Theory Comput. Systems **35**, 483–499 (2002)
DOI: 10.1007/s00224-002-1044-8

Theory of Computing Systems

Programmed Mutagenesis Is Universal

Julia Khodor and David K. Gifford

Laboratory for Computer Science, Massachusetts Institute of Technology
545 Technology Square, Cambridge, MA 02139, USA
{jkhodor,gifford}@mit.edu

Abstract. Programmed mutagenesis is a DNA computing system that uses cycles of DNA annealing, ligation, and polymerization to implement programatic rewriting of DNA sequences. We report that programmed mutagenesis is theoretically universal by showing how Minsky's 4-symbol 7-state Universal Turing Machine [11] can be implemented using a programmed mutagenesis system. Each step of the Universal Turing Machine is implemented by four cycles of programmed mutagenesis, and progress is guaranteed by the use of alternate sense strands for each rewriting cycle. The measured efficiency of an in vitro programmed mutagenesis system suggests that segregating the products of DNA replication into separate compartments would be an efficient way to implement molecular computation.

1. Introduction

Programmed mutagenesis is a DNA computation method that implements the sequence-specific rewriting of DNA molecules. Programmed mutagenesis is an in vitro mutagenesis technique based on oligonucleotide-directed mutagenesis [2]. Like oligonucleotide-directed mutagenesis, programmed mutagenesis does not mutate existing DNA strands, but instead uses DNA polymerase and DNA ligase to create copies of template molecules with engineered mutations. Every time a programmed mutagenesis reaction is thermal cycled a rewriting event occurs. Because the technique relies on sequence-specific rewriting, multiple rules can be present in a reaction at once, with only certain rules being active in a given rewriting cycle. Furthermore, the system's ability to accommodate inactive rules allows it to proceed without human intervention between cycles. We have previously demonstrated the experimental practicality of the key primitive operations required for implementation of programmed mutagenesis systems [7].

Certain DNA computing systems are known to be restricted in the types of computations they can perform. For example, the "generate-and-test" methods proposed by

Adleman [1] and Lipton [9] are useful for certain types of combinatorial problems, but these DNA computing methods cannot be used to implement general computation.

We show that programmed mutagenesis is capable of general computation by showing how it can be used to implement a Universal Turing Machine. A Turing machine is an abstract model of a programmed computer. A Universal Turing Machine is a machine that is capable of simulating any Turing machine, given the description of that machine and the input to the machine [15]. We provide a constructive reduction of a Universal Turing Machine to programmed mutagenesis, and show how to encode the tape of a Turing machine into a DNA molecule.

An example programmed mutagenesis system that implements a unary counter is shown in Figure 1. The template (I) contains an encoding of a series of symbols XZZZZZ embedded in a noncoding region. The machine is called a unary counter because we can think of the counter as being incremented every time the system is thermocycled. We say that the number of symbols other than Z (i.e., X and Y) in the coding region minus one is the current count in the counter. Thus, template I carries the count of zero, since it contains one symbol other than Z. Every mutagenic cycle rewrites another Z into either X or Y, incrementing the counter by one.

The entire region is cloned into a plasmid using Eco RI and Hind III restriction enzymes. The outer primer MLP (machine left primer) is part of the noncoding region and the outer primer MRP (machine right primer) is a part of the plasmid sequence.

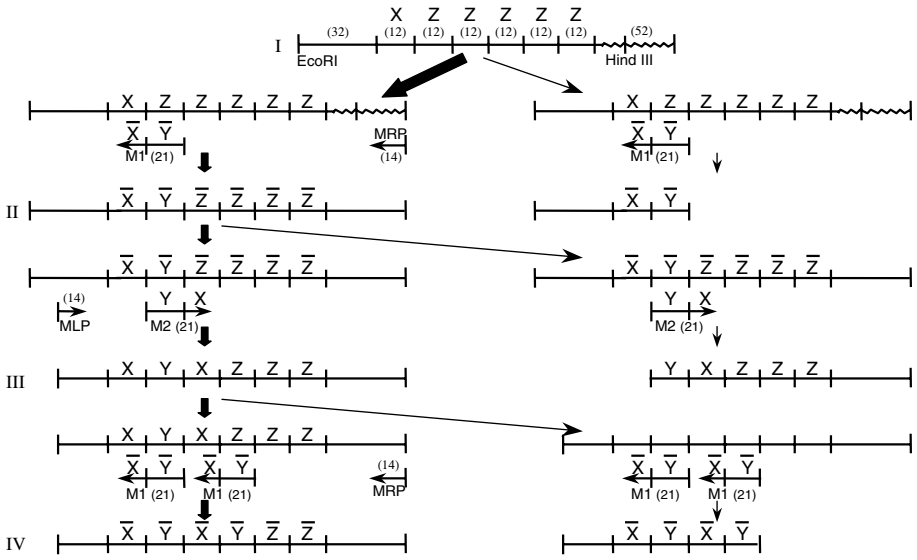

Fig. 1. Schematic representation of the unary counter. M1 and M2 are mutagenic rule oligonucleotides; MRP and MLP are perfectly matched outside oligonucleotides. Note that a rule incorporated in the previous cycle becomes part of the template for the following cycle. Bold arrows denote the transitions which carry the computation forward. The right side of the figure shows the events which lead to the creation of the characteristic bands for each cycle. These characteristic bands are the results of failed ligation events, so named because they represent the result of a failed ligation of the successful extension of the perfectly matched outside primer to the successful extension of the mutagenic rule oligonucleotide.

Each symbol used in the system (X, Y, and Z) is encoded by a 12-nucleotide long sequence of DNA. X and Y both differ from Z by two mismatches and from each other by four mismatches. The system was designed such that any oligonucleotide binding with two or fewer mismatches would be able to bind, extend, and be ligated to, and any oligonucleotide binding with more than two mismatches would not be able to interact with the template.

Each oligonucleotide able to anneal in the system is expected to be extended by the polymerase to the end of the available template or until a product of another extension is encountered. When one strand is extended to encounter another oligonucleotide positioned on the template, a ligation event is expected to occur. Ligation is not 100% efficient, and results of failed ligations are expected and are termed characteristic bands of a particular cycle. Such characteristic bands, as well as full-length products are illustrated in Figure 1.

Mutagenic oligonucleotide M1 participates in creating a first cycle product (II) that contains a different sequence than the first cycle template (I). This change permits mutagenic oligonucleotide M2 to bind to product II in cycle two, producing another new product III that incorporates M2. Product III contains a sequence that permits oligonucleotide M1 to bind in a yet another location in the third cycle yielding product IV.

Thus a sequence of related novel products (II $\rightarrow$ III $\rightarrow$ IV) is created in a specific order. Outer primers and ligation are used to create full-length products, and all of the enzymes used in the system are thermostable which allows the system to be thermal cycled for progress. The practical feasibility of the underlying specific annealing, polymerization, and ligation operations are examined in the context of a multiple-cycle experiment in Section 4.

In the remainder of this paper we prove that programmed mutagenesis is a universal model of computation (Section 2), compare programmed mutagenesis with other DNA computing systems (Section 3), and present experimental efficiency data for the unary counter machine (Section 4).

2. Programmed Mutagenesis Systems Are Universal

We begin this section with a formal definition of a Turing machine and a description of the particular Universal Turing Machine we model. We discuss challenges in encoding a Turing machine in DNA in Section 2.2, give a proof outline in Section 2.3, give the distance metric of the encoding in Section 2.4, demonstrate our encoding scheme in Section 2.5, explain the rewrite rules themselves in Section 2.6, and finally discuss correctness of the encoding in Section 2.7.

2.1. *Turing Machines and an Example of a Universal Turing Machine*

A Turing machine is a general model of computation first introduced by Turing in 1936 [15]. The Turing machine model uses an infinite tape as its unlimited memory and finite number of states and symbols to encode and compute a problem. A Turing machine has a head which can move both left and right on the tape, reading and rewriting only the symbol it is currently pointing to. The behavior of a Turing machine is governed by its

transition function, which, given the current state and symbol being read, determines the new state the machine will enter, the new symbol to be written on the tape, and the direction of motion of the tape head (left or right). There are several equivalent descriptions of Turing machines, differing only in minor details. However, since we rely on a particular universal Turing machine for the remainder of this proof, we include here the particular formal definition of a Turing machine we use, followed by the formal definition of the Universal Turing Machine we explore.

Definition 1. A Turing machine is the 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{halt}})$, where Q, Σ, and Γ are all finite sets and

1. Q is a set of states,
2. Σ is the input alphabet,
3. Γ is the tape alphabet, which includes the blank symbol,
4. δ: $(Q \times \Gamma) \rightarrow (Q \times \Gamma \times \{L, R\})$ is the transition function,
5. q_0 is a start state, and
6. q_{halt} is a halt state.

A Turing machine begins computation with its head on the leftmost cell of the input string, and proceeds by following the transition function. If it ever enters q_{halt}, it halts. If the head ever reaches the last tape cell on either side, and the transition function indicates moving off the tape, the machine appends a single tape cell with a blank symbol on it.

An instantaneous description of a Turing machine is its configuration—the current state of the finite control, the current tape contents, and the current head location. Figure 2 shows how the instantaneous description of a Turing machine progresses during a single computational step. Corresponding DNA representations are depicted in the bottom portion of Figure 2, and are discussed below.

The smallest known Universal Turing Machine was described by Minsky [11], and it has four symbols and seven states. In this machine, Σ and Γ are identical, with 0 playing the role of the blank symbol as well.

Definition 2. Minsky's Universal Turing Machine (MUTM) is the 6-tuple $(Q, \Sigma, \Sigma, \delta, q_1, q_3)$, where

1. Q=$\{q_1, q_2, q_3, q_4, q_5, q_6, q_7\}$,
2. Σ=$\{y, 0, 1, A\}$, with 0 also acting as a blank symbol, and
3. δ is given by the following transition table (where the state remains the same unless specified by a number after the /):

	q_1	q_2	q_3	q_4	q_5	q_6	q_7
y	0L	0L/1	yL	yL	yR	yR	0R
0	0L	yR	HALT	yR/5	yL/3	AL/3	yR/6
1	1L/2	AR	AL	1L/7	AR	AR	1R
A	1L	yR/6	1L/4	1L	1R	1R	0R/2

Note that the HALT state is achieved only if while the machine is in state q_3 it encounters a 0.

2.2. *A Formal Model of Programmed Mutagenesis*

An important observation about programmed mutagenesis is that the rules become part of the template for the next cycle. Thus, the rules do not take the familiar form of antecedent → consequent, but rather the rules in solution are consequents, searching for their antecedents. In fact, this property is inherent in the underlying biological technique of oligonucleotide-directed (or site-directed) mutagenesis. It is, therefore, intuitive that in modeling the MUTM we need to place state onto the tape (and into the rules) explicitly. The decision of exactly how to do that is an essential part of this proof and is explained below.

As described above, our model of programmed mutagenesis relies on a distance constraint based on mismatches in rewrite rules to sequence the steps of a program. In practice the number of mismatches is not the only factor determining whether a given rule may bind to and rewrite a given sequence of DNA, but it is the most influential. The distance constraint is reinforced by three biochemical factors. These three factors are the destabilizing effect of mismatches on duplex stability, polymerase effects, and ligase effects. Polymerase and ligase cannot function if mismatches are too close to their action sites. Other factors influencing oligonucleotide's ability to bind a given sequence of DNA, extend a given sequence, or serve as a suitable template for a ligation reaction include mismatch geometry, type of mismatch, enzymes and buffers used, and other biochemical parameters. It is impractical to try to model all the parameters influencing the efficiency of binding, extension, and ligation of DNA rewrite rules, in part because insufficient information exists to construct a reliable model. Therefore, our formal model below uses the number of mismatches as the sole determining factor for a primer (rewrite rule) to bind a given DNA sequence, extend, and to be ligated into a longer strand.

We formally model programmed mutagenesis as a DNA-based string rewrite system in which a single strand of DNA is rewritten in a sequence-specific manner with the use of a set of DNA rewrite rules to produce a single strand of DNA as output. Our formal model of programmed mutagenesis describes rewrite rules that are either 41 or 28 bases long. Also part of the model are the following four assumptions:

1. For a rule of length 41, the allowed mismatch distance is four mismatches.
2. For a rule of length 28, the allowed mismatch distance is three mismatches.
3. Any DNA sequence for which the distance in mismatches to its target on the template is above the specified number of mismatches will not bind and extend.
4. If only one rule can bind to a spot, that rule executes. If more than one rule can bind to a spot, equal percentages of each rule execute (thus creating parallel branches of computation). Thus, a copy operation occurs if no rules bind.

It is possible to formulate alternative formal models of programmed mutagenesis that use rules of different lengths and different distance constraints. We have experimentaly found above length and distance constraints to be reasonable.

2.3. *Proof Outline*

We have designed an encoding in our model of programmed mutagenesis to simulate the MUTM directly. There are three key ideas in this construction.

1. In order to represent the location of the head and the current state on the tape explicitly, we extend the alphabet to include q_i/s for every $q_i \in Q$ and every $s \in \Sigma$. We further expand the alphabet, but this is the essential decision.
2. We introduce scratch space (#) between each two tape symbols. This allows us to have read and write parts of each rule. We use the scratch space to transmit information by first writing to it the information on the new state and the direction of movement (using the q_i/s cell as the read part of the rule). We then use the freshly written information as the anchor for the next step of the rule.
3. We execute each rule of the MUTM by four rewrite rules:
   - Read the current state and symbol and save the information about the new state and the direction of the rule in the scratch space; begin the transition to the new tape symbol.
   - Finish the transition to the new tape symbol.
   - Read the new state information (in the scratch space), begin transition to the the new state/symbol pair.
   - Finish transition to the the new state/symbol pair; return the scratch space to its original state (#).

   The rules are executed over four biological cycles. Only the first three biological cycles, however, include rewrite events, while the fourth simply copies the template strand using the outside primer. Furthermore, the rules act on alternative strands of DNA (3′-to-5′ for cycles 1 and 3 and 5′-to-3′ for cycle 2), and we use this property to drive the computation forward.

   This proces allows us to have one-to-one rules, because each antecedent takes a different path to the consequent. We extend the alphabet to allow this as described below. To be precise, we have one-to-one rules for three out of four steps. Step 2 can be and is executed as many-to-one.

The instanteneous description of the DNA encoding of the MUTM at the beginning of each computational step is the single instance of a state/symbol character on the tape, indicating the precise head position and the state of the finite control, and all the other symbols on the tape. We represent states and symbols by nucleotide sequences and enact state transitions by primer extension reactions. The bottom portion of Figure 2 depicts the instanteneous descriptions of the DNA representation of a Turing machine before and after a single transition depicted in the top portion of the figure. The four rewrite rules which execute the computational step are indicated on the transition arrow. All possible transitions within the computational step are discussed in Section 2.7 and are illustrated in Figure 4.

2.4. *Distance Metric and Alphabet Extensions*

As discussed above, in addition to the four tape symbols (y, 0, A, 1) we introduce symbols q_i/s that are six mismatches away from s for all q_i. Also, if q_i/s is to be rewritten

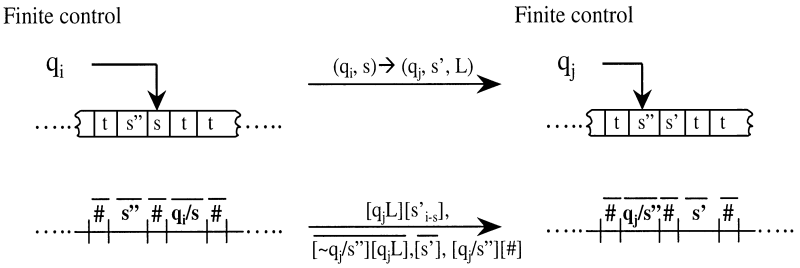

Fig. 2. Schematic representation of a state transition of the MUTM and the corresponding DNA representations. The transition rule of the Turing machine and the corresponding DNA rewrite rules necessary to execute the computation step are indicated on the arrows.

according to the rule $(q_i, s) \rightarrow (q_j, s', L/R)$, i.e., if the rule rewrites s to s', then q_i/s is also six mismatches away from s'.

We now introduce additional symbols needed to complete the proof. We summarize the discussion below by presenting the key distances between tape symbols in Table 1.

- For every transition rule $(q_i, s) \rightarrow (q_j, s', L/R)$, where s may or may not be the same as s', we introduce a new symbol s'_{i-s}, 28 bases long, which is three mismatches away from q_i/s and three mismatches away from s'. Because of the

Table 1. Key distances between tape symbols. (a) Distances for elements of size 28. The distances given are for a rule $(q_i, s) \rightarrow (q_j, s', D)$, where the next symbol in the direction of D is s''. s, s', s'' may or may not be the same. (b) The distances for elements of size 13.

(a)

s	s'	s''	q_i/s	q_k/t	s'_{i-s}	s'_{i-t}	s'_{k-s}	t_{k-v}	$\sim q_j/s''$	q_j/s''	$\sim q_k/s''$	
s	0											
s'	≥ 12	0										
s''	≥ 12	≥ 12	0									
q_i/s	6	6	≥ 6	0								
q_k/t	≥ 6	≥ 6	≥ 6	≥ 6	0							
s'_{i-s}	9	3	≥ 12	3	≥ 5	0						
s'_{i-t}	≥ 12	3	≥ 12	≥ 7	≥ 5	≥ 5	0					
s'_{k-s}	9	3	≥ 12	≥ 5	≥ 7	≥ 4	≥ 5	0				
t_{k-v}	≥ 12	≥ 12	≥ 12	≥ 5	≥ 9	≥ 5	≥ 5	≥ 5	0			
$\sim q_j/s''$	≥ 9	≥ 9	4	≥ 5	≥ 5	≥ 5	≥ 5	≥ 5	≥ 5	0		
q_j/s''	≥ 6	≥ 6	6	≥ 6	≥ 6	≥ 5	≥ 5	≥ 5	≥ 5	3	0	
$\sim q_k/s''$	≥ 9	≥ 9	4	≥ 5	≥ 5	≥ 5	≥ 5	≥ 5	≥ 5	≥ 5	≥ 5	0

(b)

	#	$q_i D_1$	$q_j D_2$ for $i \neq j$ or $D_1 \neq D_2$
#	0	1	1
$q_i D_1$	1	0	2
$q_j D_2$	1	2	0

particular method of encoding used here, s'_{i-s} is guaranteed to be at least five mismatches away from any other q_k/s''.

- For every rewrite rule $(q_i, s) \rightarrow (q_j, s', D)$, where $D \in \{L, R\}$, we introduce a new symbol: $q_j D$, 13 bases long, which is one mismatch away from #. Because of the transition table of this particular machine, it turns out there are only nine symbols in this class, corresponding to the following (new state, direction) pairs: $(q_1, L), (q_2, L), (q_2, R), (q_3, L), (q_4, L), (q_5, R), (q_6, R), (q_7, L), (q_7, R)$. Under the particular encoding used, any $q_i D_1$ is at least two mismatches away from any $q_j D_2$ if $i \neq j$ or $D_1 \neq D_2$.
- For every rewrite rule of the MUTM $(q_i, s) \rightarrow (q_j, s', D)$, where the next tape symbol in the direction of D is s'', we introduce a new symbol $\sim q_j/s''$, 28 bases long, which is four mismatches away from s'' and three mismatches away from q_j/s''. We also guarantee that it is at least five mismatches away from any s'''_{k-t}.

2.5. *Encoding Scheme*

We now demonstrate the encoding scheme that allows us to hold to the distance metric above.

We represent scratch space symbols with nine coding bases surrounded on either side by two spacer bases (CC and GC). We employ a spacer so that no mismatch in the coding positions is adjacent to the site of action of either polymerase or ligase. We have found empirically that such a mismatch would prevent the enzymes from working. The # symbol is a sequence of nine G's surrounded by the buffer bases. As illustrated in Table 1(b), each of the $q_i D$'s is different from # in one of these nine positions, and, therefore, different from each other in two positions.

The 28-base long symbols are represented by 24 coding bases surrounded, once again, by two spacer bases on each side. The relationships between the 28-base long symbols are represented in Table 1(a) and Figure 3. Directions of the arrows in Figure 3 represent the direction of rewrite, and all the arrows represent a distance of six mismatches. As can be seen in Figure 3, encoding of symbol 1 is only related to encoding of A, and has no connection with encodings for either y or 0. We capitalize on this observation in creating the following encoding scheme.

We designate the odd positions of the coding section as characteristic bases of y, 0, and A. We designate the even positions as characteristic of A and 1. As can be seen in Table 2, all odd coding bases for y are A's, for 0 are C's, and for A are T's. Even bases are random, and are the same for y, 0, and A, and different for 1. Thus, y, 0, and A are 12 mismatches away from each other, as are A and 1, but y and 0 are 24 mismatches away from 1.

If a q_i/s is rewritten into an s', we encode it by changing six of the characteristic bases of s into those of s'. For q_j/s's that are rewritten to s, we change six of the bases that are not characteristic bases of s. Under our encoding, all of these symbols are at least six mismatches apart.

If q_i/s is rewritten into an s', we encode s'_{i-s} by changing three of the six characteristic bases of s into those of s'. For q_j/s which is rewritten into an s, we change three of the six bases that are not characteristic bits of s back into the characteristic bases of s. Under our encoding there are only five pairs of symbols of this type that are four

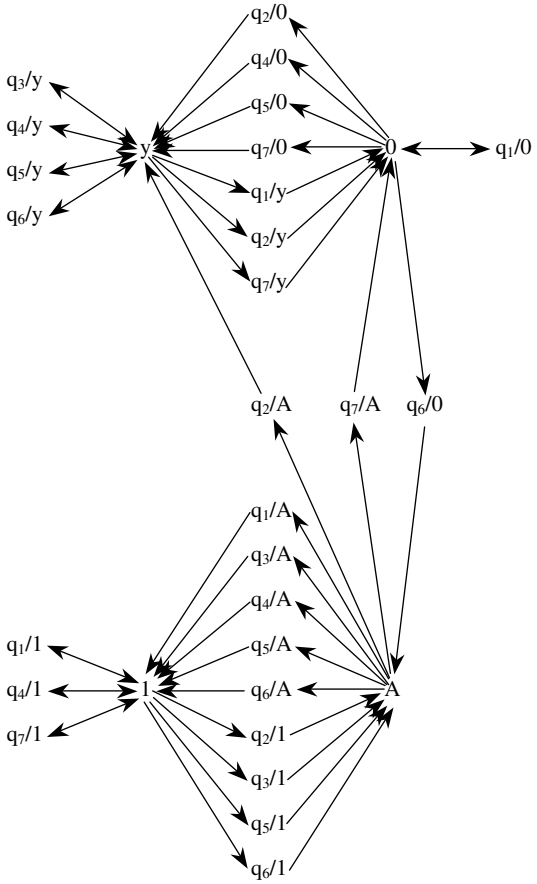

Fig. 3. Transitions between the symbols used to encode the MUTM. All arrows represent the distance of six mismatches and the direction of an arrow represents the direction of the rewrites in the system.

mismatches apart. The rest of the pairs are at least five mismatches apart. We show in Section 2.7 why even the presence of symbols that are four mismatches apart cannot lead the system to execute an illegal rewrite.

To encode $\sim q_i/s$, we start with s and change three of the six bases that are different between s and q_i/s to those of q_i/s and a fourth to a base that is different from both s and q_i/s. Under our encoding, all of these symbols are at least five mismatches apart.

We used the above algorithm to generate 85 symbol encodings for the 28-base long symbols, and 10 symbol encodings for 13-base long symbols. Note that we do not need to encode $q_3/0$ or $\sim q_3/0$ because of our readout mechanism described in Section 2.6 below. We wrote a simple program to calculate mismatch distances between 85 encodings of length 28. The distance matrix generated by this program, along with the complete list of encodings can be found at http://www.psrg.lcs.mit.edu/biocomp/universal.html. We have also verified that any rewrite rule binding off-frame would be more than the allowed number of mismatches away from its target on the template.

Table 2. Encodings of and mismatches in the tape symbols of the MUTM.

y	C	C	A	T	A	C	A	C	A	G	A	T	A	A
			X		X		X		X		X		X	
0	C	C	C	T	C	C	C	C	C	G	C	T	C	A
			X		X		X		X		X		X	
A	C	C	T	T	T	C	T	C	T	G	T	T	T	A
				X		X		X		X		X		X
1	C	C	T	A	T	G	T	G	T	C	T	A	T	T

y	A	G	A	A	A	T	A	G	A	C	A	G	G	C
	X		X		X		X		X		X			
0	C	G	C	A	C	T	C	G	C	C	C	G	G	C
	X		X		X		X		X		X			
A	T	G	T	A	T	T	T	G	T	C	T	G	G	C
		X		X		X		X		X		X		
1	T	C	T	T	T	A	T	C	T	G	T	C	G	C

2.6. *Rewrite Rules*

We now describe the actual DNA rewrite rules needed to implement the MUTM. First, we have outside primers in the machine that are used to create the full-length product by ligating the result of the outside primer extension to the inner mutagenic primer. Second, the rules below are shown in the same orientation regardless of which strand they act on, but we include the $5'$ and $3'$ markers. In total, it takes 101 DNA oligonucleotides to implement the 27 rewrite rules that constitute the transition function of the MUTM.

For every rewrite rule of the Turing machine $(q_i, s) \rightarrow (q_j, s', L)$, where the next tape symbol to the left is s'', we need the following DNA rewrite rules:

1. $5'[q_j L][s'_{i-s}]3'$.
2. $3'\overline{[s']}5'$.
3. $3'\overline{[\sim q_j/s''][q_j L]}5'$.
4. $5'[q_j/s''][\#]3'$.

For every rewrite rule of the Turing machine $(q_i, s) \rightarrow (q_j, s', R)$, where the next tape symbol to the right is s'', we need the following DNA rewrite rules:

1. $5'[s'_{i-s}][q_j R]3'$.
2. $3'\overline{[s']}5'$.
3. $3'\overline{[q_j R][\sim q_j/s'']}5'$.
4. $5'[\#][q_j/s'']3'$.

Note: Rules 2 and 3 execute simultaneously. Also, there are only four rules of type 2 (one for each alphabet symbol). Finally, the four rules act over three DNA replication cycles.

The fourth biological cycle consists of simply making a copy of the template molecule using an outside primer. This allows us to get the template into the mode where a rule of type 1 can act on it. The three rewriting cycles plus a copying cycle is the structure which guarantees that the DNA machine can never go back through the computational steps of the MUTM. In Section 2.7 we go through all possible biological transitions within a computational step and demonstrate that while within the cycles that comprise a computational step some random walk through the states of the template is possible, the DNA machine can never go back through computation.

We also need rules that will allow us to extend the tape if necessary. We can achieve this with the use of loopout structures. If we say that our template is embedded between the sequences $[left - wing]$ and $[right - wing]$, then for each $[q_i R]$ we need the rule $5'\overline{[q_i R][0][\#][right - wing]}3'$. For each $[q_j L]$, we need the rule $5'\overline{[left - wing][\#][0][q_j L]}3'$.

A simple way to detect when the computation is complete is to sense the appearance of $0q_3 L$ on the tape with a molecular beacon. Molecular beacons are hairpin structures with a flourophore and a quencher on the $5'$ and $3'$ ends respectively in which a perfect match for the loop portion of the molecule forces the hairpin to open, thus separating the flourophore from the quencher and creating a fluorescent signal [16].

2.7. *Correctness*

In this section we discuss the transitions in one computation cycle of the MUTM step by step, and consider which rewrite rules can bind at which stages according to our formal model. Figure 4 illustrates discussion below by presenting all possible transitions in a computation cycle. Bold arrows indicate the transitions that carry the computation forward. We consider here a transition that would be effected by a Turing machine rule $(q_i, s) \to (q_j, s', D)$. Without loss of generality, Figure 4 assumes $D = L$.

- At the first rewrite, all possible $q_i D$'s are the same distance away from #, so only the rules which have the component which is three away from q_i/s will bind. However, there is exactly one such rule: the one that rewrites q_i/s into s'_{i-s} (and # into $q_j D$). The other set of rules of the correct sense ($5'$-to-$3'$) is the $[q_k/s''][\#]$ set. None of these rules can bind now, since the best they can do is to match # and rewrite an s'' into q_k/s'', but even in this case the distance is at least six mismatches. Of course, the rule that effected the last transition, that is the rule $[q_i/s][\#]$ (or $[\#][q_i/s]$) can bind here. This would lead to no rewrite, and the next transition can only be effected by an outside primer, returning the template to it original state.
- At the second rewrite, the rules that contain the complement of $q_j D$ are the closest to the target sequence on the template, and of those only the one containing the complement of $\sim q_i/s''$ is the allowed distance away. Any rule containing the complement of any $q_k D$ for $j \neq k$ would be at least six mismatches away from the target (two from the scratch space mismatches and four from the symbol mismatches). Finally, none of these rules can bind to the s'_{i-s} spot because the distance between any $\sim q_k/s''$ and s'_{i-s} is at least five. At the same time, only the complement of s' can bind to s'_{i-s}. The rules containing other s'''_{k-t}'s cannot bind because they are of the wrong sense (and thus it does not matter that five pairs

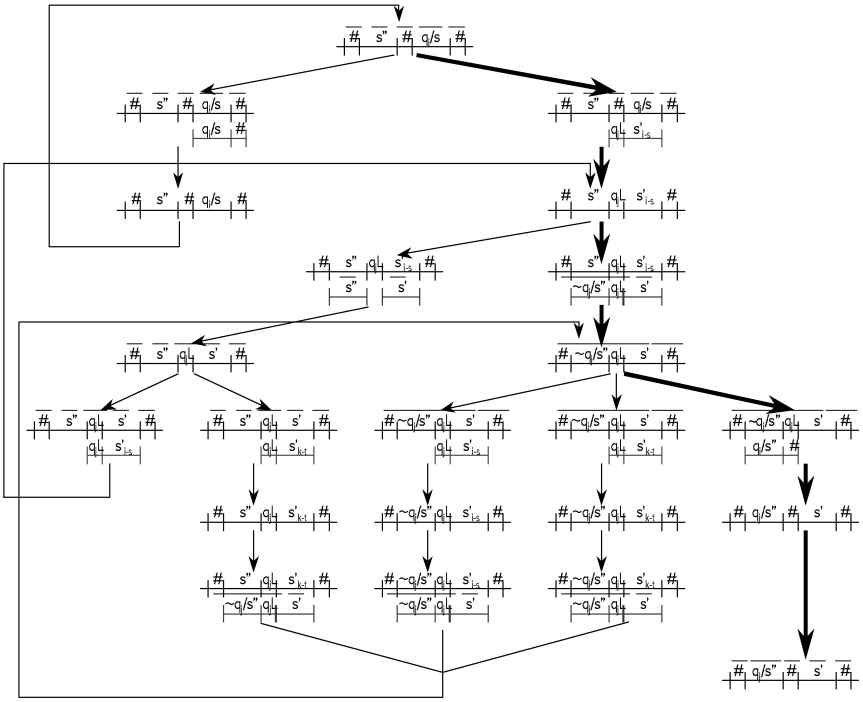

Fig. 4. Schematic representation of all possible rewrites in a computational cycle. Bold arrows denote the transitions which carry the computation forward.

of these symbols are four mismatches apart). Of course, the complement of s'' can bind to s'', thus preventing the rewriting rule from binding. However, all that would do is enable the first cycle rule to bind again on the third cycle, reproducing the second cycle template again.

- Now we actually have several rules that can bind. Both the last cycle and the first cycle rules can bind to the same spot—the complement of $q_j D$. Additionally, other first step rules can bind—those that have the complements of $q_j D$ and of some s'_{k-t} for some k and $t = \{s, s', s'''\}$. Notice that no rule containing $q_l D$ can bind at this stage, since it would be at least five mismatches away from the template target sequence (two in the scratch space and at least three in the symbol space). In this situation, if the last cycle rule executes, we are done. If, however, a first cycle rule executes, we have on the template (if $D = L$, without loss of generality) sequence $[q_j][\sim q_j/s''][q_j L][s'_{k-t}]$. However, this sequence can only be rewritten by the second cycle rules, thus returning the template to the previous state.

As demonstrated above, under our model of programmed mutagenesis and encoding scheme no incorrect rewrite is made, and we guarantee forward progress. Thus, we show that the above encoding scheme does indeed implement the MUTM, and, consequently, that programmed mutagenesis is a universal model of computation.

3. Programmed Mutagenesis and Other DNA Computing Systems

We call a computing system *composable* when a first computation results in a single molecule that can be used directly by a second computation as input without modification. Programmed mutagenesis systems are composable because both the input and output from a computation can be represented as a single strand of DNA. Programmed mutagenesis is the first composable system to be proven universal.

Our approach builds on other models proposed for DNA-based computing. These can be subdivided into five categories.

- A generate-and-search approach was pioneered by Adleman [1] in 1994. In this approach all possible candidate solution molecules are generated, and then filters are applied to select only those molecules that contain solutions. This was the first experimental demonstration of any computational system based on DNA, but this approach is not universal.
- Splicing systems were pioneered by Head [5] in 1987. When these systems have finite sets of axioms and rules which define splicing, they are limited to generating regular languages [12]. If both of these sets remain finite, the only way to increase the computational power of the system is to introduce certain control mechanisms. Splicing systems utilizing a number of such control mechanisms have been shown to be universal. However, the control systems proposed require the researcher's intervention and are vulnerable to experimental error. Furthermore, these mechanisms might prove difficult to implement in a laboratory.
- Self-assembly of two-dimensional DNA structures was shown to be universal by Winfree [17] in 1995. Winfree proposed using ligation after self-assembly to produce a single reporter strand of DNA. This reporter strand would be used for reading the output of the self-assembly computation. A challenge in the experimental implementation of this approach might be that larger computations, which include more different varieties of tiles, will require longer time intervals for each step of the assembly because self-assembly relies on the correct tile assembly followed by the correct computational structure assembly. Further, minimizing the error rate also requires longer time intervals per assembly step. Recently, the first experimental demonstration of computation by self-assembly [10] has been produced. This is a significant advancement in DNA computing because it demonstrates the first DNA-only universal computational system which requires only ligase to create an output molecule. Certain proposed implementations of self-assembly systems can be composable, but the one experimentally implemented to date is not.
- Autonomous string systems based on hairpin formation were introduced by Hagiya et al. [4]. In these systems each molecule works not only as a data carrier but also as a computing unit. Some interesting experimental systems based on this principle have been constructed [8], [14]. However, these systems are not, by themselves, universal. They are also not composable.
- Programmed mutagenesis is an example of a string-rewrite system for DNA computing. Rothemund [13] proposed implementing a universal string rewrite system (in fact, modeling the same MUTM machine) using restriction enzymes

and ligases. This model requires researcher intervention since different enzymes and oligonucleotides need to be added on alternative steps, necessitating the use of a separation operation. No experimental confirmation of this method has been reported so far. Both Beaver [3] and Kari and Thierin [6] have also proposed using systems based on string-rewriting. While both of these models were proven to be theoretically universal, both have drawbacks:

1. Beaver's model proposes using the substitution operation, that is, rules of the form xyz $\rightarrow$ xuz. This model, however, requires separations after each substitution step to guard against the possibility of a rule performing a mixture of substitution and deletion, such as rewriting the sequence xyyz on the template into the sequence xuz, or insertion, such as rewriting the sequence xz on the template into the sequence xuz. Since separation operations are required, this model is also vulnerable to experimental error.
2. Kari and Thierin [6] proposed using insertion/deletion ("insdel") systems to implement universal computation. While the insdel systems represent a theoretically interesting computational model, new techniques will be required to implement them in practice. With current techniques there are two problems with the proposed rules for the insdel systems. First, there is no way to control the length of the deleted sequences in the insdel systems. The deletion rules in the insdel systems are of the form xzy $\rightarrow$ xy, which with the currently available techniques have to be implemented by an oligonucleotide encoding sequence xy. This oligonucleotide would bind any occurrence of the contexts x and y, and would, therefore, delete any sequence embedded between these contexts. Thus, if the template read xzyy, the rule xy would, with about equal probability, delete sequences z and zy. Second, there is no way to prevent insertion rules in the insdel systems from performing substitutions. For example, an insertion rule of the form xz $\rightarrow$ xyz, represented by an oligonucleotide encoding the sequence xyz, and given a template xzzzz would be approximately equally likely to perform an insertion (rewriting the template into xyzzzz), or any of the three possible substitutions (rewriting the template into xyzzz, xyzz, or xyz).

An additional interesting property of programmed mutagenesis is that all the machinery necessary to implement it is present in a cell. We have also demonstrated that the key aspects of programmed mutagenesis function properly [7]. Furthermore, once the reaction is put together, and thermal cycling begins, programmed mutagenesis requires no intervention on the part of a researcher. In the next section we look at some experimental data from the unary counter machine.

4. Experimental Efficiency of a Programmed Mutagenesis System

We have constructed the unary counter machine shown in Figure 1, and have operated it through three cycles to gather efficiency data. The cycle reactions contained 0.02 uMolar of M1 and M2 oligonucleotides, $\sim$ 0.2 uMolar of double stranded template DNA, Taq thermostable ligase (40 U) (NEB, #MO208), Vent thermostable polymerase

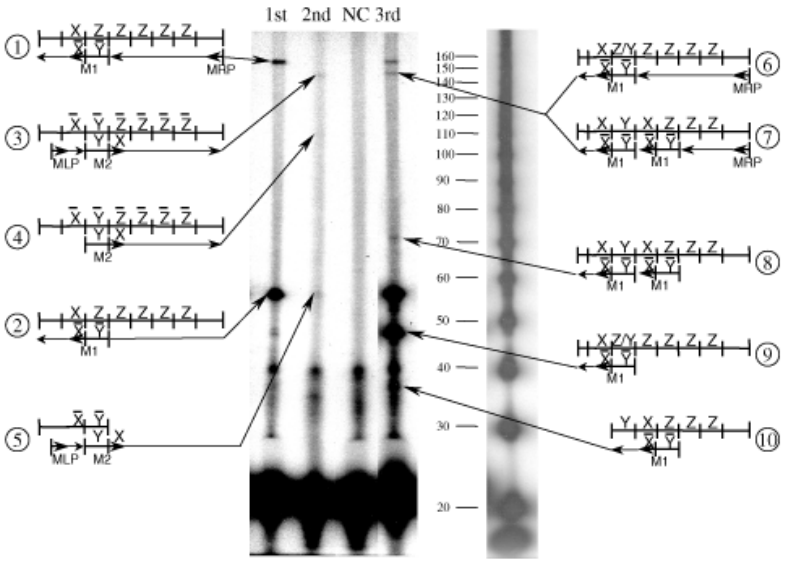

Fig. 5. Operation of the unary counter machine through cycles one to three, plus negative control (NC) at 0.02 μM of rule oligonucleotides. Negative control reactions contain the same components as second cycle reactions, with the exception of M1, first cycle mutagenic primer. In the absence of M1, no product labeled II in Figure 1 is supposed to be produced, and no second cycle product is therefore possible. We expect negative control lanes to contain no product. Oligonucleotide M1 is end-labeled with ^{32}P in cycles one and three, while M2 is labeled in cycle two and negative control. Cartoons represent the products contained in the corresponding bands. The band labeled 3 is the full-length product of cycle two (and the template for cycle three), designated by III in Figure 1. The band labeled 8 is the characteristic length product of cycle three, produced when the product of extension of MRP fails to ligate to the product of extension of the mutagenic primer M1 annealed in the third cycle location. Other bands as illustrated.

(0.25 U) (NEB, #MO254), 0.2 uMolar of outer primers, and 1x ThermoPol buffer (NEB) supplemented with 1mM NAD. These reactions were thermal cycled for 1 minute at 94 °C, and 30 minutes at 45 °C for the indicated number of cycles. Primer and template sequences are available upon request.

Figure 5 shows our experimental data. To estimate the amount of products designated by II, III, and IV in Figure 1 we 32P end-labeled M1 (for II and IV) and M2 (for III). Failed ligations produce products of characteristic lengths, and we estimated the ligation efficiency of the third cycle by assuming it is the same as the first cycle. We include cartoon depictions of the products on the gel. Notice that we cannot ascertain directly that cycle three has occurred, or what the efficiency of that cycle is, from the full-length product band because in cycle three the full-length band is a mixture of product IV and a shortened version of product II (as illustrated by the cartoons 7 and 6 in Figure 5, respectively). However, both the characteristic product of cycle three (cartoon 8 in Figure 5) and the coloring of the characteristic product of cycle two (cartoon 10 in Figure 5) are present, and indicate that cycle three has indeed taken place. We directly quantitated the amount of product in the characteristic band of cycle three, and estimated the amount of product

IV present by assuming that the ligation efficiency of the third cycle is the same as that in the first cycle.

The results of the reactions were run on polyacrylimide denaturing gels, and bands were quantiated on a phosphoroimager. The results showed that 5.3×10^{-3} of template I is converted to product II in cycle one, 1.3×10^{-1} of product II is converted to product III in cycle two, and 3.5×10^{-2} of product III is converted to product IV in cycle three. In addition, no product III was generated in the cycle two negative control (NC) reaction in the absence of primer M1 and the presence of M2.

The major impediment to continued cycling of the machine is that as long as template I is present, its products will increase exponentially with cycle number. However, if the Watson and Crick strands resulting from each DNA replication are separated into different compartments, then the compartment that receives product II will only contain a single computational state and thus will not repeat earlier computational steps. In vivo programmed mutagenesis might be an effective way to evolve DNA sequences computationally and could potentially assist in sequence specific control of cellular function. It is also possible that gene conversion events and other natural systems for DNA evolution could implement a more complex computational substrate than is now understood.

References

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

[2] Ausubel, I., and Frederick, M. (eds.), 1997, *Current Protocols in Molecular Biology*, Section 8.5, Wiley, New York.

[3] Beaver, D., 1995, Molecular Computing, Technical Report, Department of Computer Science and Engineering, Penn State University.

[4] Hagiya, M., Arita, M., Kiga, D., Sakomoto, K., and Yokoyama, S., 1999, Towards parallel evaluation and learning of Boolean mu-formulas with molecules, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 48, pp. 57–72, AMS, Providence, RI.

[5] Head, T., 1987, Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviors, *Bulletin of Mathematical Biology*, **49**, 737–759.

[6] Kari, L., and Thierin, G., 1996, Contextual insertions/deletions and computability, *Information and Computation*, **131**, 47–61.

[7] Khodor, J., and Gifford, D.K., 1997, Design and implementation of computational systems based on programmed mutagenesis, *Proceedings of* 3^{rd} *DIMACS Workshop on DNA-Based Computers*.

[8] Komiya, K., Sakamoto, K., Gouzu, H., Yokoyama, S., Arita, M., Nishilkawa, A., and Hagiya, M., 2000, Successive state transitions with IO interface by molecules, *Proceedings of* 6^{th} *DIMACS Workshop on DNA-Based Computers*.

[9] Lipton, R., 1995, DNA solution to computational problems, *Science*, **268**(5210), 542–545.

[10] Mao, C., LaBean, T.H., Reif, J.H., and Seeman, N.C., 2000, Logical computation using algorithmic self-assembly of DNA triple-crossover molecules, *Nature*, **407**, 493–496.

[11] Minsky, M.L., 1967, *Computation*: *Finite and Infinite Machines*, Prentice-Hall, Englewood Cliffs, NJ.

[12] Păun, Gh., Rozenberg, G., and Salomaa, A., 1998, *DNA Computing*, Springer-Verlag, Berlin.

[13] Rothemund, P., 1996, A DNA and restriction enzyme implementation of Turing machines, *Proceedings of* 1^{st} *DIMACS Workshop on DNA-Based Computers*, pp. 75–119.

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

[15] Turing, A.M., 1936, On computable numbers, with an application to the Entscheidungsproblem, *Proceedings of the London Mathematical Society*, *Series* 2, **42**, 230–265.
[16] Tyagi, S., and Kramer, F.R., 1996, Molecular beacons: probes that flouresce upon hybridization, *Nature Biotechnology*, **14**, 303–308.
[17] Winfree, E., 1995, On the computational power of DNA annealing and ligation, *DNA Based Computers*: *DIMACS Workshop*, vol. 27, pp. 199–221.

Received October 31, 2001, *and in revised form February* 19, 2002. *Online publication July* 24, 2002.