The Power of Surface-Based DNA Computation *

Weiping Cai, Anne E. Condon, Robert M. Corn, Elton Glaser,
Zhengdong Fei, Tony Frutos, Zhen Guo, Max G. Lagally,
Qinghua Liu, Lloyd M. Smith, Andrew Thiel

University of Wisconsin
Madison, WI 57306 USA

August 13, 1996

Abstract

A new model of DNA computation that is based on surface chemistry is studied. Such computations involve the manipulation of DNA strands that are immobilized on a surface, rather than in solution as in the work of Adleman. Surface-based chemistry has been a critical technology in many recent advances in biochemistry and offers several advantages over solution-based chemistry, including simplified handling of samples and elimination of loss of strands, which reduce error in the computation.

The main contribution of this paper is in showing that surface-based DNA chemistry efficiently supports general circuit computation on many inputs in parallel. To do this, an abstract model of computation that allows parallel manipulation of binary inputs is described. It is then shown that this model can be implemented using fairly standard chemistry, in which inputs are encoded as DNA strands and the strands are repeatedly modified in parallel on a surface using the chemical processes of hybridization, exonuclease degradation, polymerase extension or ligation. Thirdly, it is shown that the model supports efficient circuit simulation in the following sense: exactly those inputs that satisfy a circuit can be isolated, and the number of parallel operations needed to do this is proportional to the size of the circuit. Finally, results are presented on the power of the model when another resource of DNA computation is limited, namely strand length.

*Corn, Fei, Frutos, Guo, Liu, Smith and Thiel are in the Chemistry Department, Condon and Glaser are in the Computer Sciences Department and Cai and Lagally are in the Materials Sciences Department. Email address for further communication: condon@cs.wisc.edu.

1 Introduction

Following Adleman [1], Lipton [9], and others, we recently proposed a new DNA computing scheme based on surface chemistry [10]. In this paper, we define an abstract model of surface-based DNA computation and describe the power and limitations of this model.

In contrast with Adleman's solution-based experiment, surface-based DNA computations manipulate DNA strands that are immobilized on a surface using chemical linkers. This means that a key operation used in solution-based DNA computations, that of selectively separating strands into separate test tubes, cannot be performed. Also the number of DNA strands involved in the computation is limited since the strands are restricted to two rather than three dimensions. (The number of strands that can fit on a 1 cm^2 planar surface is roughly 10^{12}.) Nevertheless, it is our premise that surface-based chemistry will be important to advances in DNA computation for the following reasons.

First, surface-based chemistry has been a critical technology in recent advances in biochemistry, including protein sequencing, DNA synthesis and peptide synthesis [14]. Second, handling of samples is simpler and more readily automated when the samples are immobilized on surfaces rather than in solution, loss of strands is effectively eliminated and a much greater degree of control is obtainable in each chemical "operation." This is important since errors, particularly in separating strands into distinct test tubes, is a primary obstacle to Adleman's approach. Third, the work of the company Affymetrix on DNA synthesis on "addressed" surfaces [13] can be viewed as a restricted form of computation. Our proposed DNA computing method is complementary to Affymetrix arrays in that the ability to address strands spatially is lost but the scale is much greater. Finally, experiments on surfaces will provide an ideal means to develop chemistry, algorithms and understanding of errors that will be useful in further, larger-scale DNA computations that may not necessarily be wholly based on surfaces.

For all of these reasons, we believe that it is important to understand how general-purpose computation can be performed on surfaces. Towards this end, we introduce an abstract model of computation that can be implemented using standard surface chemistry. Roughly, the operations in the model allow strands to be marked or selected depending on their value, bits to be appended to the free (non-immobilized) end of selected strands, selective destruction of strands, and erasure of appended bits from selected strands. The time to do the mark operation is proportional to the length of the strands marked, but the time to do the remaining operations is constant. We then show that surface-based DNA chemistry efficiently supports general circuit computation on many inputs in parallel: Given a circuit that accepts a language $L_n \subseteq \{0,1\}^n$, there is a surface-based DNA algorithm that, given a set S of binary inputs that are represented as DNA strands, identifies exactly those inputs in S that satisfy the circuit and destroys the rest. The values of gates are appended in parallel to strands until the output is computed. The number of operations needed to do this is proportional to the size of the circuit. The time is proportional to the square of the number of gates, since the mark operation is done once per gate.

In addition to keeping the number of operations to a minimum in a DNA computation, it is also desirable to keep the maximum strand length small. One reason for this is that long strands are more likely to form unwanted secondary structures that cause errors in the computation. Another reason is that the time for some operations is linear in the length of the strand, and hence the operations are faster when the strand is short. Therefore, we prove results about the

power of surface-based computation when (i) modification of strands by appending information onto them is not allowed, and (ii) the number of bits that can be appended to strands at any given time during the computation is bounded. In each case, we classify exactly the restricted type of circuit that can be efficiently simulated by a DNA algorithm.

In Section 1.1 we present our abstract model of surface-based DNA computation. We then summarize our results on the power of the model in Section 1.2. The chemical basis for the abstract model is given in Section 2. Proofs of some results are given in Section 3.

1.1 Abstract Model

The following surface-based model is motivated by several goals. It should be built on chemical processes that are reasonably well understood and reliable. It should be conceptually simple, yet the operations should exploit the chemical processes in as general a way as possible. This is because it is important to keep the number of operations to a minimum in a DNA computation; the more general the operation set, the fewer operations are needed in a computation.

Operations are performed on a multiset S of strands, where each strand is a sequence of bits. Several practical and strategic considerations detailed in Section 2 motivate the following word-based structure for organizing bits within strands. Each strand is a finite sequence of short, variable-length binary words numbered consecutively starting from 1. The length of a word is the number of bits in that word. Over all strands, the ith word has the same length. The value of a word is the vector of values of the bits within that word. For simplicity we assume that all strands in the initial set (i.e. before any operation is performed) have the same number, say n, of bits. We now list the operations that can be applied to a set S of strands.

mark(constraint-list): all strands satisfying all constraints in the constraint-list are identified as marked. Each constraint specifies, for a distinct i, a set of values for a subset of the bits in the ith word. A strand satisfies this constraint if and only if the value of the ith word in the strand agrees with the values of the bits specified in the constraint. For simplicity we assume that when the mark operation is applied, if the ith word is constrained then there are at least i words present in all strands on the surface.

unmark: unmark all marked strands.

destroy-marked (destroy-unmarked): all marked (resp. unmarked) strands are removed from the set S.

append-marked(new-word): a word with value new-word is appended to all marked strands. For simplicity, we assume that when the append-marked operation is applied, all marked strands have the same number, say k, of words and the number of the appended word is $k+1$. Thus, the operation is really append($k+1$, new-word).

append-unmarked(high-word,new-word): a word with value new-word is appended to all unmarked strands in which the value of the highest-numbered word is high-word. For simplicity, we assume that when the append-unmarked operation is applied, all unmarked strands have the same number, say k, of words and the number of the appended word is $k+1$.

erase-marked: all words that have been appended to marked strands since the start of the computation (not counting the set initialization phase) are erased, leaving just the initial strands (still marked). The motivation for including this operation is to keep strands short.

We note that by using the mark operation, strands can be selectively marked for further "processing" such as append or destroy. The mark operation replaces the separate (or extract) operation of Adleman and Lipton, which divides strands into separate test tubes for further processing. The destroy-unmarked operation is very similar to an operation of Amos et al. [3] although their model is solution-based. There is no read-out operation in the above set. In this paper we ignore readout and focus on what sets can be recognized, as described in the next section. The problem of read-out is essentially the same for surface-based DNA computation as for solution-based DNA computation.

1.2 Power of Model

A DNA algorithm A $(= A_n)$ is a sequence of operations applied to an initial multiset S of n-bit strands. The output of the algorithm, denoted by $A_n(S)$, is the multiset of strands on the surface at the end of the algorithm. Thus, a DNA algorithm performs a computation on a (huge) set of inputs in parallel, where each computation is completely independent of the others. We say that A_n recognizes, or accepts, a subset L_n of $\{0,1\}^n$ if and only if $A_n(\{0,1\}^n) = L_n$. To characterize the power of DNA algorithms, we adopt standard conventions used for circuits. Consider a (nonuniform) infinite family of DNA algorithms $\{A_n\}$, one algorithm per input size. We say that this family accepts language L if and only if the language accepted by A_n is $L \cap \{0,1\}^n$.

Given an initial multiset S of n-bit strands, important resources of a DNA algorithm are (i) the number of operations, (ii) the maximum number of bits, or alternatively of words, in a strand during the computation (we will see later that the time to do the mark operation is linear in the number of words in the strand), and (iii) the types of operations. We distinguish between computations that use the append operation and those that do not. This is because computations that use append rely on more complicated chemistry and results in longer strands.

Let Surface-Time($t(n)$) be the set of languages accepted by a family of DNA algorithms with $O(t(n))$ operations on n-bit strands. The first result is that Surface-Time($t(n)$) is exactly the set of languages that are accepted by a family of unbounded fan-in circuits with $O(t(n))$ gates. Let Circuit-Size($t(n)$) be the latter set. The constants in the O-notation in the following theorems are made explicit in the proofs.

Theorem 1 *Surface-Time*$(t(n))$ $=$ *Circuit-Size*$(t(n))$.

This theorem is analogous to the theorem of Boneh et al. [5] that the Circuit SAT problem can be solved using a solution-based model of DNA computation.

We are also interested in computations of languages when the append operation is not allowed. In this case, the erase operation is also unnecessary. We let Restricted-Surface-Time($t(n)$) be the corresponding set of languages. A language in this class can be expressed as the set of satisfying assignments of the following type of boolean formula with $O(t(n))$ gates. The formula is represented as a depth-three tree with the output gate (root) at level 0 and the n input variables and their negations labeling the leaves at level 3. The root is an **and**-gate; the gates at level 1 are either **or**-gates or **nor**-gates, and the gates at level 2 are **and**-gates. Call such a formula a depth-three-sat formula. (Equivalently, a depth-three-sat formula is a CNF formula in which the "literals" are replaced by and's or nand's of variables and their complements.) Let Depth-Three-Sat($t(n)$) be the set of languages that are accepted by a family of depth-three-sat formulas with $O(t(n))$ gates.

Theorem 2 *Restricted-Surface-Time*$(t(n))$ = *Depth-Three-Sat*$(t(n))$.

Let Surface-Time,Length$(t(n), l(n))$ be the set of languages that are computable by a family of DNA algorithms with $O(t(n))$ operations in which the maximum number of bits that are appended to the initial strand is $O(l(n))$ throughout the computation. From Theorem 1 it follows immediately that Circuit-Size$(t(n))$ = Surface-Time,Length$(t(n), t(n))$. To obtain a characterization of Surface-Time,Length$(t(n), l(n))$ for general $l(n)$, we need to define a new type of circuit, which we call a *blocked* circuit. The gates of such a circuit are organized into ordered blocks numbered $1, 2, \ldots,$. A subset of the gates of a block are designated as output gates. All inputs to gates in the ith block are either from other gates within that block, from the output gates of the $(i-1)$st block (if $i > 1$), or from inputs to the circuit. We let Circuit-Size,Block$(t(n), l(n))$ be the set of languages accepted by blocked circuits with $O(t(n))$ gates, $O(l(n))$ gates per block and $O(1)$ output gates per block. The proof of the following theorem is omitted in this extended abstract.

Theorem 3 *Surface-Time,Length*$(t(n), l(n))$ = *Circuit-Size,Block*$(t(n), l(n))$.

All of the above results can readily be extended to function computation, defined as follows. We say that a family $\{A_n\}$ of algorithms computes a function f if $A_n(\{0,1\}^n)$ is the set of strands of the form $xyf(x)$, where $x \in \{0,1\}^n$ and y is an arbitrary strand.

2 Chemical Realization of Model

We now describe how the model of Section 1.1 can be implemented using surface-based DNA chemistry. Preliminary work on implementing the mark and destroy operations on single-word strands is already underway [10].

2.1 Set Initialization; Word Design

A DNA strand, or oligonucleotide, can be thought of as a string over the alphabet (i.e. set of bases or nucleotides) $\{A, C, G, T\}$. A DNA strand has chemically distinct ends, known as the $3'$ and $5'$ ends. A DNA oligonucleotide is used to encode the value of a word plus the index of that word. A *strand* in our abstract model is simply the concatenation of multiple words with indices $1, 2, \ldots, k$ for some k. Also, a common sequence of bases known as a primer is placed at the $3'$ end of each strand to enable readout using the polymerase chain reaction (PCR).

To encode binary truth assignments for (up to 70) variables, Adleman and Lipton proposed that one-bit words be used, with 20 bases per word to identify the bit value and the variable (or word) index [2, 9]. The exact encoding of words was not specified. Each possible word can be synthesized using standard solid phase DNA synthesis, in which a large quantity of a desired DNA molecule is built up nucleotide by nucleotide on a support particle in sequential coupling steps. For example, a support with the nucleotide "A" attached may have the "A" reacted with a "C" to form a string of length 2 (known as a dimer), washed and the "C" coupled with "G" to form a string of length 3 (a trimer) and so on. Adleman [2] described how words can be concatenated together using ligation so that all 2^n n-bit words can be created in n ligation steps.

In our preliminary experimental work on an initial multiset $S = \{0,1\}^5$ [10], we are using one base per bit, plus labels on either end. This can extended to multiple words as follows. The word index is encoded using the labels and between the labels the bases A or T are used in half of the bit positions to represent 0 and 1 respectively, and C or G are used in the remaining positions to represent 0 and 1 respectively. The DNA synthesis scheme described above can easily be adapted for parallel synthesis of multiple-word strands, when this method of encoding is used. Namely, a mixture of two nucleotides is used at coupling steps that correspond to bits of a word, and just one nucleotide is used at coupling steps that correspond to the label of a word. For example, if two nucleotides are used together in five coupling steps, 32 different molecules are made and are present on the support. In our experimental work, an Applied Biosystems DNA synthesizer is used to generate the initial set; with this system a set of strands with 50 bases can be generated in an afternoon for about $100. In addition to the possibility of parallel synthesis of words, this method has the advantage of high information density, i.e. bits per base. However, words with the same index that differ in only one bit necessarily differ in only a single base, which makes implementation of a reliable mark operation more challenging as we explain later.

Schemes intermediate between the two already described are also possible, in which there are multiple bits per word but more than a single base difference between distinct words. However, parallel synthesis of words is no longer possible. To generate the set of all possible n-bit strands with w bits per word and greater than one base mismatch between any pair of words requires $2^w n/w$ synthesis steps plus n/w ligation steps.

Attachment chemistry describes the molecules at the interface of the surface and the oligonucleotides to be attached to the surface. Both the surface and one end of the oligonucleotides are specially prepared to enable this attachment. A good attachment chemistry ensures that the properly prepared oligonucleotides are immobilized to the surface at a high density, and that other oligonucleotides exposed to the surface later (for example, during hybridization) do not bind non-specifically to the surface. In our preliminary experiments on single-word strands [10], we use a glass surface and an attachment chemistry developed in the Smith laboratory [7] to attach oligonucleotides at the $5'$ end. The glass surface is modified with amino-reactive isothiocyanate functionalities in a multi-step process. For multi-word strands, it is necessary to attach oligonucleotides to the surface at the $3'$ end, in order to use polymerase extension in the mark operation as described later. A different chemistry can be used to do this.

2.2 Operations

We now describe how each of the operations of our abstract model can be realized on surfaces. All operations are based on standard chemical processes, namely hybridization, polymerase extension, ligation, and exonuclease degradation.

mark: Strands are marked simply by making them double-stranded. Under suitable conditions (such as temperature and salt concentration), single-stranded DNA *hybridizes*, or anneals, to form a double-stranded DNA molecule, or duplex, with its Watson-Crick complement, in which complementary pairs of bases form a bond. The Watson-Crick complement of a DNA strand is the sequence obtained by replacing each A with a T and vice versa, similarly replacing each C with a G and vice versa, and with the distinct chemical ends in the opposite order. For example, the Watson-Crick complement of $(5')$ACCTG$(3')$ is $(3')$TGGAC$(5')$. We begin by

describing how the mark operation works in the case that all strands have a single word, then extend this to the case of multiple-word strands with one constraint, and finally the general case of multiple-word strands with multiple constraints.

Single-word strands: First, the set of DNA oligonucleotides F that are Watson-Crick complementary to the strands that satisfy the constraint are synthesized. If there are w bits in the word and j of these are constrained then there are 2^{w-j} distinct strings in F. Then each of these hybridizes to its complement on the surface (if present). GC content has a very strong effect upon hybridization stability and hence upon hybridization conditions; hence if more than one bit is stored in a word, the GC content must be kept constant over all values of this word.

Hybridization discrimination refers to the degree to which only perfect matches occur in a hybridization reaction involving multiple strands. The fewer matches between two strands (or substrands), the less stable the hybridization between them. In previous studies [7] it was determined that excellent discrimination of even single-base mismatches is obtained using sequences of 15 bases. More experimentation is needed to determine if reliable single-base mismatch discrimination is possible when combinatorial mixtures are used.

Multiple-word strands, at most one constraint: In this case the method used for the single-word strands is combined with polymerase extension. Suppose that there is a single constraint on the ith word. Then, as in the previous case, a set F of DNA oligonucleotides that are Watson-Crick complementary to the constrained word values are synthesized and annealed to the oligonucleotides to be marked. The word labels ensure that annealing takes place at the proper word. This annealed word is used as a primer from which to initiate DNA synthesis. DNA polymerase adds nucleotides to the $3'$ hydroxyl of the primer polynucleotide, copying the complementary sequence to the end. Thus, marked strands are doubly-stranded from the constrained word to the free terminus. In the special case that the constraint-list is empty, polymerase extension may be done starting from the Watson-Crick complement of the initial primer that is common to every strand. The time to do this is linear in the length of a strand, since it involves processing the strand base by base. The time in the case of multiple-word constraints is also linear, as described next. This is a drawback of surface-based operations.

Multiple-word strands, multiple constraints: In this case, the result of the mark operation is that the strands to be marked are doubly-stranded starting from the lowest-numbered constrained word and strands that are not marked remain single-stranded. We now describe a scheme for the case of two constraints. This can be generalized to more than two constraints. Suppose that the words to be constrained are numbered a and $a+b$ and call the constraints constraint-a and constraint-$(a+b)$.

The idea is first to create a double-stranded region that extends between the two constrained words only on the strands to be marked and then to use polymerase extension as before to extend the double-stranded region all the way to the free terminus. The double-stranded region may be created one word at a time, using hybridization and ligation. One order in which to process the words is $a, a+1, \ldots, a+b$. Let F_a and F_{a+b} be the set of oligonucleotides representing the Watson-Crick complements of words that satisfy constraint-a and constraint-$(a+b)$ respectively. Let F_k be the set of oligonucleotides representing the Watson-Crick complements of all possible values for word number k, $a < k < a+b$. First, hybridization takes place using F_a, only on the strands satisfying constraint-a. Then hybridization followed by ligation is done using F_{a+1}. As a result, when $k = a+1$, strands that satisfy the constraints F_a have a two-word

strand annealed to them (at words numbered a and $a+1$) and strands that do not satisfy F_a have a one-word strand annealed to them (at word $a+1$). The one-word strands are "melted off" by raising the temperature. Assuming words (including labels) are about 15 bases long, sufficient difference between the melting temperature of one-word and two-word strands makes this possible. This process is repeated for $F_{a+2}, \ldots, F_{a+b-1}$. Finally, hybridization followed by ligation using F_{a+b} is done.

The strands that satisfy F_{a+b} but not F_a now have a single word annealed to them and this is melted off. The strands that satisfy both F_a and F_{a+b} (i.e. the strands to be marked) have a b-word strand annealed to them, and strands that satisfy only F_a have a $(b-1)$-word strand annealed to them. If b is small, the difference in melting temperature between these two types of strands may be used to melt off the strands with just $b-1$ words. For larger b, this scheme can be adapted by changing the order in which the words are processed, alternately working down from word $a+b$ and up from word a until they meet in the middle. Then at the end, strands to be marked have a strand of b words annealed to them whereas strands that are not to be marked have a strand of at most $\lceil b/2 \rceil$ words annealed to them, and the difference in melting temperature between these is greater. Further optimizations are possible that involve use of the append operation (details omitted).

unmark: This is done simply by washing the surface in distilled water and if necessary, raising the temperature. In the absence of salt, which stabilizes the double-stranded pairs, the complementary strands of DNA denature from the oligonucleotides on the surface and are washed away, leaving only the original single-stranded DNA attached to the surface.

destroy-marked, destroy-unmarked: Double-stranded or single-stranded DNA molecules may be selectively destroyed using enzymes known as exonucleases, which chew up DNA molecules from the end in, and exist with specificity for either the single-stranded or double-stranded form. We currently use the enzyme Exonuclease I to destroy single-stranded oligonucleotides.

append-marked(new-word): Since marked strands are double-stranded, the append operation can be implemented via ligation at the free terminus. The DNA polymerase employed in the mark operation will affect the nature of the free terminus of the marked strands. DNA polymerases which contain a $3' \rightarrow 5'$ proofreading activity (most naturally occurring procaryotic DNA polymerases do possess this activity) yield a blunt end. In this case, blunt-end ligation can be used to implement the append operation. Ligation will only occur at the duplex, and not at (unmarked) single-stranded oligonucleotides. If the $3' \rightarrow 5'$ proofreading activity is removed (such engineered polymerases are commercially available) they show a new non-templated $3'$ adenylation activity. That is, the base "A" is added to the $3'$ hydroxyl of the duplex creating a one base overhang, or "sticky end". A cloning vector developed in the Smith laboratory some years ago exploited this non-templated adenylation activity to facilitate the cloning of PCR products (TA cloning [11]). Ligation to "sticky ends" (using a complementary one base T overhang on the word to be ligated to the surface-bound duplex) is substantially more efficient than blunt-end ligation. Hence, in the case of a mark followed by an append with no destroy operation intervening, sticky-end ligation can be used.

append-unmarked(word1-value, word2-value): This can be done using hybridization of a splint oligonucleotide followed by ligation, exactly as in Adleman's experiment [1].

erase-marked: This operation is implemented using a restriction enzyme which recognizes a short sequence of bases, called a restriction site, and cuts the strand at that site when the

sequence is double-stranded. The restriction enzyme may have a 4, 6, or 8 base pair recognition sequence, and may leave a blunt end, 3' overhang, or 5' overhang. The restriction site is included in the first appended word and must be chosen to have a recognition sequence not represented in the combinatorial word structure. For example, the restriction enzyme Msp I recognizes the 4 base-pair sequence $(5')$CCGG$(3')$ (when annealed to its complement $(3')$GGCC$(5')$) [8].

3 Proofs of Results

Theorem 1 *Surface-Time*$(t(n))$ = *Circuit-Size*$(t(n))$.

Proof: Let C be a circuit with n inputs and t gates. Without loss of generality, assume that the output of C is an **and**-gate, all gates of C are either **or**-gates or **and**-gates, and the n inputs are available both in negated and unnegated form.

There are four operations in the DNA algorithm per gate, plus five additional operations, two at the start and three at the end. Initially a trivial word with a single 0 bit is appended to all strands, using the mark operation with empty constraint list and the append-marked(0) operation. Then for every strand in parallel, the value of each gate is computed in topological order and is appended to each strand.

If the gate is an **and**-gate, a mark operation marks those strands that set the **and**-gate to true. The mark operation has one constraint per input to the **and**-gate. Then an append-marked operation appends a word containing the bit "1", which is the value of the gate for marked strands. Finally, two append-unmarked operations append a word containing the bit "0" to unmarked strands. Two operations are needed since the append-unmarked operation requires knowledge of the last word on a strand. This can only have two possible values, since each appended word contains just one bit. (This is the reason for the initial append operation.)

If the gate is an **or**-gate, the mark operation marks exactly those strands that set the gate to false. Then an append-marked operation appends a word containing the bit "0", which is the value of the gate for marked strands. Finally, two append-unmarked operations append a word containing the bit "1" to unmarked strands.

The last three instructions of the DNA algorithm are: mark all of those strands for which the output of the circuit is "1," destroy-unmarked strands, and erase. The strands remaining on the surface are exactly those from the initial set that satisfy the circuit. Clearly the number of operations in the DNA algorithm is at most $4t + 5$.

Conversely, let A be a DNA algorithm with n inputs and t operations. We describe how to construct a circuit that accepts the language $A(\{0,1\}^n)$. In this extended abstract, we assume that each append operation appends just one bit. Let $n + l$ be the maximum number of bits on a strand at any point in the computation. In addition to the n input gates (corresponding to the initial n bits on a strand), the circuit has l "initial" gates with arbitrary fixed constant value. The circuit also has one output **and**-gate.

Gates are added to the circuit for each operation in order that the operations appear in the algorithm A. To guide the construction of the circuit, two sets of gates are maintained. The values of these sets change as operations are processed; consider the sets as a function of the number of operations already processed. One set is called *current-bits*. For each i, $1 \leq i \leq n+l$, this set contains one pair $(i, gate)$ that shows which gate represents the current value of the

ith bit of a strand whose initial n-bit value is the input to the circuit. Initially, this contains one pair for each of the n input gates and each of the l initial gates. (If a bit i does not currently exist on the input strand, the set *current-bits* still contains a pair (i,gate) which is meaningless). There is also a set called *current-marked* which is initially empty. Throughout the construction, the following invariant is maintained: a strand in the initial multiset S is marked after t operations of the algorithm A if and only if when this strand is the input to the circuit, some gate currently in the set *current-marked* has value "1."

For each mark operation, an **and**-gate is added to the circuit that takes as input the gates (or their negations) of the circuit that currently represent the constrained bits in the constraint-list of the mark operation. This **and**-gate is added to the set *current-marked*. For each unmark operation, the set *current-marked* is reset to empty, and no changes are made to the circuit. For each erase operation, no changes are made. (Think of this as follows: entries in the set *current-bits* that correspond to erased bits are no longer valid, but will never be referenced anyway in the case that the input strand is unmarked. This is because the mark operation is restricted to only refer to bits that are actually present on a strand.)

It remains to consider the destroy and append operations. For each such operation, a new **or**-gate is added that takes as input all gates in the set *current-marked*. Call this the indicator gate. The output of the indicator gate is "1" if and only if the input strand is currently marked. If the operation is a destroy-marked operation, the negated output of the indicator gate is input to the output **and**-gate. Thus, a strand that is currently marked (and therefore destroyed by the algorithm A) fails to satisfy the circuit. If the operation is a destroy-unmarked operation, the indicator gate is input to the output **and**-gate. Thus, in this case a strand that is currently unmarked (and therefore destroyed by the algorithm A) fails to satisfy the circuit.

Suppose that the operation is append-marked(new-word). Recall that we are assuming that the appended word has just one bit in it; hence new-word is a bit. In this case, one new gate – call it new-gate – is added that represents the bit new-word. Suppose that this is the ith bit in the strand. If the indicator gate has value 0, the ith bit still has the value of the gate given in the set *current-bits*. If the indicator gate has value 1, the ith bit should have the value new-word. A simple constant-size circuit can ensure that new-gate has the correct value. (This circuit takes as input the value of the indicator gate, a gate with value new-word, and the gate representing the current value of the bit; its output is new-gate). Finally, the pair $(i, \text{new-gate})$ is added to the set *current-bits*, replacing the entry $(i, *)$ already in the set. The append-unmarked operation is similar to the append-marked operation. □

In the simulation of a circuit by a surface-based DNA algorithm described above, all appended words contain only one bit which represents the value of a single gate. Multiple bits per word are possible at the cost of increasing the number of operations of the algorithm. To obtain w bits per word, gates can be processed in groups of size w, with a separate phase for each of the 2^w possible values of the gates. Thus, a circuit family with $t(n)$ gates can be "simulated" by a family of DNA algorithms with $O(2^w t(n)/w)$ operations and with a total of $t(n)/w$ w-bit words appended to each initial strand.

Theorem 2 *Restricted-Surface-Time($t(n)$) = Depth-Three-Sat($t(n)$).*

Proof: Let F be a depth-three-sat formula with n inputs and t gates. The following restricted DNA algorithm computes exactly the set of satisfying inputs to F and uses at most a constant times t operations from the set {mark, destroy-marked, destroy-unmarked, unmark}.

```
for each or-gate OR of F do{
     for each and-gate AND of F that is a child of OR do
        { mark exactly those strands that satisfy AND }
     destroy-unmarked
     unmark                }

for each nor-gate NOR of F do{
     for each and-gate AND of F that is a child of NOR do
       { mark exactly those strands that satisfy AND }
     destroy-marked
     unmark                   }
```

Conversely, let A be a restricted DNA algorithm with n inputs and t operations. Without loss of generality we can assume that the sequence of operations in A has the following structure. It is a sequence of blocks, where each block is a sequence of mark operations followed by a destroy (either destroy-marked or destroy-unmarked) operation, followed by an unmark operation. replaced by a **nor**-gate. If A is organized into m blocks with a total of l mark operations (and m destroy and m unmark operations at the end of each block) then there is a depth-three-sat formula with a total of m **or**-gates and **nor**-gates at level 1 and l **and**-gates at level 2 that accepts the same language as A (details omitted). □

4 Conclusions

We have shown how to evaluate the output of a boolean circuit in parallel on many inputs using surface-based manipulation of DNA strands. The inputs are encoded using single-stranded DNA and the "computation" is done using the chemical processes of hybridization, ligation, polymerase extension and exonuclease degradation.

There are many directions for further research. (i) Detailed designs for information storage on DNA strands need to be developed, and the chemical processes needed for each operation need to be refined and tested. (ii) Errors inherent in the operations need to be quantified and algorithmic methods for performing robust computation in the face of these errors need to be developed. (iii) It may be desirable to add new operations to the model, specifically ways to erase words from strands in a more flexible manner than that described here, or to duplicate strands on the surface. (iv) It would be interesting to understand how the operations of our model may be useful in "computations" involving general DNA strands, i.e. not just those designed to encode binary information. For example, function computation in our model is akin to synthesis of sets of DNA. The company Affymetrix performs synthesis of complex sets of DNA using light-directed methods, in which bases are added one at a time to strands as a function of the location of a strand on the surface. In contrast, our model allows words to be added to a strand as a function of its content rather than its location. Finally, readout strategies need to be developed for both surface- and solution-based computation.

References

[1] Adleman, L. M. (1994). Molecular Computation of Solutions to Combinatorial Problems. Science, 266, 1021-1024.

[2] Adleman, L. M. (1995). On Constructing a Molecular Computer, Manuscript, Computer Science Department, University of Southern California.

[3] Amos, M., Gibbons, A., and Hodgson, D., Error-resistant implementation of DNA computations, Proc. Second Annual Meeting on DNA-Based Computers (to be published by the American Mathematical Society).

[4] Baum, E. (1996), DNA sequences useful for computation, Proc. Second Annual Meeting on DNA-Based Computers (to be published by the American Mathematical Society).

[5] Boneh, D., Dunworth, C., and Lipton, R. J. (1995), On the computational power of DNA, Princeton CS Technical Report CS-TR-489-95.

[6] Deaton, R., Murphy, R. C., Garzon, M., Franceschetti, D. R., and Stevens Jr., S. E. (1996), Good encodings for DNA-based solutions to combinatorial problems, Proc. Second Annual Meeting on DNA-Based Computers (to be published by the American Mathematical Society).

[7] Guo, Z., Guilfoyle, R. A., Thiel, A. J., Wang, R. and Smith, L. M. (1994). Direct fluorescence analysis of genetic polymorphisms by hybridization with oligonucleotide arrays on glass supports. Nucl. Acids Res., 22, 5456-5465.

[8] Jentsch, S. (1983), J. Bacteriol 156, 800-808.

[9] Lipton, R. J. (1995). DNA Solution of Hard Computational Problems. Science, 268, 542-545.

[10] Liu, Q., Guo, Z., Condon, A.E., Corn, R.M., Lagally, M.G., and Smith, L.M. (1996), A Surface-Based Approach to DNA Computation, Proc. Second Annual Meeting on DNA-Based Computers (to be published by the American Mathematical Society). Available on the web at http://www.cs.wisc.edu/ condon/papers/dna.ps.

[11] Mead, D. A., Pey, N. K., Herrnstadt, C., Marcil, R. A. and Smith, L. M. (1991), A Universal Method for the Direct Cloning of PCR Amplified Nucleic Acid. Biotechnology, 9(7), 657-663.

[12] Mir, K. U. (1996), A restricted genetic alphabet for DNA computing, Proc. Second Annual Meeting on DNA-Based Computers (to be published by the American Mathematical Society).

[13] Pease, A. C., Solas, D., Sullivan, E. J., Cronin, M. T., Holmes, C. P. and Fodor, S. P. (1994) Light-generated oligonucleotide arrays for rapid DNA sequence analysis, Proc. Nat. Acad. Sci. USA, 91, pp 5022-5026.

[14] Smith, L. M. (1988), Automated synthesis and sequence analysis of biological macro-molecules, Analytical Chemistry 60, 381A-390A.