cannot be used to make a FANOUT gate. For simplicity, Figure 4.2 shows the green walker with only one tail domain, the one used to trap it. If the circuit design requires the green walker to block subsequent tracks, it can be given a 3' tail (in addition to its 5' tail) that is the reverse complement of the tail domain on the walker it will block.

4.1.3 Removing the Walker

This strategy blocks a walker by removing it from the origami tile. The walker is no longer able to reach the end of its track once it has been separated from the tile, rendering it blocked. Figure 4.3 shows an example of a mechanism that implements this strategy. In this mechanism, the blue walker displaces the strand with the green walker's binding site from the junction and exposes the toehold of another strand on

the junction that will remove the green walker from the origami tile.

The blue walker differs from the green walker in two ways. First, the blue walker is in reverse orientation to the green walker; its toehold domain is at the 5' end. Second, the blue walker has an extra "tail" domain at the 3' end that plays a role in blocking the green walker. The junction anchorage consists of three strands: a tether strand attached to the origami tile, a remover strand that is initially fully bound to the tether, and an auxiliary strand, also bound to the tether, that has a single-stranded 5' extension that acts as a green anchorage. The remover strand has the same nucleotide sequence as the green anchorage, with the exception of a single nucleotide mismatch in the restriction enzyme recognition sequence.

If the green track is unblocked, the green walker steps onto the green binding site on the auxiliary strand of the junction anchorage. Once the restriction site for the nicking enzyme is completed, the auxiliary strand is cut and the green walker steps on as normal. If the blue walker arrives first, it binds to the blue toehold domain on the tether strand and displaces the auxiliary strand, removing the binding site for the green walker. The 3' tail domain on the blue walker is then in competition with the remover strand for a binding site on the tether strand, partially exposing a green toehold domain on the remover strand. When the green walker arrives at the junction, it binds to this toehold initiating a strand-exchange reaction in which the green walker hybridises to the remover strand and displaces it from the junction anchorage. A restriction site mismatch in the remover strand prevents enzymatic cleavage of the remover-green walker duplex. Hence, the green walker is blocked by removing it from the track.

This blocking mechanism relies on the blue walker staying bound at the junction, so it cannot be used to make a FANOUT gate. The advantage of this mechanism is that the green walker is removed from the track, reducing the chance that flexibility of the tile will allow it to step over the junction. Shortening the green toehold domain in

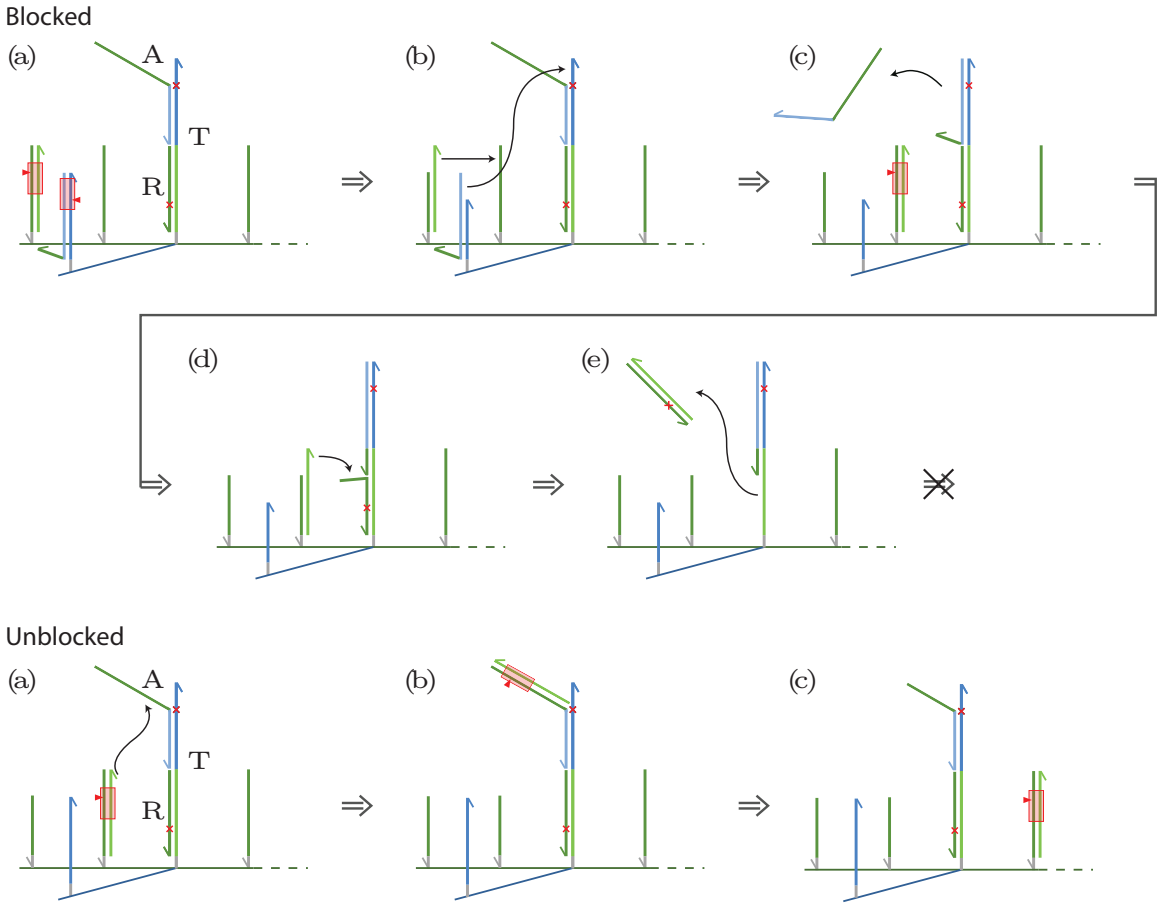


Figure 4.3: Removing the walker. The mechanism uses a three-strand junction: T = tether; A = auxiliary strand; R = remover. The restriction site is shown by a red box, and a mismatch in the restriction site is shown by a red cross. The blue walker displaces the auxiliary strand, exposing the toehold of the remover strand that can displace the green walker from the tile. (Blocked, a) The blue walker is likely to arrive at the junction anchorage first. (Blocked, b) The blue walker steps onto its binding site at the top of the tether strand, displacing the auxiliary strand. (Blocked, c) The auxiliary strand diffuses away from the tile. (Blocked, d) The tail domain on the blue walker exposes the green toehold of a remover strand. When the green walker arrives at the junction, it binds to the remover toehold that was exposed by the blue walker. (Blocked, e) The green walker hybridises to the remover strand, forming an inert duplex which is displaced from the track. Due to the complexity of the mechanism, the unblocked case is shown as well. For the unblocked case, the green walker steps through the junction by hybridising to the auxiliary strand.

its binding site on the auxiliary strand at the junction can help prevent errors caused by extended reach. To scale this mechanism up to more complex circuits, the green walker should be able to both block a track and be blocked by another walker. This can be accomplished by giving the green walker a 5' tail domain so that it can act as a blocking walker at subsequent junctions.

4.2 Finding Orthogonal Walker Sequences with Similar Stepping Rates

The blocking walker should arrive at the junction before the walker that it blocks to avoid a missed-chance error. If the walkers have the same rate of stepping, this can be accomplished by adjusting the lengths of each track. However, a walker's stepping rate is sequence-dependent, and the walkers must have orthogonal sequences so that a walker does not step onto another walker's track. It is therefore important to find two orthogonal sequences for the walkers where the walkers will step at the same rate.

The first walker is the 26 nucleotide walker from [10] with an eight nucleotide toehold (highlighted in blue) on its 3' end:

5' - CGA TGT TAG TTG GGC TGA **GGT TCG AT** - 3'.

Another walker sequence was designed using NUPACK [107] in such a way that it is orthogonal to the walker above, but has similar G-C content and free energy when hybridised to its reverse complement. NUPACK showed that the following 24 nucleotide walker was a promising candidate:

5' - TGT GAT GAT TGA GGC TGA **GGT GAG** - 3'.

This second walker sequence has the same restriction site for the Nt.BbvCI nicking enzyme, but has a six nucleotide toehold in contrast to the eight nucleotide toehold in the walker above.

The above walkers' stepping rates were measured as they each stepped down short tracks of three anchorages (Figure 4.4). A TEX615 fluorophore was tethered to the origami tile near the start of each track, and a Cy5 fluorophore was tethered near the end of each track. Both walkers had a 5' Iowa Black dark quencher modification. The **1** track and **x** track were arranged on the origami tile in the same positions as the first three anchorages in the NOT track in Figure 4.6. The 26 nucleotide walker was selected to be the **1** walker and the 24 nucleotide walker was selected to be the **x** walker. Measurements for each walker were taken in separate cuvettes, but at the same time and in the same fluorescent spectrophotometer to ensure the same experimental conditions.

The output signals from the fluorophores at the start and end of the tracks are shown in Figure 4.5. As the walkers step away from the first anchorage, the TEX615 fluorophore signal increases as the quencher on the walker moves away. As walkers accumulate at the end of the track, the quencher on the walker causes the Cy5 signal to decrease. The normalised intensity vs. time traces (Figure 4.5) for each walker are nearly identical, indicating that the two walker sequences above have very similar stepping rates. They are therefore good candidates for use in the NOT and NOR gates.

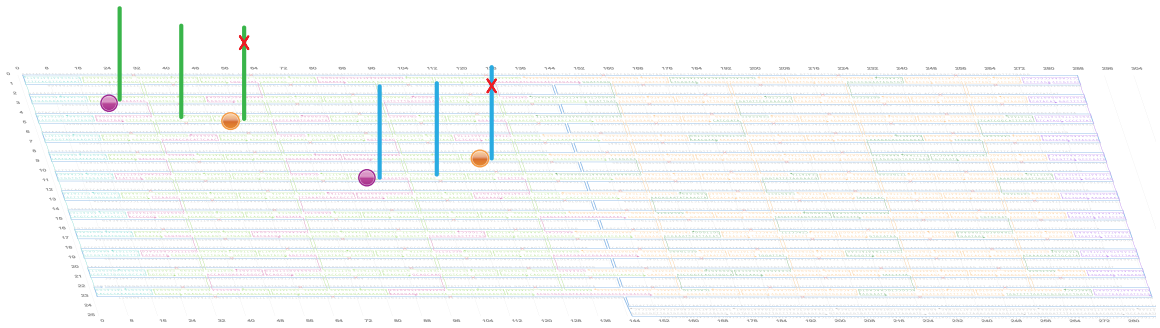


Figure 4.4: CADnano schematic of a minimal twist DNA origami tile with the locations of the two tracks (green and blue) superimposed. Each anchorage is a 5' extension of an origami staple. The 26 nucleotide walker was designated as the **1** walker (green) while the 24 nucleotide walker was designated as the **x** walker (blue). An Iowa Black dark quencher was tethered to the 5' end of each walker so that fluorophores tethered to the origami tile would quench when the walker was nearby. TEX615 fluorophores (purple circles) were placed at the start of each track and Cy5 fluorophores (orange circles) were placed at the end of each track. The final anchorage in each track has a restriction site mismatch (red cross) to stop the walker once it reaches the end of the track.

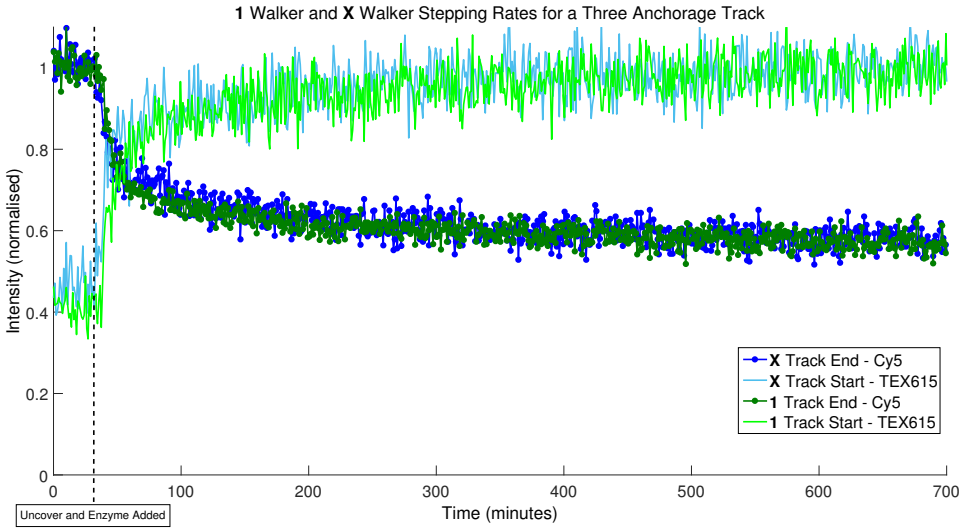


Figure 4.5: Normalised intensity vs. time signals for Cy5 and TEX615. The signals for each walker are nearly identical, indicating that the two walker sequences have very similar rates of stepping.

4.3 NOT: Logical Negation

The blocking mechanism chosen for the NOT gate was the "destroying the track" blocking strategy from Section 4.1.1 due to its simplicity: the junction anchorages can be loaded onto the tile during self-assembly of the origami, and the mechanism does not rely on branched structures. As discussed in Section 1.4.3, strain at the tile crossover locations can introduce a global right-handed twist. While this effect is reduced in the minimal twist tile used in this thesis, residual twist may lead to different walker stepping rates at different locations on the tile. Therefore, the anchorages in the NOT gate were arranged on the origami tile by extending the **1** track from Figure 4.4 to help ensure that both the **x** and **1** walkers step at the same rate (Figure 4.6). The timing with which the walkers arrive at the junction was controlled by making the **1** track longer than the **x** track: the **x** track has two anchorages before the junction whereas the **1** track has six anchorages before the junction.

Including two junction anchorages instead of one allows the blocking walker to create a larger gap in the **1** track (Figure 4.7). It is therefore less likely for the **1** walker to reach the end of its track once the **x** walker has blocked the track. To make the gap larger, the **1** track has an anchorage missing between the junction and the **1** track's final anchorage. When intact, the length of the junction anchorages enables the **1** walker to step onto the final anchorage. If the junction anchorages are cut, the gap is too large for the **1** walker to reach the end of the track.

When the **x** walker steps through the junction anchorages, the enzyme cuts the junction anchorages so that the **1** walker's binding sites on the junction will diffuse away from the tile. Once the **x** walker has stepped through all if its track anchorages, it is free to diffuse among the cut anchorages in the **x** track where it may introduce leaks or unexpected behaviour. To prevent this, an additional anchorage is added to the end of the **x** track with a restriction site mismatch that stops the **x** walker's movement. Once the **x** walker is hybridised to the end anchorage, it will remain

there. To prevent the **x** walker from binding to the end anchorage prematurely and leaving the junction anchorages uncut, a gap is introduced in the **x** track between the junction anchorages and the **x** end anchorage (Figure 4.7).

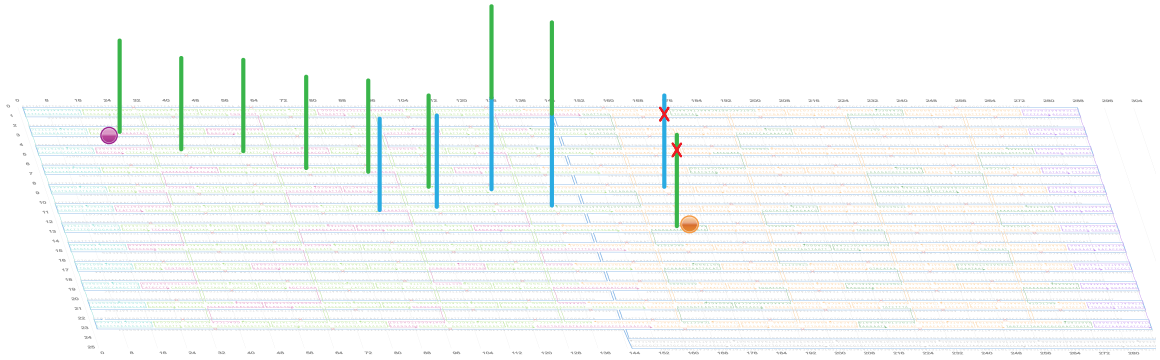


Figure 4.6: The location of the track anchorages in the NOT gate. Anchorages are 5' extensions of staple strands, and are shown superimposed on the CADnano schematic of the minimum twist tile. The **1** track (green) has a TEX615 fluorophore reporter (purple circle) tethered near the first anchorage and a Cy5 fluorophore reporter (orange circle) tethered near the final anchorage. The **x** track (blue) intersects the **1** track at the location of the two long junction anchorages. The junction anchorages have a binding site for the **x** walker at the bottom (nearest the tile) and a binding site for the **1** walker at the top (near the 5' end). The final anchorages of both the **1** and **x** tracks each contain a restriction site mismatch (red cross) that inhibits the walker's movement once bound.

To test whether the blocking mechanism was effective, a NOT gate was assembled in the state shown in Figure 4.7, bottom. The junction anchorages were shortened to mimic the state where the **x** walker has already stepped through the junction and reached the end of its track. This experiment determines whether the blocked track prevents the **1** walker from reaching the end of the track.

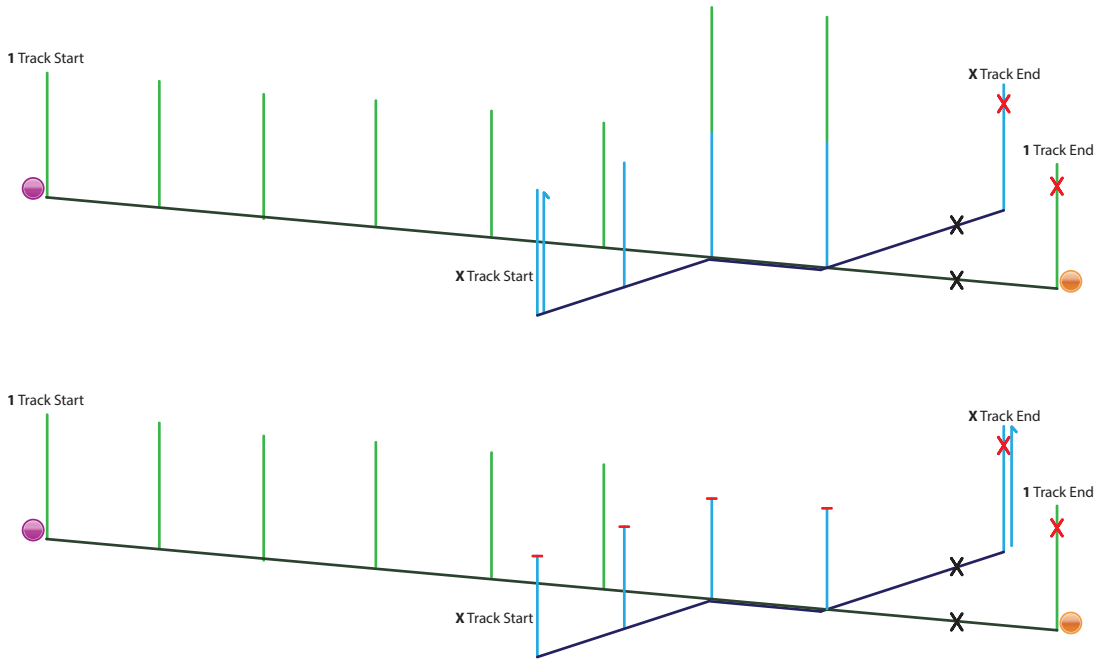


Figure 4.7: Anchorages in the NOT gate before the **x** walker has started stepping (top) and after the **x** walker has reached the end of its track (bottom). The **x** walker blocks the **1** track by creating a gap in the **1** track. The movement of the **1** walker is reported by TEX615 (purple circle) and Cy5 (orange circle) fluorophores. Missing anchorages in the **1** track and **x** track are indicated by black crosses.

The normalised fluorophore signals for the pre-blocked NOT gate are shown in Figure 4.8. When the enzyme is added to the solution, the **1** walker steps away from the first anchorage and TEX615 signal increases. When the **1** walker is unable to traverse the gap in the track, it diffuses among the first six anchorages of its track via blunt strand exchange. The walker will spend time hybridised to the first anchorage as it diffuses, quenching TEX615. In the ensemble fluorescence measurement, this causes the TEX615 signal to decrease again.

Despite the gap in the **1** track, the signal from the Cy5 reporter at the end of the **1** track decreases over time, indicating that walkers are still able to reach the end of the track. However, this percentage is small. As shown in Table 4.1, the intensity of the Cy5 signal decreased by 8% after 300 minutes. By 700 minutes, the intensity of the Cy5 signal had decreased by 12%. While this indicates the presence of a reaction

leak whereby the **1** walker can traverse the gap in the track, it also indicates that correct operation predominates. This demonstrates proof of principle for the blocking strategy and track design.

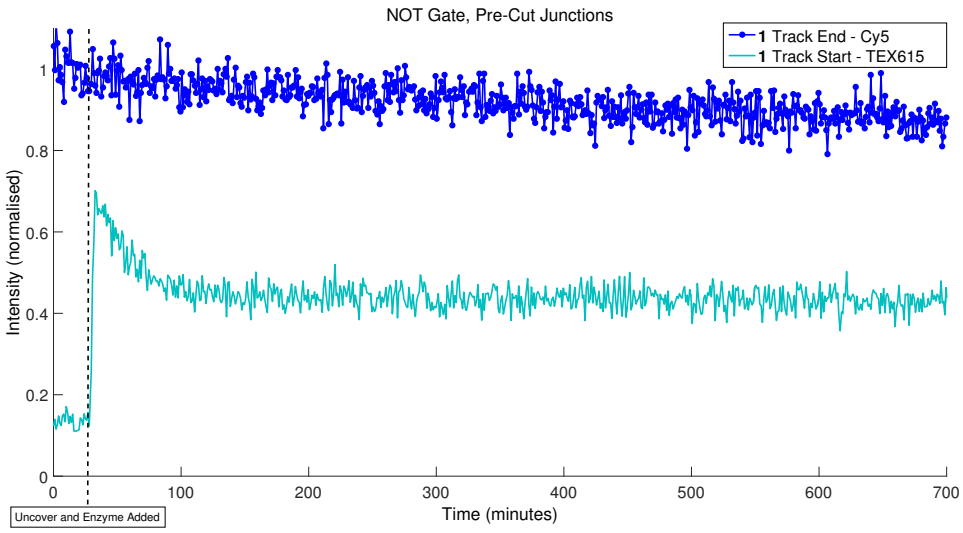


Figure 4.8: Cy5 and TEX615 signals from the pre-blocked track shown in Figure 4.7, bottom. The TEX615 fluorophore (light blue trace) is tethered in close proximity to the first anchorage of the **1** track, and rapidly unquenches when the walker begins stepping. When the walker cannot traverse the gap in its track, it diffuses on the six anchorages before the junction, causing some walkers to re-bind to the first anchorage and quenching the signal from TEX615. The Cy5 fluorophore (dark blue trace) is tethered near the final anchorage in the **1** track. It shows slow quenching, indicating some walkers can step to the end of a blocked track, but this percentage is small when compared with an unblocked track.

NOT Gate, Pre-Cut Junction		
Loading	Cy5 Quenched at 300 min.	Cy5 Quenched at 700 min.
87%	8%	12%

Table 4.1: Percentages of walker loading and Cy5 intensity 300 minutes and 700 minutes after the enzyme was added to the solution. The **1** walker, hybridised to the first anchorage in its track, anneals into 87% of the origami which indicates a high percentage of correct loading. At a time point 300 minutes after the enzyme has been added, the Cy5 fluorophore has an 8% decrease in signal. At 700 minutes after the enzyme has been added, the decrease in signal decreases to 12%. This shows that the blocking strategy is effective, but still suggests the presence of a leak whereby the **1** walker can traverse a blocked junction.

For a NOT gate, the **x** walker must convert the **1** track from an unblocked state (Figure 4.7, top) to a blocked state (Figure 4.7, bottom). To demonstrate that the action of the **x** walker can achieve a blocking effectiveness similar to that of the pre-cut junctions, a tile was assembled as in Figure 4.6 with intact junction anchorages, a **1** walker loaded on the first anchorage of the **1** track, and a **x** walker loaded on the first anchorage of the **x** track. Both walkers started stepping down their respective tracks when the nicking enzyme was added to the solution.

Figure 4.9 shows the signals from TEX615 and Cy5 for the case where the **x** walker blocks the **1** track and the case where the **1** track is unblocked. When the **x** walker blocks the **1** track, the TEX615 signal at the beginning of the **1** track has a similar shape to the signal in the pre-cut junction experiment: the TEX615 signal increases as the **1** walker steps away from the first anchorage, then decreases again as the walker is unable to cross the junction and diffuses on the first six anchorages of its track.

In contrast, the TEX615 signal from the unblocked case increases rapidly as the walker steps away, followed by a decrease in signal that is small compared to the blocked case. This slight decrease is due to the missing anchorage between the second junction anchorage and the final anchorage in the **1** track. In order to reach the end of the track, the **1** walker steps from the second junction anchorage across the gap created by the missing anchorage and onto the final anchorage in the track which contains a mismatch in the restriction site. This mismatch is essential for stopping the movement of the walker, but it can also decrease the rate at which the walker hybridises to the anchorage. Crossing the gap to hybridise to the final anchorage is a slow step, and the walker may diffuse on the track before binding to the final anchorage.

The signal from the Cy5 fluorophore tethered at the end of the **1** track decreases much more rapidly in the unblocked case than in the blocked case. More **1** walkers

arrive at the end of the track to quench Cy5 when the track is unblocked. At 300 minutes after the enzyme has been added to solution, the Cy5 signal decreased by 51% when the track was blocked as opposed to 21% when the track was blocked (Table 4.2). A 21% decrease in signal shows less blocking effectiveness than the 8% decrease in the pre-cut junction experiment. However, a decrease in effectiveness is expected: when the **x** walker cuts the junctions, there is a possibility of missed-chance error. Successful blocking also relies on the correct loading of the blocking walker and the correct assembly of the blocking walker's track.

Over a 300 minute timescale, the ratio of Cy5 signal decrease in the unblocked vs. blocked case is,

$$\frac{51\%}{21\%} = 2.4.$$

Over the longer 700 minute timescale, however, this ratio decreases to,

$$\frac{58\%}{32\%} = 1.8.$$

This suggests that this system behaves differently over short and long timescales. Over short timescales of 300 minutes or less, correct operation predominates. After 700 minutes, however, slow leak reactions have accumulated that lead to incorrect operation whereby a significant portion of **1** walkers are able to reach the end of the track. These leaks can affect the asymptotic behaviour of the output signals when the experiment is allowed to run for a long time.

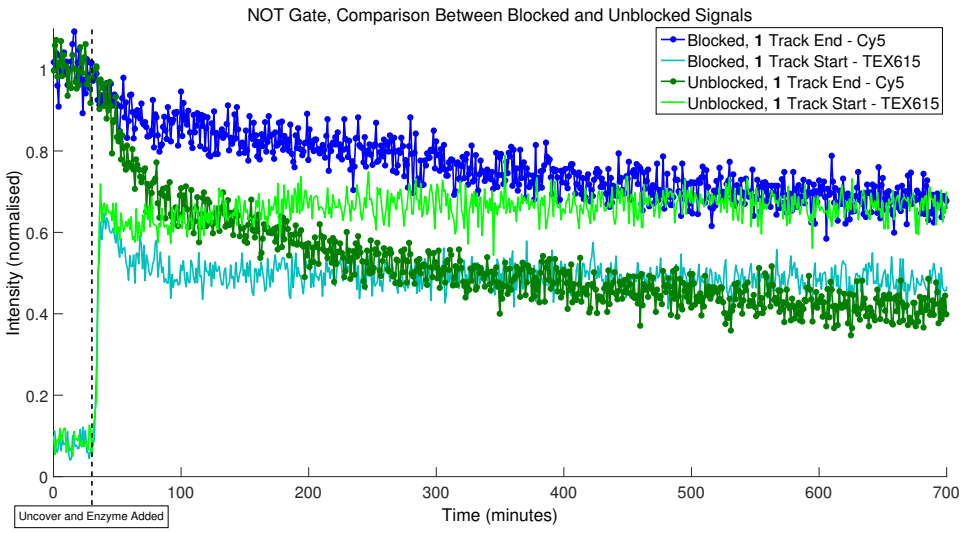


Figure 4.9: A comparison between the fluorophore signals when the track is blocked (blue traces) and unblocked (green traces). In the unblocked case, the TEX615 signal (located at the start of the **1** track, shown in light green) has a rapid initial unquenching. It slightly requenches, indicating that the gap after the junction can slow the **1** walker reaching the end of its track. The signal then resumes a steady unquenching, characteristic of a walker that can make it to the end of its track. The Cy5 signal (located at the end of the **1** track, shown in dark green) decreases as the **1** walker binds to its final anchorage. In the blocked case, TEX615 (shown in light blue) shows a similar shape to the TEX615 curve in Figure 4.8 which is characteristic of a blocked walker. There is a gradual decrease in the Cy5 signal (dark blue), indicating that some **1** walkers still make it to the end of the track via reaction leaks, but not as much as in the unblocked case.

NOT Gate Blocked and Unblocked Signals			
	Loading	Cy5 Quenched at 300 min.	Cy5 Quenched at 700 min.
Blocked	91%	21%	32%
Unblocked	91%	51%	58%

Table 4.2: Benchmarks for the NOT gate data in Figure 4.9. The similar loading percentages (left column) of the **1** walker between the blocked and unblocked case allow for valid comparison between the two sets of signals. The percentage of Cy5 quenched at 300 minutes gives a measure of the gate's performance over a shorter timescale, while the percentage of Cy5 quenched at 700 minutes shows how the gate performs over a long timescale. Comparing the ratio between blocked and unblocked Cy5 signals at 300 minutes vs. 700 minutes shows that the gate is more reliable over shorter timescales.

4.4 NOR: A Functionally Complete Operator

A NOR gate is functionally complete: any other logic gate can be constructed out of NOR gates. Building a NOR gate is therefore important for demonstrating proof of principle that DNA walker circuits can emulate propositional logic. Extending the NOT gate to a NOR gate requires adding an additional **y** track that intersects the **1** track. The track diagram in Figure 2.2 shows the **x** and **y** tracks intersecting the **1** track at separate junctions. However, creating two junctions instead of one would require lengthening the **1** track. To avoid this, the **x** and **y** tracks intersect the **1** track at the same junction. The arrangement of anchorages on the origami tile for the NOR gate is the same as the NOT gate, with the addition of the **y** track (Figures 4.10-4.11).

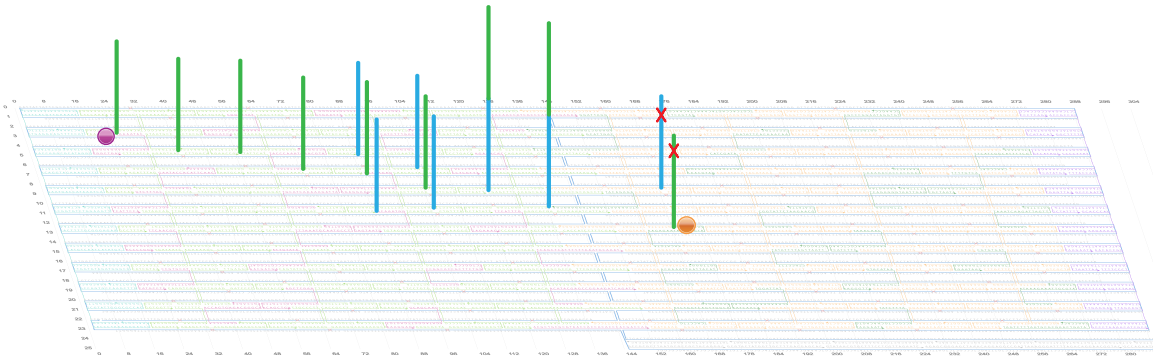


Figure 4.10: Anchorage locations for the NOR gate superimposed on the DNA minimal twist origami tile. Note that the anchorage locations for the **1** and **x** tracks are identical to those in the NOT gate shown in Figure 4.6. A **y** track has been added that intersects the **1** track at the same intersection as the **x** track. As in the NOT gate, TEX615 (purple circle) is tethered near the first anchorage in the **1** track and Cy5 (orange circle) is tethered near the final anchorage in the **1** track.

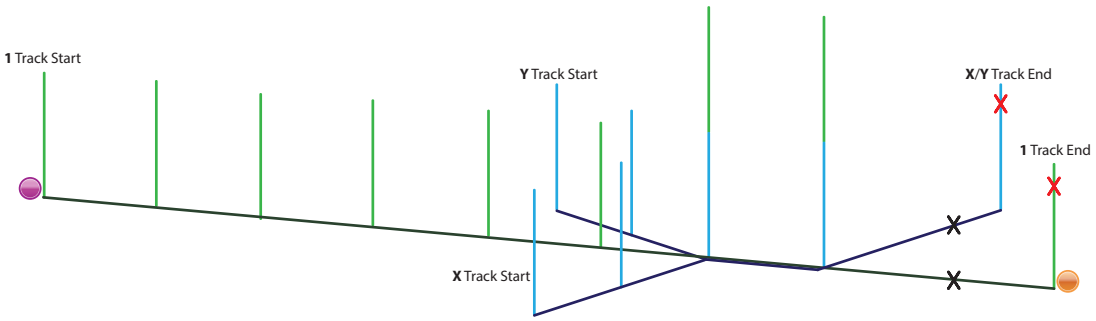


Figure 4.11: A simplified track diagram of the NOR gate with the origami tile omitted. The **1** and **x** tracks are the same as the NOT gate in Figure 4.7. A **y** track (with its starting anchorage labelled) has been added to create a NOR gate. Both the **x** and **y** track anchorages are shown in the same colour (light blue), as both the **x** and **y** walkers have the same sequence. This simplification avoids the challenge of finding three orthogonal walker sequences with similar stepping rates while allowing both the **x** and **y** tracks to use the same junction.

Finding three orthogonal sequences with the same stepping rate would present a challenge. To overcome this obstacle, the same sequence was used for both the **x** and **y** walkers. Giving the **x** and **y** walkers different address sequences in loading (the orange domain in Figure 3.1 from Chapter 3) allows each walker to be loaded onto the correct track even though the walkers share the same sequence. However, this introduces a possible leak whereby the **x** walker could step onto the **y** track (and vice versa) instead of onto the junction anchorages. Comparing the performance of the NOR gate when x = True and y = False to the performance the blocked NOT gate will indicate how significant this leak is.

The signals from the Cy5 fluorophore at the end of the **1** track are shown in Figure 4.12 for each of the four possible inputs to the NOR gate. Similar to the results of the NOT gate, the case where the **1** track is unblocked (i.e., propositions $x = y$ = False) shows a rapid decrease in Cy5 signal. When the **1** track is blocked by one or both of the **x** and **y** walkers, there is a more gradual decrease in Cy5 signal.

For the cases where the **1** track is blocked by at least one walker, the signal when x = True and y = False shows a larger decrease in Cy5 signal than the Cy5 signals

when y = True and x is either True or False. In addition, the signal when x = True and y = True is nearly identical to the signal when x = False and y = True. This suggests that the **y** walker blocks more effectively than the **x** walker. One possible explanation is that the tile is more flexible along the axis of the **y** track than along the axis of the **x** track, allowing the **y** walker to reach the junction more quickly.

The percent decrease in Cy5 signal is shown in Table 4.3. For the cases when y is True and x is either True or False, blocking is effective over a 300 minute timescale. The Cy5 signal at the end of the **1** track decreases by 15%. The ratio of Cy5 quenched for these two cases versus the unblocked case is,

$$\frac{47\%}{15\%} = 3.1.$$

This outperforms the case where y is False and x is True, where the ratio is,

$$\frac{47\%}{25\%} = 1.9.$$

While the NOR gate is effective over a 300 minute timescale, performance is less effective over the longer 700 minute timescale. For the two cases when y is True, the ratio of the Cy5 signal for the blocked case versus the unblocked case goes from 47%/15% = 3.1 at 300 minutes to 56%/25% = 2.2 at 700 minutes. There is also a decrease in performance for the case where x is True and y is False, where the ratio goes from 47%/25% = 1.9 over 300 minutes to 56%/33% = 1.7 over 700 minutes. Similar to the NOT gate, the NOR gate has a number of slow reaction leaks that affect the gate's reliability over longer timescales.

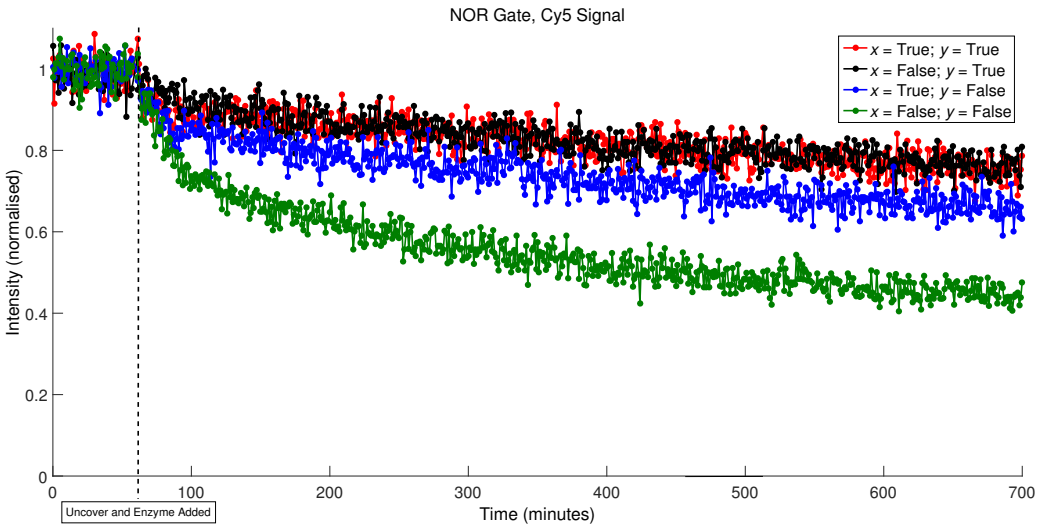


Figure 4.12: Signals from Cy5 at the end of the **1** track when the **1** track is unblocked (dark green), blocked by the **x** walker alone (dark blue), blocked by the **y** walker alone (red), and blocked by both the **x** and **y** walkers (black). The unblocked signal shows the greatest decrease over time, indicating that more **1** walker reaches the end of the track when the track is unblocked. The other three signals show a more gradual decrease compared to the unblocked signal, showing that the blocking walker(s) impeded the **1** walker's movement.

NOR Gate Signals			
	Loading	Cy5 Quenched at 300 min.	Cy5 Quenched at 700 min.
x=T;y=T	79%	15%	25%
x=F;y=T	81%	15%	25%
x=T;y=F	80%	25%	33%
x=F;y=F	81%	47%	56%

Table 4.3: Statistics for the NOR gate. The percentages of origami tiles loaded with a **1** walker (left column) are similar for each case, so the four cases can be compared to one another. The percentage of Cy5 quenched is benchmarked over 300 minutes and over 700 minutes. These benchmarks show that the gate is more reliable over shorter timescales.

The difference in Cy5 signal at 700 minutes between the case where $x = y =$ False and the case where x is True and y is False is $56\% - 33\% = 23\%$. For the NOT gate, the difference in Cy5 signal between the blocked and unblocked cases is $58\% - 32\% = 26\%$. These two percentages are similar, indicating that the leak

whereby the **x** walker can step onto the **y** track does not significantly affect the gate's performance.

4.5 Discussion

There are two ways to interpret the output signals from the NOT and NOR gates. The first is to require a correct output from as many individual tiles as possible. In this case, the Cy5 signal should decrease very little when the track is blocked, indicating that few **1** walkers have reached the end of the track. The percent decrease in Cy5 signal gives a measure of the gate's performance. The second is to require a correct output from the ensemble fluorescence measurement of many tiles in solution, where the measurement of performance is whether the Cy5 signals from the blocked and unblocked cases are clearly distinguishable from one another. In this case, the percent decrease in Cy5 signal becomes less important, as the gate's performance is measured by the ratio of percent decrease in Cy5 signal between the blocked and unblocked cases.

The application determines whether the output should be interpreted from each individual tile or from an ensemble measurement. If DNA walker circuits were used to perform an *in vivo* function such as molecular diagnostics or logic-gated drug synthesis, it would be imperative that the circuit on each individual tile operates correctly. In these applications, there is an undesirable effect if there are errors on more than a few tiles. If the circuit was only needed to evaluate the truth value of a propositional formula, an ensemble output would be suitable.

Regardless of whether the output is taken for each individual tile or as an ensemble measurement, the results for the NOT and NOR gate suggest that DNA walker circuits are only reliable over shorter timescales. When the track is blocked, there is a small percent decrease in Cy5 signal over 300 minutes, but a larger percent decrease over

the longer 700 minute timescale. While the correct operation of each individual circuit predominates over shorter timescales, there are a number of slow reaction leaks that allow errors to accumulate over longer time periods. In addition, the ratio of percent decrease in Cy5 signal between the blocked and unblocked cases gets smaller as run time increases. For both the NOT and NOR gates, there was a large ratio at 300 minutes, indicating that the blocked and unblocked Cy5 signals were clearly distinguishable from one another. As time increased to 700 minutes, the ratio grew smaller, making it more difficult to distinguish between the Cy5 signals for the blocked and unblocked cases.

A notable feature in the results for both the NOT and NOR gates is that there is a decrease in Cy5 signal very soon after the enzyme is added. The results in [100–102] all show a delay between the time when the enzyme is added to solution and the time when the signal from the fluorophore at the end of the track begins to decrease. In the data presented for the NOT and NOR gate, the shape of the Cy5 signal suggests there is a rate limiting process for the walker to reach the end of the track, and that the stepping rate is fast compared to the rate of this process. For both the NOT and NOR gates, the shape of the TEX615 signal in the unblocked case suggests a possible explanation: the rate of stepping between the second junction anchorage and the final anchorage is slow, causing the walker to diffuse on the first six anchorages of the track before it steps to the final anchorage. If the rate of stepping between the other anchorages is fast by comparison, then the Cy5 signal shows the rate of this single process.

While this is a possible explanation, further work is needed to fully characterise the dynamics of walker stepping. This illustrates an important challenge, namely that it is difficult to characterise the rates of stepping down a non-uniform track by using fluroescence spectrophotometry alone. Further advances in single molecule imaging and high speed atomic force microscopy are needed for direct observation of walker

stepping over long timescales. These advances would allow for better identification of the sources of error and facilitate better track design. In the absence of these techniques, however, several possible leaks can still be investigated. The following chapter tests and quantifies the error due to overstepping, tile flexibility, and MCEs caused by the extended reach of the long junction anchorages.

Chapter 5

Sources of Error in DNA Walker Circuits

The previous chapter demonstrates how to build and measure NOT and NOR gates that operate by using DNA walker blocking interactions. However, for both of these gates, there is a sizeable percentage of tiles in the ensemble measurement that output an error. These error tiles have a walker that reaches the end of its track even though it should have been blocked by another walker. This chapter explains and quantifies several different sources of these errors in order to pinpoint the challenges that would have to be overcome to build highly reliable DNA walker circuits.

An error is said to occur when a walker is able to reach the end of a blocked track, meaning the blocking mechanism has failed to halt the walker's movement. This failure can occur due to one or more of the following reasons:

- intermolecular transfer between origami tiles, where a blocked walker steps onto the final anchorage of the track on another origami tile in the solution,
- incorrect loading of the blocking walker or the blocking walker's track,
- overstepping, whereby a walker can step multiple anchorages forward at once,

- flexibility of the tile, a type of overstepping where a blocked walker is brought into close proximity to the final anchorage when the tile flexes,
- extended reach of the long junction anchorages, a type of overstepping whereby the walker can step onto the junction anchorage from several anchorages away.

Intermolecular transfer was quantified in [100] where it was found to occur at a negligible rate at the 0.06μM concentration of origami tiles used in these experiments. In addition, the loading data shown in Section 3.3 suggests that the blocking walker loads reliably. Therefore, it is unlikely that either of these sources cause a significant percentage of the errors observed. Overstepping (Section 5.1), tile flexibility (Section 5.2), and the reach of the junction anchorages (Section 5.3) are investigated and quantified in this chapter. While this list of possible errors is not exhaustive, it shows progress towards fully understanding and characterising the kinetics of DNA walker stepping.

5.1 Overstepping

There is a distance of about 6nm between two consecutive anchorages in the tracks used in the NOT and NOR gate [100]. The walker-anchorage duplex that extends from the origami tile is a double helix consisting of 26 base pairs (bp), and hence it is about (26bp)(0.33nm/bp) = 8.6nm in length [98]. A single-stranded anchorage is (26bp)(0.68nm/bp) = 18 nm in length [22]. This means that the exposed toehold of the walker in the walker-anchorage duplex can reach the 5' end of an anchorage two 6nm spaces away. This section investigates the probability with which the walker "oversteps" the next anchorage in its track to bind instead to the anchorage two spaces away. The probability of overstepping is determined by using a binary decision experiment, where the walker is presented with the option to take one step or two steps.

To measure how often a walker oversteps, the walker is modified with an Iowa Black dark quencher on its 5' end. After the walker is loaded onto its starting position on the tile, it can step onto an anchorage one space (6nm) away or onto another anchorage located two spaces (12nm) away (Figure 5.1). Each of the two anchorages that the walker can step onto has a single-nucleotide mismatch in the enzyme's restriction site to stop the walker's movement once it has stepped onto the anchorage. A fluorophore is tethered near the "one step" anchorage and another fluorophore with different excitation/emission wavelengths is tethered near the "two steps" anchorage. The relative quenching of these fluorophores shows how often the walker will take two steps instead of one.

The fluorophores chosen were Cy5 and TEX615, the same pair that was used in the previous chapter. This experiment requires comparing Cy5 quenching to TEX615 quenching, but the two fluorophores may have different rates of photobleaching. To correct for any bias, the experiment was run twice: once with TEX615 tethered near the one step anchorage and Cy5 tethered near the two steps anchorage (Figure 5.1, top) and then again with the positions of the fluorophores exchanged (Figure 5.1, bottom). For each anchorage, the corrected signal at time t was found by averaging the signals from when TEX615 was bound near the anchorage and when Cy5 was bound near the anchorage:

$$|S(t)| = \frac{S_{TEX}(t) + S_{Cy5}(t)}{2}. \tag{5.1}$$

This corrected signal eliminates the bias caused by photobleaching, and the corrected signals for each anchorage can be compared to determine how often the walker oversteps.

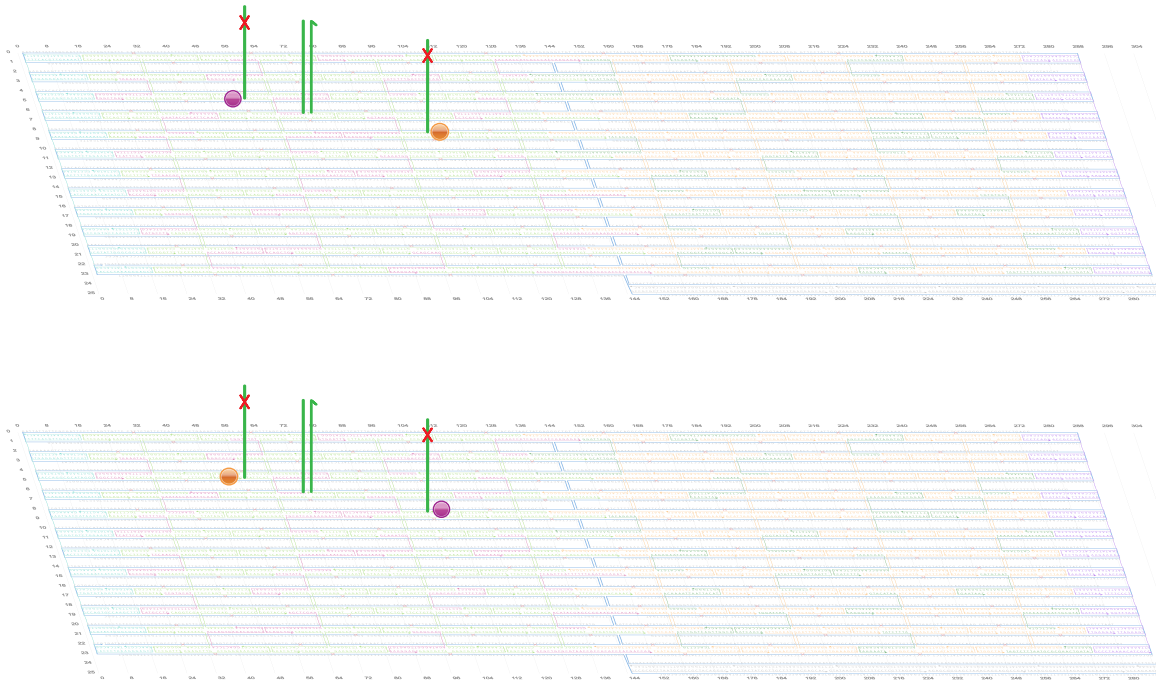


Figure 5.1: Binary decision experiment for measuring how often a walker will overstep. A walker is loaded into a starting anchorage, where it can step onto an anchorage one space away (leftmost anchorage) or onto an anchorage two spaces away (rightmost anchorage). TEX615 (purple circle) and Cy5 (orange circle) are tethered near each of these two anchorages. When a walker binds to the anchorage, its Iowa Black dark quencher will quench the fluorophore. Each of the anchorages has a restriction site mismatch (red cross) to stop the walker's movement once bound. To correct for the Iowa Black quenching spectrum, the experiment was run with TEX615 tethered near the one space anchorage and Cy5 tethered near the two space anchorage (top), and then repeated with the fluorophores exchanged (bottom). The signals were averaged as shown in Equation 5.1.

Figure 5.2 shows the fluorophore signals corrected according to Equation 5.1. The signal from the fluorophore tethered near the one step anchorage shows a very sharp decrease between 0 and 20 minutes after the enzyme was added to solution, followed by a much more gradual decrease. The signal from the fluorophore tethered near the two steps anchorage shows a much more gradual decrease over the first 600 minutes.

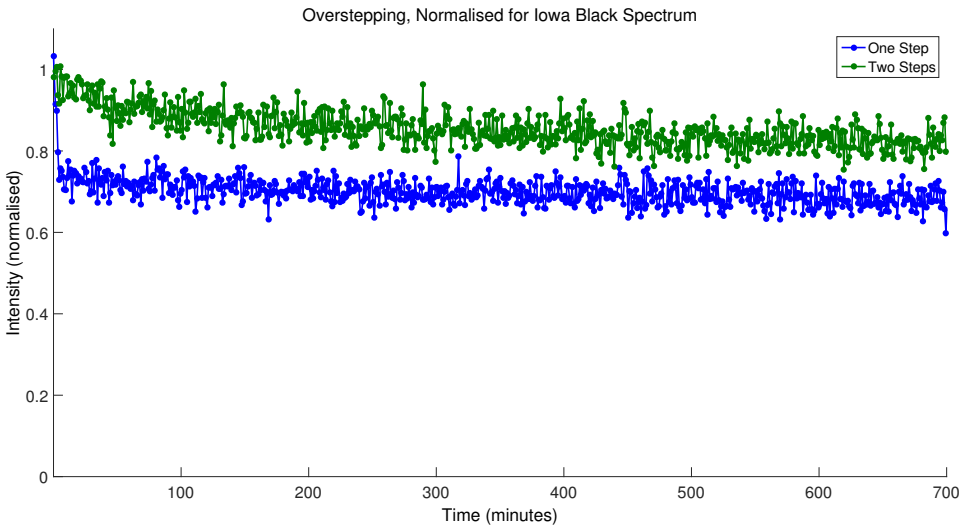


Figure 5.2: Signals from the fluorophores tethered near the one space anchorage and the two space anchorage, normalised as shown in Equation 5.1. The enzyme is added at $t = 0$ and stepping was measured over a timescale of 700 minutes. The signal from the fluorophore tethered near the one step anchorage (blue trace) decreases very sharply in the first few minutes of the experiment, and then reaches a steady state. The signal from the fluorophore tethered near the two step anchorage (green trace) decreases much more gradually, reaching a steady state after about 600 minutes.

The data in Figure 5.2 shows that the system behaves differently over short and long timescales. The very sharp decrease in signal from the fluorophore tethered near the one step anchorage shows that the process of the walker taking one step dominates over short timescales: the one step signal when $t \leq 30$ minutes has decreased by a large margin while the signal from the fluorophore near the two steps anchorage has decreased very little. Over longer timescales, the decrease in signal from the two steps fluorophore becomes significant while the signal from the one step fluorophore changes very little after the initial decrease. This suggests that the probability of overstepping is negligible over short timescales, but can become significant over longer timescales.

This experiment has an important implication for track design: leaving out one anchorage in a track to create a 12nm gap is not sufficient to block a walker. Over longer timescales, the walker will be able to overstep the gap to reach the end of the track. This is why the gaps created by the blocking walkers in the NOT and NOR

gate in the previous chapter were made to be large. A large gap in the track is needed for a walker to remain blocked over long timescales.

While overstepping becomes significant when the system is allowed to run for a long time, the fact that single-step behaviour dominates over short timescales indicates that overstepping is a rare event on a track with 6nm between each anchorage. Overstepping is only likely to occur if the walker has stalled for a long period of time with no adjacent anchorage available. Hence, it is unlikely that a walker can step down its track more rapidly than expected by overstepping anchorages along the way.

5.2 Tile Flexibility

DNA walker circuits perform localised computation, where interactions between walkers are controlled by geometric constraints imposed by the origami tile; species that are spaced far away from each other on the tile should not interact. For example, the previous chapter describes a blocking mechanism used in the NOT and NOR gates that works by creating a gap in the track that a walker cannot traverse. The walker is confined to the portion of its track before the junction and it cannot reach the final anchorage due to the geometric constraint imposed by the tile. Errors may occur when the tile flexes, bringing species that should have been kept far apart into close proximity with one another. In the case of DNA walker circuits, tile flexing may allow a walker to circumvent a blocked junction to reach the end of its track.

The experimental design in Figure 5.3 tests whether flexibility in the tile will allow a walker to traverse large gaps on the origami tile. The experiment uses two anchorages: a start anchorage and an end anchorage. As in the previous chapter, the walker's movement is reported by signals from a TEX615 fluorophore located near the start anchorage and a Cy5 fluorophore located near the end anchorage. The end anchorage is located either a half-tile or full tile away from the start anchorage. For

the half-tile gap, the start and end anchorages are placed in the same locations as the start and end anchorages of the **1** track in the NOT and NOR gates described in the previous chapter. Whether or not the walker can cross the gap to step onto the end anchorage gives some intuition for how flexibility can affect the reliability of DNA walker circuits.

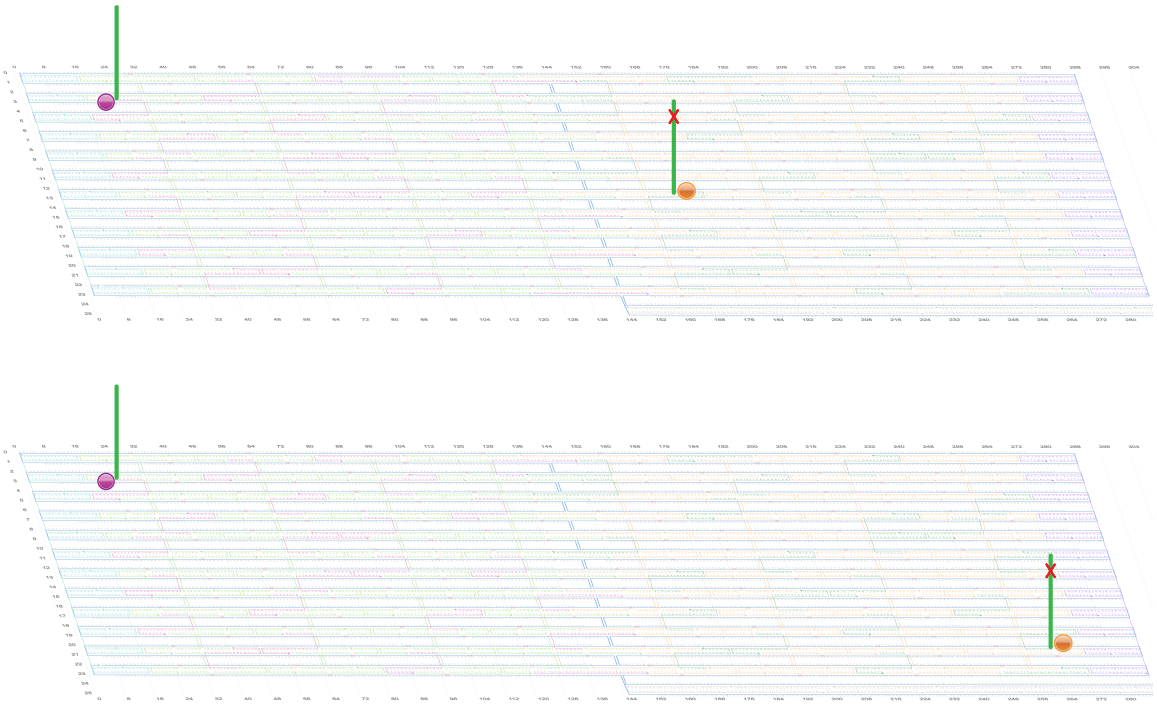


Figure 5.3: Experimental design that tests whether tile flexibility allows a DNA walker to traverse large gaps between anchorages. A TEX615 fluorophore (purple circle) is tethered near a start anchorage, and a Cy5 fluorophore (orange circle) is tethered near an end anchorage placed either a half-tile (top) or full tile (bottom) away. The end anchorage has a restriction site mismatch to stop the walker's movement once bound.

Figure 5.4 shows the fluorescence signals for a walker attempting to traverse the gap when the end anchorage is located a half-tile or a full tile away. The signals for both cases are nearly identical: there is very little decrease in the Cy5 signal, and a very slow increase in the TEX615 signal. Immediately after the enzyme is added, there is a slight decrease in Cy5 signal of about 5%. When the uncover strands and

enzyme are added to the solution, there is a small dilution that can weaken the signal. A decrease in signal of 5% is within the expected range of this dilution effect, and this decrease in Cy5 signal is not likely to be caused by the walker reaching the end of the track. Following the initial signal decrease, the Cy5 signal remains constant, indicating that no significant amount of walker is able to step onto the end anchorage.

For both the half-tile track and full tile trials, the TEX615 signal shows a gradual but steady increase. While the increase is small, it still more than expected given the constant Cy5 signal. A possible explanation is that there is a very slow leak whereby the exposed toehold on the walker can weakly interact with other strands in solution, displacing the walker from the start anchorage. If the walker cannot reach the end anchorage, it will remain hybridised to the start anchorage with its toehold exposed. It is possible that there is a weak interaction between the walker and another oligonucleotide in the solution that displaces the walker from the tile.

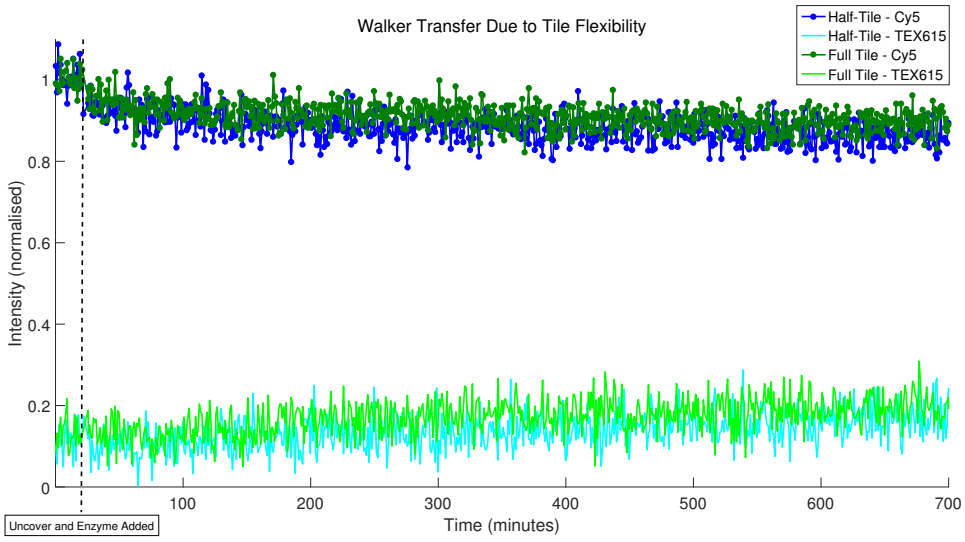


Figure 5.4: Fluorescence signals from Cy5 and TEX615 for a walker traversing a half-tile gap (shown in blue) and a full tile gap (shown in green). The signals for the half-tile and full tile trials are nearly identical. The very slight decrease in Cy5 signal after the enzyme was added is likely due to a weakening signal from dilution effects rather than from walker movement. The Cy5 signal maintains a steady state, indicating that no dectectable amount of walker reaches the end anchorage. The TEX615 signal has a steady increase throughout the experiment. This is likely due to the slow unbinding of the walker from its start anchorage.

The fact that there is little change in the TEX615 and Cy5 signals in Figure 5.4 provides evidence that flexibility is not likely the cause of errors observed in the NOT and NOR gate. One possible reason is that tile flexibility and walker stepping happen on different timescales. The tile flexing may bring the walker into close proximity to its end anchorage, but the effect is fleeting and there is not enough time for the walker to take a step. This explanation is speculative, however, and a more thorough characterisation of the flexibility of an origami tile would require more investigation using other biophysical techniques.

While this experiment demonstrated very little stepping between the start and end anchorages, it does provide a useful control for the logic gates from the previous chapter. As previously mentioned, the start and end anchorages in the half-tile flexibility test were in the same locations as the start and end anchorages in the **1** track in the previous chapter. This experiment demonstrates that the track is necessary for the directed motion of a walker; the walker cannot step directly from the start anchorage to the track's end anchorage.

5.3 Extended Reach of the Junction Anchorages

As described in Chapter 4, the blocking mechanism used in the NOT and NOR gates requires two long junction anchorages that each have a toehold for the **1** walker at the 5' end. The extended length of the anchorages means that the walker may be able to hybridise to them from several anchorages away, causing the walker to arrive at the junction more quickly than if it had stepped down every anchorage along the way. To account for this extended reach, the toehold for the **1** walker was shortened from eight nucleotides to two nucleotides on both junction anchorages. This lowers the rate with which the **1** walker will bind to the junction anchorage, compensating for the extended reach. However, it is important to test whether the reach of the

junction anchorages can still cause an error.

To test whether the extended reach of the junction anchorages allows the walker to hybridise to them from several anchorages away, a tile was assembled with only the **1** track from the previous chapter. As shown in Figure 5.5, the reach of the junction anchorages was determined by whether a walker could step down a track that was missing:

(a) anchorages 5 and 6,

(b) anchorages 4, 5, and 6,

(c) anchorages 3, 4, 5, and 6,

(d) anchorages 2, 3, 4, 5, and 6.

Creating a larger and larger gap in the track tests whether the junction anchorages are long enough to help the walker cross each gap, determining how far they can reach.

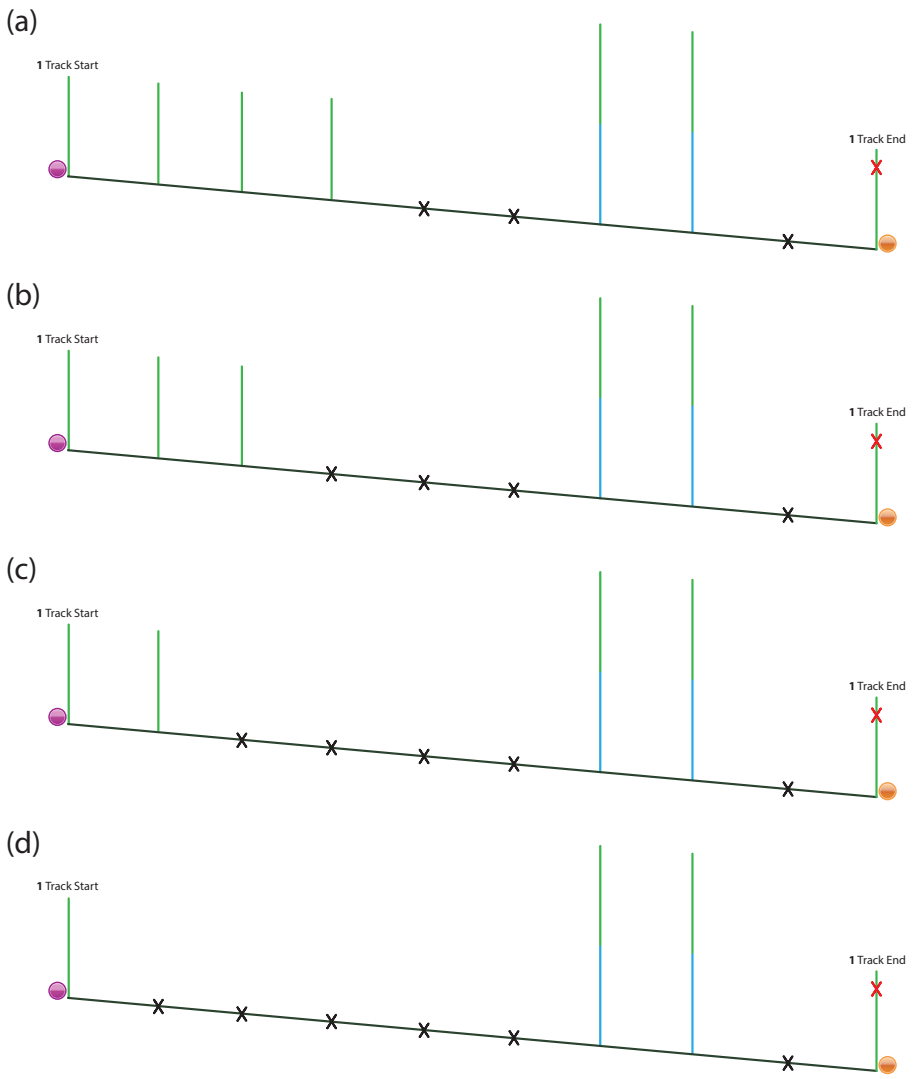


Figure 5.5: Testing the reach of the junction anchorages. A gap is created by leaving out anchorages from the track (black crosses). If the signal from the Cy5 fluorophore (orange circle) decreases, it indicates that the reach of the junction anchorages was long enough to enable the walker cross the gap.

Figure 5.6 shows the TEX615 and Cy5 signals from each of the four trials shown in Figure 5.5. The Cy5 signals for (b)-(d) are nearly identical and all show a gradual decrease over time. However, the Cy5 signal for (a) shows a sharper decrease than the other three, indicating that the walker reaches the end of the track at a faster rate in (a) than in (b)-(d). This data suggests that the junction anchorages can hybridise to a walker located three anchorages (about 18nm) away. However, if the walker is

farther away, the rate at which the junction anchorage hybridises to the walker is slow when compared to (a).

While case (d) shows that a significant number of walkers are able to hybridise to the junction anchorage and reach the end of the track, the rate at which the walkers reach the end of the track is still much slower than the rate for the unblocked track in the NOT and NOR gate. The data from the overstepping experiment in Section 5.1 shows that over a short timescale, the behaviour of stepping onto the next anchorage in the track dominates. Hence, the leak whereby the walker forgoes stepping onto an adjacent anchorage 6nm away to instead step onto a junction anchorage that is farther away is not expected to contribute significantly to errors in DNA walker circuits.

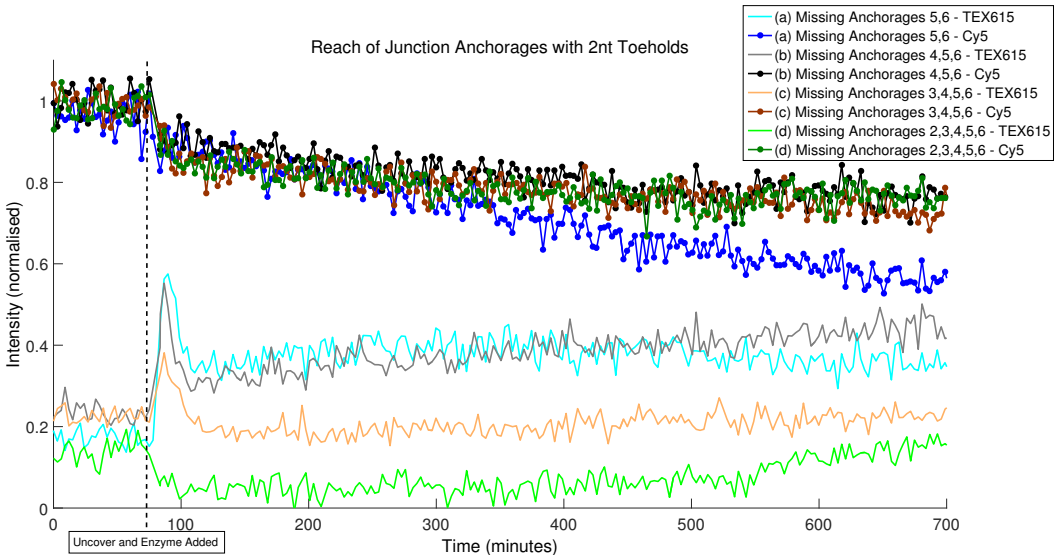


Figure 5.6: Signals from TEX615 and Cy5 for cases (a)-(d) described above. There is a significant decrease in Cy5 signal for case (a) while the decrease in Cy5 signal for cases (b)-(d) is more gradual, suggesting that the junction anchorage has difficulty reaching a walker that is more than three track anchorages away. The TEX615 signal rises and then decreases again, which is characteristic of a walker diffusing on the track anchorages that precede the junction before it eventually reaches the final anchorage in the track.

5.4 Discussion

The experiments in this chapter represent a step towards investigating and quantifying the different sources of error that may arise in DNA walker circuits. While the aim of this investigation was to identify a main leak reaction that was responsible for most of the error observed in the NOT and NOR gates from the previous chapter, a principal source of error was not discovered here. The results suggest that none of these three possible error sources are the main cause of error that arises in the NOT and NOR gates. Each of these sources is likely only responsible for a small percentage of the overall error observed in a circuit.

It is possible that a leak reaction exists that was not considered here, and that this reaction is responsible for the majority of the error observed in DNA walker circuits. However, it is also possible that there is no main source of error, and that the error observed is the cumulative effect of many minor leak reactions. Determining which of these is the case is difficult, as the list of possible errors investigated in this chapter is not exhaustive and there are many more sources of error that can be investigated. However, a complete characterisation of possible error sources would be a substantial undertaking and is beyond the scope of this thesis. Optimisation of the system would be a worthwhile topic of future work.

While the evidence presented here does not suggest that any of these three sources contributes significantly to overall error in the DNA walker circuits, it does provide good justification for the use of the blocking mechanism and the DNA walker mechanism in constructing these circuits. If the DNA walkers were prone to overstepping with high probability, or if the extended reach of the junction anchorages was likely to greatly increase the rate at which a walker could reach the end of the track, there may be cause for resignation that these circuits could never operate with high reliability. However, this is not what the data suggests: these experiments show that the DNA walker mechanism and the blocking mechanism both operate effectively. This

motivates future work in optimising the system, and suggests that these circuits may be able to operate with high reliability if further improvements were made.

Chapter 6

Conclusions and Future Work

The work presented in this thesis demonstrates how the blocking interactions between DNA walkers can be used as primitive operations to evaluate propositional formulae. Viewing DNA walker circuits as a distributed system provides a natural way in which to model these circuits, enabling software to automatically design them using only the formula that the circuit should evaluate and an error tolerance. Some circuits can be simplified by using blocking logic, which is shown to be both sound and complete, and therefore provides a set of rules for finding equivalent circuits. These DNA walker circuits can be implemented experimentally, which is demonstrated by using a NOT and NOR gate as proof of principle. The output of these gates is determined by using fluorescent spectrophotometry to track the movement of the designated output walker as it steps down the track. These logic gates can be prone to error, but many possible sources of error can be investigated and quantified in the pursuit of making DNA walker circuits a reliable and efficient technology. This chapter describes the conclusions that can be drawn from this work, as well as some more speculative ideas about the future of this technology and its applications.

A limitation of the formal system $\mathcal{L}$ described in Chapter 2 is that it only describes circuit designs where a walker can block once. It cannot capture the behaviour where

a walker can block a track and continue stepping to block additional tracks. For any circuit that uses a walker that can block two tracks, there is an equivalent circuit in $\overline{\mathcal{L}}$ that uses two separate tracks, each associated with the same logical proposition, in place of the walker that blocks twice. A useful extension of blocking logic would be incorporating multiple blocking actions from a single walker, as the mechanism used in Chapter 4 has this feature.

This limitation of $\mathcal{L}$ comes from the fact that the syntax encodes a tree rather than a more general graph. A NOR gate can be encoded in the syntax of $\mathcal{L}$ as $x(y1)$, where the output walker is the **1** walker. It is the tree's root, and the **x** and **y** walkers are its children. Applying rule **R2** to write $x(y1) \equiv y(x1)$ is just a fact about trees, stating that it does not matter which order the children are in. To represent a circuit where walkers can block more than once, a node must have multiple parents which is not allowed in a tree data structure. In this case, the circuit must be encoded as a directed graph. However, the syntax of $\mathcal{L}$ was not designed to represent this data structure, and would therefore have to be modified. Creating this extension to $\mathcal{L}$ would be valuable tool for identifying additional ways in which circuits could be simplified by introducing multiple blocking actions by a single walker.

The biggest challenge for creating efficient, reliable, and scalable DNA walker circuits is eliminating as much error as possible. When a mechanism involves multiple processes, each with a probability of error, these sources of error compound to create a significant overall probability of an error occurring in the system. The assembly and operation of a DNA walker circuit is, at its most fundamental level, a chemical reaction. Chemical reactions rarely have 100% yield. The challenge is getting each constituent process to function with as close to 100% yield as possible.

To reduce error, the process of DNA walker stepping must first be fully understood. This is a challenge due to the limited ways in which a walker's movement can be measured. In this thesis, a walker's movement was measured by modifying

the walker with a quencher and determining when the quencher reduced the signal from fluorophores on the origami tile. This only provides information about what happens at the beginning and end of the track, and interesting behaviour may be difficult to infer from the ensemble measurement. Valuable future work in the DNA walker field would be perfecting the use of single molecule imaging for visualising a walker's behaviour. This was done using high speed atomic force microscopy in [102], but only for a few steps. Work is currently being done to track a walker's movement using DNA PAINT [44, 46], and this would be an invaluable contribution to the field.

Despite the inherent difficulties with eliminating error and the challenge of inferring a walker's movements, the experiments in Chapter 5 demonstrate that this may be an achievable goal. The sources of error investigated did not suggest that a particular source was primarily responsible for a majority of the error. While such a source of error may exist that was not investigated or quantified in Chapter 5, it appears more likely that the overall error is an accumulation from many small sources. Subtle modifications to the DNA mechanism may make it possible to reduce or eliminate the error contributions from some of these sources. Making DNA walker circuits operate with high efficiency and low error is a challenge in optimisation engineering for future work; the scope of this thesis is to demonstrate proof of principle for experimental implementation, together with corresponding theoretical tools.

While eliminating error remains a challenge, it is worthwhile to predict what DNA walker circuits could be used for if the probability of error was sufficiently low. One possible application is using DNA walker circuits for the controlled, stepwise, logic-gated synthesis of chemical reactants (Figure 6.1). One reactant is tethered to the output walker while the other reactant is tethered to the end of the track. The walker is able to pull one reactant into the other if the gate evaluates to True, facilitating a chemical reaction. A circuit that operates with minimal error is important in this application, as a missed-chance error or leak reaction would cause a false positive,

enabling a reaction to proceed that should have been prevented.

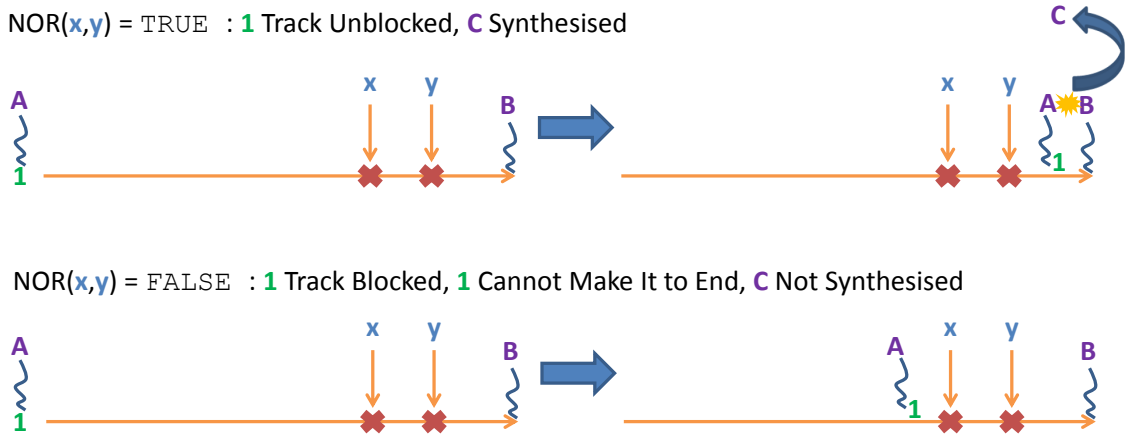


Figure 6.1: A possible application of DNA walker circuits to produce controlled, logic-gated chemical reactions. Chemical reactants A and B can react to form C if they are brought into close proximity. By tethering A to the **1** walker and B to the end of the track, the reactants can only come into contact with one another if $\mathsf{NOR}(x, y) = \text{True}$ which allows the **1** walker to reach the end of the track.

Another interesting application would be the creation of a predator-prey interaction between DNA walkers. If a tile were assembled with an anchorage domain on the 5' end of every staple, two DNA walkers would be able to freely step on this "field of anchorages". Creating a mechanism where one walker could irreversibly destroy the other would create a very simple predator-prey interaction between the walkers. As the walkers step on the field of anchorages, each anchorage that a walker steps on will be shortened by the action of the nicking enzyme. Interesting behaviour may arise as the number of intact anchorages on the tile gets smaller and smaller, leading to a state where the two walkers are separated by regions of cut anchorages. The average time that it takes for one walker to destroy another could be measured, and patterns may emerge in the walkers' behaviour.

Appendices

Appendix A

Axioms and Inference Rules of Propositional Logic

The following appendix shows the axioms and inference rules for propositional logic system introduced in Section 2.3. Attention is restricted to logical operators of arity less than or equal to two; as well as allowing for clear presentation, this is also the relevant case for DNA walker circuits. Propositions are indicated by lower case characters, and the usual characters from propositional logic are used to represent logical connectives: AND ($\wedge$), OR ($\vee$), NOT ($\neg$), and IMPLIES ($\Rightarrow$). Boolean values are given by True and False, and p, q, and r are propositional variables that can be True or False.

A.1 Axioms

1. Commutativity

$$p \wedge q \equiv q \wedge p$$

$$p \vee q \equiv q \vee p$$

$$p = q \equiv q = p$$

2. Associativity

$$p \wedge (q \wedge r) \equiv (p \wedge q) \wedge r$$

$$p \vee (q \vee r) \equiv (p \vee q) \vee r$$

3. Distributive Rule

$$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$$

$$p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$$

4. De Morgan's Laws

$$\neg(p \wedge q) \equiv \neg p \vee \neg q$$

$$\neg(p \vee q) \equiv \neg p \wedge \neg q$$

5. Double-Negatives

$$\neg\neg p \equiv p$$

6. Tautology

$$p \vee \neg p \equiv \text{True}$$

7. Contradiction

$$p \wedge \neg p \equiv \text{False}$$

8. Implication

$$p \Rightarrow q \equiv \neg p \vee q$$

9. Equivalence

$$(p \equiv q) \equiv (p \Rightarrow q) \wedge (q \Rightarrow p)$$

10. Simplification of OR

$$p \vee p \equiv p$$

$$p \vee \text{True} \equiv \text{True}$$

$$p \vee \text{False} \equiv p$$

$$p \vee (p \wedge q) \equiv p$$

11. Simplification of AND

$$p \wedge p \equiv p$$

$$p \wedge \text{True} \equiv p$$

$$p \wedge \text{False} \equiv \text{False}$$

$$p \wedge (p \vee q) \equiv p$$

12. Identity

$$p \equiv p$$

A.2 Inference Rules

1. Transitive Property

$$\frac{p \equiv q,\ q \equiv r}{p \equiv r}$$

2. Substitution

$$\frac{p \equiv q}{f(p) \equiv f(q),\ f(q) \equiv f(p)}$$

3. Conditional Substitution

$$\frac{p_1, p_2, ..., p_n, p_1 \wedge p_2 \wedge ... \wedge p_n \Rightarrow (p_1 \equiv p_2)}{f(p_1) \equiv f(p_2),\ f(p_2) \equiv f(p_1)}$$

4. Modus Ponens

$$\frac{p_1;\ p_1 \Rightarrow p_2}{p_2}$$

Appendix B

Soundness and Completeness of Blocking Logic

This appendix proves soundness and completeness for the blocking logic, and this logic can be used to prove all of the rules stated in Section 2.7. Some topics from Section 2.7, such as the definition of the formal system, are repeated here to make this appendix self-contained. The proof for completeness presented here is a proof by construction, and it uses some techniques common for proving the completeness of other propositional logics (for example, see [72, 97] for completeness proofs of several Hilbert systems).

There is a formal system $\mathcal{L} = (\Sigma, \Omega, \mathcal{T}, \mathcal{A})$ that is defined as follows. Let Σ be a finite set of propositional variables and Ω be a set with the following two elements:

- 1, a constant that represents a variable that is always True,
- $\otimes$, a binary operator with the truth table shown by Table B.1.

In Ω, the 1 constant performs the same function as the 1 track from Chapter 2 and the $\otimes$ operator formalises how a walker on one track can block the track of another walker.

X	Y	$X \otimes Y$
False	False	False
True	False	False
False	True	True
True	True	False

Table B.1: Truth table for the $\otimes$ operator. The expression $X \otimes Y$ signifies that X blocks Y, where this expression evaluates to True if Y reaches the end of the track.

More preliminaries are needed before defining $\mathcal{T}$ and $\mathcal{A}$, but sets Σ and Ω can be used to define a formal language of well-formed formulae. Each well-formed formula corresponds to a DNA walker circuit. The set $\overline{\mathcal{L}}$ is the language of well-formed formulae in $\mathcal{L}$, and can be defined inductively as the smallest set such that:

- $C \in \Sigma \cup \{1\}$ is a well-formed formula,
- if C_1 and C_2 are well-formed formulae, then $C_1 \otimes C_2$ is a well-formed formula.

This definition already specifies the context-free grammar for $\overline{\mathcal{L}}$, but the grammar can also be defined explicitly in Backus-Naur Form:

$$\langle C \rangle ::= 1 \mid b_i \mid \langle C \otimes C \rangle. \tag{B.1}$$

Equation B.1 shows how compound formulae are constructed from multiple blocking interactions. The nonterminals of the grammar are designated with angled brackets while the terminals are 1 constants and propositional variables. To simplify notation, the $\otimes$ symbols are omitted for the remainder of this paper so that C_1C_2 is understood to mean $C_1 \otimes C_2$.

Defining the set $\mathcal{A}$ for $\mathcal{L}$ first requires a definition for semantic entailment. For a formula $C \in \overline{\mathcal{L}}$, let $P_C \subseteq \Sigma$ be the set of all propositional variables in C. The function $v : P_C \to \{\text{True, False}\}$ assigns a truth value to each propositional variable in the formula and is called a valuation of C For a valuation v, the function $v^* : C \to \{\text{True, False}\}$ maps the formula to a truth value using the truth table for $\otimes$. This leads to the following definition.

Definition B.0.1 (semantic entailment)**.** *Let $C \in \overline{\mathcal{L}}$. For a valuation v of C, if $v^*(C) = True$ then v is said to semantically entail C, which is written $v \models C$.*

If C is true for any valuation, it is called a tautology and is written $\models C$. If C is not true for any valuation, it is called a contradiction and written $\not\models C$. With definitions for valuation and semantic entailment, $\mathcal{T}$ and $\mathcal{A}$ can be defined. The set $\mathcal{T}$ is the set of inference rules for $\mathcal{L}$. This system only uses one inference rule, Modus Ponens (MP), which is defined as follows:

$$\text{MP} := \frac{C_1 \mid C_2C_11}{C_2}, \quad C_1, C_2 \in \overline{\mathcal{L}}.$$

This means that C_2 can be deduced from C_1 and C_2C_11. The set $\mathcal{A}$ is the set of all axioms for $\mathcal{L}$. For formulae $C_1, C_2, C_3 \in \overline{\mathcal{L}}$, the axioms are the following three tautologies:

A1. $((C_3C_11)(C_2C_11)1)((C_3C_21)C_11)1$,

A2. $(C_1C_21)C_11$,

A3. $(C_2(C_1(C_21)1)1)(C_11(C_21)1)1$.

The main purpose of $\mathcal{L}$ is deciding which formulae are provable from other formulae. In order for $\mathcal{L}$ to be used in this way, it needs to be made rigorous what "provable" means in the context of blocking logic. A formula is provable if it is syntactically entailed in the logic, which is defined as follows.

Definition B.0.2 (proof and syntactic entailment)**.** *Let $X \subseteq \overline{\mathcal{L}}$ be a set of formulae. A proof is a sequence $\xi_1, \xi_2, ..., \xi_m$ such that for all $k \leq m$ one of the following conditions hold:*

- $\xi_k \in X \cup \mathcal{A}$,
- *there exists indicies $i, j < k$ where $\xi_j = \xi_k\xi_i1$.*

If there is such a sequence, ξ_m is said to be syntactically entailed by X, which is written $X \vdash \xi_m$.

Syntactic entailment is a formal way of saying that a formula has a proof in $\mathcal{L}$. Informally, the above definition of proof and syntactic entailment states that a formula is provable from a set of formulae if they can be linked by the repeated use of MP. Hence, the definition defines ξ_m as provable if it is "one use of MP away" from X and $\xi_1, \xi_2, ..., \xi_{m-1}$. One important property of syntactic entailment is monotonicity, which is implied by the definition. Let $q \in \overline{\mathcal{L}}$ and $X, X' \subset \overline{\mathcal{L}}$ such that $X \subset X'$. If $X \vdash q$, then $X' \vdash q$ by the monotonicity property.

Clearly, if a formula is syntactically entailed by $\mathcal{A}$ then that formula is a tautology as well. However, for blocking logic to be useful in simplifying DNA walker circuits, it is necessary to strengthen this claim so that any formula that has a proof in $\mathcal{L}$ is a tautology. This is soundness for $\mathcal{L}$:

$$\text{For } C \in \overline{\mathcal{L}}, \text{ if } \vdash C \text{ then } \models C. \qquad \textbf{(Soundness)}$$

The soundness property states that if a formula is syntactically entailed in $\mathcal{L}$, then it also must be semantically entailed in $\mathcal{L}$. This guarantees that proofs in $\mathcal{L}$ will only produce true formulae, and this property is proved in the following theorem.

Theorem B.0.1 (Soundness for $\vdash$)**.** *Let $X \subset \overline{\mathcal{L}}$ be a set of formulae. For a formula q, if $X \vdash q$ then $X \models q$.*

Proof. Suppose $X \vdash q$. Clearly $X \models q$ if $q \in \mathcal{A}$, so assume $q \notin \mathcal{A}$ and proceed by induction on m, the number of steps in the proof from X to q. If $m = 1$, then $q \in X$ so any valuation that satisfies X must also satisfy q. Therefore, $X \models q$ in this case. Now assume that soundness holds for all integers less than m, in particular for ξ_i and $\xi_j = q\xi_i 1$ with $i, j < m$. Then $X \models q$ holds by applying MP. □

Soundness shows that any formula that has a proof is also a tautology, which

shows that $\mathcal{L}$ can produce rules that show two DNA walker circuits have the same truth table. However, it is possible to imagine two DNA walker circuits that have the same truth table, but where there is no proof in $\mathcal{L}$ to show this equivalence. It is important to guarantee that this is not the case. Therefore, it is most useful to show that if two circuits are equivalent, then there is a proof in $\mathcal{L}$ that shows this equivalence. This property is known as completeness of $\mathcal{L}$.

$$\text{For } C \in \overline{\mathcal{L}}, \text{ if } \models C \text{ then } \vdash C. \qquad \textbf{(Completeness)}$$

Completeness states that if two formulae are equivalent in the sense that they have the same truth table, then there is a proof in $\mathcal{L}$ that shows this equivalence.

The proof of completeness is much more involved than the proof of soundness, and requires a number of preliminaries. While there are several possible strategies to prove completeness, the proof presented here is a constructive proof. It relies on the deduction theorem, which shows an important relationship between syntactic entailment and the $\otimes$ operator. Before proving the deduction theorem, we will need the following simple lemma.

Lemma B.0.2. $\vdash CC1$ *for any* $C \in \Sigma \cup \{1\}$.

Proof. $\models CC1$ is clear from the truth table of $\otimes$, and in particular $\models (CC1)C1$. Setting $C = C_1 = C_2 = C_3$ in Axiom A1 and using MP twice gives $\vdash CC1$. □

With this lemma in hand, we can prove the deduction theorem. Note that there is a stronger version of the deduction theorem whereby $X, q \vdash C$ is shown to be both necessary and sufficient condition for $X \vdash Cq1$. However, for the logic developed here, the following version is sufficient.

Theorem B.0.3 (Deduction Theorem)**.** *Let* $q, C \in \overline{\mathcal{L}}$ *and* $X \subset \overline{\mathcal{L}}$*. If* $X, q \vdash C$*, then* $X \vdash Cq1$.

Proof. Proceed by induction on m, the length of the proof for $X, q \vdash C$. If $m = 1$, then $C \in X \cup \mathcal{A} \cup \{q\}$. If $C = q$ then $X \vdash qq1$ by Lemma B.0.2 and monotonicity. If $C \in \mathcal{A} \cup X$, then clearly $X \vdash C$. From Axiom 2, we have $\models (Cq1)C1$, so using soundness and MP gives $X \vdash Cq1$. Now assume the theorem holds for $< m$, in particular for $X, q \vdash \beta$ and $X, q \vdash C\beta 1$. This means that $X \vdash \beta q1$ and $X \vdash (C\beta 1)q1$. From Axiom A1, we can write $\models ((Cq1)(\beta q1)1)((C\beta 1)q1)1$. Using MP twice on Axiom A1 gives $X \vdash Cq1$, as was to be demonstrated. □

The following lemma has eight parts; they are labelled L1 - L8 because they will be referred to often. The results up to this point were proved without needing features common in propositional logics such as double-negatives and contraposition, but these are proven now in L2-L5. They are used to prove L6-L8, which in turn are needed in the completeness proof.

Lemma B.0.4. *For $C_1, C_2, C_3 \in \overline{\mathcal{L}}$, the following are tautologies provable from $\mathcal{A}$:*

L1. $((C_3C_11)C_21)((C_3C_21)C_11)1$,

L2. $C_1(C_111)1$,

L3. $(C_111)C_11$,

L4. $(C_1C_21)((C_21)(C_11)1)1$,

L5. $((C_11)(C_21)1)(C_2C_11)1$,

L6. $(C_2C_11)(C_11)1$,

L7. $((C_2C_111)(C_21)1)C_11$,

L8. $(C_2(C_2(C_11)1)1)(C_2C_11)1$.

Proof. To prove L1, it is enough to prove $\{(C_3C_21)C_11, C_2\} \vdash C_3C_11$ and then use the deduction theorem to give the result. Suppose that $(C_3C_21)C_11$ and C_2 hold, and

observe that $\models (C_2C_11)C_21$ by Axiom A2. Applying MP to Axiom A2 deduces,

$$C_2C_11. \tag{B.2}$$

From Axiom A1, $\models ((C_3C_11)(C_2C_11)1)((C_3C_21)C_11)1$. By hypothesis, $(C_3C_21)C_11$ holds so using MP deduces $(C_3C_11)(C_2C_11)1$. Note that C_2C_11 also holds by Equation B.2, so applying MP again deduces C_3C_11. Applying the deduction theorem proves L1. To prove L2, first note that Axiom A3 gives

$$\models (C_1(C_11(C_11)1)1)(C_111(C_11)1)1. \tag{B.3}$$

L1 shows that $(C_1(C_111(C_11)1)1)(C_11(C_11)1)1$ is provable from Equation B.3, and because $\models (C_11(C_11)1$ holds from Lemma B.0.2, deduce $C_1(C_111(C_11)1)1$ using MP. Equivalently, $C_111(C_11)1 \vdash C_1$. Axiom A2 shows that $\models (C_111(C_1)1)(C_111)1$ which means that $C_111 \vdash C_111(C_1)1$. The $\vdash$ relation is transitive, so it follows that $C_111 \vdash C_1$ which proves L2. For L3, Axiom A3 shows that,

$$\models (C_111(C_1(C_1111)1)1)(C_11(C_1111)1)1.$$

Note that $C_11(C_1111)1$ holds from L2, so deduce $C_111(C_1(C_1111)1)1$ using MP. From Axiom A2, $\models (C_1(C_1111)1)C_11$ so $C_1 \vdash C_111$ which proves the result. To prove L4, suppose $(C_11)(C_21)1$ holds and write $(C_2(C_1(C_21)1)(C_11(C_21)1)1$ using Axiom A3. Applying MP deduces that,

$$C_2(C_1(C_21)1. \tag{B.4}$$

From Axiom A2, we can write $\models (C_1(C_21)1)C_11$. Together with Equation B.4, this means that C_2C_11. Hence, $C_11(C_21)1 \vdash C_2C_11$ which completes the proof of L4. For L5, suppose that C_2C_11 holds and observe that $C_1(C_111)1$ holds from L2, so $C_111 \vdash C_2$. Using L3 shows that $(C_211)C_21$, which means that $C_111 \vdash C_211$, and equivalently $(C_211)(C_111)1$. From L4, $((C_11)(C_21)1)((C_211)(C_111)1)1$ holds. Therefore, $C_2C_11 \vdash (C_11)(C_21)1$ which proves the result. To prove L6, first show that $\{C_1, C_11\} \vdash C_2$.

For C_1, Axiom A2 shows that $\models (C_1(C_21)1)C_11$. Applying MP gives $C_1(C_21)1$. For C_11, Axiom A2 gives $\models ((C_11)(C_21)1)(C_11)1$ therefore $(C_11)(C_21)1$ from MP. From Axiom A3, $\models (C_2(C_1(C_21)1)1)(C_11(C_21)1)1$ so applying MP twice shows that $\{C_1, C_11\} \vdash C_2$. Applying the deduction theorem gives $C_11 \vdash C_2C_11$, which in turn shows $(C_2C_11)(C_11)1$, completing the proof of L6. To prove L7, observe that $\{C_1, C_2C_11\} \vdash C_2$ by MP and apply the deduction theorem to show that,

$$C_1 \vdash C_2(C_2C_11)1. \tag{B.5}$$

Applying the deduction theorem again gives $\vdash (C_2(C_2C_11)1)C_11$. Letting $C_1 = C_2C_11$ in L5 gives $((C_2C_11)(C_21)1)(C_2(C_2C_11)1)1$. Using Equation B.5 shows that $C_1 \vdash (C_2C_11)(C_21)1$ which proves the L7. Finally for L8, assume C_2C_11 and $C_2(C_11)1$. Applying L5 and MP to C_2C_11 gives $(C_11)(C_21)1$ and applying L5 and MP to $C_2(C_11)1$ gives $(C_111)(C_21)1$. By Axiom A3,

$$(C_2(C_11(C_21)1)1)(C_111(C_21)1)1.$$

The formula $(C_111)(C_21)1$ and $(C_11)(C_21)1$ holds, so applying MP twice gives C_2. This means that $\{C_2C_11, C_2(C_11)1\} \vdash C_2$. Applying the deduction theorem twice gives $(C_2(C_2(C_11)1)1)(C_2C_11)1$ which concludes the proof. □

Before proving the main lemma needed in the completeness theorem, it is helpful to introduce some notation. Suppose that formula C has propositional variables $b_1, b_2, ..., b_n$. Depending on the valuation made by v, function v^* is the homomorphic extension of v and determines the resulting truth value of the whole formula. Define a new formula C^v as follows:

$$C^v = \begin{cases} C & \text{if } v^*(C) = \text{True}, \\ C1 & \text{if } v^*(C) = \text{False}. \end{cases} \tag{B.6}$$

Informally, this new function uses the valuation of C to create a new formula that

always evaluates to True. In a similar fashion, v is used to define the following new formulae from all propositional variables b_i in C:

$$b_i^v = \begin{cases} b_i & \text{if } v(b_i) = \text{True}, \\ b_i 1 & \text{if } v(b_i) = \text{False}. \end{cases}$$

For example, if the variable b_i was valued as False by v, the new formula $b_i^v = b_i 1$ will evaluate to True.

The overall strategy of the completeness proof is to first show that the formulae b_i^v syntactically entail C^v for any valuation v. This is shown by Lemma B.0.5. With this result in hand, the completeness proof in Theorem B.0.6 removes the dependence on v and shows that if C is a tautology, then C is syntactically entailed by all b_i regardless of valuation. The first step, however, is proving the following lemma.

Lemma B.0.5. *For circuit C with propositional variables $b_1, b_2, ..., b_n$, for any valuation v of C it holds that $b_1^v, b_2^v, ..., b_n^v \vdash C^v$.*

Proof. Proceed by induction on the structure of C. The syntax of C given in Equation B.1 indicates that there are three cases to check.

Case: $C = 1$.

Applying the deduction theorem to $1 \vdash 1$ gives $\vdash 111$. Applying Modus Ponens gives $\vdash 1$, so $b_1^v, b_2^v, ..., b_n^v \vdash 1$ by monotonicity.

Case: $C = b_i$.

Here, $b_1^v, b_2^v, ..., b_n^v \vdash b_i^v$ follows immediately from $b_i^v \vdash b_i^v$ and monotonicity.

Case: $C = C_1 C_2$.

The induction hypothesis is that $\left\{b_1^v, b_2^v, ..., b_j^v\right\} \vdash C_1^v$ and $\left\{b_{j+1}^v, b_{j+2}^v, ..., b_n^v\right\} \vdash C_2^v$. Note that $\left\{b_1^v, b_2^v, ..., b_j^v\right\}$ and $\left\{b_{j+1}^v, b_{j+2}^v, ..., b_n^v\right\}$ are not necessarily disjoint. By mono-

tonicity,

$$b_1^v, b_2^v, ..., b_n^v \vdash C_1^v,$$

$$b_1^v, b_2^v, ..., b_n^v \vdash C_2^v.$$

Proceed by exhaustion on the following cases.

Subcase: $v^*(C_1) = v^*(C_2) = \text{False}$.

The definition in Equation B.6 shows that,

$$C_1^v = C_1 1,$$

$$C_2^v = C_2 1,$$

$$C^v = C1 = C_1 C_2 1.$$

Lemma L6 shows that $\models (C_1 C_2 1)(C_2 1)1$. Applying MP gives,

$$\frac{C_2 1 \mid (C_1 C_2 1)(C_2 1)1}{C_1 C_2 1 = C^v}.$$

Because $\{b_1^v, b_2^v, ..., b_n^v\} \vdash C_2^v$ by the induction hypothesis, $\{b_1^v, b_2^v, ..., b_n^v\} \vdash C^v$ by definition of $\vdash$.

Subcase: $v^*(C_1) = \text{False},\ v^*(C_2) = \text{True}$.

From Equation B.6,

$$C_1^v = C_1 1,$$

$$C_2^v = C_2,$$

$$C^v = C = C_1 C_2.$$

Using L7 shows that $\models ((C_1 C_2 11)(C_1 1)1)C_2 1$ so applying MP three times deduces $C_1 C_2 = C^v$. Therefore, $\{b_1^v, b_2^v, ..., b_n^v\} \vdash C^v$ from the induction hypothesis and the definition of $\vdash$.

Subcase: $v^*(C_1) = \text{True},\ v^*(C_2) = \text{False}$.

Here, $C_1^v = C_1$ and $C_2^v = C_2 1$ with $C^v = C1 = C_1C_21$. L7 shows that $\models (C_1C_21)(C_21)1$, so applying Modus Ponens deduces $C_1C_21 = C^v$. Therefore, $\{b_1^v, b_2^v, ..., b_n^v\} \vdash C^v$ by the induction hypothesis and the definition of $\vdash$.

Subcase: $v^*(C_1) = v^*(C_2) = \text{True}$.

In this case, $C_1^v = C_1$ and $C_2^v = C_2$ with $C^v = C1 = C_1C_21$. Note that $\models (C_1C_21)C_11$ by Axiom A2. Applying MP deduces $C_1C_21 = C^v$ so $\{b_1^v, b_2^v, ..., b_n^v\} \vdash C^v$ by the induction hypothesis and definition of $\vdash$.

By the syntax for C defined in Equation B.1, this concludes all possible cases. Therefore, it holds that $b_1^v, b_2^v, ..., b_n^v \vdash C^v$, as was to be demonstrated. □

The above preliminaries are enough to prove completeness for $\mathcal{L}$. Lemma B.0.5 shows that C^v is syntactically entailed by formulae $b_1^v, b_2^v, ..., b_n^v$. The proof of completeness works by assuming that C is a tautology and showing that we can iteratively eliminate each b_i^v to show that C is always syntactically entailed in $\mathcal{L}$.

Theorem B.0.6 (Completeness)**.** *If* $\models C$ *then* $\vdash C$.

Proof. Assume $\models C$, which means that $C^v = C$. For any v, it holds that $\{b_1^v, b_2^v, ..., b_n^v\} \vdash C^v$ from Lemma B.0.5, so substitution for C^v gives,

$$b_1^v, b_2^v, ..., b_n^v \vdash C.$$

For each b_i^v, either $b_i^v = b_i$ or $b_i^v = b_i 1$. Pick two valuations v_1, v_2 for C so that,

$$\begin{aligned} v_1(b_1) &= v_2(b_1), \\ v_1(b_2) &= v_2(b_2), \\ &\vdots \\ v_1(b_{n-1}) &= v_2(b_{n-1}), \end{aligned}$$

but for b_n,

$$v_1(b_n) = \text{True}, \;\; v_2(b_n) = \text{False}.$$

First for v_1, there is $b_n^{v_1} = b_n$. Applying the deduction theorem shows that,

$$b_1^{v_1}, b_2^{v_1}, ..., b_{n-1}^{v_1} \vdash C b_n 1. \tag{B.7}$$

Similarly, for v_2 there is $b_n^{v_2} = b_n 1$ and the deduction theorem gives,

$$b_1^{v_2}, b_2^{v_2}, ..., b_{n-1}^{v_2} \vdash C(b_n 1)1. \tag{B.8}$$

Therefore, both Cb_n1 and $C(b_n1)1$ are both provable from $\left\{b_1^v, b_2^v, ..., b_{n-1}^v\right\}$, regardless of whether $v = v_1$ or $v = v_2$. Lemma L8 shows that $\models (C(C(b_n1)1)1)(Cb_n1)1$. Applying Modus Ponens to Equation B.7 gives,

$$b_1^v, b_2^v, ..., b_{n-1}^v \vdash C(C(b_n 1)1)1. \tag{B.9}$$

Applying Modus Ponens again shows that,

$$b_1^v, b_2^v, ..., b_{n-1}^v \vdash C. \tag{B.10}$$

Hence, b_n^v has been eliminated. Repeating the process for each $i \in \{1, 2, ..., n-1\}$ gives $\vdash C$ which completes the proof. □

Appendix C

DNA Sequences

C.1 Minimum Twist Origami Tile

The following staple sequences were used together with the M13mp18 vector to create a DNA origami tile with minimal twist. All sequences are written in the 5' to 3' direction.

Name	**Sequence**
r-1t0g	CGA TCT AAA GTT TTG TCG TCT TTT TGC GCC G
r-1t8e	AGG CTT TTA CGA TAA AAA CCA AAA TTA AGA C
r-1t16e	AAC GCC ATT CAG CTC ATT TTT TAA ATG CAA AT
r-3t0g	TAG TAA ATG AAT TTT CTG TAT GGG GTG AAT TT
r-3t8e	GGT AAT AGC AAC ACT ATC ATA ACA TCA TTG T
r-3t16e	TCC TGT AGT TGT TAA AAT TCG CAT TAA TGC C
r-5t0g	AAA CAA CTT TCA ACA GTT TCA GCA ATT GTA T
r-5t8e	CCA ATA CTC GCC AAA AGG AAT TAA ACT GGC T
r-5t16e	TGA GCG AGA GCA AAT ATT TAA ATG ATC TAC A
r-7t0g	AAT AGA AAG GAA CAA CTA AAG GAC TCC AAA A
r-7t8e	TGA ATC CCG AAT ACC ACA TTC AAC GAA GAA AA
r-7t16e	GGA ACA AAA GAA AAG CCC CAA AAG AGT CTG G
r-1t2f	ACA ATG ACC GGT CGC TGA GGC TTC GAT TAT A
r-1t10f	CGA GCT TCA GGT CAG GAT TAG AGG CTA TAT T
r-1t18f	ACC AGG CAG TTG GGA AGG GCG ATC TCA CAA T
r-3t2f	CTT AAA CAC GCT TTT GCG GGA TCA ATA CAC T
r-3t10f	CCC GAA AGC CTT TTG ATA AGA GGT ACA TTT C
r-3t18f	CCA GCT TTG CTA TTA CGC CAG CTA TAG CTG T
r-5t2f	CGG TTT ATA AAG ACA GCA TCG GAA AGG CAC CA

Name	Sequence
r-5t10f	AAG CGG ATC TTA GAG CTT AAT TGG CGA ACG A
r-5t18f	CAG TAT CGG CTG CAA GGC GAT TAG GGT ACC G
r-7t2f	AAA AGG CTT ACA GAG GCT TTG AGT AAA CGG G
r-7t10f	GTC TTT ACA GCT CAA CAT GTT TTA TCA TTC CA
r-7t18f	CCG TGC ATG GTT TTC CCA GTC ACG CAT GCC TG
r-1t2e	GAT ATA TTA ACA ACC ATC GCC CAC CAA GAG AA
r-1t10e	ACT CCA ACA AAG CGA ACC AGA CCG CAG CCT T
r-1t18e	GCG CAA CTA AGC GCC ATT CGC CAA ACA AAC A
r-3t2e	TTA AAG GCG CTT GAT ACC GAT AGC CAG ACG T
r-3t10e	TAA TTG CTA CTT CAA ATA TCG CGC CAG AGG G
r-3t18e	GCC TCT TCC CGG CAC CGC TTC TGG TCT GGC CT
r-5t2e	CAG CAG CGC AGC TTG CTT TCG AGA TTT TGC T
r-5t10e	GCG GAT GGT GCA TCA AAA AGA TTA GAT AGC GT
r-5t18e	GGG GAT GTG CCT CAG GAA GAT CGA TTA AAT G
r-7t2e	GCA ACG GCC CAA AAG GAG CCT TTG GAG TGA G
r-7t10e	AAT GCT GTC CTG ACT ATT ATA GTA TAT TCA T
r-7t18e	AAC GCC AGC TGC CAG TTT GAG GGT CTC CGT G
r-1t4f	CCA AGC GCG CCT GAT AAA TTG TGC CCT GAC G
r-1t12f	TTC ATT TGA CTA ATA GTA GTA GCA GAG AAA GG
r-1t20f	TCC ACA CAT GGG GTG CCT AAT GAT GCC CCA G
r-3t4f	AAA ACA CTG CTC CAT GTT ACT TAA CAA AGC T
r-3t12f	GCA AAT GGC ATA CAG GCA AGG CAA GAG TAA TG
r-3t20f	TTC CTG TGT AAT TGC GTT GCG CTA GAG AGT T
r-5t4f	ACC TAA AAG ACG GTC AAT CAT AAA GAA CCG G
r-5t12f	GTA GAT TTA GCA ATA AAG CCT CAT AGA ACC C
r-5t20f	AGC TCG AAG TCG GGA AAC CTG TCG ACA GCT GA
r-7t4f	TAA AAT ACA ACT TTG AAA GAG GAC TGG CTG AC
r-7t12f	TAT AAC AGC GGT TGT ACC AAA AAG CCT TTA T
r-7t20f	CAG GTC GAG AAT CGG CCA ACG CGG GTG GTT T
r-1t4e	GTA TCA TCG AAA CAA AGT ACA ACC GCC GCC A
r-1t12e	TCA ATT CTG GGC GCG AGC TGA AAT CAT TAC C
r-1t20e	GTA AAG CCA CAT ACG AGC CGG AAG ATC AGA TG
r-3t4e	CGC GAC CTC ATC TTT GAC CCC CAG GCA GGG AG
r-3t12e	CAA TAA ATT CAA TAA CCT GTT TAA GTA CCT T
r-3t20e	ACT CAC ATT GAA ATT GTT ATC CGC GGT GCG G
r-5t4e	GAG GCG CAC GAA AGA GGC AAA AGG TCA CCC T
r-5t12e	CAA AAT TAA GTT TGA CCA TTA GAT CAT TTT T
r-5t20e	GCT TTC CAT TCG TAA TCA TGG TCG GCG AAA G
r-7t4e	AAC TGA CCG TAA TGC CAC TAC GAC GAG GGT A
r-7t12e	AGC TAA ATT TGA TTC CCA ATT CTC TGA ATA T
r-7t20e	GCA TTA ATC TCT AGA GGA TCC CCA GTT GGG T
r-1t6f	AGA AAC ACT TTA ATT TCA ACT TTA CCT CGT TT
r-1t14f	CCG GAG ACG TTC TAG CTG ATA AAT TAA ATT T

Name	Sequence
r-1t22f	CAG GCG AAA AAA TCC CTT ATA AAT AGC GGT C
r-3t6f	GCT CAT TCT TAT GCG ATT TTA AGC GAG GCA T
r-3t14f	TGT AGG TAT AGC TAT TTT TGA GAT GTA AAC G
r-3t22f	GCA GCA AGA GAT AGG GTT GAG TGT AAA GGA GC
r-5t6f	ATA TTC ATC AGT CAG GAC GTT GGT AAT GCA G
r-5t14f	TCA TAT ATC AGG TCA TTG CCT GAA CAG GAA G
r-5t22f	TTG CCC TTA AGA GTC CAC TAT TAG AAC GTG G
r-7t6f	CTT CAT CAT AAT AAA ACG AAC TAA TCA GTT G
r-7t14f	TTC AAC GCA GAG AAT CGA TGA ACG ACC CCG GT
r-7t22f	TTC TTT TCA CGT CAA AGG GCG AAC CCG ATT T
r-1t6e	TGA GAT GGC AGA ACG AGT AGT AAC ATT ACC A
r-1t14e	ATT CAA CCA GTC AAA TCA CCA TCT TTT CGA G
r-1t22e	AAA TCG GCA ATC CTG TTT GAT GGC AAA CCC T
r-3t6e	GAA TTA CCA GTG AAT AAG GCT TGT CGA AAT C
r-3t14e	GGA GAG GGA AGA TTC AAA AGG GTT TAA CAT C
r-3t22e	ATA GCC CGC GGT CCA CGC TGG TTG TGA GCT A
r-5t6e	CAT TAT ACT ACC CAA ATC AAC GTA GCC GGA AC
r-5t14e	AAG GCT ATT TTA AAT GCA ATG CCT AGA ATT AG
r-5t22e	TTT GGA ACC ACC GCC TGG CCC TGC ACT GCC C
r-7t6e	ATC TAC GTA GAG TAA TCT TGA CAG GGA ACC G
r-7t14e	AGC AAA CAA AGG ATA AAA ATT TTG AGC ATA A
r-7t22e	GGA CTC CAA CCA GTG AGA CGG GCA TGC CAG CT
r-1t8f	ACC AGA CGG CAA AAG AAG TTT TGT TTT AAT T
r-1t16f	TTG TTA AAC AAA AAT AAT TCG CGT GCC GGA A
r-1t24h	ACG CTG CGC GTA ACC ACC ACA CCC ACG ACC AG
r-3t8f	AGT AAG AGT AAA ATG TTT AGA CTG AGA GGA AG
r-3t16f	TTA ATA TTC CAG CTT TCA TCA ACC ACT CCA G
r-3t24h	GGG CGC TAG GGC GCT GGC AAG TGT CAA AAG A
r-5t8f	ATA CAT AAG CGG AAT CGT CAT AAC AGA AGC A
r-5t16f	ATT GTA TAT AAC AAC CCG TCG GAT GAC GAC GA
r-5t24h	CGA GAA AGG AAG GGA AGA AAG CGT GTT CCA G
r-7t8f	AGA TTT AGC CTC AAA TGC TTT AAA AAA TCA G
r-7t16f	TGA TAA TCC GGC GGA TTG ACC GTC ATC GTA A
r-7t24h	AGA GCT TGA CGG GGA AAG CCG GCA AGA ACG T
r1t0g	ATT CCA CAG ACA GCC CTC ATA GTT AGC GTA A
r1t8e	GAA TAC CCA CGC AGT ATG TTA GCG AAT TAG A
r1t16e	TAT ATA ACA AGA CAA AGA ACG CGA CAA CGC C
r3t0g	TCG TCA CCA GTA CAA ACT ACA ACG CTC AGT AC
r3t8e	AGA AGG AAT ACA TAA AGG TGG CAA ATT ATC A
r3t16e	CTA CCT TTT ATA TTT TAG TTA ATT AGT AGG GC
r5t0g	AAT AGG AAC CCA TGT ACC GTA ACG GGT TGA T
r5t8e	AAA AGT AAC AAA GAC ACC ACG GAA AAA TAT TG
r5t16e	GTG AAT TTA TTT AAT GGT TTG AAA AAT TCT T

Name	Sequence
r7t0g	CAC CAC CCT CAT TTT CAG GGA TAA TCA CCG T
r7t8e	GCA ATA GCC AAT CAA TAG AAA ATG CGA CAT T
r7t16e	AGC TTA GAA AAT AAG GCG TTA AAA AAA GCC T
r1t2f	GGA TTA GGA GAG GCT GAG ACT CCT GCA TAA CC
r1t10f	TAC AGA GAG TCA AAA ATG AAA ATA GGA AGC AA
r1t18f	TCA AGA AAA AAA GAA GAT GAT GAT TCA GGC T
r3t2f	CAG GCG GAG AAC CTA TTA TTC TGG TCA GAC G
r3t10f	GCA TTA GAA ATC CAA ATA AGA AAA TAT AGA A
r3t18f	TTC ATT TGG AGG CGA ATT ATT CAT GAT TGT T
r5t2f	ATA AGT ATT ATA AAC AGT TAA TGA TAA ATC C
r5t10f	GAA CAA AGT TAC AAA ATA AAC AGC GAG GCG T
r5t18f	GTA CAT AAT TGC TTT GAA TAC CAA AAG GGT TA
r7t2f	ACT CAG GAA CGG GGT CAG TGC CTG CAG TCT C
r7t10f	TCA GAG AGA CGA GCG TCT TTC CAG GAG GTT T
r7t18f	CTT GCT TCC ATC GGG AGA AAC AAT GCA CGT A
r1t2e	AAG TAT TAA TTA GCG GGG TTT TGC CTG TAG C
r1t10e	TGT TTA ACG AAT AAC ATA AAA ACT AAT AAC G
r1t18e	ACC TGA GCA CAA AAT TAA TTA CAT GGT TGG GT
r3t2e	CTA TTT CGT AAG TGC CGT CGA GAA CTG AGT T
r3t10e	TTT ATC CCC GGG AGA ATT AAC TGA AGT TAC C
r3t18e	ATC GCG CAA ATT ACC TTT TTT AAC TGA GAG A
r5t2e	AGT GCC CGA GCC CGG AAT AGG TGT GCA AGC CC
r5t10e	TTT GCC AGT CAG AGG GTA ATT GAG TTT TTA AG
r5t18e	TCG CCT GAA TCA ATA TAT GTG AGA GTC AAT A
r7t2e	AAG TTT TAG GTT TAG TAC CGC CAC TCA GAG C
r7t10e	CAA CGC TAA TAA CCC ACA AGA ATA TGA AAT A
r7t18e	ACC TTT TAT GTA AAT CGT CGC TAA TAG CGA T
r1t4f	GCA TTG ACG AAC CAC CAC CAG AGC GGA GAT TT
r1t12f	GCG CCC AAT TTT CAT CGT AGG AAA GGT GGC A
r1t20f	ATG GCA ATA TCA TAT TCC TGA TTC ATA AAG T
r3t4f	ATT GGC CTC TCA GAG CCA CCA CCA ATG AAA C
r3t12f	GGC TTA TCG CAC TCA TCG AGA ACC CGA CAA A
r3t20f	TGG ATT ATA CAA AGA AAC CAC CAG AAT CAA CA
r5t4f	TCA TTA AAC CTC AGA GCC GCC ACG TAG CGA C
r5t12f	TTT AGC GAC ATT CCA AGA ACG GGG ACG ACA A
r5t20f	GAA CCT ACA AAA GTT TGA GTA ACA TCT AAA A
r7t4f	TGA ATT TAC CGG AAC CAG AGC CAC AGA CTG T
r7t12f	TGA AGC CTA ACC AAT CAA TAA TCA GAA CGC G
r7t20f	AAA CAG AAT TAA ATC CTT TGC CCA ATA GAT T
r1t4e	CGC CAC CAA GGA GGT TGA GGC AGA AAC ATG A
r1t12e	CGT TTT TAT AGC AAG CAA ATC AGC GAT TTT T
r1t20e	GAA TTA TCT CAT CAA TAT AAT CCT TTC AAT T
r3t4e	CCG CCA CCT GAT ATT CAC AAA CAA CCC CCT GC
r3t12e	CAA GTA CCC GGT ATT CTA AGA ACG CCA TAT TA

Name	Sequence
r3t20e	TTT GCG GAA CTT CTG AAT AAT GGG TTA CAA A
r5t4e	ACC GCC TCG CCA GAA TGG AAA GCT GAG TAA C
r5t12e	TTC CTT ATA CCT CCC GAC TTG CGG AGC CTA A
r5t20e	TTA ATT TTC ATA TCA AAA TTA TTT AAC GGA T
r7t4e	CAA AAT CAC CGT TCC AGT AAG CGC TGG TAA T
r7t12e	CAT GTA GAT AAA TCA AGA TTA GTT AAT CTT AC
r7t20e	AAC TCG TAA TAA AGA AAT TGC GTA GTA ACA GT
r1t6f	TTA GCA AGA ATC ACC AGT AGC ACA TTG GGC T
r1t14f	CCA GTA ATA TTT AGG CAG AGG CAA ATA TGA T
r1t22f	CAA TCA ATC TGA ACC TCA AAT ATT GGT TCC G
r3t6f	CAT CGA TAG ACT TGA GCC ATT TGG AAA CGT AG
r3t14f	AGG TAA AGG AAT CGC CAT ATT TAA GAA AAC T
r3t22f	GTT GAA AGC AGC AAA TGA AAA ATA CAG AGA T
r5t6f	AGA ATC AAT ATT CAT TAA AGG TGA CAT ATA A
r5t14f	TAA ACA ACA AAG CCA ACG CTC AAC TCA TCT TC
r5t22f	TAT CTT TAA CCG CCT GCA ACA GTG TAA GAA TA
r7t6f	AGC GCG TTT GAG GGA GGG AAG GTT AAG TTT A
r7t14f	CCT GTT TAT CAT ATG CGT TAT ACA TAC CGA C
r7t22f	AGA GCC GTA TAA AAC AGA GGT GAA TGG CTA T
r1t6e	GCC AGC AAG CCG GAA ACG TCA CCC TCA GAG C
r1t14e	AAC ATG TAA AGA GAA TAT AAA GTA AAG CAA GC
r1t22e	TCA CCT TGA TCT GGT CAG TTG GCA AAG GAG CG
r3t6e	CCG TCA CCG CAG CAC CGT AAT CAC CTC AGA A
r3t14e	TTA ATT GAT AAT TCT GTC CAG ACT ATT AAA C
r3t22e	AGA GCC AGG AAT TGA GGA AGG TTA TTA TCA T
r5t6e	ACG GAA ATG TTT GCC TTT AGC GTC CAC CGG A
r5t14e	ACC AGT ATA TGT TCA GCT AAT GCG GCT GTC T
r5t22e	GTA TTA ACG GAG CAC TAA CAA CTG AAC GTT A
r7t6e	CAA CCG ATT TCA TCG GCA TTT TCG TTC ATA AT
r7t14e	GTT TAG TAT CAA CAA TAG ATA AGT TTA CGA G
r7t22e	AGC AGA AGC AAT AGA TAA TAC ATA ATT CGA C
r1t8f	TCC TTA TTA AAA GAA CTG GCA TGA TAG CGA G
r1t16f	CCA ATC GCT ATA TGT AAA TGC TGC CAA TAG G
r1t24h	TAA TAA AAG GGA CAT TCT GGC CAC TAA AGC A
r3t8f	AAA ATA CAA CCG AGG AAA CGC AAA GGG AAG C
r3t16f	TTT TCA AAT TAA CCT CCG GCT TAT TAA CAA T
r3t24h	AGA ACC CTT CTG ACC TGA AAG CGC CAC GCT G
r5t8f	AAG AAA CGG CAG ATA GCC GAA CAA ACA CCC T
r5t16f	TGA CCT AAA TCA AAA TCA TAG GTT GGA AAC A
r5t24h	CGT GGC ACA GAC AAT ATT TTT GAG GCG GTC A
r7t8f	TTT TGT CAT ATC TTA CCG AAG CCC CGC TAA TA
r7t16f	CGT GTG ATT TAA GAC GCT GAG AAG TGA ATA AC
r7t24h	TAG TCT TTA ATG CGC GAA CTG ATA GAA CCA CC
r9t0i	TTT TCC TCA GAA CCG CCA CCC CCT CAG A

Name	Sequence
Rem9	GCG TAC TAT GGT TGC TTT GAC GAG CAC
r9t2f	ACC GCC ACT TTT ATG ATA CAG GAG TGT ATC ATA CAT
r9t18f	TTT TCC CTT TTT GTC AGA TGA ATA TAC AGA TTT TCA
r-9t12e	CCC TGT AAT TTT ACG GTG TCT GGA AGT TAA TAT GCA
Rem2	TTT TGA CGC TCA ATC GTC TGA AAT GGA T
Rem10	GTA TAA CGT GCT TTC CTC GTT AGA ATC
r9t4f	GGC TTT TGT TTT TAG CGT TTG CCA TCT TGT CAT AGC
r9t20f	GGT TTA ACT TTT ATT AGA CTT TAC AAA CTT GAG GAT
r-9t14e	AAA ACT AGT TTT TAC TTT TGC GGG AGA ACA TTA TGA
Rem3	AGG AAA AAC GCT CAT GGA AAT ACC TAC A
Rem11	AGA GCG GGA GCT AAA CAG GAG GCC GAT
r9t6f	CCC CTT ATT TTT CGC CAA AGA CAA AAG GTC ATA TGG
r9t22j	TTA GAA GTT TTT ATT AAA AAT ACC GAA CGC CCT AAA ACA TCG CCT TTT
r-9t16e	AGG TCA CGT TTT CAT GTC AAT CAT ATG TGT AAT CGT
Rem4	AAC AAT ATT ACC GCC AGC CAT TGC AAC
Rem12	TAA AGG GAT TTT AGA CAG GAA CGG TAC
r9t8f	TTT ACC AGT TTT ATA AGA GCA AGA AAC ATG AGT TAA
r-9t2i	ACT TTT TCT TTT TTT TCA CGT TGA AAA TAT TGC GAA TAA TAA TTT TTT
r-9t18e	AAA CGA CGT TTT TTG GTG TAG ATG GGC GAA TGG GAT
Rem5	ACT ATC GGC CTT GCT GGT AAT ATC CAG
Rem13	GCC AGA ATC CTG AGA AGT GTT TTT ATA
r9t10f	GCC CAA TAT TTT CTA CAA TTT TAT CCT GGC TAT TTT
r-9t4e	GGT GTA CAT TTT ATG AGG AAG TTT CCA TGA CTA AAG
r-9t20e	GGC GGT TTT TTT GCC AGT GCC AAG CTT GAC GTT GTA
Rem6	ATC ACT TGC CTG AGT AGA AGA ACT CAA
Rem14	ATC AGT GAG GCC ACC GAG TAA AAG AGT
r9t12f	GCA CCC AGT TTT TAA TAT CCC ATC CTA ATC CTG AAC
r-9t6e	ACA TTA TTT TTT GAC CAG GCG CAT AGG CAG ATG AAC
r-9t22e	TAT CAC GGT TTT GCG TAT TGG GCG CCA GCG GGG AGA
Rem7	AGC AAT ACT TCT TTG ATT AGT AAT AAC
Rem15	CTG TCC ATC ACG CAA ATT AAC CGT TGT
r9t14f	AAG AAA AAT TTT AAT CAT AAT TAC TAG ATA AGA ATA
r-9t8e	GAA AAC GAT TTT ACA GGT AGA AAG ATT CAC GGA ACA
r-9t24j	TTT TAA CCC TAA AGG GAG CCA AAC CGT C
Rem8	GCC GCG CTT AAT GCG CCG CTA CAG GGC
r9t0i	TTT TCC TCA GAA CCG CCA CCC CCT CAG A
r9t16f	AAC ACC GGT TTT TAG AAT CCT TGA AAA CTT AAT TAA
r-9t10e	ACT AAA GTT TTT GAA TGA CCA TAA ATC AAC AGT TCA
Rem1	TAT TTA CAT TGG CAG ATT CAC CAG TCA C
Rem9	GCG TAC TAT GGT TGC TTT GAC GAG CAC
r9t2f	ACC GCC ACT TTT ATG ATA CAG GAG TGT ATC ATA CAT
r9t18f	TTT TCC CTT TTT GTC AGA TGA ATA TAC AGA TTT TCA

Name	Sequence
r-9t12e	CCC TGT AAT TTT ACG GTG TCT GGA AGT TAA TAT GCA
Rem2	TTT TGA CGC TCA ATC GTC TGA AAT GGA T
r9t16f	AAC ACC GGT TTT TAG AAT CCT TGA AAA CTT AAT TAA
r-9t10e	ACT AAA GTT TTT GAA TGA CCA TAA ATC AAC AGT TCA
Rem1	TAT TTA CAT TGG CAG ATT CAC CAG TCA C
Rem10	GTA TAA CGT GCT TTC CTC GTT AGA ATC
r9t4f	GGC TTT TGT TTT TAG CGT TTG CCA TCT TGT CAT AGC
r9t20f	GGT TTA ACT TTT ATT AGA CTT TAC AAA CTT GAG GAT
r-9t14e	AAA ACT AGT TTT TAC TTT TGC GGG AGA ACA TTA TGA
Rem3	AGG AAA AAC GCT CAT GGA AAT ACC TAC A
Rem11	AGA GCG GGA GCT AAA CAG GAG GCC GAT
r9t6f	CCC CTT ATT TTT CGC CAA AGA CAA AAG GTC ATA TGG
r9t22j	TTA GAA GTT TTT ATT AAA AAT ACC GAA CGC CCT AAA ACA TCG CCT TTT
r-9t16e	AGG TCA CGT TTT CAT GTC AAT CAT ATG TGT AAT CGT
Rem4	AAC AAT ATT ACC GCC AGC CAT TGC AAC
Rem12	TAA AGG GAT TTT AGA CAG GAA CGG TAC
r9t8f	TTT ACC AGT TTT ATA AGA GCA AGA AAC ATG AGT TAA
r-9t2i	ACT TTT TCT TTT TTT TCA CGT TGA AAA TAT TGC GAA TAA TAA TTT TTT
r-9t18e	AAA CGA CGT TTT TTG GTG TAG ATG GGC GAA TGG GAT
Rem5	ACT ATC GGC CTT GCT GGT AAT ATC CAG
Rem13	GCC AGA ATC CTG AGA AGT GTT TTT ATA
r9t10f	GCC CAA TAT TTT CTA CAA TTT TAT CCT GGC TAT TTT
r-9t4e	GGT GTA CAT TTT ATG AGG AAG TTT CCA TGA CTA AAG
r-9t20e	GGC GGT TTT TTT GCC AGT GCC AAG CTT GAC GTT GTA
Rem6	ATC ACT TGC CTG AGT AGA AGA ACT CAA
Rem14	ATC AGT GAG GCC ACC GAG TAA AAG AGT
r9t12f	GCA CCC AGT TTT TAA TAT CCC ATC CTA ATC CTG AAC
r-9t6e	ACA TTA TTT TTT GAC CAG GCG CAT AGG CAG ATG AAC
r-9t22e	TAT CAC GGT TTT GCG TAT TGG GCG CCA GCG GGG AGA
Rem7	AGC AAT ACT TCT TTG ATT AGT AAT AAC
Rem15	CTG TCC ATC ACG CAA ATT AAC CGT TGT
r9t14f	AAG AAA AAT TTT AAT CAT AAT TAC TAG ATA AGA ATA
r-9t8e	GAA AAC GAT TTT ACA GGT AGA AAG ATT CAC GGA ACA
r-9t24j	TTT TAA CCC TAA AGG GAG CCA AAC CGT C
Rem8	GCC GCG CTT AAT GCG CCG CTA CAG GGC

C.2 NOR Gate

C.2.1 1 Track Strands

Name	Sequence
G.walker.5'Q	/5IAbRQ/ CGA TGT TAG TTG GGC TGA GGT TCG AT
NOR-G.A.1-TEX615-Tether	AAA AGG CTT ACA GAG GCT TTG AGT AAA CGG GTT TAT CCC ACT CCC TAC ACT TCG
NOR-G.A.1	ATC GAA CCT CAG CCC AAC TAA CAT CGT TTT AAA ATA CAA CTT TGA AAG AGG ACT GGC TGA C
NOR-G.A.2	TCT TGT ACC GTT TGC AGA CTA TCG AAC CTC AGC CCA ACT AAC ATT TTA ACT GAC CGT AAT GCC ACT ACG ACG AGG GTA
NOR-G.A.3	TCT TGT ACC GTT TGC AGA CTA TCG AAC CTC AGC CCA ACT AAC ATT TTA TAT TCA TCA GTC AGG ACG TTG GTA ATG CAG
NOR-G.A.4	TCT TGT ACC GTT TGC AGA CTA TCG AAC CTC AGC CCA ACT AAC ATT TTC ATT ATA CTA CCC AAA TCA ACG TAG CCG GAA C
NOR-G.A.5	TCT TGT ACC GTT TGC AGA CTA TCG AAC CTC AGC CCA ACT AAC ATT TTA GTA AGA GTA AAA TGT TTA GAC TGA GAG GAA G
NOR-G.A.6	TCT TGT ACC GTT TGC AGA CTA TCG AAC CTC AGC CCA ACT AAC ATT TTG GTA ATA GCA ACA CTA TCA TAA CAT CAT TGT
NOR-G.END	TCT TGT ACC GTT TGC AGA CTA TCG AAC TTC AGC CCA ACT AAC ATT TTC GTT TTT ATA GCA AGC AAA TCA GCG ATT TTT
NOR-G.END-Cy5-Tether	AAC ATG TAA AGA GAA TAT AAA GTA AAG CAA GCT TTC CTT TTA CTT CCA CTA CCC A

C.2.2 Blocking Tracks

Name	Sequence
B.walker	TGT GAT GAT TGA GGC TGA GGT GAG
NOR-B.6nt.A.1-tether	ACT TCT CAT CAT CCA CCC TAT TTG CAA ATG GCA TAC AGG CAA GGC AAG AGT AAT G
NOR-B.6nt.A.1-anch-address	CTC ACC TCA GCC TCA ATC ATC ACA TTT TAG GGT GGA TGA TGA GAA GTC TCA CCT CAG CCT CAA TCA TCA CAT TTT AGG GTG GAT GAT GAG AAG T
NOR-B.6nt.A.2	TTT CTA CAT TCA CCT ACA CTC TCA CCT CAG CCT CAA TCA TCA TTT TAA TTG CTA CTT CAA ATA TCG CGC CAG AGG G
NOR-B.6nt.END	TTT CTA CAT TCA CCT ACA CTC TCA CTT CAG CCT CAA TCA TCA TTT GAA TAC CCA CGC AGT ATG TTA GCG AAT TAG A
NOR-R.6nt.A.1-tether	GTG GAG TGG AGA TGA GTT GAT TTG CTC ATT CTT ATG CGA TTT TAA GCG AGG CAT
NOR-R.6nt.A.1-anch-address	CTC ACC TCA GCC TCA ATC ATC ACA TTT TCA ACT CAT CTC CAC TCC AC
NOR-R.6nt.A.2	TTT CTA CAT TCA CCT ACA CTC TCA CCT CAG CCT CAA TCA TCA TTT GAA TTA CCA GTG AAT AAG GCT TGT CGA AAT C

C.2.3 Junction Anchorages

Name	Sequence
NOR-B.G.J.1.2ntGTH	CCT CAG CCC AAC TAA CAT TCC TCA CCT CAG CCT CAA TCA TCA TTT CGA GCT TCA GGT CAG GAT TAG AGG CTA TAT T
NOR-B.G.J.2.2ntGTH	CCT CAG CCC AAC TAA CAT TCC TCA CCT CAG CCT CAA TCA TCA TTT ACT CCA ACA AAG CGA ACC AGA CCG CAG CCT T

C.2.4 Cover/Uncover, Remover, and Fluorophore Strands

Name	Sequence
B.cover	GGT GAG AGT GTA GGT GAA TGT AGA AAT CAC CTC TTA TAC CAC
B.uncover	GTG GTA TAA GAG GTG ATT TCT ACA TTC ACC TAC ACT CTC ACC
G.cover	GGT TCG ATA GTC TGC AAA CGG TAC AAG AAA GTC GCT TCG CAC A
G.uncover	TGT GCG AAG CGA CTT TCT TGT ACC GTT TGC AGA CTA TCG AAC C
G.remover	ATC GAA CTT CAG CCC AAC TAA CAT CG
addressable.Cy5	TGG GTA GTG GAA GTA AAA GG /3Cy5Sp/
addressable.TEX615	CGA AGT GTA GGG AGT GGG AT /3TEX615/

Appendix D

Experimental Protocols

D.1 NOR Gate Protocol

To simplify notation for this protocol, the **1**, **x**, and **y** tracks/walkers are referred to as "green," "blue," and "red," respectively.

D.1.1 Resuspensions

Resuspend DNA to the following concentrations:

Strand Type	**Purification**	**Resuspension Conc.** (μM)
Track Anchorages	PAGE	25
Walkers	HPLC	150
Junction Strands	PAGE	25
Covers	Desalting	300
Uncovers	Desalting	600
Fluorophore Tethers	Desalting	50
Addressable Fluorophores	HPLC	50

D.1.2 Prepare Mixes

D.1.2.1 Track Mix

Anchorage stocks should all be at 25 μM.
The following track mix makes a 30 μL solution with each anchorage staple at 2.5μM.

Strand	**Volume** (μL)
A.2_G	3
A.3_G	3
A.4_G	3
A.5_G	3
A.6_G	3
A.END_G	3
A.2_B	3
A.END_B	3
A.2_R	3
H_2O	3
Total:	30

D.1.2.2 Green Walker-A.1 Duplex

Makes 20μL of 5μM stock
Add 1μL of 150μM walker to 14μL of H_2O to make a 10μM dilution

- A.1_G: 4μL of 25μM stock
- 9.5 μL of 10μM walker dilution (0.95*10μL*10μM gives 95% of A.1 molarity)
- 2μL of 10x Origami Buffer
- 4.5μL of H_2O

Anneal in a thermocycler, dropping from 95°C to 20°C at a rate of 5°C per minute.

D.1.2.3 Blue Walker-A.1 Address Duplex

Makes 20μL of 5μM stock
Add 1μL of 150μM walker to 14μL of H_2O to make a 10μM dilution

- A.1_B Address: 2μL of 50μM stock
- 9.5 μL of 10μM walker dilution
- 2μL of 10x Origami Buffer
- 6.5μL of H_2O

Anneal in a thermocycler, dropping from 95°C to 20°C at a rate of 5°C per minute.

D.1.2.4 Red Walker-A.1 Address Duplex

Makes 20μL of 5μM stock
Add 1μL of 150μM walker to 14μL of H_2O to make a 10μM dilution

- A.1_R Address: 2μL of 50μM stock
- 9.5 μL of 10μM walker dilution
- 2μL of 10x Origami Buffer
- 6.5μL of H_2O

Anneal in a thermocycler, dropping from 95°C to 20°C at a rate of 5°C per minute.

D.1.3 Master Mix

	Stock	**Excess**	**Volume**(μL)
MilliQ Water	-	-	3.00
10x Origami Buffer	-	-	7.10
Staple Mix	0.69	5	28.80
Staple Replace	10.33	5	1.924
Track Mix	2.50	3	3.75
Blue A.1 tether	5.0	3	1.875
Red A.1 tether	5.0	3	1.875
Blue/Red Cover	300	200	2.50
Green Cover	300	200	2.50
Junction 1	25	5	0.80
Junction 2	25	5	0.80
G_start_TEX615 Tether	10	3	1.20
G_end_Cy5 Tether	10	3	1.20
TEX615 3' Fluorophore	10	5	2.00
Cy5 3' Fluorophore	10	5	2.00
M13	-	1	8.93
Total:			70.254

Before adding M13: Vortex on lowest setting to mix.
Add M13 and mix by pipetting up and down.
Anneal in a thermocycler, dropping from 95°C to 20°C at a rate of 1°C per minute.

D.1.4 Anneal walker-A.1 duplexes and junction into the origami

To the origami annealed according to the master mix above, add:

- 0.724μL of the 5μM green walker-A.1 duplex prepared above

- 4.0μL (5x excess to origami) of the 5μM blue walker-A.1 address duplex prepared above
- 4.0μL (5x excess to origami) of the 5μM red walker-A.1 address duplex prepared above

Incubate at 37°C for 2 hours, holding at 4C after.

D.1.5 Prepare purification columns

1. Each origami will require 3 columns
2. Add 500 μL of Sephacryl to each column
3. Spin all columns for 1 min at 1000xg (4k RPM for desk centrifuge)
4. Add another 500 μL of Sephacryl to each column, repeat step 3
5. 2 x 3 min spin at 1000xg to remove excess buffer

D.1.6 Purify Origami

1. Add annealed origami to first column, spin for 3 min at 1000xg
2. Repeat in series for second and third column

D.1.7 Dilute samples for fluorometer

1. Measure volume after purification with a pipette. Usually about 60-70 μL.
2. Add 1x Origami buffer to a final volume of 150 μL.
3. Add 1.5 μL of 5M NaCl.
4. Vortex on lowest setting to mix.

D.1.8 Fluorimetry

Excitation/emission wavelengths for fluorophores:

- Cy5: 648nm/668nm
- TEX615: 596nm/613nm

Test the voltage to find a suitable value. Usually about 700-750V is sufficient.

1. Put samples in the cuvettes. Numbers of the cuvettes should face towards you.
2. Place 400 μL of mineral oil on top of samples straight away

3. Start the run and wait for flat signal (20 mins)

4. Add 2.45 μL of each (green and/or blue) 600 μM uncover stock as the experiment requires. Add 1 μL of Nt.BbvCI. Mix by pipetting 30x with pipette set to 25 μL

5. Leave to run until the reaction goes to completion

6. Add 1 μL of 50 μM of remover to get an unquenched baseline. Wait until signal levels.

7. Add 1.5μL of green walker at 150 μM for fully quenched baseline

References

[1] Oligonucleotide stability study. Technical report, Integrated DNA Technologies, 2014.

[2] Cleavage close to the end of DNA fragments. Technical report, New England Biolabs, 2016.

[3] Fluorescence fundamentals. Technical report, ThermoFisher Scientific, 2016.

[4] Optimizing restriction endonuclease reactions. Technical report, New England Biolabs, 2016.

[5] L.M. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266(5187):1021–1024, 1994.

[6] H.T. Allawi and J. SantaLucia Jr. Nearest neighbor thermodynamic parameters for internal GA mismatches in DNA. *Biochemistry*, 37:2170–2179, 1998. PMID: 9485363.

[7] E.S. Andersen, M. Dong, M.M. Nielsen, K. Jahn, R. Subramani, W. Mamdouh, M.M. Golas, B. Sander, H. Stark, C.L.P. Oliveira, J.S. Pedersen, V. Birkedal, F. Besenbacher, K.V. Gothelf, and J. Kjems. Self-assembly of a nanoscale DNA box with a controllable lid. *Nature*, 459:73–76, 2009.

[8] J. Bardeen and W.H. Brattain. The transistor, a semi-conductor triode. *Phys. Rev.*, 74:230–231, 1948.

[9] J. Bath, S.J. Green, K.E. Allen, and A.J. Turberfield. Mechanism for a directional, processive, and reversible DNA motor. *Small*, 5:1513–1516, 2009.

[10] J. Bath, S.J. Green, and A.J. Turberfield. A free-running DNA motor powered by a nicking enzyme. *Angewandte Chemie*, 117:4432–4435, 2005.

[11] S. Berber, Y. Kwon, and D. Tománek. Unusually high thermal conductivity of carbon nanotubes. *Phys. Rev. Lett.*, 84:4613–4616, 2000.

[12] A. Bianco. Graphene: safe or toxic? The two faces of the medal. *Angewandte Chemie*, 52:4986–4997, 2013.

[13] T.A. Bickle and D.H. Kruger. Biology of DNA restriction. *Microbiological Reviews*, 57:434–450, 1993.

[14] C.H. Bigger, K. Murray, and N.E. Murray. Recognition sequence of a restriction enzyme. *Nature*, 244:7–10, 1973.

[15] M.A. Boemo, A.E. Lucas, A.J. Turberfield, and L. Cardelli. The formal language and design principles of autonomous DNA walker circuits. *ACS Synthetic Biology*, 5:878–884, 2016.

[16] J. Bornholt, R. Lopez, D.M. Carmean, L. Ceze, G. Seelig, and K. Strauss. A DNA-based archival storage system. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS'16, pages 637–649, New York, NY, USA, 2016. ACM.

[17] J.G. Brookshear. *Theory of Computation: Formal Languages, Automata, and Complexity*. The Benjamin/Cummings Publishing Company, 1989.

[18] R. Carlson. The changing economics of DNA synthesis. *Nature Biotechnology*, 27:1091–1094, 2009.

[19] C.E. Castro, F. Kilchherr, D. Kim, E.L. Shiao, T. Wauer, P. Wortmann, M. Bathe, and H. Dietz. A primer to scaffolded DNA origami. *Nature Methods*, 8:221–229, 2011.

[20] H. Chen, T. Weng, M.M. Riccitelli, Y. Cui, J. Irudayaraj, and J.H. Choi. Understanding the mechanical properties of DNA origami tiles and controlling the kinetics of their folding and unfolding reconfiguration. *JACS*, 136:6995–7005, 2014.

[21] Z. Chen, X. Lan, Y. Chiu, X. Lu, W. Ni, H. Gao, and Q. Wang. Strong chiroptical activities in gold nanorod dimers assembled using DNA origami templates. *ACS Photonics*, 2:392–397, 2015.

[22] Q. Chi, G. Wang, and J. Jiang. The persistence length and length per base of single-stranded DNA obtained from fluorescence correlation spectroscopy measurements using mean field theory. *Physica A: Statistical Mechanics and its Applications*, 392:1072 – 1079, 2013.

[23] A. Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58:345–363, 1936.

[24] G.M. Church, Y. Gao, and S. Kosuri. Next-generation digital information storage in DNA. *Science*, 337:1628–1628, 2012.

[25] R. Crawford, C.M. Erben, J. Periz, L.M. Hall, T. Brown, A.J. Turberfield, and A.N. Kapanidis. Non-covalent single transcription factor encapsulation inside a DNA cage. *Angewandte Chemie International Edition*, 52:2284–2288, 2013.

[26] F. Dannenberg, M. Kwiatkowska, C. Thachuk, and A.J. Turberfield. DNA walker circuits: Computational potential, design, and verification. In D. Soloveichik and B. Yurke, editors, *DNA Computing and Molecular Programming*, vol-

ume 8141 of *Lecture Notes in Computer Science*, pages 31–45. Springer International Publishing, 2013.

[27] F. Dannenberg, M. Kwiatkowska, C. Thachuk, and A.J. Turberfield. DNA walker circuits: computational potential, design, and verification. *Natural Computing*, 14:195–211, 2015.

[28] M.F.L. De Volder, S.H. Tawfick, R.H. Baughman, and A.J. Hart. Carbon nanotubes: Present and future commercial applications. *Science*, 339:535–539, 2013.

[29] M. Diaz. *Petri Nets: Fundamental Models, Verification, and Applications*. Wiley-ISTE, 2013.

[30] H. Dietz, S. Douglas, and W.M. Shih. Folding DNA into twisted and curved nanoscale shapes. *Science*, 325:725–730, 2009.

[31] B. Ding, Z. Deng, H. Yan, S. Cabrini, R.N. Zuckermann, and J. Bokor. Gold nanoparticle self-similar chain structure organized by DNA origami. *Journal of the American Chemical Society*, 132:3248–3249, 2010. PMID: 20163139.

[32] S.M. Douglas, I. Bachelet, and G.M. Church. A logic-gated nanorobot for targeted transport of molecular payloads. *Science*, 335:831–834, 2012.

[33] S.M. Douglas, A.H. Marblestone, S. Teerapittayanon, A. Vazquez, G.M. Church, and W.M. Shih. Rapid prototyping of 3D DNA-origami shapes with caDNAno. *Nucleic Acids Research*, 2009.

[34] J.P.K. Doye, T.E. Ouldridge, A.A. Louis, F. Romano, P. Sulc, C. Matek, B.E.K. Snodin, L. Rovigatti, J.S. Schreck, R.M. Harrison, and W.P.J. Smith. Coarse-graining DNA for simulations of DNA nanotechnology. *Phys. Chem. Chem. Phys.*, 15:20395–20414, 2013.

[35] M. Endo, M.S. Strano, and P.M. Ajayan. *Carbon Nanotubes: Advanced Topics in the Synthesis, Structure, Properties and Applications*, chapter Potential Applications of Carbon Nanotubes, pages 13–62. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[36] C.M. Erben, R.P. Goodman, and A.J. Turberfield. Single-molecule protein encapsulation in a rigid DNA cage. *Angewandte Chemie*, 118:7574–7577, 2006.

[37] C.M. Erben, R.P. Goodman, and A.J. Turberfield. A self-assembled DNA bipyramid. *Journal of the American Chemical Society*, 129:6992–6993, 2007. PMID: 17500526.

[38] L. Feng, S.H. Park, J.H. Reif, and H. Yan. A two-state DNA lattice switched by a DNA nanoactuator. *Angewandte Chemie*, 115:4478–4482, 2003.

[39] A.K. Geim. Graphene: Status and prospects. *Science*, 324:1530–1534, 2009.

[40] R.P. Goodman, R.M. Berry, and A.J. Turberfield. The single-step synthesis of a DNA tetrahedron. *Chem. Commun.*, pages 1372–1373, 2004.

[41] D. Han, S. Pal, J. Nangreave, Z. Deng, Y. Liu, and H. Yan. DNA origami with complex curvatures in three-dimensional space. *Science*, 332:342–346, 2011.

[42] J.R. Hindley and J.P. Seldin. *Lambda-Calculus and Combinators: An Introduction*. Cambridge University Press, 2008.

[43] R. Holliday. A mechanism for gene conversion in fungi. *Genetical Research*, 5:282–304, 1964.

[44] R. Iinuma, Y. Ke, R. Jungmann, T. Schlichthaerle, J.B. Woehrstein, and P. Yin. Polyhedra self-assembled from DNA tripods and characterized with 3D DNA-PAINT. *Science*, 344:65–69, 2014.

[45] D. Joyner, O. Čertík, A. Meurer, and B.E. Granger. Open source computer algebra systems: SymPy. *ACM Commun. Comput. Algebra*, 45:225–234, 2011.

[46] R. Jungmann, M.S. Avendaño, J.B. Woehrstein, M. Dai, W.M. Shih, and P. Yin. Multiplexed 3D cellular super-resolution imaging with DNA-PAINT and Exchange-PAINT. *Nature Methods*, 11:313, 2014.

[47] Y. Ke, S.M. Douglas, M. Liu, J. Sharma, A. Cheng, A. Leung, Y. Liu, W.M. Shih, and H. Yan. Multilayer DNA origami packed on a square lattice. *Journal of the American Chemical Society*, 131:15903–15908, 2009. PMID: 19807088.

[48] O. Kennard. Structural studies of DNA fragments: the G-T wobble base pair in A, B, and Z DNA; the G-A base pair in B-DNA. *Journal of Biomolecular Structure and Dynamics*, 3:205–226, 1985.

[49] D. Kim, F. Kilchherr, H. Dietz, and M. Bathe. Quantitative prediction of 3D solution shape and flexibility of nucleic acid nanostructures. *Nucleic Acids Research*, 40:2862–2868, 2012.

[50] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. *Proc. 23rd International Conference on Computer Aided Verification*, 6806:585–591, 2011.

[51] T.H. LaBean, H. Yan, J. Kopatsch, F. Liu, E. Winfree, J.H. Reif, and N.C. Seeman. Construction, analysis, ligation, and self-assembly of DNA triple crossover complexes. *Journal of the American Chemical Society*, 122:1848–1860, 2000.

[52] M.R. Lakin, R. Petersen, K.E. Gray, and A. Phillips. *DNA Computing and Molecular Programming: 20th International Conference, DNA 20, Kyoto, Japan, September 22-26, 2014. Proceedings*, chapter Abstract Modelling of Tethered DNA Circuits, pages 132–147. Springer International Publishing, Cham, 2014.

[53] M.R. Lakin, S. Youssef, F. Polo, S. Emmott, and A. Phillips. Visual DSD: A design and analysis tool for DNA strand displacement systems. *Bioinformatics*, 27:3211–3213, 2011.

[54] J.D. Le, Y. Pinto, N.C. Seeman, K. Musier-Forsyth, T.A. Taton, and R.A. Kiehl. DNA-templated self-assembly of metallic nanocomponent arrays on a surface. *Nano Letters*, 4:2343–2347, 2004.

[55] H. Li, J.D. Carter, and T.H. LaBean. Nanofabrication by DNA self-assembly. *Materials Today*, 12:24 – 32, 2009.

[56] Q. Liu, L. Wang, A.G. Frutos, A.E. Condon, R.M. Corn, and L.M. Smith. DNA computing on surfaces. *Nature*, 403:175–179, 2000.

[57] K. Lund, A.J. Manzo, N. Dabby, N. Michelotti, A. Johnson-Buck, J. Nangreave, S. Taylor, R. Pei, M.N. Stojanovic, N.G. Walter, et al. Molecular robots guided by prescriptive landscapes. *Nature*, 465:206–210, 2010.

[58] R.R.F. Machinek, T.E. Ouldridge, N.E.C. Haley, J. Bath, and A.J. Turberfield. Programmable energy landscapes for kinetic control of DNA strand displacement. *Nature Communications*, 5:1–9, 2014.

[59] J. Malo, J.C. Mitchell, C. Venien-Bryan, J. Robin Harris, W. Holger, D.J. Sherratt, and A.J. Turberfield. Engineering a 2D protein-DNA crystal. *Angewandte Chemie*, 44:3057–3061, 2005.

[60] M. Meselson and R. Yuan. DNA restriction enzyme from E. coli. *Nature*, 217:1110–1114, 1968.

[61] M.K. Molloy. Performance analysis using stochastic petri nets. *IEEE Transactions on Computers*, C-31:913–917, Sept 1982.

[62] T.E. Ouldridge, A.A. Louis, and J.P.K. Doye. DNA nanotweezers studied with a coarse-grained model of DNA. *Phys. Rev. Lett.*, 104:178101, Apr 2010.

[63] J. Pan, F. Li, T. Cha, H. Chen, and J. Choi. Recent progress on DNA based walkers. *Current Opinion in Biotechnology*, 34:56 – 64, 2015. Systems biology Nanobiotechnology.

[64] P.J. Paukstelis, J. Nowakowski, J.J. Birktoft, and N.C. Seeman. Crystal structure of a continuous three-dimensional {DNA} lattice. *Chemistry & Biology*, 11:1119 – 1126, 2004.

[65] R. Pei, S.K. Taylor, D. Stefanovic, S. Rudchenko, T.E. Mitchell, and M.N. Stojanovic. Behavior of polycatalytic assemblies in a substrate-displaying matrix. *J. Am. Chem. Soc.*, 128:12693–12699, 2006.

[66] N. Peyret, P. Ananda Seneviratne, H.T. Allawi, and J. SantaLucia Jr. Nearest-neighbor thermodynamic and NMR of DNA sequence with internal A-A, C-C, G-G, and T-T mismatches. *Biochemistry*, 38:3468–3477, 1999.

[67] A. Phillips and L. Cardelli. A programming language for composable DNA circuits. *J. R. Soc. Interface*, 6:S419–S436, 2009.

[68] G. Piccinini. *Physical Computation*. Oxford University Press, 2015.

[69] H. Pospesel. *Introduction to Logic: Propositional Logic*. Pearson, 1997.

[70] L. Qian, D. Soloveichik, and E. Winfree. *DNA Computing and Molecular Programming: 16th International Conference, DNA 16, Hong Kong, China, June 14-17, 2010, Revised Selected Papers*, chapter Efficient Turing-Universal Computation with DNA Polymers, pages 123–140. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[71] L. Qian and E. Winfree. Scaling up digital circuit computation with DNA strand displacement cascades. *Science*, 332:1196–1201, 2011.

[72] W. Rautenberg. *A Concise Introduction to Mathematical Logic*. Springer, 2009.

[73] J.H. Reif. DNA lattices: A method for molecular-scale patterning and computation. *Computing in Science & Engineering*, 4:32–41, 2002.

[74] J.H. Reif, T.H. LaBean, and N.C. Seeman. *DNA Computing*, chapter Challenges and applications for self-assembled DNA nanostructures, pages 173–198. Springer Berlin Heidelberg, 2001.

[75] P.W.K. Rothemund. Folding DNA to create nanoscale shapes and patterns. *Nature*, 440:297– 302, 2006.

[76] K. Sakamoto, H. Gouzu, K. Komiya, D. Kiga, S. Yokoyama, T. Yokomori, and M. Hagiya. Molecular computation by DNA hairpin formation. *Science*, 288:1223–1226, 2000.

[77] J. SantaLucia Jr., H.T. Allawi, and P. Arnanda Seneviratne. Improved nearest-neighbor parameters for predicting DNA duplex stability. *Biochemistry*, 35:3555–3562, 1996.

[78] J. SantaLucia Jr. and D. Hicks. The thermodynamic of DNA structural motifs. *Annu. Rev. Biophys. Biomol. Struct.*, 33:415–40, 2004.

[79] G. Seelig, D. Soloveichik, D.Y. Zhang, and E. Winfree. Enzyme-free nucleic acid logic circuits. *Science*, 314:1585–1588, 2006.

[80] N.C. Seeman. Nucleic acid junctions and lattices. *Journal of Theoretical Biology*, 99:237–247, 1982.

[81] W.B. Sherman and N.C. Seeman. A precisely controlled DNA biped walking device. *Nano Letters*, 4:1203–1207, 2004.

[82] W.M. Shih, J.D. Quispe, and G.F. Joyce. A 1.7-kilobase single-stranded DNA that folds into a nanoscale octahedron. *Nature*, 427:618–621, 2004.

[83] J. Shin and N.A. Pierce. A synthetic DNA walker for molecular transport. *Journal of the American Chemical Society*, 126:10834–10835, 2004. PMID: 15339155.

[84] M. Sipser. *Introduction to the Theory of Computation*. Thomson Course Technology, 2006.

[85] D. Soloveichik, G. Seelig, and E. Winfree. *DNA Computing: 14th International Meeting on DNA Computing, DNA 14, Prague, Czech Republic, June 2-9, 2008. Revised Selected Papers*, chapter DNA as a Universal Substrate for Chemical Kinetics, pages 57–69. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[86] J. Sponer, J. Leszczynski, and P. Hobza. Hydrogen bonding and stacking of DNA bases: A review of quantum-chemical ab initio studies. *Journal of Biomolecular Structure and Dynamics*, 14:117–135, 1996. PMID: 8877568.

[87] P. Sulc, T.E. Ouldridge, F. Romano, J.P.K. Doye, and A.A. Louis. Simulating a burnt-bridges DNA motor with a coarse-grained DNA model. *Natural Computing*, 13:535–547, 2014.

[88] P. Sulc, F. Romano, T.E. Ouldridge, L. Rovigatti, J.P.K. Doye, and A.A. Louis. Sequence-dependent thermodynamics of a coarse-grained DNA model. *The Journal of Chemical Physics*, 137, 2012.

[89] S.J. Tans, M.H. Devoret, H. Dai, A. Thess, R.E. Smalley, L.J. Geerligs, and C. Dekker. Individual single-wall carbon nanotubes as quantum wires. *Nature*, 386:474–477, 1997.

[90] C. Thachuk, E. Winfree, and D. Soloveichik. Leakless DNA strand displacement systems. In A. Phillips and P. Yin, editors, *DNA Computing and Molecular Programming: 21st International Conference*, volume 9211, pages 133–153. Springer, 2015.

[91] Y. Tian, Y. He, Y. Chen, P. Yin, and C. Mao. A DNAzyme that walks processively and autonomously along a one-dimensional track. *Angewandte Chemie International Edition*, 44:4355–4358, 2005.

[92] M.M.J. Treacy, T.W. Ebbesen, and J.M. Gibson. Exceptionally high Young's modulus observed for individual carbon nanotubes. *Nature*, 381:678–680, 1996.

[93] A.J. Turberfield, J.C. Mitchell, B. Yurke, A.P. Jr Mills, M.I. Blakey, and F.C. Simmel. DNA fuel for free-running nanomachines. *Physical Review Letters*, 90:118102, 2003.

[94] A.M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *J. of Math*, 58(345-363):5, 1936.

[95] R.D. Vale, T.S. Reese, and M.P. Sheetz. Identification of a novel force-generating protein, kinesin, involved in microtubule-based motility. *Cell*, 42:39–50, 1985.

[96] A.S. Walsh, H. Yin, C.M. Erben, M.J.A. Wood, and A.J. Turberfield. DNA cage delivery to mammalian cells. *ACS Nano*, 5:5427–5432, 2011.

[97] A. Wasilewska. An introduction to classical and non-classical logics. Book in progress.

[98] J.D. Watson and F.H.C. Crick. Molecular structure of nucleic acids. *Nature*, 171:737–738, 1953.

[99] G.M. Whitesides, J.P. Mathias, and C.T. Seto. Molecular self-assembly and nanochemistry: a chemical strategy for the synthesis of nanostructures. *Science*, 254:1312–1319, 1991.

[100] S.F.J. Wickham. *DNA origami: a substrate for the study of molecular motors*. PhD thesis, University of Oxford, 2011.

[101] S.F.J Wickham, J. Bath, Y. Katsuda, M. Endo, K. Hidaka, H. Sugiyama, and A.J. Turberfield. A DNA-based molecular motor that can navigate a network of tracks. *Nature Nanotechnology*, 7:169–173, 2012.

[102] S.F.J. Wickham, M. Endo, Y. Katsuda, K. Hidaka, J. Bath, H. Sugiyama, and A.J. Turberfield. Direct observation of stepwise movement of a synthetic molecular transporter. *Nature Nanotechnology*, 6:166–169, 2011.

[103] E. Winfree, F. Liu, L.A. Wenzler, and N.C. Seeman. Design and sef-assembly of two-dimensional DNA crystals. *Nature*, 394:539–544, 1998.

[104] R. Wing, H. Drew, T. Takano, C. Broka, S. Tanaka, K. Itakura, and R.E. Dickerson. Crystal structure analysis of a complete turn of B-DNA. *Nature*, 287:755–758, 1980.

[105] P. Yin, A.J. Turberfield, S. Sahu, and J.H. Reif. Design of an autonomous DNA nanomechanical device capable of universal computation and universal translational motion. In Claudio Ferretti, Giancarlo Mauri, and Claudio Zandron, editors, *DNA Computing*, volume 3384 of *Lecture Notes in Computer Science*, pages 426–444. Springer Berlin Heidelberg, 2005.

[106] B. Yurke, A.J. Turberfield, A.P. Mills Jr., F.C. Simmel, and J.L. Neumann. A DNA-fuelled molecular machine made of DNA. *Nature*, 406:605–608, 2000.

[107] J.N. Zadeh, C.D. Steenberg, J.S. Bois, B.R. Wolfe, M.B. Pierce, A.R. Khan, R.M. Dirks, and N.A. Pierce. NUPACK: Analysis and design of nucleic acid systems. *Journal of Computational Chemistry*, 32:170–173, 2011.

[108] D.Y. Zhang, A.J. Turberfield, B. Yurke, and E. Winfree. Engineering entropy-driven reactions and networks catalyzed by DNA. *Science*, 318:1121–1125, 2007.

[109] Y. Zhang and N.C. Seeman. Construction of a DNA-truncated octahedron. *J. Am. Chem. Soc.*, 116:1661–1669, 1994.