A DNA-based Finite State Transducer

Nathanaël Aubert

A DNA-based Finite State Transducer

Nathanaël Aubert
Under the direction of Masami Hagiya
University of Rennes 1
University of Tokyo

June 5, 2010

Abstract

Deoxyribonucleic acid (DNA) computing is an emerging field wherein the biological properties of DNA are used to carry out computation. As DNA can also be used as a building material with movable parts, the combination of those two aspects allows scientists to design nanomachines easily. However, one main concerns is the lack of automation of existing mechanisms. This limits their potential applications as it is not possible to use nanomachines in a non-controlled environment – that is, almost anything that is outside of a test tube. The goal of this work is thus to design a completely automated mechanism that will reduce the amount of operations required from the user. We have designed a finite state transducer that can eliminate the need to add control strands to the solution, one of the most common external operations. Our transducer uses polymerase to generate the output described by the input strands present in the solution. As it can work at a wide range of temperatures our transducer can facilitate the design of more complex nanomachines while making the existing ones, that rely on control strands, more autonomous. Moreover, as it can generate a wide variety of outputs, our transducer can also be useful to DNA mechanisms that are computation-intensive. For example, with nondeterministic polynomial (NP)-complete problem solvers, it can generate test strands when needed instead of having all of them crowding the solution.

Contents

Introduction

The idea of using bio-molecules to carry out computation was first presented in 1973 [1], but the true groundbreaking discovery that really launched DNA computing is the resolution of the Hamiltonian path problem using operations on DNA strands by Adleman [2] in 1994. The fact that all strands are in the same environment allows for intrinsic non-determinism, along with a very high parallelism, providing an interesting computation paradigm to work with. Adleman first attempt used only basic DNA interactions with limited applications, but since then other ways to use DNA have been proposed, mainly DNA origami and DNA actuators. DNA origami refers to a folding method of DNA to make it fit a predefined form, allowing us to use nucleotides as a building material. Authors in [3] produced, for instance, smiley faces as a proof of concept. DNA actuators are specific DNA structures that can change shape depending on their environment. The combination of DNA origami, DNA actuators and DNA computers allow us to design nanomachines with DNA, such as tweezers [4] and walkers [5]. Already we can see the diversity of DNA nanomachines in review articles such as [6].

One of the main concerns regarding DNA computing is the lack of automation of existing mechanisms, which limits their potential application as it is not possible to use them in a non-controlled environment. The goal of this internship is thus to design a completely automated mechanism that will reduce the amount of operations required from the user. Adding some control strands to the solution is a common external operation that can be effectively replaced by a finite state transducer which generates such control strands when needed.

Another possible application of this transducer is related to solving NP problems by testing every solution. A given amount of solvent can only contain a limited amount of DNA strands – depending on their length, around 10^{20} strands per liter – which is usually enough for most applications, but with NP problems this limit is easily reached. One way around this problem is to generate a first batch of solutions using our transducer, test them, then destroy them and generate the next batch, and so on. In this situation, automation is not our primary goal, so we allow ourselves all possible DNA operations.

During this internship, various ways to realize such a finite state transducer have been considered, taking advantage of different aspects of DNA, such as DNA tiling (tiling is a formal model of crystal growth that has been proven to be Turing-complete). In the end, the model we chose uses enzymes to perform extended operations. Moreover, we have chosen our transducer to be a Moore machine – meaning that output is function of the state alone – for the sake of

simplicity[1].

However, to generate quickly the output in enough quantity to be usable it is necessary to have many instances of the transducer in the solution (for instance 10^{12}). All of these instances can be seen as independent threads and, in the case of generating control strands, it is critical that they generate the same strands at approximately (DNA is less strict than silicon) the same time. The way we solve this problem is the operation *join*, as described in [7].

In the first section of this report, we review some biology notions necessary to understand the sections that follow. In section 2 we go through some related works, such as Shapiro's finite state automaton [8] and Seeman's transducer [9]. Section 3 describes two motivating examples, showing possible applications of our transducer. Section 4 describes some attempted models that, although were ultimately discarded, served as a basis for this work. Section 5 describes the final finite state transducer model. Section 6 goes further in explaining the implementation of our transducer. Finally, section 7 gives the results of our simulations and experiments.

1 Using DNA

DNA features a certain variety of operations, which can be used for multiple purpose, such as computation. In this section, we will first cover the basic biochemistry of DNA, as a basic understanding of it is necessary for the following. Then, we will see in a more detailed way the operations that are used in the rest of this report. Numerous sources such as [10] and [11] present such operations in a more extensive and general-use oriented way. Wikipedia [12] and biology books are also a good starter to understand the underlying mechanisms – even if such knowledge is not necessary for the scope of the present paper.

1.1 DNA chemistry

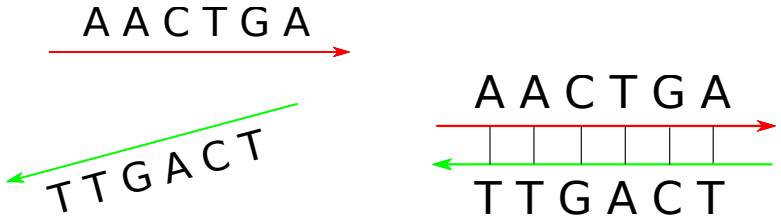
A DNA strand is an oriented string of nucleotides which are tied together by a backbone of sugar and phosphate groups. One of its ends is called $5'$ (five prime), and the other $3'$ (three prime). The nucleotides are the basic elements that define a DNA sequence. There are four different nucleotides: adenine (abbreviated A), cytosine (C), guanine (G) and thymine (T). T is complementary with A, and C with G.

A string of nucleotides, for instance AACTGA, is called a sequence. A sequence is usually written from the $5'$ end to the $3'$ end. A sequence consisting

[1]Simplicity is important in DNA computing, as everything tends to follow Murphy's law – anything that can go wrong will go wrong.

of the complementary nucleotides of another sequence in the *reverse* order is called the complementary sequence. For instance TCAGTT is the complementary sequence of AACTGA. It has to be in the reverse order, because when two DNA strands bind together – see the definition of annealing below – the $5'$ end of one of them is bound to the $3'$ end of the other, as shown on figure 1(b).

For the purpose of abstraction, a sequence is usually represented as a lower case letter. There are various conventions about its complementary sequence, as seen for instance in [2] and [13]. In this article, the complementary sequence will be represented as the same letter in upper case. The decision of where to start and where to end a sequence – after all, a strand can be considered as only one long sequence – depends on their purpose and interaction with other sequences, as shown in figure 2. The abstraction in figure 2(c) is interesting, for instance, if there is also a strand C Y in the solution, as it may bind to the red strand without completely removing the green strand.



(a) Two single strands of DNA.

(b) Double-stranded DNA.

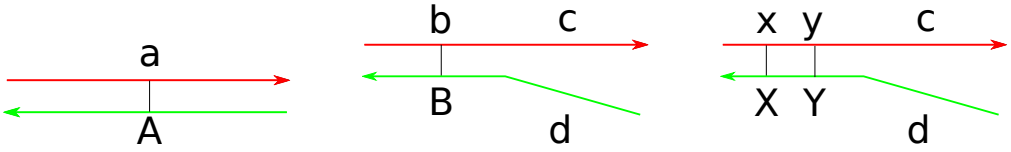
Figure 1: Example of annealing: two complementary strands bind together. The orientation of each strand, from 5' to 3', is represented by the arrow.

1.2 DNA-only operations

There are only two thermodynamically viable operations that DNA alone can perform: annealing and branch migration.

Annealing: Annealing is the simplest DNA operation, namely the binding of two complementary strands, as shown on figure 1. This operation is also called hybridization.

Usually annealing is performed between two different strands, but, if is is long enough and has some complementary subsequences, a strand can anneal to itself, forming what is called an *hairpin*.

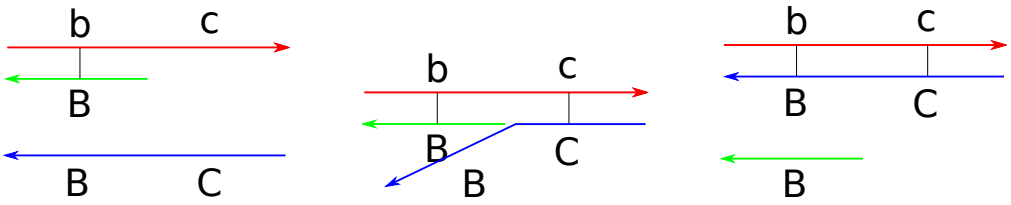


(a) Simple abstraction of the previous example

(b) Simple abstraction of two partially matched strands

(c) Another possible abstraction

Figure 2: Abstraction of DNA sequences



(a) Before the migration

(b) First step: annealing with the toe-hold

(c) After the migration

Figure 3: A simple example of branch migration

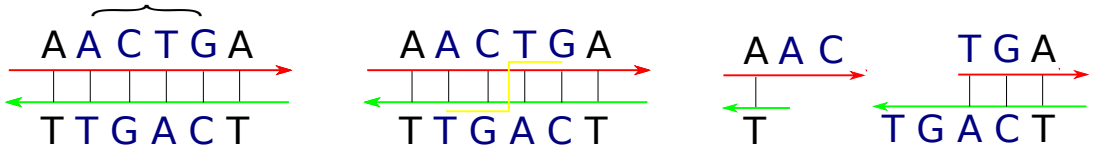
Branch migration: Branch migration occurs when a sequence bound to its complementary sequence "migrates" from the complementary one to another complementary one, as shown on figure 3. This operation alone is not very effective, so completely double-stranded DNA can be considered stable. This is why strands that are supposed to perform branch migration are designed with a *toe-hold*, for instance c in figure 3, a short sequence supposed to bind only with the third strand, so it will be in close contact, which triggers the branch migration.

1.3 User-requiring DNA operations

Some other operations can be performed by the user, allowing more complex behavior at the price of automation. Because we want our transducer to have an autonomous behavior, such operations can't be used. However, some mechanisms that we refer to use them, so knowing them is important.

Melting: Melting is the inverse operation of annealing, namely the separation of double-stranded DNA into two single strands. The most common way to do so is by heating the solution, thus the term "melting".

Another common way to separate DNA is to wash double-stranded DNA in double distilled water. The operation name remains the same, even



(a) The recognition site. (b) Restriction characteristic. (c) After restriction.

Figure 4: An example of the way a restriction enzyme works. Here, the recognition site is showed in blue. As shown on the second picture, restriction enzymes usually don't cut straight, but in a z-shape. After restriction, the two parts of DNA are completely separated.

though heating is not involved, which can be misleading.

The advantage of heating is that, since longer double-stranded DNA is more stable, by heating to an intermediary temperature, one can melt double-stranded DNA up to a certain length and leave the rest undamaged.

Merge: The simplest user-based operation. Given two test tubes, combine them, usually by pouring one into the other.

Separating by length: This operation splits one solution containing DNA into two: one with strands of a particular length, and the other with all the rest. This is achieved using electrophoresis and then extracting DNA from the gel.

Separating by subsequence: This operation splits one solution containing DNA into two: one with strands containing a particular subsequence, and the other with all the rest. It can be done by different ways, for example by introducing the complement of the subsequence at the end of which a magnetic bead can be added. This operation is sometimes called "extraction".

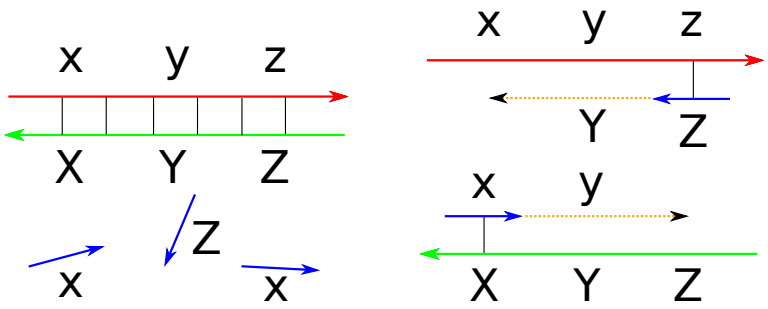
1.4 Enzymes

Enzymes add a bit of flavor to the DNA operations. The following is a list of the enzymes that are used in our work.

Restriction enzymes: Enzymes from this family recognize a particular double-stranded sequence and cut it as shown in figure 4. The recognition site and the cut form are dependent on the particular type of enzyme. Restriction enzyme can only be applied to double-stranded DNA.

(a) Two strands held together. (b) Only one strand.

Figure 5: This figure shows the way ligase works: when it comes accross two strands that are kept close to each others, it replaces the $3'$ and $5'$ ends that are in contact by a standard DNA backbone. Note: ligase itself is not represented on those figures.



(a) The template with some primers (b) The polymerase replicating strands.

Figure 6: Those are the two main configuration during PCR. First, the templates are melted and the primers (blue strands) anneal. Then, polymerase extends them until there are two templates. This will cause an exponential growth of their number.

Ligase: This enzyme concatenates two strands which are kept next to each other by being annealed to a third strand as show in figure 5

Polymerase: Polymerase is an enzyme that replicates DNA by creating the complementary strand of a single strand in the solution. However, polymerase can't work on purely single-stranded DNA, so a small part of it should be double-stranded[2]. Polymerase attaches to the $3'$ end of the double-stranded zone and extends it in the $3'$ direction, as shown in figure 6(b)

Nuclease: Enzymes from this family are used to completely destroy DNA. Depending on their type, they can be applied only to single or double stranded DNA.

[2]the (usually) small sequence used for this purpose is called *primer*.

2 Related work

In this section we go through other mechanisms that have inspired and/or are used in our transducer.

2.1 Logical gates

An important application of DNA, first proposed in [14], is to use it to design logical gates (AND, OR, XOR, ...), based on their hardware counterparts. With inputs and outputs made of DNA, these gates enable communication between different components in a given test tube. The most common implementation is to choose a particular sequence as input for a certain gate; the presence of this sequence means `True`, its absence `False`. Figure 7 shows the way an AND gate from [15] can be implemented. The most interesting feature of this design is that it relies only on basic DNA operations.

The problem with this design is that each input needs to be made unique with a different sequence, to avoid unwanted reactions between an input and a gate other than its destination. Furthermore, using this convention, NOT gates need to be the first to interact, using De Morgan's Laws to change a logical formula. Indeed, if a NOT gate is used at a depth deeper than 1, it will release its output because no input reaches it. Since the operations are non-reversible, the computed result will be wrong. Another model presented in [16] removes that problem by encoding both `True` and `False` with different sequences. In that case, another problem arises: for each input two sequences need to be designed instead of one, which increases the risk of error. Some software , such as [17], exists to facilitate sequences design.

It has also been shown that it is possible to perform some operations related to thread management, such as *join* and *fork* using DNA gates [7]. The main idea is that an AND gate is not different from a *join* between the threads that generated both input signals. The output is then, in this perspective, a signal that allows the computation to continue. If the *join* is performed between more than two threads, the gate is simply a n-way AND gate. However, the gates used in this fashion are a bit different from the previous non-reversible one. If no care is taken, the computation may fail even if no thread is blocked: since there is a large quantity of structures in the solution, the signal from each thread may anneal to a different *join* gate, leaving the threads waiting for each other. To prevent this, *join* gates are made from reversible AND gates, which means that a signal can still separate from the gate as long as the computation has not yet finished – *i.e.* as long as both inputs are not yet annealed. In our work, since many of our transducers are present in the solution at the same time, *join* gates can help by keeping them at the same pace, so that they won't

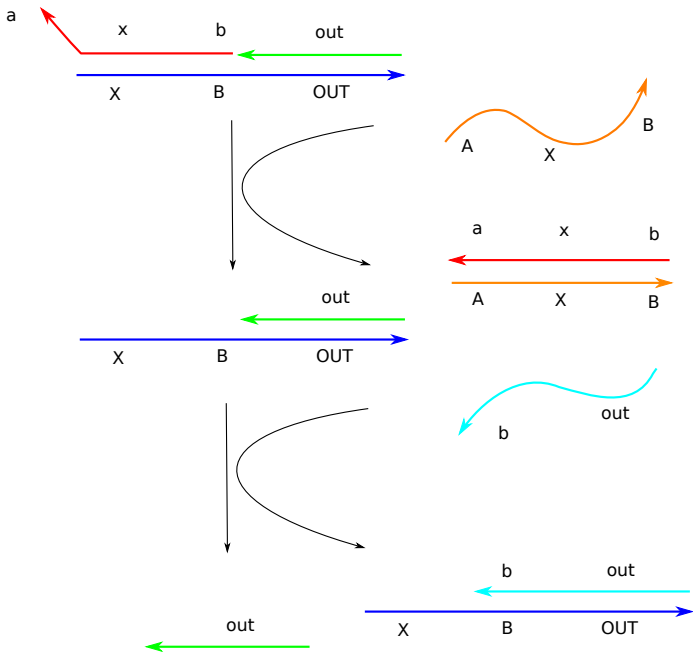


Figure 7: The two input strands are required to release the output strand. The a and b sequences act as toe-holds.

generate different, and possibly contradictory, signals at the same time.

2.2 DNA catalysts

In DNA gates, DNA plays the same role that electric or digital circuits serve in more conventional computers. For this reason, it is clear that DNA computing also needs means to amplify signals. This can be achieved using catalyst gates, as shown in [18]. Catalysts are molecular species that help a chemical reaction without being modified by it. This means that the same catalyst can help in multiple reactions. In the scope of DNA computing, the most important reaction to assist is the signal generation. The catalyst works by turning a template as well as a generic input in the solution – such as fuel strands – into the desired output. Depending on the catalyst, the template may be reusable or not. In their work [18], Zhang *et. al.* describe a very simple enzyme-free catalyst – actually a short single strand – that releases two outputs from the substrate they where annealed to. It relies heavily on the fact that hybridized short sequences tends to spontaneously separate.

Because Zhang's catalyst is enzyme-free, it has to consume the template – the substrate containing the two outputs – but, because we don't have this restriction, we were able to design an enzyme driven catalyst gate that have reusable templates [19] and can match more precisely the needs of our transducer.

2.3 Seeman's PX-JX$_2$ switch and finite state transducer

In his article [20], Seeman presents a DNA structure that can switch between two stable conformations with the addition or removal of control strands – one strand for each conformation. Without control strands the structure oscillates between the two. The two conformations only differ by half a revolution one end of the structure, as shown in figure 8. Such a structure thus allows the rotation of some parts of a more complex structure, which is a useful asset for a nanomachine.

Using this switch Liao and Seeman designed a simple finite state transducer [9]. They use a structure containing two switches, which has in total 4 different conformations. This mechanism can be assimilated to a simple finite state automaton in which it is possible to go from any of the four states to any other (Fig. 9). Using rigid helper DNA structures, one side of the transducer is thermodynamically favored. The state of the transducer is defined by the control strands in the test tube. Depending of the rotation of each of this

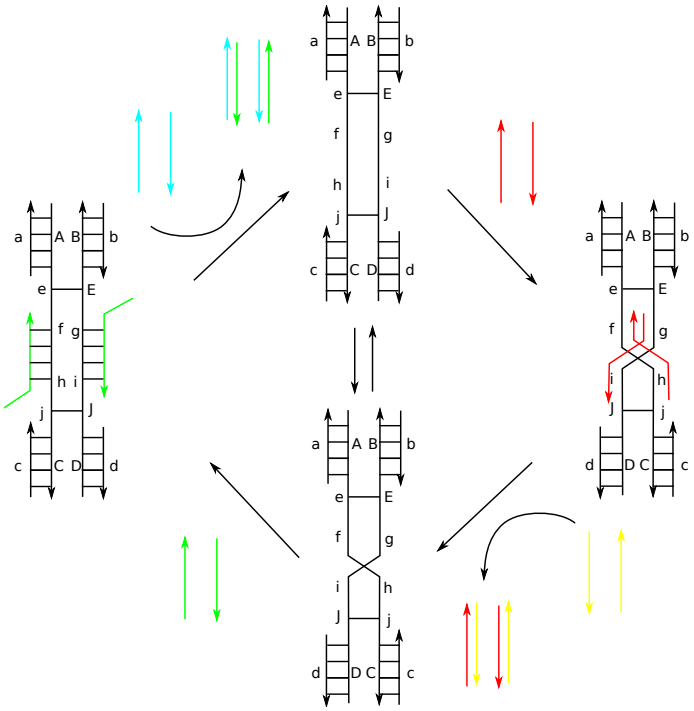


Figure 8: The PX-JX$_2$ switch.

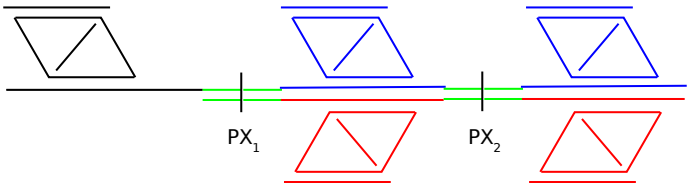


Figure 9: A transducer with two switches, both in PX position. Helping strands then anneal to the diamond structures and fill the gaps.

switches, different helper structures will be on the favored side. Output will then be assembled using these structures and some other helper strands. Helper strands are used for the sake of genericity: the main mechanism remains the same, but by changing the helping strands one can configure arbitrarily long outputs. The benefit of this transducer is that the output sequence doesn't have to be related to the input. However, the size of the output is bounded to 2 sequences –one defined by each switch– regardless of their length.

2.4 Shapiro's finite state automaton

In [8], a finite state machine is described where both the transition rules and the input are implemented using double-stranded DNA. The interesting part of this research is that the current state of the machine and the next inputted character are encoded in the sticky end of the input.

The computation takes place as follows: first, the input anneals to the appropriate transition rule. Each transition rule contains the recognition site of the *Fok*I restriction enzyme. Since this enzyme cuts at a location 10 nucleotides down this site, the new state is at this location, which depends on the number of nucleotides in the transition rule. This also means that the transition rules are unaffected by the presence of the restriction enzyme until they are activated. The drawback of this method is that the different states are differentiated by length alone, which allows only as many states as the number of nucleotides passed over by the restriction enzyme.

In Shapiro's work, a particular sequence marks the end of the input, so different double-stranded DNA, called output-detection molecules, can anneal to the sequence, depending of the final state. They are different in size, so they are easy to differentiate using electrophoresis. Because the ending sequence is very long, the final result is the longest double-stranded DNA string in the solution, which makes it easy to find.

Shapiro's automaton consumes both input and transition rules as it uses them. Because they are not reusable, this automaton is not suitable to be

used as the central part of our transducer. However, we can use it to compute the "address" of the next inputted character, i.e. to "add one" to the current pointer.

2.5 Self-assembly

Self-assembly is a completely different approach in DNA computing: it regards DNA as a building material instead of mere computational support. Indeed, DNA can self-assemble into arbitrarily complex structures thanks to its annealing property. A single strand of DNA can be locally (over a distance of a hundred nucleotides) considered as rigid, but can still bend provided the sequence is long enough. Moreover, double-stranded DNA is much more rigid, so annealing can be used as a bond between different parts.

These properties allow the construction of bi- or tri-dimensional structures. Some of these structures can be seen in [21] or [22]. They also allow the creation of specific material, like DNA crystals – a motif replicated in every direction. Although crystals are interesting, they have limited application –mostly, building simple shapes, such as surfaces, nanotubes, etc. More possibilities come from the bending property as it makes it possible to build arbitrary bi-dimensional structures: a long DNA strand is bent and kept in a specific shape by annealing to "staple" strands. This method, called DNA origami, has been described in [3].

In our case, the most interesting application is DNA tiling. Tiling, as a generic way to perform computation is well known to be turing-complete. Because of this, Winfree designed a DNA implementation of tiling [23]. DNA strands are designed to self-assemble into DNA tiles, which then in turn self-assemble. This can be achieved by a variety of means. For instance, the solution can be set to a high temperature where tiles can self-assemble but bonding between two tiles is unstable. The solution is then cooled, and computation can take place. It is useful to create tiles before computing. Otherwise, partially assembled tiles may assemble in unwanted configurations –configurations that wouldn't be accepted if the tiles were complete before assembly– and thus block computation. Another way to avoid this complication is to first let tiles self-assemble in separate solutions, and then merge them. [24] gives a more in-depth summary of DNA tiling computation.

2.6 Whipslash

This machine, detailed in [25], is made of a very long DNA strand. Its body contains the transition rules, and its state is described by the last sequence. Because the sequence is long enough, the structure can bend and hybridize to

itself, forming an hairpin structure. After the state hybridizes, the polymerase enzyme appends the next state to the end of the strand. Between transition rules are stop sequences that block the polymerase. They are designed in such a way that the polymerase would need to add elements that are not in the solution. The strand then separates and does the next transition. The processing time of this structure is independent of its concentration in the solution, as it anneals to itself.

There are some major drawbacks to this design: first, the current state is as likely to hybridize with the end of a transition rule –for instance the one that generated it– as it is to hybridize, as desired, with the beginning of a transition rule. A correction of the model has been proposed in [26], where transitions are disabled after being used, but that triggers different problems, like the sheer fact that a given instance of a transition rule can be used only once. The second problem is a thermodynamic one: the melting step –the last one of the cycle– can't happen without user intervention, such as heating. Having external operations is a problem because we want to create a completely autonomous machine.

3 Motivating examples

In this section, we show some possible applications of our transducer, both from the automation and the generation perspective.

3.1 Automating Seeman's PX-JX_2 switch

As said in the related work section, Seeman's PX-JX_2 switch is an important asset for DNA nanomachine design. By making it more autonomous, our transducer could improve the ability of this switch to integrate with other mechanisms.

It would be possible, for instance, to get a transducer with four states: two generating the PX control strand and the JX_2 control strand respectively , and two generating their complementary strands– thus putting the switch back in the neutral position. The transducer can also be very simple and loops through all those states so the switch passes regularly from one conformation to the other. This would be even more critical with more complex actuator, such as the three states structure presented in [27]

Strict timing can hardly be controlled with such a simple mechanism, but in most applications, it's the actions order that is important, rather than the time at which they are executed. Some kinds of timing can still be easily implemented by adding some "ghost" states between the generation of the two

control strands. For instance, if we add four states that don't generate anything to the previous transducer between the generation of the PX control strand and its complementary strand, the switch will stay, on average, longer in the PX configuration than in the JX_2 configuration.

3.2 Sequence generation

Our transducer can also be used to generate strands for computation intensive mechanisms. The intrinsic non-determinism of DNA computing helps to solve NP-complete problems, since virtually all possibilities can be tested at once, for instance with 3-SAT [13]. However, there is a material limit, as every possible solution needs to be generated first. Furthermore, there is also a limit to the number of DNA strands that fit in a given amount of solution (depending on their length, between 10^{18} and 10^{20}).

Using a finite state transducer to generate the strands can solve those two problems. For example, if we have a mechanism testing strands made of 4 different sequences, each of which is one of two different possibilities, we have in total 16 different strands. If we use a transducer to generate them, it needs only two states (one for each kind of sequence) and 4 transition strands (meaning that we can transition from any state to any other, including the current one, so every possible strand can be generated). This means that we have to generate, in total, only 5 different strands. The difference gets larger as the solutions to the problem get longer.

On the other hand, by limiting what transition strands we use, we can only cover partially the solution space, limiting the amount of strands in the test tube. If none of the generated strands solve the problem, we can "clean" the tube with nuclease targeting specifically those strands and the transition strands (or simply start all over again) and use different transition strands to explore the next part of the solution space.

We can note that this application constitute a tradeoff between time and space. In a conventional computer, we would be referring to memory space, whereas in DNA computing we refer to the number of strands that fit in a given solution. Usually, DNA computing relies heavily on space to be as fast as possible, but this kind of model has its limits. Our transducer can "convert" part of the space at the cost of time, making those mechanisms more feasible.

4 First attempts

The first attempt to create a finite state transducer was strongly inspired by computer science models. It was an attempt to use DNA strands to directly

encode the current state, inputs, etc. and to use DNA operations to have them interact.

This attempt relied heavily on proxy branch-migration – a branch-migration occurring not with the closest sequence but with the next one or even further – requiring a "jump". Because this kind of branch-migration proved extremely ineffective – it is slow in comparison to other possible reactions such as polymerase extension – the proxy model was judged impossible. However, this early version formed the basis of our work, in such elements as breaking down the structure into different specialized substructures, using the polymerase enzyme to easily generate specific sequences from a generic – and thus easily replaceable – buffer of nucleotides and using specific sequences ("pointers") to get the next input.

In a second attempt, we tried a very different approach, using DNA tiling (Fig. 10). The tiling was designed to grow in two directions. One direction represents the consecutive states of the transducer, while the other is used to carry out computation. The line on one side is made of the input and at the end of the computation the output strands are generated by polymerase from the line on the other side (Fig. 11). This approach is interesting because the computation for different output characters happen at almost the same time. Moreover, once computation is achieved, outputs can be generated multiple times from the same structure. However, for very long inputs, the resulting structure will be very large, as at least two tiles are required for each character[3], making the structure unstable. Another problem comes from the fact that in this model it is not possible to change the input once the computation starts, so this mechanism does not adapt in changing environments. Finally, the generated output is hard to use, since it is completely annealed to the transducer.

5 Overview of the current version

Like the first version, this version is strongly inspired by computer science designs. In the same way that, in computer science, complex mechanisms such as computer are actually made of a lot of very simple part the current version of the transducer is composed of different substructures which have all their specific purpose. By relying on simple parts, it gets easier to verify that they work as expected and that they interact correctly together.

[3]Verification tiles can be added, making wrong structures less probable.

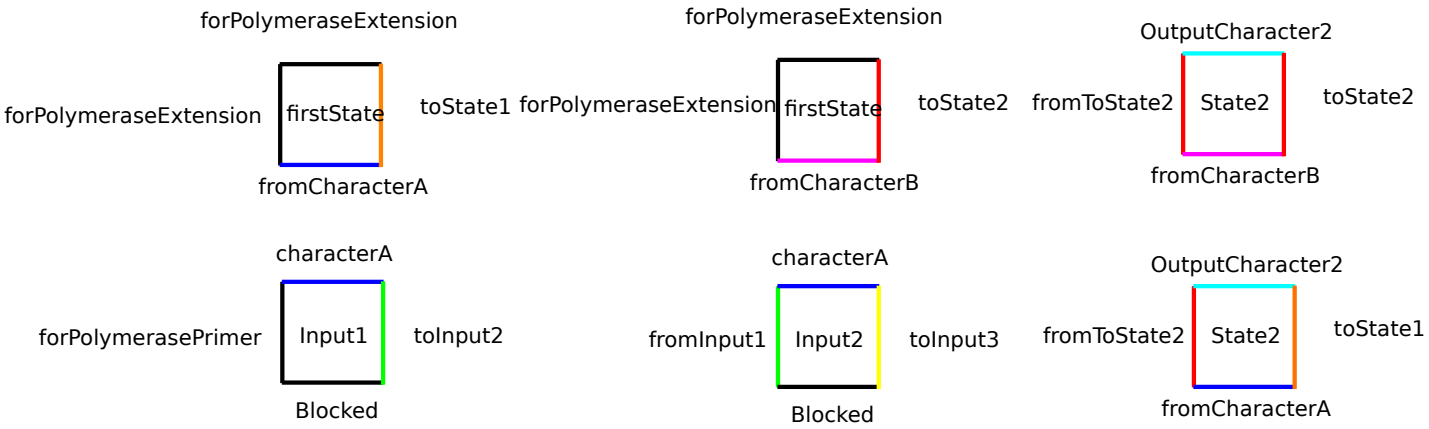


Figure 10: Partial instance of a tile set for a transducer and its input. Note that this tile set is minimal, as the state tile is directly used to generate the output. However, since incorrect partial structures can form, this means incorrect outputs may also be found in the solution.

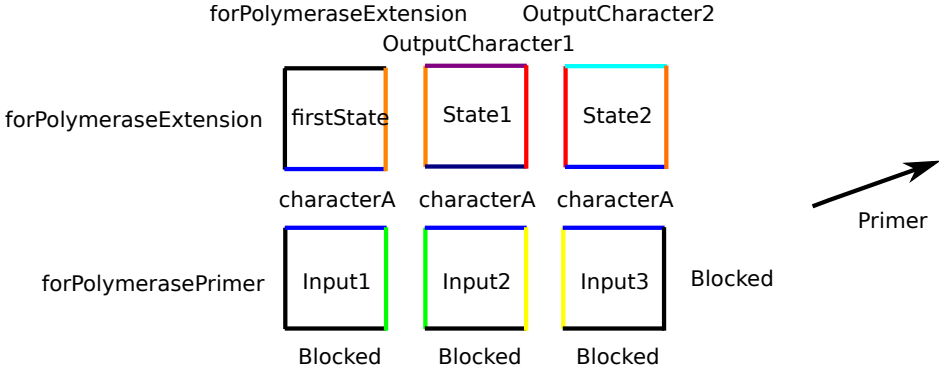


Figure 11: Correct computation of the output. The output strand is generated by the elongation of a primer annealing to a side of the mechanism, which means that the output actually starts with a header sequence.

5.1 Structure

Our transducer is made of the following parts and strands.

state strand: This strand forms the core by storing the current state of our transducer. As the other parts are shared, this strand can be assimilated to the whole mechanism, even if it is useless by itself. This strand also contains a pointer to the next input character to read.

transition function: It maps every combination of input characters and states into the next state of the transducer. It is actually composed of numerous single strands, one for each possible combination. It is possible to have a non-deterministic transducer by having more than one possibility for a given combination of states and inputs.

input: The input is made of a string of characters. Each character is represented independently by a single strand that contains the index of this character in the string as well as the sequence representing it.

output: Outputted characters are partially double-stranded DNA. Their length can be arbitrary, but we expect long outputs to be more useful, as they can have more diversity. Since they are supposed to be control strands for other mechanisms, we define each output character independently.

primers: Primers are used to start the polymerization by annealing to the targeted sequence.

pointer cap: Small non-reusable structures that are actually almost complete pointers. They are completed by being ligated to a state strand after the state update. There are – obviously – as many types of pointer caps as different pointers.

remover cap: Similar to the previous structure, they assemble with former states after they been separated from the updated ones and contain the recognition sites for the restriction enzyme that then frees the pointer to the outputted character.

Our model also uses five different enzymes (polymerase, ligase and three different restriction enzymes).

5.2 Schematic workings of the transducer

To start the process, all structures are put in the solution together. Since every strand and enzyme is there, no additional operations are needed.

The processing of each input character happens in 5 steps, as shown in Fig.**??**. First the state strand anneals to the corresponding input strand. Then

the transition function strand corresponding to both the current state and the input character anneals to those areas. Next, using the polymerase enzyme, the state strand is extended to contain information on the next state, as well as a recognition site for a far-cutting restriction enzyme. The restriction enzyme can then cut between the current and previous states. The old part is then used to generate the outputted character, while the new part is used as the next state strand. This means that one character is generated each time there is a state transition, so there is no real temporality. This also means that there is no character outputted from the initial state.

5.3 Simplified model

In case automation is not needed, we can use operations such as adding and extracting strands or melting. We can also add input characters only when they are needed. Since the input doesn't have to be completely in the solution at all times, we can greatly simplify the way state transition works. The state strand doesn't need a pointer to the next input character, as we add only the transition strands corresponding to this character. Once the state strand has been annealed to the right transition strand and extended by polymerase, we separate it and remove all the transition strands. Similar to before, the elongated state is separated into the new state and an outputted character.

This version contains fewer structures, and the remaining ones are simpler. As exterior operations also help with controlling the mechanism, this version is much more stable than the automated one. This version is obviously not designed to help automate other nanomachines, so its use should be restrained to generating strands for computation-heavy mechanisms. In that case, as explained in the motivating example, we want more complex outputs made of more than one output character, so additional structures are needed to concatenate them.

6 Wetware implementation

In this section a more detailed version of each "steps" is presented.

6.1 Polymerase-based catalyst

While working on our transducer, we had problems using polymerase in a fully automated way: when used, polymerase will completely close the strand it is working on. There is two main ways to re-open the newly created double strand, melting it or using restriction enzyme.

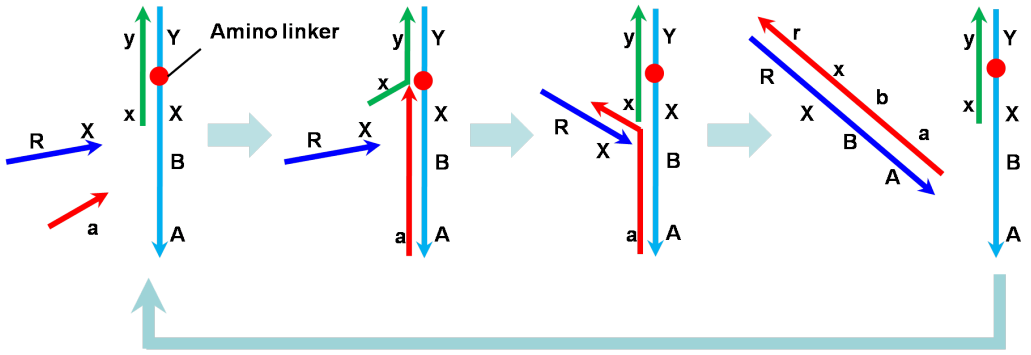


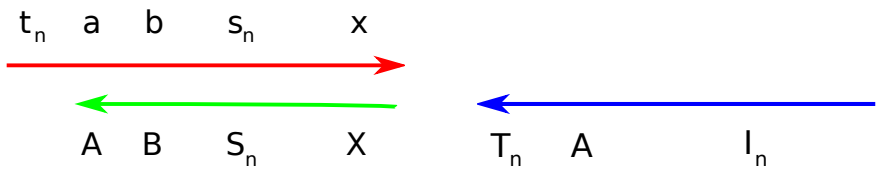
Figure 12: Polymerase based catalyst. At the beginning, the first primer hybridizes to the structure. Then, polymerase extends the primer to the position of the amino-linker. By branch migration, the 3′ end of the primer is freed, which allows the second primer to hybridize. Finally, the two primers are released as a long double strand by another intervention of the polymerase enzyme.

Melting proved impractical as it is an external operation. Furthermore, the idea of directly using restriction enzyme wasn't satisfying, as this would mean destroying the template that the polymerase used, thus reducing the life-span of our system.

Instead, we developed a polymerase-based catalyst to solve this issue [?]. This catalyst works as an intermediary for the generation of the structure by polymerase, transforming first one primer into the template and then using another primer with this new template. Because the created double strand does not contain the original template it is safe to alter, and thus restriction enzyme can be used.

Our device operates in four steps, as shown in Fig. 12, and relies on the use of an amino-linker, which can stop polymerase. This small molecule can be inserted between two DNA backbones, taking the place of nucleotides in such a way that the modified DNA strand acts normally in every aspect except that no nucleotide can anneal to the linker. Because it is small, we expect that the time overhead for "jumping" the linker will be reasonable (less than a second).

The device is left intact after its use and thus can be described as a catalyst like Zhang's catalyst gate [?]. We anticipate its use in implementing PCR without any temperature control and serving as a template to generate a given strand from a primer. In our transducer, we use this catalyst specifically to update the state strand.



(a) The state strand. (b) An inputted character.

Figure 13: Strands involved in the first step. The sequence t_n is complementary to the sequence T_n, and thus serves as a pointer to the correct character. The sequence s_n is the current state. AB is the recognition site for the restriction enzyme that re-opens the state strand in case polymerase closes it. The sequence X is a remainder of the state strand generation and is inactive. I_n is the inputted character.

6.2 First and second step: input reading

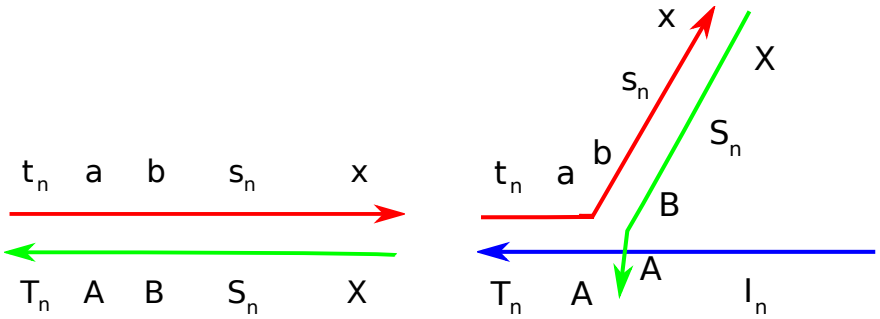
As explained in the previous section, the state strand (Fig. 13(a)) contains both the current state of the transducer and a pointer to the next input character. To prevent interaction with incorrect transition strands – strands with the correct state, but an incorrect input character – the state part is double-stranded. There is, however, a toe-hold on the pointer part, so that the state strand can still anneal to the correct input strand (Fig. 14(b)). The pointer part t_n contains a recognition site for a different restriction enzyme whose restriction area is further that the end of the current double strand, inactivating it.

The transition strand is made of our polymerase-based catalyst, with an extra sequence to anneal to the input strand (Fig. 15(a)). When both the state strand and the correct transition strand are annealed to the input (Fig. 15(b)), the continuous contact between those two structures allows for branch migration, annealing the state part to the transition strand.

The state "lock" is then removed, and stays in the solution as waste, since it cannot interact with any other structure. However, while it is still there – before step two –, it can serve as a primer for the polymerase enzyme, closing completely the state strand (Fig. 14(a)). In this case, a recognition site for a restriction enzyme is completed and is used to cut the generated nucleotides. As restriction enzymes are faster than polymerase, the structure is open most of the time. After step two, the recognition site is split between two strands and is thus rendered inactive.

6.3 Third step: state update

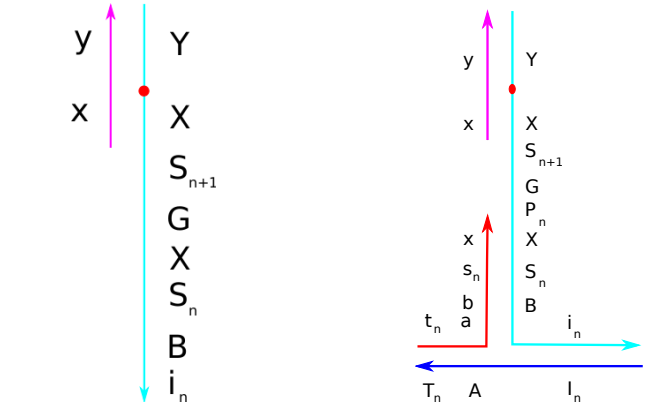
After all three structures have annealed together, the $3'$ end of the state strand can be extended following the template of the transition strand (Fig. 16(a)).



(a) The state strand closed by polymerase.

(b) After annealing.

Figure 14: The two possible outcomes of the interaction of the input strand, state strand and the environment. Note that even in the right configuration, the green strand can hybridize completely again by branch-migration, allowing polymerase to extend it into the left configuration.



(a) A transition strand.

(b) After annealing.

Figure 15: Second step. Input characters are short so that if the wrong transition strand anneals before step 1 occurs, it will be blocked, at worst, only briefly. The red disk is the amino-linker. P_n is a pointer to the output character corresponding to the current state. Because the recognition site AB is annealed to two different strands, it is inactive. In the same way, the recognition site in t_n will not activate.

This step works mostly as described in the catalyst section: polymerase is stopped by the amino-linker (Fig. 16(b)), then branch-migration releases the end of the extended state strand (Fig. 16(c)). The generic primer present in the solution can anneal to this toe-hold and generate the final double strand, separating it from the rest (Fig. 16(d)).

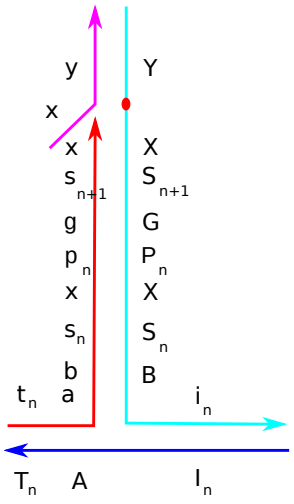
6.4 Fourth step: next pointer generation

The new double strand is then long enough to activate the recognition site present in t_n. In the third step, a long sequence is generated, between the old state and the new one, to be used for computing the next pointer. As in Shapiro's automaton, the length of t_n is specific, and the cut will occur at a place that is specific to t_{n+1} (Fig. 17(a)). Then a spare t_{n+1} hybridizes at the cut and is glued to it by ligase. By convention, the length of our pointers decreases with n. We also ensure that after reading the last input character the pointer count is reset and the transducer loops, starting all over again – and thus the transducer can be used more than once.

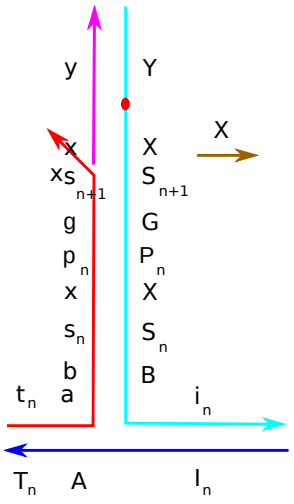
The downside of this step is that it requires a large number of additional structures, consisting of the pointer "ends". Also, an unwanted polymerization before the ligase acts can leak a state sequence in the solution, generating in turn unwanted structures. Because ligase is set in higher concentration and is faster that polymerase, the latter is infrequent, although it is not entirely avoided.

Another cause of concern comes from the transition between pointers. Because we don't want the transducer to be stuck nor for it to drop a part of the input, the exact number of inputted characters should be known beforehand. This means that we can't use this system with a mechanism where the input string length may vary. Moreover, because restriction enzyme can only cut to a finite distance, this system may not be enough for very long input strings. In that case, however, it would be possible to use Shapiro's finite state automaton in more than one step: the structure annealing after the first cut is a temporary step which continues the computation until the next pointer is decided. However, since in the current model there is no direct information on the current pointer after the first cut, some modifications would have to be made.

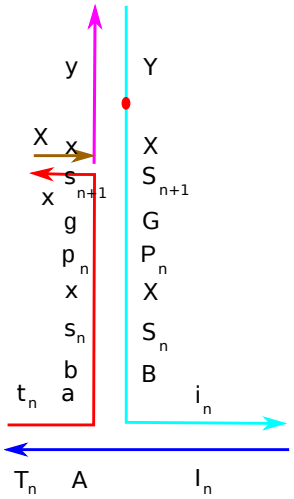
It would be possible to use a slightly modified version of our catalyst gate to generate the next pointer instead of using the caps. This would assess a lot of the previous problems: those additional structures would be reusable, there wouldn't be unwanted hybridizations and the number of inputs could vary, as the catalyst gate only needs to make the correct updater active at the same moment a new input is made active. This pointer generator could also help in



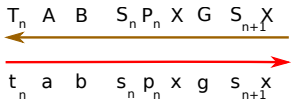
(a) State strand extention.



(b) Release of the toehold.

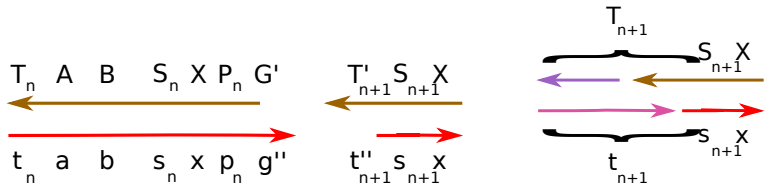


(c) Annealing of the second primer.



(d) Release of the toehold.

Figure 16: Step three.



(a) After the restriction.

(b) New state strand.

Figure 17: Fourth step. The G′ and g″ are the remains of the sequence G after the restriction. In the same fashion, T'_{n+1} and t''_{n+1} are ends of their respective sequences. The corresponding pointer cap can then complete the structure into a complete state strand.

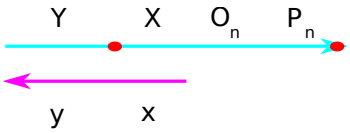


Figure 18: An output updater.

implementing the *join* operation, which is still lacking.

6.5 Fifth step: output update

The four previous steps form a complete and operational finite state automaton that is already releasing an output depending on both the current and previous states as well as the index of the current input character, making it some kind of transducer. After this, all that remains is to transform this intermediary output into the desired sequence. First, the pointer to the correctly outputted character is used as a toe-hold for hybridizing with the right output generator – a structure similar to the transition strands (Fig. 18). Then, as in step three, the strand is elongated and then removed. This step uses a different primer that contains a recognition site for a restriction enzyme, so the output character will be released. This restriction happens in such a way that the pointer actually stays with the output, so it will not incorrectly generate more output.

This mechanism is not perfect, as it has obvious unwanted interactions. For example, it is possible to anneal to a transition strand instead, generating completely erroneous structures. One workaround to this problem is to have much more output generator in the solution than transition strand. However, this fix will only limit the number of errors; it will not get rid of them.

6.6 Simplified model

The simplified model has already been explained in the previous section and relies primarily on polymerase. Our polymerase-based catalyst gate is not needed here, as we can add components when they are used so there is no need to focus on re-usability. However, as we just started implementing it, most of the details change very quickly.

7 Simulations and experiments

DNA strands with amino-linker and enzyme are expensive molecules. Also it takes months to receive a shipment of special DNA containing amino-linker. For these reasons, it is imperative to simulate these mechanisms first to be reasonably sure that they will work.

7.1 Simulations

Simulations are conducted on different mechanisms independently to guaranty that they work well. The first step is to define the length of the various sequences. At this stage, separating these sequences between "short" (around 6 nucleotides) and "long" (around 15 nucleotides or more) is enough. In the simulator, a "short" strand has a positive probability to denaturate by itself – separate from a strand it is annealed to. Of course, if a strand is made of more than one "short" strand and two or more of them are annealed, this possibility is removed. The next step is to decide what enzymes are present in the solution and consequently what information is needed regarding the way they interact with our sequences. This information includes which sequence is a recognition site, where the amino-linker is, the working temperature, etc. Then the starting configuration (mostly what strands are present at the beginning) is set and the simulation is started. It works in steps: from the current configuration, every possible DNA operation is considered. Some of them are executed at random, following kinetic considerations – namely the reaction speed and the reactant concentration in the solution. It should go without saying that this process is computationally intensive, which is another reason to test small subparts of our system instead of the whole mechanism.

7.1.1 Simulation of the polymerase-based catalyst

The first mechanism to be simulated is the catalyst designed to automate the use of polymerase in our transducer – and possibly other nanomachines (Fig. 12). We simulate how our catalyst works in combination with the two primers

for the purpose of genericity, considering that this helping mechanism can be used in other machines than our transducer.

In this mechanism, the only sequence that needs to be short is the sequence used for releasing the end of the extended first primer. The reason is simply that there is a chance that the second primer accidentally anneals to the strand of the catalyst, blocking it. As polymerase can not act in this fashion, however, this configuration does not produce any unwanted structures. The first primer can be either short or long, as this factor does not change how the mechanism works. However, primers are traditionally short, as short strands are cheaper and easier to produce. The sequence annealing the short strand of the catalyst to the long one has to be long, as it absolutly must stay annealed to it.

This simulation shows that the number of outputted structures is superior to the initial concentration of the catalyst (Fig. 19). However we can also see that it leaks an unwanted intermediary structure, consisting of a slightly elongated second primer (Fig. 19(c)). The reason for this is that after it has been slightly elongated this primer is still short. However, we can also see that its concentration decreases after a time, as this intermediary structure can still be used in place of the second primer. This means that in the end this intermediary strand will be completely consumed. If the simulation is conducted for a long enough time (over three hours of simulated time, with standard kinetic parameters), the only remaining structures are the catalyst gate and the intended output, proving that this gate works as expected.

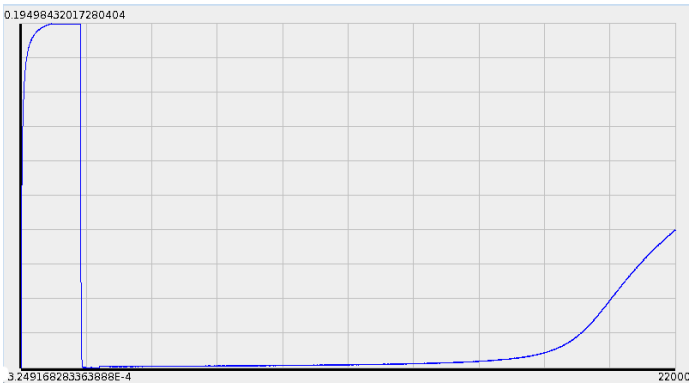
7.1.2 Simulation of steps 1 to 3

Currently, the simulator doesn't support restriction enzymes, so the only part that can be simulated is from step 1 to 3. Step 1 was slightly modified such that restriction enzyme is not required (it is, however, still required for the real mechanism): the "lock" strand is said to have an amino-linker at its end and is thus inextensible. This simulation is a success, and the results are largely similar to those of the catalyst gate alone.

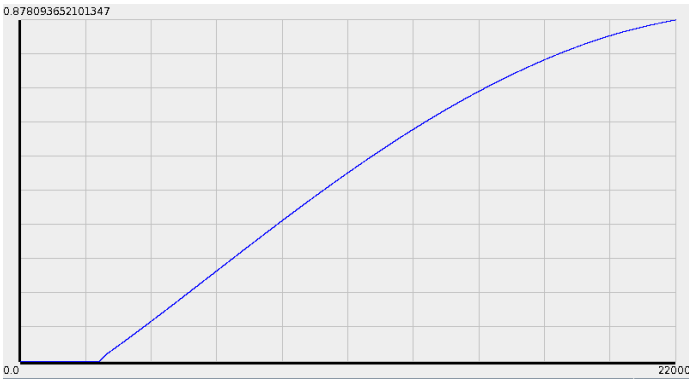
7.2 Experiments

Once the simulation with the abstract sequences has completed successfully, concrete sequences are designed, using computer assistance. Design consists of selecting the length and interactions for sequences and then use a generation algorithm that will try to minimize unwanted interactions.

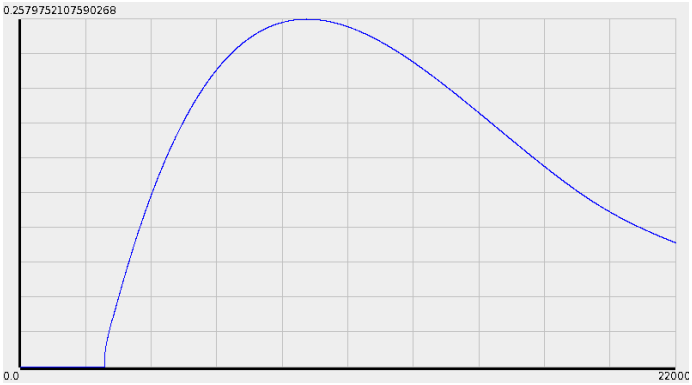
The catalyst gate experiment is the only one to have been planned at the moment, but more experiments will be conducted as soon as positive results are obtained. The sequences for this experiment are shown on table 1. Having



(a) Catalyst concentration.



(b) Output concentration.



(c) Unwanted structure.

Figure 19: Concentration of different species in the solution. The simulator is not able to deal with complex structures, so we emulate adding the complete catalyst gate by adding first the two strands it is made of, give them enough time to completely anneal, and then add the primers. This is the reason why the output and the unwanted strand don't appear immediately.

Strand name	Nucleotides ($5' \rightarrow 3'$)
primer1	CGCAAGAACAGTAGC
primer2	CTCCGAAACCCTAACAAAGGAAAGGACGGG
gateShort	CCCGTCCTTTCCTTTGTGGTGGTGTGGTAAGAT
gateLong	ATCTTACCACACCACCACAAAGGAAAGGACGGGACGACATGCTGAGAGGCTACTGTTCTTGCG
gateLong-aminoC3	ATCTTACCACACCACCAC-linker-AAAGGAAAGGACGGGACGACATGCTGAGAGGCTACTGTTCTTGCG
gateLong-aminoC18	ATCTTACCACACCACCAC-linker-AAAGGAAAGGACGGGACGACATGCTGAGAGGCTACTGTTCTTGCG
gateShort-2Aloop	CCCGTCCTTTCCTTTAAGTGGTGGTGTGGTAAGAT
gateShort-2Tloop	CCCGTCCTTTCCTTTTTGTGGTGGTGTGGTAAGAT
longPrimer1	CGCAAGAACAGTAGCCTCTCAGCATGTCGTCCCGTCCTTTCCTTT
output1	CGCAAGAACAGTAGCCTCTCAGCATGTCGTCCCGTCCTTTCCTTTGTTAGGGTTTCGGAG
output2	CTCCGAAACCCTAACAAAGGAAAGGACGGGACGACATGCTGAGAGGCTACTGTTCTTGCG

Table 1: Concrete sequences. Primer1 is the first primer to anneal to the catalyst gate. It is then extended into longPrimer1 and released by the extension of primer2. Then they together form the double-stranded structure output1-output2. GateLong and gateShort form the catalyst gate and come in different varieties. GateLong doesn't have an amino-linker and is here for control. GateLong-aminoC3 has a short – and thus weak – amino-linker that may not stop polymerase, but doesn't slow down the mechanism. GateLong-aminoC18 has a long amino-linker that will noticeably slow down the mechanism, but will stop any kind of polymerase. The two kinds of gateShort differ only by the two nucleotides used to jump the amino-linker – A and T respectively.

been prepared, the multiple mixes are left for three hours to react in different conditions – ambient temperature, 37°C, and PCR. Electrophoresis is then performed to see what happened to each of the mixes (Fig. 20).

Conclusion

During this internship, we designed a DNA-based finite state transducer and partially tested it. Different approaches were considered, but in the end a model which works similar to the usual representation of a Turing machine was chosen. However, other valid models were developed – or, in some cases, are being developed – along the way, namely the tiling model and the simplified model, which both have distinct benefits and drawbacks.

The main model has some particular limitations, as it relies on expensive materials, uses specific non-reusable parts and has a low, but non-zero output

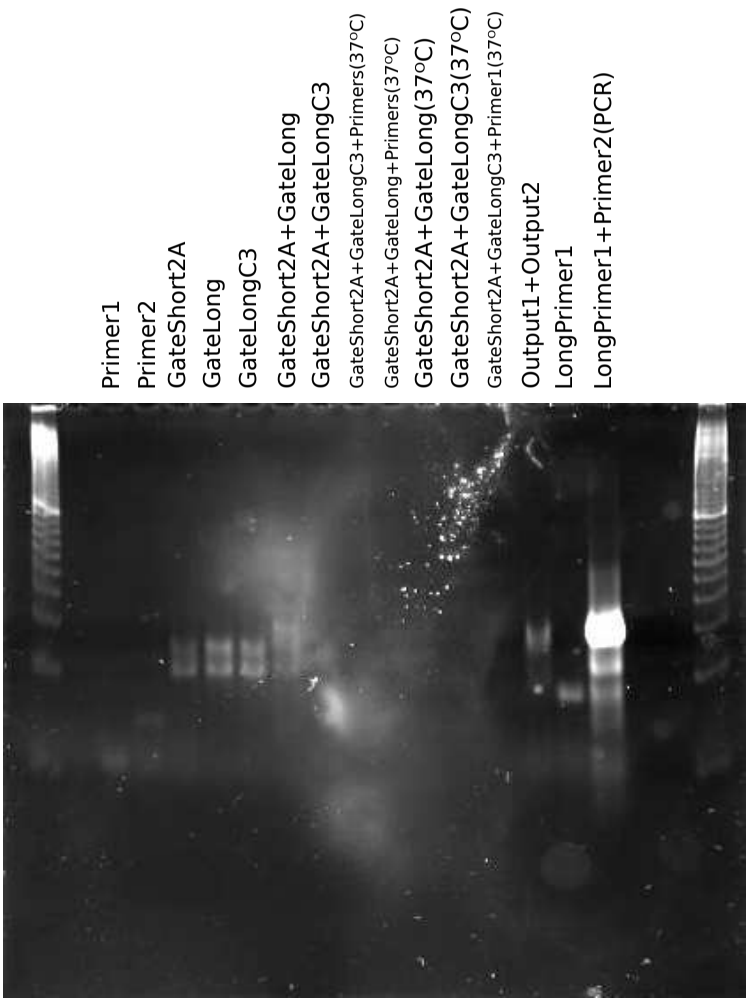


Figure 20: Electrophoresis results. Bands indicate the presence of DNA. Markers on the side are used as a reference. The lower the band, the smaller the DNA structure is. You will notice a damaged area appears in the middle of the results. The important results of this experiment should have appeared in this damaged area. Moreover, some mixes seems to have been contaminated as, for example, there is more than one band in the fourth and fifth wells. Future experiments have been planed to get usable results. Except for the PCR, each mix is made of 5μL of each species at the concentration of 50nM.L^{-1}. In the case of PCR, we use a more concentrated species ($5\mu\text{M.L}^{-1}$)

of erroneous structures. It also requires further simulations and experiments to be completely validated.

In our future work, we will try to modify the current transducer to improve its autonomy, especially reducing wastes and non-reusable parts as much as possible. We also want to add the *join* operation between two output generations. Also, different biological approaches could lead to unexpected improvements. We might also be able to use different enzymes such as CRE recombinase that can remove parts between specific tag sequences. Furthermore it is possible to use methylation of nucleotides to prevent restriction enzyme from working on those particular nucleotides while still being active everywhere in the solution. We have also considered trying different approaches still and looking for new implementations.

We also wish to extend this transducer, as it may be very easily turned into a Turing machine. DNA-based Turing machines are a particularly interesting test for design.

To conclude, DNA computing is definitively an interesting new field that has a variety of promising applications in such fields as parallel computing, nanomachine design and cellular automata.

References

[1] M. N. Vaintsvaig and E. A. Liberman, "Formal Definition of Cell Molecular Computer", *Biofizika*, vol. 18, 1973, pp. 939–942

[2] L. M. Adleman, "Molecular Computation of Solutions to Combinatorial Problems", *Science*, vol. 266, 1994, pp. 1021–1024

[3] P. W. K. Rothemund, "Folding DNA to create nanoscale shapes and patterns", *Nature*, vol. 440, 2006, pp. 297–302

[4] 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*, vol. 406, 2000, pp. 605–608

[5] J.-S. Shin and N. A. Pierce, "A Synthetic DNA Walker for Molecular Transport", *JACS*, vol. 126, No. 35, 2004, pp. 10834-10835

[6] J. Bath and A. J. Turberfield, "DNA nanomachines", *Nature nanotechnology*, vol. 2, May 2007, pp. 275–284

[7] L. Cardelli, "Strand Algebras for DNA Computing", *Proceedings of the 15th International Meeting on DNA Computing and Molecular Programming*, 2009, pp. 119–128

[8] Y. Benenson, T. Paz-Elizur, R. Adar, E. Heina, Z. Livneh and E. Shapiro, "Programmable and autonomous computing machine made of biomolecules", *Nature*, vol. 414, 2001, pp. 430–434

[9] S. Liao and N.C. Seeman, "Translation of DNA Signals into Polymer Assembly Instructions", *Science*, vol. 306, 2004, pp. 2072–2074

[10] C. C. Maley, "DNA Computing and Its Frontiers", *Molecular Computing*, Sienko, Adamatzky, Rambidi and Conrad, editors, MIT press, 2003, ch.5, pp. 153–189

[11] T. Yokomori, M. Hagiya, S. Kobayashi, K. Komiya, F. Tanaka, "Molecular Computing Machineries – Computing Models and Wet Implementation", chapter from the *Handbook of Natural Computing*, to be published

[12] Wikipedia article on DNA, http://en.wikipedia.org/wiki/DNA

[13] R. S. Braich, N. Chelyapov, C. Johnson, P. W. K. Rothemund and L. Adleman, "Solution to a 20-variable 3-SAT problem on a DNA computer", *Science*, vol. 296, 2002, pp. 499–502

[14] M. N. Stojanovic, T. E. Mitchell and D. Stefanovic, "Deoxyribozyme-Based Logic Gates", *Journal of the American Chemical Society*, vol. 124, 2002, pp. 3555–3561

[15] G. Seeling, D. Soloveichik, D. Y. Zhang, E. Winfree, "Enzyme-Free Nucleic Acid Logic Circuits", *Science*, vol. 314, 2006, pp. 1585–1588

[16] Y. Sakai, Y. Mawatari, Y. Yamasaki, K. Shoda and A. Suyama, "Construction of AND gate for RTRACS with the capacity of extension to NAND gate", *Proceedings of the 15th International Meeting on DNA Computing and Molecular Programming*, 2009, pp. 32–39

[17] I. Kawamata, F. Tanaka and M. Hagiya, "Automatic Design of DNA Logic Gates Based on Kinetic Simulation", *Proceedings of the 15th International Meeting on DNA Computing and Molecular Programming*, 2009, pp. 8–17

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

[19] N. Aubert, F. Tanaka and M. Hagiya, "A Polymerase-based Catalyst for Automated Mechanisms", *Proceedings of the 16th International Meeting on DNA Computing and Molecular Programming*, to be published

[20] H. Yan, X. Zhang, Z. Shen and N. C. Seeman, "A robust DNA mechanical device controlled by hybridization topology", *Nature*, vol. 415, 2002, pp. 62–65

[21] E. Winfree, F. Liu, L. A. Wenzler and N. C. Seeman, "Design and self-assembly of two-dimensional DNA crystals", *Nature*, vol. 394, 1998, pp. 539–544

[22] Y. He, T. Ye, M. Su, C. Zhang, A. E. Ribbe, W. Jiang and C. Mao, "Hierarchical self-assembly of DNA into symmetric supramolecular polyhedra", *Nature*, vol. 452, 2008, pp. 198–201

[23] E. Winfree, *Algorithmic self-assembly of DNA*, thesis report, 1998

[24] J. H. Reif and T. H. LaBean, "Autonomous Programmable Biomolecular Devices Using Self-Assembled DNA Nanostructures", *Communications of the ACM (CACM)*, vol. 50, issue 9, 2007, pp. 46–53

[25] K. Sakamoto, D. Kiga, K. Komiya, H. Gouzu, S. Yokoyama, S. Ikeda, H. Sugiyama and M. Hagiya, "State transition by molecules", *BioSystems*, vol. 52, no. 1-3, 1999, pp. 81–91

[26] J. A. Rose, K. Komiya, S. Yaegashi and M. Hagiya, "Displacement Whipslash PCR: optimized architecture and experimental validation", *DNA Computing, 12th International Workshop on DNA-based Computers*, vol. 4287, 2006, pp. 393–403

[27] B. Chakraborty, R. Sha N.C. Seeman, "A DNA-Based Nanomechanical Device with Three Robust States", *Proc. Nat. Acad. Sci. (USA)*, vol. 105, 2008, pp. 17245–17249