Automated Design and Verification of Localized DNA Computation Circuits

Michael A. Boemo[1], Andrew J. Turberfield[1], and Luca Cardelli[2,3]

[1] University of Oxford,
Department of Physics, Clarendon Laboratory, Parks Road, Oxford OX1 3PU, UK
(michael.boemo,a.turberfield1)@physics.ox.ac.uk
[2] University of Oxford,
Department of Computer Science, Wolfson Building, Parks Road, Oxford OX1 3QD, UK
[3] Microsoft Research Cambridge, Station Road, Cambridge CB1 2FB, UK
luca@microsoft.com

Abstract. Simple computations can be performed using the interactions between single-stranded molecules of DNA. These interactions are typically toehold-mediated strand displacement reactions in a well-mixed solution. We demonstrate that a DNA circuit with tethered reactants is a distributed system and show how it can be described as a stochastic Petri net. The system can be verified by mapping the Petri net onto a continuous time Markov chain, which can also be used to find an optimal design for the circuit. This theoretical machinery can be applied to create software that automatically designs a DNA circuit, linking an abstract propositional formula to a physical DNA computation system that is capable of evaluating it.

Computation with DNA has been the subject of much interest from the points of view of both pure computer science and nanomedicine. A 2009 paper by Andrew Phillips and Luca Cardelli showed how DNA strand displacement can be thought of as a formal computing language [8]. Further work by Matthew Lakin and colleagues produced Microsoft Visual DSD, a computational tool for the design and analysis of such reactions [6]. In the field of nanomedicine, Benenson et al. created a biomolecular DNA computing system that can produce an mRNA inhibitor to control gene expression [2].

These papers all consider DNA strands as freely floating reactants in a well-mixed solution. There are no topological or geometric constraints that prevent two species from interacting. Such constraints can be introduced by tethering DNA reactants to rigid structures. Yin and colleagues designed a DNA Turing machine that operates by DNA walkers moving on a rigid lattice [11]. Another method utilizes the tethering of walkers to a DNA origami tile [9]. In a 2005 paper, Jonathan Bath and colleagues introduced a DNA walker powered by a nicking enzyme that is capable of traversing a track of single-stranded DNA anchorages (Figure 1) [1]. Shelley Wickham and colleagues built on this design in a 2011 paper that demonstrated how the walker could be programmed to navigate a series of tracks on an origami tile [10]. The result was a DNA walker that could perform a computation, namely a binary decision tree. This is a "local" computation that is performed by reactants that are tethered to the origami tile.

Localized DNA computation has also been the subject of theoretical and computational study. A recent paper by Dannenberg et al. analyzed the computational potential of localized DNA circuits [3]. Lakin and colleagues incorporated tethered

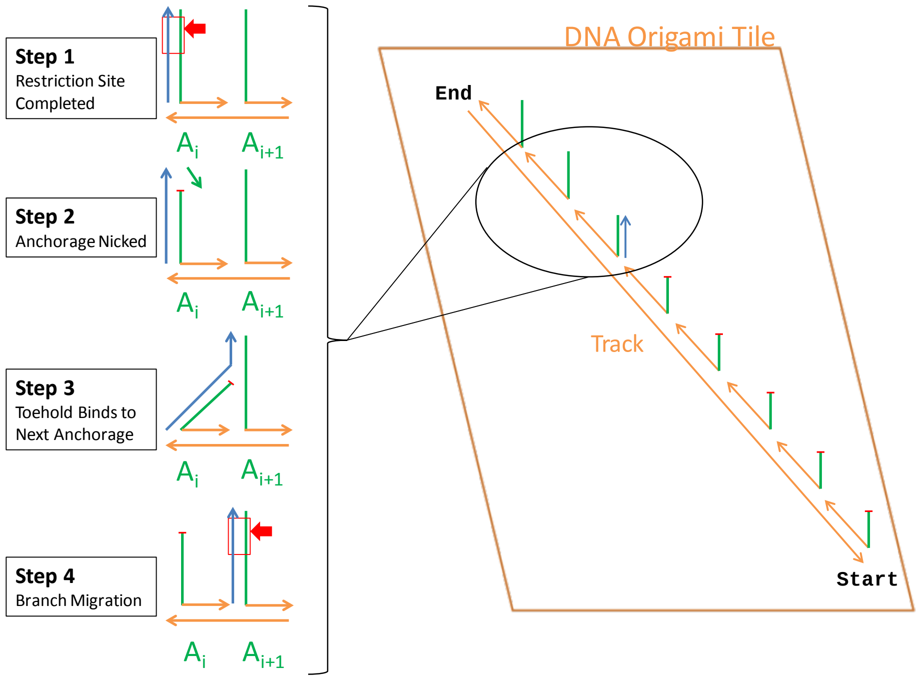

Fig. 1: The “burnt-bridges” DNA walker from [1]. Single-stranded oligonucleotides are shown as arrows, with the arrowhead indicating the 3’ end. *Step 1* A DNA walker (shown in blue) is bound to a complementary single-stranded anchorage (A_i), completing the restriction site (red box) for a nicking enzyme. *Step 2* The nicking enzyme cuts the 5’ end of anchorage A_i, exposing a 6-nucleotide toehold on the 3’ end of the DNA walker. *Step 3* The toehold on the walker binds to the next anchorage in the track, anchorage A_{i+1}. *Step 4* The DNA walker moves from anchorage A_i to A_{i+1} via a toehold-mediated branch migration reaction. By continuously repeating Steps 1-4, a DNA walker can navigate down a track of single-stranded anchorages.

DNA reactants into Microsoft Visual DSD, allowing for topological and geometric constraints in DNA circuits [7]. This software brings the functionality of Visual DSD to localized DNA computation circuits: It can perform probabilistic model checking, detect leaks, and provide information about reaction rates.

We demonstrate how localized DNA circuits can be analyzed as distributed systems. As such, they can be automatically designed and verified by software. The input to the localized DNA circuit can be posed using the language and grammar of the propositional calculus (Section 1). This input is then abstracted into a directed graph that captures the topology of the circuit (Section 2). Because the localized DNA circuit is a distributed system, it can be modeled as a stochastic Petri net. Analysis of this Petri net determines the geometry of the circuit (Section 3).

1 The Propositional Calculus of DNA Localized Computation Circuits

The language of a propositional calculus is composed of two parts: The first is a set of propositional variables, or atomic statements that each hold exactly one truth value (**1** or **0**); the second is the set of logical connectives, or operators that act on the propositional variables. A propositional formula is a string of propositional variables and logical connectives that is said to be well-formed if it follows the rules of the grammar.

Localized DNA computation systems can be designed to evaluate propositional formulae, and their action can be written in the formal language and grammar of the propositional calculus. The set of all logical connectives Ω can be partitioned into disjoint subsets according to their arity, or the number of arguments each connective takes:

$$\Omega = \Omega_0 \cup \Omega_1 \cup \Omega_2 \cup \cdots \cup \Omega_n.$$

For localized DNA computation systems, attention is restricted to nullary, unary, and binary logical connectives where,

$$\begin{aligned} \Omega_0 &= \{\mathbf{0}, \mathbf{1}\}\,, \\ \Omega_1 &= \{\neg\}\,, \\ \Omega_2 &= \{\vee, \wedge, \rightarrow, \leftrightarrow\}\,. \end{aligned}$$

The rules of the propositional calculus can be used to search for a simpler form of a propositional formula in order to make the corresponding DNA computation system more efficient. For example, the propositional formula

$$(x \wedge y) \vee (x \wedge z) \tag{1}$$

has three logical connectives, and hence will require three logic gates. In this case, it is possible to find an equivalent form that requires only two:

$$(x \wedge y) \vee (x \wedge z) \equiv x \wedge (y \vee z). \tag{2}$$

There are libraries available that can implement heuristics for such a search, e.g. SymPy [4].

2 Directed Graph Abstraction

An input propositional formula, like those described in the previous section, can be used to find the topology for a localized DNA circuit. Every propositional variable is represented by a track, or linear array of DNA anchorages tethered to an origami tile, and a DNA walker. If the propositional variable holds the value **1**, then the walker will begin walking at the start of its track. Tracks that always take the value **1** have a DNA walker that will always start walking.

DNA walkers are able to perform universal Boolean logic if they are able to block other walkers on tracks that intersect their own. It is straightforward to construct a NOR gate, which is a functionally complete operator, using track blockage (Figure

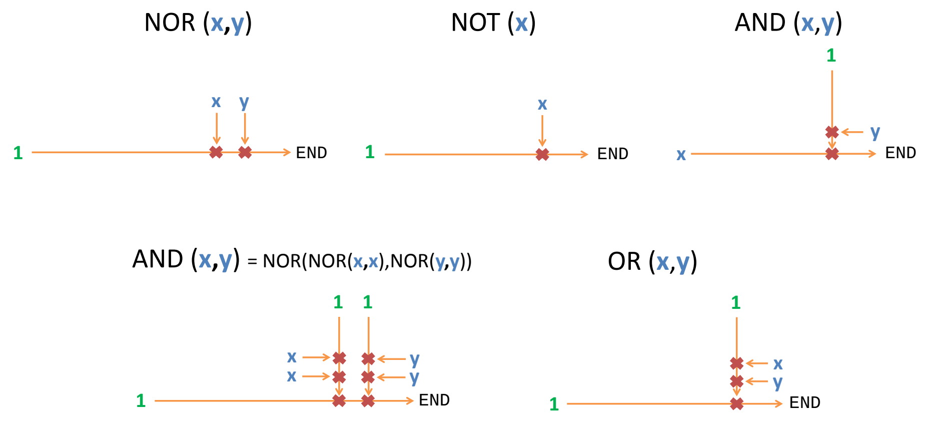

Fig. 2: Track diagrams showing one possible choice of design for NOR, NOT, OR, and AND logic gates that use the interaction (blockage) between DNA walkers. Each walker stays on its own track, and all walkers begin stepping at the same time. Walkers denoted by **1** will always walk while **x** and **y** are walkers that will only start walking if their respective propositions are true. Walkers can block another track at the junctions marked with a red cross. The gate evaluates to **1** if the walker whose track has END at the end of it is indeed able to make it to the end without being blocked.

2, top left). Figure 2 shows both an AND gate constructed out of NOR gates and an alternative design that is simpler and uses fewer tracks.

A formula written in the propositional calculus, together with the chosen design for each gate, is enough to completely determine the topology of a corresponding DNA walker circuit. In the context of DNA localized computation systems, topology refers to "connectedness" of the tracks. Two tracks are said to be topologically connected if they intersect so that the walkers on both tracks can interact with each other.

More formally, for any chosen gate design, it is possible to describe the topology of each gate in terms of a directed graph $G_D = (V, E)$ where V is a set of vertices and E is a set of edges represented as ordered pairs. For the NOR gate,

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

The set V can be interpreted as the gate having three tracks, one for each of walkers $\{\mathbf{1}, x, y\}$. The set of ordered pairs E indicates that the x walker blocks the **1** walker and the y walker blocks the **1** walker. Figure 2 shows one possible choice of gate design. Once a choice is made, directed graphs can be constructed for each gate as shown in Table 1. Directed graphs for the gates can be pieced together to form one directed graph for the whole circuit, capturing the topology of a DNA circuit that evaluates the propositional formula.

Adding the directed graph abstraction between the propositional formula and its resulting track diagram has a number of advantages. At the topological level,

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

Table 1: The sets of edges E and vertices V in the directed graph that correspond to each logic gate. Subscripts are used to differentiate between unique tracks.

the system becomes easier to analyze and simplify by automated means. The most immediate example is detecting certain redundancies, which can informally be thought of as double negatives. For any tracks a and b, if there exists a path $\{(a, \mathbf{1}_i), (\mathbf{1}_i, \mathbf{1}_j), (\mathbf{1}_j, b)\} \subset E$, then we may replace this path by $\{(a, b)\}$. In logical terms, this is the equivalent of writing $\neg(\neg b) = b$.

FANOUT

A key use of the directed graph structure is that it can represent circuits that cannot be posed in the propositional calculus. The most immediate example is FANOUT, which can be written as a directed graph:

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

As shown in Figure 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 universal Boolean logic. It can, however, be useful in simplifying track designs by using fewer walkers overall.

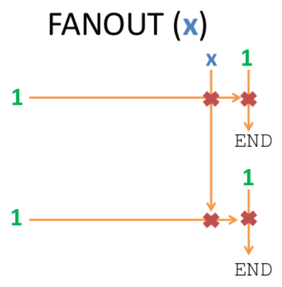

Fig. 3: A track diagram of a possible design choice for a FANOUT gate. This design requires a walker that can block a track and keep walking, blocking other tracks thereafter.

3 Localized DNA Circuits as Distributed Systems

This section shows how introducing another piece of information, a tolerance for the probability of error, allows one to use the circuit topology to find a circuit geometry. A geometry defines the length of each track and specifies the locations where the tracks intersect. The tools needed to find this information come from analyzing the localized DNA circuit as a distributed system.

A distributed system is a network of autonomous computers that can perform a coordinated action by passing messages between different computers in the network. When the anchorages on an origami tile are viewed as networked computers and the walker is viewed as the the message, a localized DNA computation system becomes a distributed system. The advantage of looking at the DNA circuit in this light is that it can be readily represented and analyzed as a Petri net.

3.1 Stochastic Petri Nets

The "burnt-bridges" walker-track system from [1] can be modeled as shown in Figure 4, upper. The initial marking shows the DNA walker, represented by a token, on the first anchorage G1. The stochastic Petri net allows each walker a stepping rate, or the 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. Such 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 nicked by the nicking enzyme. A reusable track can be represented in a similar fashion (Figure 4, lower). In both cases, walkers are assumed to only step forward and remain on the last anchorage once they reach the end of their track.

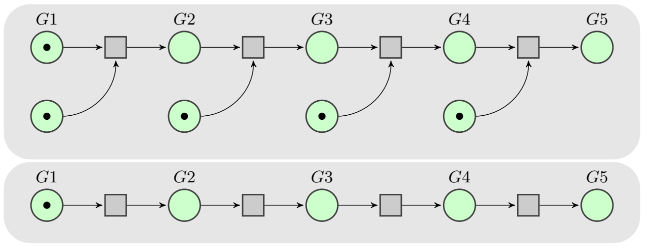

Fig. 4: *upper* Stochastic Petri net representation of the "burnt-bridges" walker from [1]. This track can only be used once, as the tokens in the bottom row of nodes are used up as the walker steps. *lower* Stochastic Petri net of a reusable track. Each transition only requires one token to fire, and no tokens are used up in the process.

Junctions between tracks are needed to implement the entire localized DNA circuit as a stochastic Petri net. Figure 5 shows a Petri net where a designated blocking walker (blue walker) can block another walker (green walker) if the blue walker arrives at the junction first. If the green walker arrives at the junction first, it steps through to the end of its track as normal.

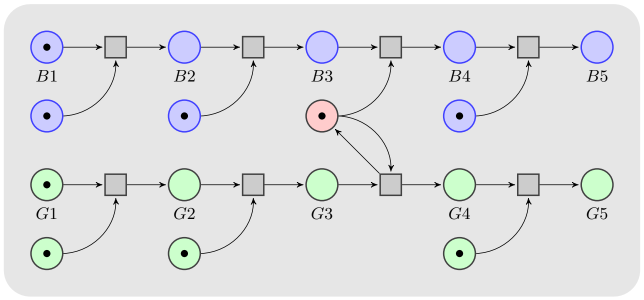

Fig. 5: Two tracks, green and blue, with a blocking junction on the third anchorage of each track (G3 and B3). If the blue walker arrives at the junction first, it can block the green track by using up the token of the shared node (shown in red).

A stochastic Petri net can be mapped directly onto a Markov process. If the DNA system can be posed as a Petri net, then it can be mapped onto a continuous time Markov chain (CTMC). Using this CTMC, the probability of certain properties of the system can be computed using techniques from formal verification.

3.2 Designing a System Using Formal Verification Techniques

The localized computation systems discussed thus far operate under the assumption that all DNA walkers, if they walk at all, begin walking at the same time and walk at the same rate. This imposes certain length restrictions on the tracks. In the NOT gate, for example, the track for the **1** walker must be sufficiently longer than the track for the x walker so that the **1** walker does not arrive at the junction first. If it does, a missed chance error has occurred because the x walker has missed its chance to block the **1** walker.

By representing the DNA system as a Markov chain and analyzing it for each possible combination of track lengths, one can search for a design that is optimal in the sense that it is compact and minimizes the probability of missed chance errors. Starting from the state corresponding to the initial marking of the Petri net, the system is allowed to evolve according to the CTMC. Earlier, it was assumed that all walkers can only step forward, if they step at all, and that walkers do not move forward once they are blocked or reach the end of their track. Hence, the Markov chain is an absorbing Markov chain, and an absorbing state will always be reached if the system runs to equilibrium. The absorbing states can be divided into two groups: one group that corresponds to a missed chance error for at least one junction, and another group that corresponds to correct operation. Analysis of the CTMC determines the probability with which the system ends up in missed chance error state or a correct state after it is allowed to run for a long time.

Prism provides a natural scripting language for representing complex systems as Markov chains in continuous time [5]. It also allows the user to evaluate the probability of certain conditions, such as the probability of eventually ending up in a certain state. For example, if a walker B is intended to block a walker G, Prism should check the following property:

```
P = ? [B=end & G>intersection].
```

In Prism's language, this is the probability that the G walker has moved through the junction before the B walker has arrived to block it. This statement measures the probability of a missed chance error for that junction. Due to the modular nature of the Prism language, the code can be automatically generated from the directed graph structures described in the previous section.

Model checking can be used to determine the system with the shortest tracks that still has a missed chance error 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 missed chance errors. The output is the assignment of a natural number to each track for the smallest number of anchorages needed to stay within the specified tolerance for missed chance error. Assuming that a track is always blocked on its penultimate anchorage, this is sufficient to determine the track geometry of the system.

3.3 Illustrative Example 1: Track Design

This theoretical machinery forms the foundation for software that can design track systems. The only inputs required are a propositional formula, a choice of gate design, and a tolerance for missed chance error. The plot in Figure 6 was automatically generated in this way.

The simplified propositional formula in Equation 2 can be arranged into a parsing tree based on its logical connectives. Prism can be used to find the lengths (in anchorages) of each track that minimizes missed chance error. Using a tolerance of 0.15 probability for missed chance error results in the following optimized track lengths:

$$\mathbf{1}_1 = 14 \text{ anchorages}, \mathbf{1}_2 = \mathbf{1}_3 = 7 \text{ anchorages}, x = y = z = 2 \text{ anchorages}.$$

Using the lengths of each individual track shown above and the blocking topology from the directed graph, a track design is generated that evaluates the original propositional formula (Figure 6).

3.4 Illustrative Example 2: DNA Mechanism

Figures 7 and 8 show an example of a DNA walker mechanism that implements the junction in Figure 5, whereby one walker (shown in blue) blocks another (shown in green). The notation used is similar to that of Microsoft Visual DSD. Single-stranded oligonucleotides are represented by arrows, as in Figure 1. Domains are printed as short strings of characters, such as B, and a domain's reverse complement is appended with an asterisk. For example, the reverse complement of B is B^*. Certain domains contain a restriction site for a nicking enzyme. When a restriction site is completed by hybridization to its reverse complement, the location where the nicking

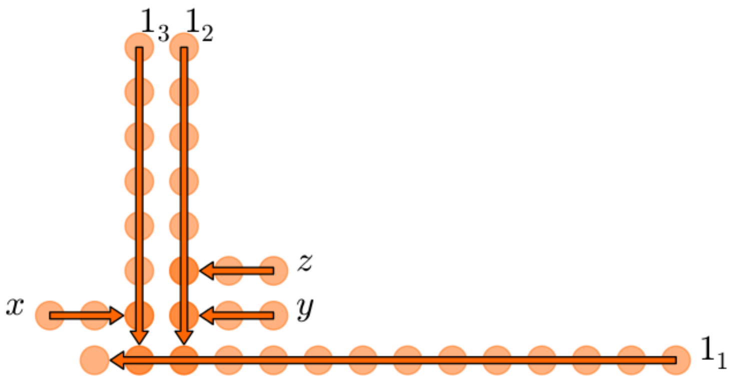

Fig. 6: Final design of the DNA localized computation system that can evaluate Equation 2. Each anchorage is represented by a light orange circle. The individual tracks, along with their directionality, are indicated by orange arrows. Each track is labelled at its starting anchorage by the name of the walker that walks along it.

enzyme will cut is indicated by a small red arrow. The mechanism requires that the nicking enzyme does not cut at certain domains that closely resemble the restriction site. These domains contain a restriction site mismatch, where one nucleotide in the restriction site has been altered. Domains containing a restriction site mismatch are indicated with an underscore. Locations where the anchorages are tethered to the origami tile are shown with an orange dot.

The green walker is made up of a short toehold domain G_t at the 3' end and a longer primary domain G_p at the 5' end. The blue walker is similar, with two important differences. 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 G_t* "tail" domain at the 3' end. A junction anchorage is located where the two tracks intersect and acts as a transducer, whereby the blue walker can convert the junction anchorage from a normal anchorage to a trap that will block the green walker.

If the green track is unblocked (i.e. the blue walker has not arrived at the junction anchorage) then the green walker steps onto the auxiliary $[G_t{}^* \; G_p{}^* \; B_p]$ strand that is hybridized to the junction anchorage. The auxiliary strand resembles a normal green track anchorage (Figure 7). The restriction site for the nicking enzyme is completed, the junction anchorage is cut, and the green walker steps on as normal. If the blue walker arrives at the junction anchorage first, it binds to the B_t* toehold domain on the junction anchorage and displaces the auxiliary green anchorage from the junction (Figure 8). This strand then diffuses away in solution. The 3' tail on the blue walker also displaces the G_t* domain in the $[G_t{}^* \; \underline{G_p}{}^*]$ trapping strand. When the green walker arrives at the junction, it binds to the toehold on the trapping strand that was exposed by the blue walker. The green walker hybridizes to the trapping strand and displaces it from the junction anchorage. A restriction site mismatch in the trapping strand makes the trap-green walker duplex inert and the duplex diffuses away in solution. Hence, the green walker has been blocked and removed from the track.

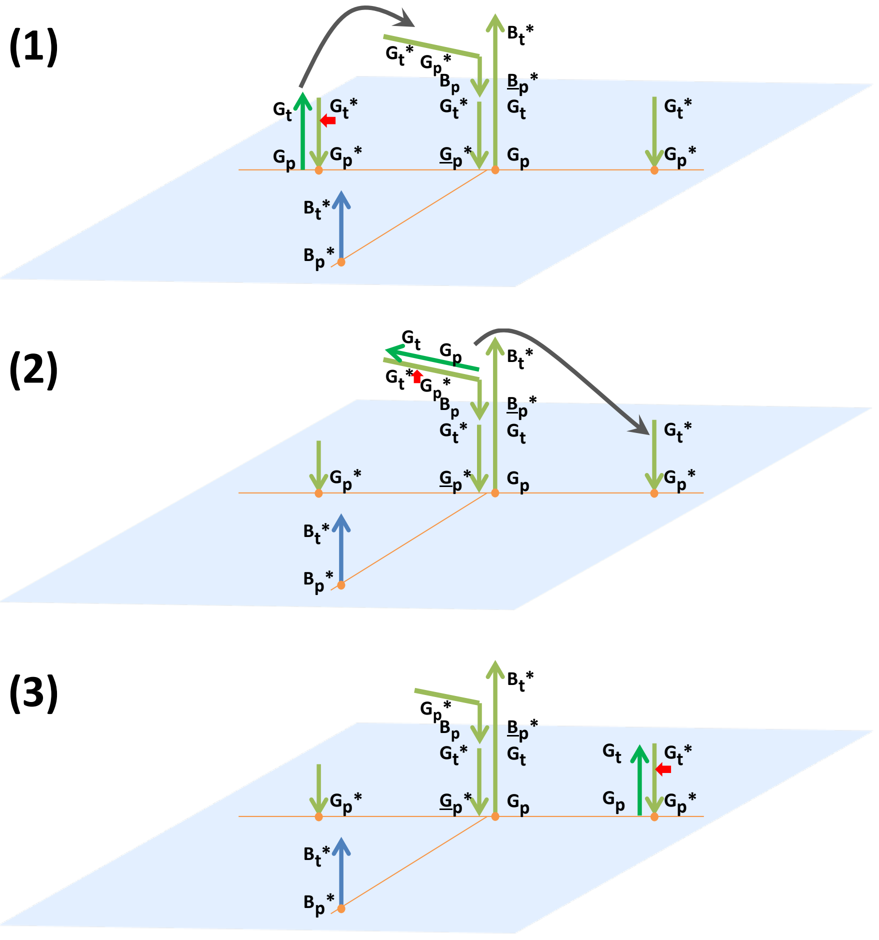

Fig. 7: The green walker arrives at the junction first. It can move through the junction and proceed down its track.

4 Conclusions

We have shown that localized DNA computation circuits can be analyzed as distributed systems. A propositional formula, a choice of gate design, and an error tolerance are enough to determine the geometry of the track system. There are simple blocking mechanisms for junction anchorages at the intersection between tracks that make such systems realizable.

The software and theory developed here can be improved upon by using it together with previously developed tools. One challenge in designing any DNA circuit is preventing unwanted interactions between domains. As the circuit gets more complex, these leaks become more and more difficult to identify by hand. Microsoft DSD

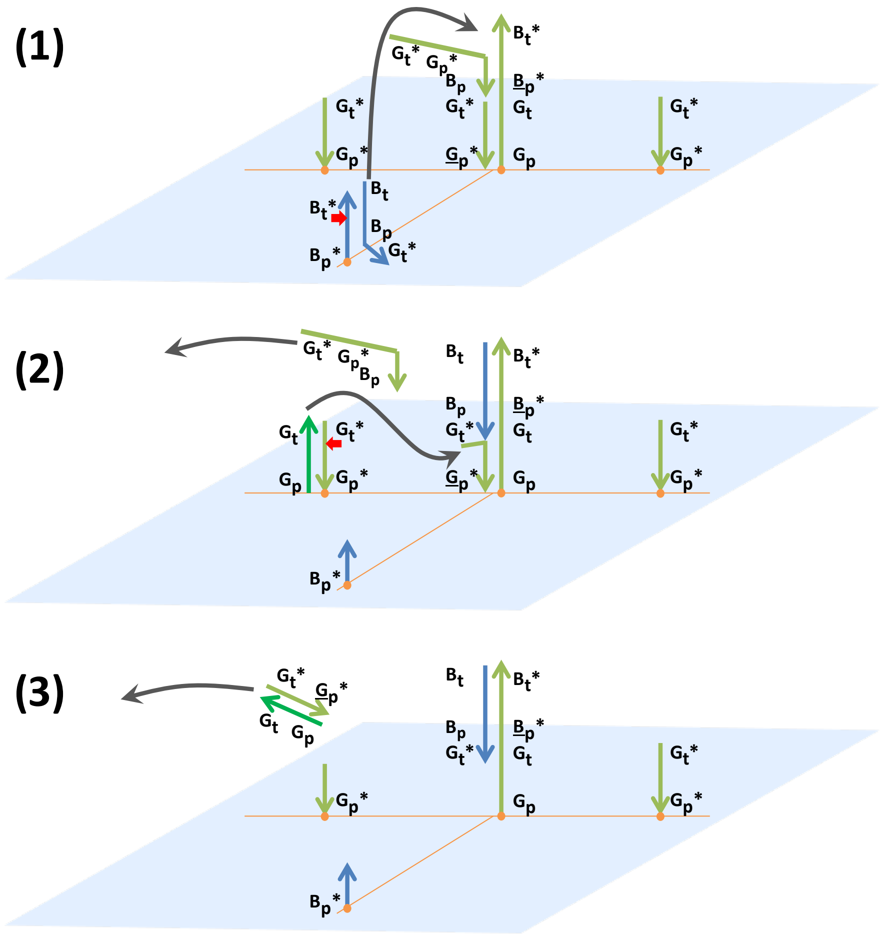

Fig. 8: The blue walker arrives at the junction first, exposing the toehold of a trap for the green walker. When the green walker arrives, it is trapped and the track is blocked.

already has automated means of detecting such leaks at the domain level. At the sequence level, NUPACK is an easy-to-use tool that can find suitable nucleotide sequences for a given design [12]. The three pieces of software can work together to first design a localized DNA circuit using the methods described above, then compile Microsoft DSD code to check for errors at the domain level, and finally compile NUPACK code to generate nucleotide sequences for each strand.

A challenge of working with DNA localized computation systems is making the system compact enough to fit on an origami tile while maintaining a low probability of missed chance error at the junctions. We can imagine a logic gate on an origami tile that, if it evaluates to **1**, activates a messenger strand that can set another walker stepping on a different origami tile. Such a mechanism can help alleviate the compactness issue. It would also increase the ability of localized DNA circuits to scale up and work with a higher reliability.

GitHub Repository

The software written to generate track designs is open source and freely available via the GitHub clone URL `https://github.com/MBoemo/DLCC.git`.

Acknowledgements

The authors would like to thank Andrew Phillips (Microsoft Research Cambridge), Marta Kwiatkowska (Oxford Computer Science), Alex Lucas (Oxford Physics), and Jonathan Bath (Oxford Physics) for guidance and helpful conversations.

References

1. Bath, J., Green, S.J., Turberfield, A.J.: A free-running DNA motor powered by a nicking enzyme. Angewandte Chemie 117, 4432–4435 (2005)
2. Benenson, Y., Gil, B., Ben-Dor, U., Adar, R., Shapiro, E.: An autonomous molecular computer for logical control of gene expression. Nature 429, 423–429 (2004)
3. Dannenberg, F., Kwiatkowska, M., Thachuk, C., Turberfield, A.J.: DNA walker circuits: Computational potential, design, and verification. In: Soloveichik, D., Yurke, B. (eds.) DNA Computing and Molecular Programming, Lecture Notes in Computer Science, vol. 8141, pp. 31–45. Springer International Publishing (2013)
4. Joyner, D., Čertík, O., Meurer, A., Granger, B.E.: Open source computer algebra systems: SymPy. ACM Commun. Comput. Algebra 45, 225–234 (2011)
5. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. Proc. 23rd International Conference on Computer Aided Verification 6806, 585–591 (2011)
6. Lakin, M.R., Youssef, S., Polo, F., Emmott, S., Phillips, A.: Visual DSD: A design and analysis tool for DNA strand displacement systems. Bioinformatics 27, 3211–3213 (2011)
7. Lakin, M., Petersen, R., Gray, K.E., Phillips, A.: Abstract modelling of tethered DNA circuits. In: Murata, S., Kobayashi, S. (eds.) DNA Computing and Molecular Programming, Lecture Notes in Computer Science, vol. 8727, pp. 132–147. Springer International Publishing (2014)
8. Phillips, A., Cardelli, L.: A programming language for composable DNA circuits. J. R. Soc. Interface 6, S419–S436 (2009)
9. Rothemund, P.W.K.: Folding DNA to create nanoscale shapes and patterns. Nature 440, 297– 302 (2006)
10. Wickham, S.F.J., Bath, J., Katsuda, Y., Endo, M., Hidaka, K., Sugiyama, H., Turberfield, A.J.: A DNA-based molecular motor that can navigate a network of tracks. Nature Nanotechnology 7, 169–173 (2012)
11. Yin, P., Turberfield, A.J., Sahu, S., Reif, J.H.: Design of an autonomous DNA nanomechanical device capable of universal computation and universal translational motion. In: Ferretti, C., Mauri, G., Zandron, C. (eds.) DNA Computing, Lecture Notes in Computer Science, vol. 3384, pp. 426–444. Springer Berlin Heidelberg (2005)
12. Zadeh, J.N., Steenberg, C.D., Bois, J.S., Wolfe, B.R., Pierce, M.B., Khan, A.R., Dirks, R.M., Pierce, N.A.: NUPACK: Analysis and design of nucleic acid systems. J Comput Chem 32, 170–173 (2011)