Arithmetic and Logic Operations with DNA

Vineet Gupta[†], Srinivasan Parthasarathy, Mohammed J. Zaki

[†] Department of Chemistry, University of Rochester, Rochester, NY 14627,
`vtga@uhura.cc.rochester.edu`
Department of Computer Science, University of Rochester, Rochester, NY 14627,
{`srini,zaki`}`@cs.rochester.edu`

Abstract

A lot of current research in DNA computing has been directed towards solving difficult combinatorial search problems. However, for DNA computing to be applicable on a wider range of problems, support for basic computational operations such as logic operations like AND, OR and NOT and arithmetic operations like addition and subtraction is necessary. Unlike search problems, which can be solved by generating all possible combinations and extracting the correct output, these operations mandate that only a unique output be generated by specific inputs. The question of suitability of DNA for such simple operations has so far largely been unaddressed. In this paper we describe a novel method for using DNA molecules to solve the basic arithmetic and logic operations. We also show that multiple rounds of operations can be performed in a single test tube, utilizing the output of an operation as an input for the next. Furthermore, the operations can be performed in a linear series or a series-parallel fashion and operators can be mixed to form any operation sequence.

1 Introduction

In recent work, Adleman [1] and Lipton [8] presented the idea of solving difficult combinatorial search problems using DNA molecules. These studies showed that DNA computing may have an advantage over electronic computers for such problem domains due to the massive parallelism inherent in DNA reactions. However, for DNA computing to be applicable on a wider range of problems, support for simple computational operations is necessary [5, 12, 14]. Boolean operators such as AND, OR and NOT, and arithmetic operators such as addition and subtraction are the fundamental operations of an electronic computer. In contrast to search problems, which can be solved by generating all possible combinations and extracting the correct output, these operations mandate that only a unique output be generated by specific inputs. Although DNA computers might not improve on current silicon technology for these operations, as Adleman [2] has pointed out, "they can contribute to our understanding of the nature of computation." There has been some work in simulating boolean circuits and performing additions and matrix multiplications with DNA. Ogihara and Ray [9] have shown how DNA computers can simulate Boolean circuits with a small overhead. Oliver [10] has shown how DNA based methods can be used to calculate the product of Boolean matrices or matrices containing positive, real numbers. Guarnieri *et al.* [6] have proposed a clever way to add two binary numbers. The novelty of their approach is the introduction of a place holder for the carry position while performing additions. [1] As they themselves point out, a limitation of their approach is that

[1] In contrast to boolean operators, ADD can generate two output bits (e.g. 1+1 = 10). In this case that bit position is assigned a value of zero (0) and the one (1) is carried over. The carried over bit is added to the next bit position and the value of that position adjusted accordingly. The operation is repeated as many times as there is a carry over.

the output strand of one operation cannot serve as the input strand for another round of addition. This has been a constraining factor so far in performing a sequence of operations in a single vessel. In this paper we show that this limitation can be overcome (in DNA), allowing a number of basic series and series-parallel bit operation sequences in solution.

The rest of the paper is organized as follows. We present our bit encoding scheme, and our approach to performing a single bit operation in solution in section 2. We extend this framework to handle a *series* or a linear sequence of mixed operations in section 3. We then show how to combine multiple series to perform *series-parallel* operations in section 4. Finally we conclude in section 5, with a discussion on the advantages and limitations of our approach, as well as directions for future work.

2 Bit Encoding and Single Operations

Table 1 A. The truth table for various binary operations

Input 1	Input 2	Output			
		NAND	AND	XOR	ADD
0	0	1	0	0	0
0	1	1	0	1	1
1	0	1	0	1	1
1	1	0	1	0	10

Table 1 B. The truth table encoding using dinucleotide "bits"

Level 1	Level 2	Level 3 (Interpretation of the Duplex for the Output)			
Input Strand	Operand Strand	NAND	AND	XOR	ADD
UA = 0	PT = 0	1	0	0	0
	AT = 1	1	0	1	1
AU = 1	TA = 0	1	0	1	1
	TP = 1	0	1	0	10

In an electronic computer an operator is succinctly represented by a truth table, a table of all possible combinations of the input bit values and their corresponding output values, as shown in table 1 A. Our approach is to encode these truth tables in DNA using a three-level scheme. An operation is represented in terms of DNA hybridization. For each binary operation, the two bit strings are represented with two different DNA single strands. The first string is called the “input” and the second the “operand” strand. Each bit is represented with a dinucleotide unit, and a bit string with a sequence of dinucleotides. The natural DNA bases Adenine (A), Thymine (T), Uracil (U) and a non-natural base 7-deaza-adenine (P) (shown in figure 1) are used for constructing the dinucleotides. A non-natural base is needed in order to realize all possible input and operand bit values in DNA; a base which can pair with the specificity of A and yet is chemically distinct. 7-deaza-adenine [2] base-pairs selectively with U and T just like A which makes it ideal for our application. The input DNA strand is constructed with dinucleotides 5’-AU representing bit 1, and 5’-UA, the bit 0, where 5’ and 3’ denote directionality of the DNA strand. The operand strand is constructed with

[2]2-aminopurine and 2-aminoadenine could potentially be used too.

dinucleotides 3’-TA and 3’-PT, representing bit 0, and the dinucleotides 3’-AT and 3’-TP, representing bit 1. The table 1 B shows how these dinucleotides can base-pair to allow for all possible combinations of the input strand and the operand strand. Since, as we will show later in the paper, the operand strands do not have to be sequenced to obtain the final result, but instead carry an output strand, we do not have to chemically distinguish between A and P at any stage. DNA strands carrying TA (similarly AT) might have a different output appended to it than the DNA strand carrying TP (similarly PT) depending upon the result of the calculation. If the input strand is constructed in the 5’ to 3’ direction the operand strand is constructed in the 3’ to 5’ direction and vice versa. The input strand is constructed using only A and U (level 1), while the operand strand with A, T, and P (level 2). This keeps the input distinct from the operand, yet retains the same base-pairing structure and allows all possible combinations of the input and operand bits (see table 1). As a result of an operation the input strand hybridizes with its complementary operand strand to form a double stranded DNA complex. This output is interpreted according to the truth table (level 3) of the operator applied. Table 1 B shows our encoding scheme along with the truth tables for different boolean and arithmetic operations such as NAND, AND, XOR and ADD (addition). The truth table encoding can easily be extended to a number of other operators.

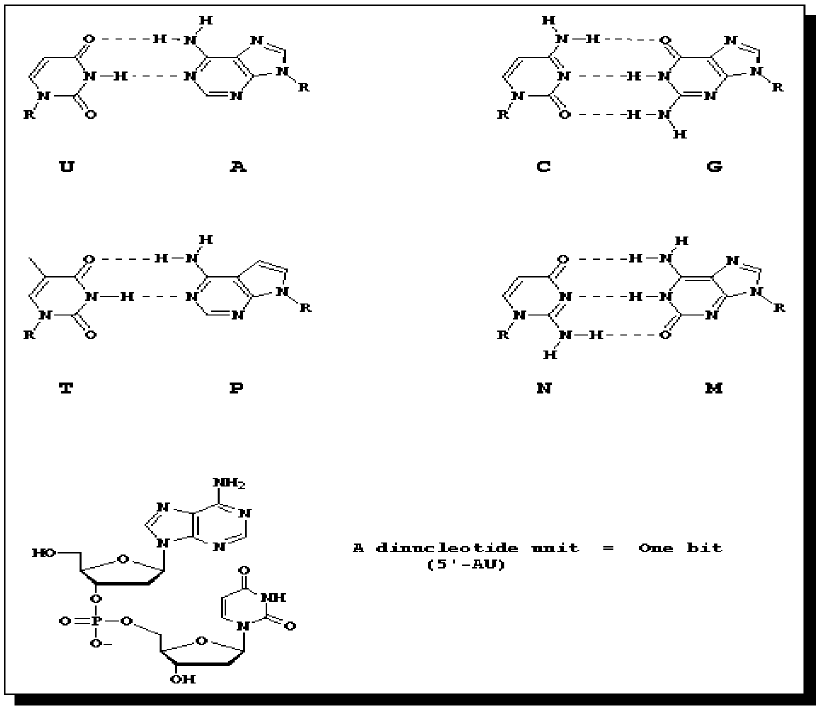

Figure 1: Chemical structure of the bases used; Adenine (A), Thyamine (T), Uracil (U), 7-deaza-adenine (P), Guanine (G), Cytidine (C), iso-Guanine (M) and iso-Cytidine (N). Also shown are some example base-pairs, and a dinucleotide “bit” unit.

The NAND operator is a *universal* boolean operator, which means that any boolean operation can be represented in terms of a sequence of NAND operators. Our first example shows the execution, in DNA, of the operator NAND on the two binary strings 1001 and 0101, as depicted in table 2. The input bit string 1001, is represented as the DNA sequence 5’-AUUAUAAU. All 16 possible DNA sequences are used to represent the operand 0101, since there are two alternative dinucleotides for each bit in the level 2 encoding. However, only one of the operand strands carries a DNA sequence complementary to the sequence of the input strand. This strand (3’-TAATPTTP) is thus the only one that can hybridize with the input strand to yield a unique output duplex. The output, when decoded using the truth table for NAND, yields the correct answer of 1110. In fact, the same output can be interpreted differently if decoded using a different operator's truth table. For instance, if we were to perform the operation (1001 XOR 0101) instead of (1001 NAND 0101), the input and all of the operand strands would still be the same, and would yield the same output duplex. However, interpreting this output duplex according to the truth table for XOR (see table 1 B) would generate a different but correct answer of 1100. This shows that single operations can be performed efficiently with DNA using the proposed approach.

Table 2 Example of a NAND operation

Input Strand	Operand Strands		Unique Output
1 0 0 1	0 1 0 1		**1 1 1 0**
5′-AUUAUAAU	3′-TATPPTAT	3′-TATPPTTP	5′-AUUAUAAU
	3′-TATPTAAT	3′-TATPTATP	3′-TAATPTTP
	3′-TAATPTAT	**3′-TAATPTTP**	
	3′-TAATTAAT	3′-TAATTATP	
	3′-PTTPPTAT	3′-PTTPPTTP	
	3′-PTTPTAAT	3′-PTTPTATP	
	3′-PTATPTAT	3′-PTATPTTP	
	3′-PTATTAAT	3′-PTATTATP	

3 Mixed Operation Series

In order to extend this framework to handle a *series* operation, i.e. a linear sequence of operations, we have to address two main issues: 1) how can the output of one operation, a double strand, be used as the input for the next operation, and 2) how to ensure that the operations take place only in the desired order. As shown in the first example from table 2, *each input strand can bind with only one of the operand strands producing a duplex representing the unique output*. As a solution to the first problem we attach, *a priori*, the corresponding output value to each of the operand strands in the form of a single stranded DNA. The DNA sequence of the output is constructed using the level 1 encoding from table 1 B. Thus, each operand strand carries a covalently linked output strand which does not interfere with its operation of hybridizing with the input strand, but instead results in a duplex with a sticky-end upon one such complexation. This sticky-end can then act as the input strand for the next operation. Not only does this allow a series operation involving a single operator, but also mixed operators, since the output strand attached to the operand is dependent only on a particular operator.

To solve the second problem, we use a unique DNA sequence tag on each strand. This tag is just a length 4 DNA segment comprised of four bases, the natural bases Guanine(G) and Cytidine (C) along with the non-natural bases iso-Guanine (M) and iso-Cytidine (N), shown in figure 1. This allows for a maximum of 256 unique tags, which gives us the option of performing 256 consecutive operations. This number can be increased by simply using a longer tag sequence. Each input strand is linked with a unique tag, while the operand strand next in order in the sequence, is linked with the complementary tag. The input can thus only hybridize with the next operand. A unique tag for the next step is also linked to the output strand, since it serves as the input for the next operation. The tags also prevent the formation of unproductive duplexes. For example, since each of the strands is made up of an operand and an output, they may become self-complementary. The tags prevent this from happening. Using this approach, we generate unique DNA sequences as inputs and obtain a unique final output resulting from the desired sequence of operations.

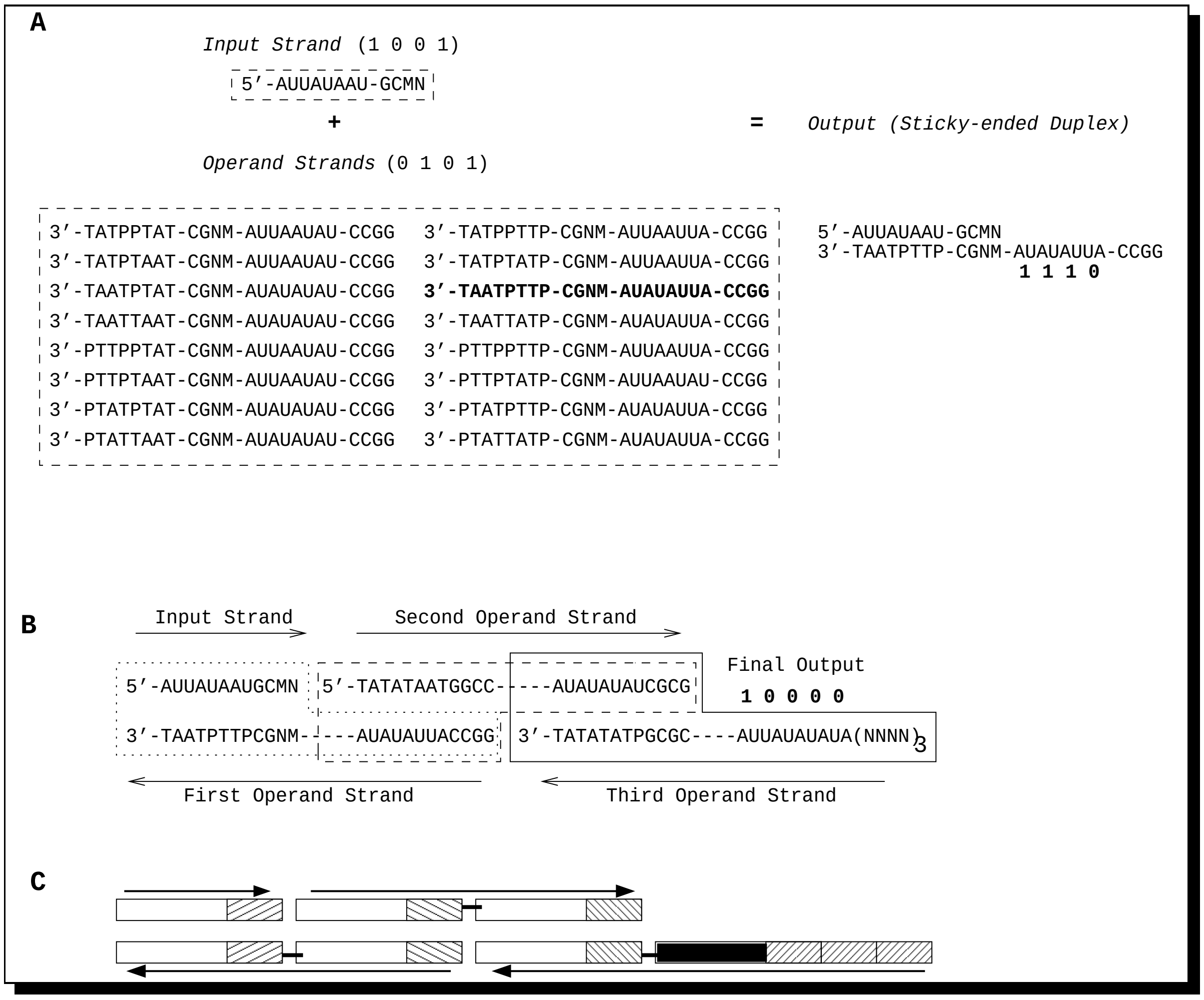
A
Input Strand (1 0 0 1)
5′-AUUAUAAU-GCMN
+
Operand Strands (0 1 0 1)
=
Output (Sticky-ended Duplex)
3′-TATPPTAT-CGNM-AUUAAUAU-CCGG
3′-TATPTAAT-CGNM-AUUAAUAU-CCGG
3′-TAATPTAT-CGNM-AUAUAUAU-CCGG
3′-TAATTAAT-CGNM-AUAUAUAU-CCGG
3′-PTTPPTAT-CGNM-AUUAAUAU-CCGG
3′-PTTPTAAT-CGNM-AUUAAUAU-CCGG
3′-PTATPTAT-CGNM-AUAUAUAU-CCGG
3′-PTATTAAT-CGNM-AUAUAUAU-CCGG
3′-TATPPTTP-CGNM-AUUAAUUA-CCGG
3′-TATPTATP-CGNM-AUUAAUUA-CCGG
3′-TAATPTTP-CGNM-AUAUAUUA-CCGG
3′-TAATTATP-CGNM-AUAUAUUA-CCGG
3′-PTTPPTTP-CGNM-AUUAAUUA-CCGG
3′-PTTPTATP-CGNM-AUUAAUAU-CCGG
3′-PTATPTTP-CGNM-AUAUAUUA-CCGG
3′-PTATTATP-CGNM-AUAUAUUA-CCGG
5′-AUUAUAAU-GCMN
3′-TAATPTTP-CGNM-AUAUAUUA-CCGG
1 1 1 0
B
Input Strand
Second Operand Strand
5′-AUUAUAAUGCMN
5′-TATATAATGGCC-
---AUAUAUAUCGCG
Final Output
1 0 0 0 0
3′-TAATPTTPCGNM-
---AUAUAUUACCGG
3′-TATATATPGCGC----AUUAUAUAUA(NNNN)3
First Operand Strand
Third Operand Strand
C

Figure 2: Example of a series operation. A) The first operation : (1001 NAND 0101); B) The whole series : (((1001 NAND 0101) XOR 0001) ADD 0001); C) A template of the structure generated (The arrows denote directionality, the unfilled boxes the input and operands, the black box the output, and the pattern-filled boxes the tags).

In figure 2, we present an example of how three different operators – NAND, XOR and ADD – can be performed in succession on DNA. The example solves the series operation sequence (((1001 NAND 0101) XOR 0001) ADD 0001). The first two bit strings are the same as those used in our first example from table 2. The input strand (5’-AUUAUAAU-GCMN) has the bit string 5’-AUUAUAAU (1001) linked with the tag 5’-GCMN. Figure 2 A shows the 16 possible representations of the operand strand (0101) that could possibly bind with the input (similar to our first example). Only one strand hybridizes with the input, namely the strand 3’-TAATPTTP-CGNM-AUAUAUUA-CCGG, where 3’-TAATPTTP is the bit string 0101, 3’-CGNM is the complement of the tag 5’-GCMN, 3’-AUAUAUUA is the attached output (1110) of the operation and 3’-CCGG is the tag for the next operation. The result of the first operation is thus a duplex with a single stranded overhang (3’-AUAUAUUA-CCGG), which is ideally suited to serve as the input strand for the next operation – XOR 0001.

Our second operand strand, representing 0001, also has 16 possible representations (not shown). Only 5'-TATATAATGGCC-AUAUAUAUCGCG (see figure 2 B) hybridizes with the output of the previous operation, producing a duplex with the overhang 5'-AUAUAUAUCGCG as the output (1111) of the current operation. Out of the 16 possible third operand strand (0001) DNA sequences, for ADD 0001, only 3'-TATATATPGCGCAUUAUAUAUA(NNNN)$_3$ hybridizes with the output of the previous round of operations. 5'-(NNNN)$_3$ is a special tag signifying that the sequence of operations has completed and is used for extracting the result. The unique result of our operation 3'-AUUAUAUAUA, when decoded using the truth table, is 10000, which is the correct result of our operation sequence. This examples shows that our scheme generates a unique output from a series operation in solution.

4 Series-Parallel Operations

In order to emulate the more complex boolean circuits as in an electronic computer, support for *series-parallel* operation sequences is desirable. A series-parallel operation sequence merges two or more series operation sequences. In figure 3 we show how the above architecture can be extended to handle a series-parallel operation sequence in a single test tube. The output of the series operation (from the previous example, figure 2 B), a duplex with an overhang is used as both the input and the operand complex, performing the operation $\mathcal{I}$ NAND $\mathcal{O}$, where $\mathcal{I}$ = $\mathcal{O}$ = (((1001 NAND 0101) XOR 0001) ADD 0001). However, instead of the typical operand strand which has an output attached to it, the operand complex for the series-parallel operation has two attached outputs – the output of the series-parallel operation followed by the output of the series operation sequence, shown in figure 3 A). The final step is a copy-operation – AND 11111 (which simply replicates the output). The result of this operation sequence is a three-arm junction [15, 17] with the correct output (01111) as the overhang, ideally suited for performing more operations.

5 Discussion and Conclusion

Only a theoretical model is presented in this paper. However, it is based on well-established techniques of biological chemistry. To practically carry out the operations in a test tube, we can attach the input strand 5'-AUUAUAAUGCMN to a magnetic bead, and add all the operand strands for the different operations to be performed, to the solution. The solution can then be heat-denatured, annealed and ligated. [3] All strands with the magnetic bead attached can be filtered off, and the output strand end-sequenced to produce the result.

For an n bit long string ($2n$ length DNA strand), with tag length m, our technique must satisfy some constraints. The length of an input/operand strand is $m + 2n$. Since the mismatch tolerance of a duplex increases with its length, at present duplexes only about 20 base-pairs long can distinguish single-base mismatches [16], introducing the constraint $m + 2n \leq 20$ on our system. Furthermore, for m operations, we need at least $m \cdot 2^n$ distinct DNA strands in the solution. For nanomolar quantities of each, thermodynamical limitations would put the upper bound at approximately 10^{14} different strands. The first constraint, though, clearly subsumes the second. The total length of the output complex, $2n \cdot m$, might also be limited by various factors that affect the reliability, such as the hybridization conditions, nearest-neighbor interactions (sequence context), and the efficiency of both the successive ligations and the result extraction process. We use two-fold (internal and end) mismatch discrimination system to prevent formation of unwanted (mismatched) duplexes/complexes. Each internal mismatch is at least two base long since we use dinucleotides

[3]Ligation can be performed either enzymatically, using DNA ligase enzyme, or chemically using Letsinger's method [7]. The DNA strands will have to be properly end-modified for either of the cases. The modifications should not affect their hybridizing properties.

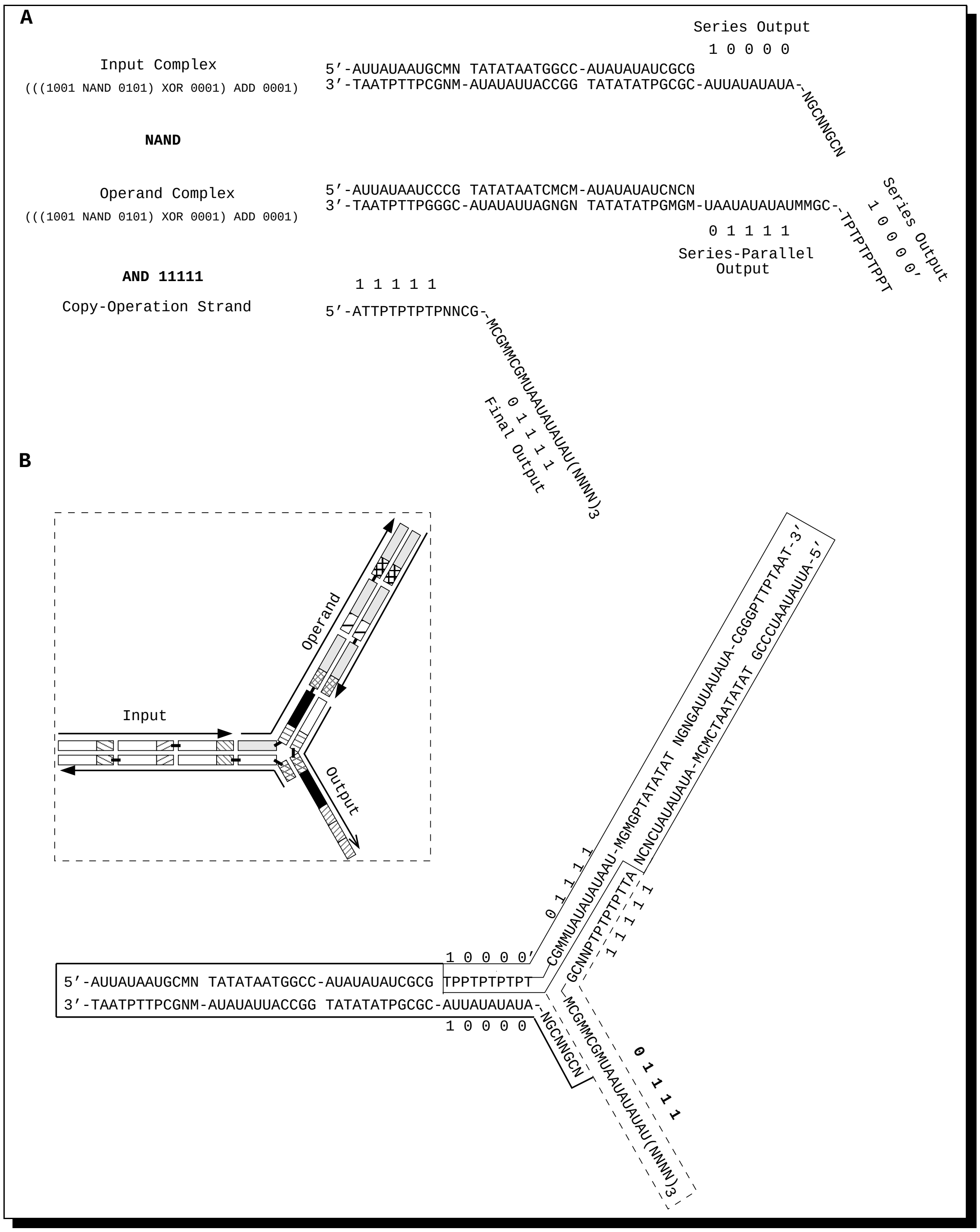
A
Series Output
1 0 0 0 0
Input Complex
(((1001 NAND 0101) XOR 0001) ADD 0001)
5′-AUUAUAAUGCMN TATATAATGGCC-AUAUAUAUCGCG
3′-TAATPTTPCGNM-AUAUAUUACCGG TATATATPGCGC-AUUAUAUAUA-NGCNNGCN
NAND
Operand Complex
(((1001 NAND 0101) XOR 0001) ADD 0001)
5′-AUUAUAAUCCCG TATATAATCMCM-AUAUAUAUCNCN
3′-TAATPTTPGGGC-AUAUAUUAGNGN TATATATPGMGM-UAAUAUAUAUMMGC-TPTPTPTPPT
Series Output
1 0 0 0 0′
0 1 1 1 1
Series-Parallel Output
AND 11111
Copy-Operation Strand
1 1 1 1 1
5′-ATTPTPTPTPNNCG-MCGMMCGMUAAUAUAUAU(NNNN)3
0 1 1 1 1
Final Output
B
Operand
Input
Output
5′-AUUAUAAUGCMN TATATAATGGCC-AUAUAUAUCGCG TPPTPTPTPT
3′-TAATPTTPCGNM-AUAUAUUACCGG TATATATPGCGC-AUUAUAUAUA-NGCNNGCN
1 0 0 0 0′
1 0 0 0 0
CGMMUAUAUAUAAU-MGMGPTATATAT NGNGAUUAUAUA-CGGGPTTPTAAT-3′
NCNCUAUAUAUA-MCMCTAATATAT GCCCUAAUAUUA-5′
GCNNPTPTPTPTTA
0 1 1 1 1
1 1 1 1 1
MCGMMCGMUAAUAUAUAU(NNNN)3
0 1 1 1 1

Figure 3: Example of a series-parallel combination operation (Input NAND operand); **Input**: (((1001 NAND 0101) XOR 0001) ADD 0001) = 10000 (same as in figure 2); **Operand**: same as **Input**; **Intermediate Result**: 10000 NAND 10000 = 01111; **Copy-operation**: 01111 AND 11111 = 01111 (Final Output). A) The sequence of the input complex, operand complex, and the copy-operation strands; B) The three-arm junction generated as a result of the series-parallel operation. (Inset) A template of the three-arm junction structure.

as single ”bits”. Thus the stability of the perfectly matched duplex is much higher than that of a mismatched duplex. Moreover, ligases are very sensitive to end mis-matches. Only perfectly matched duplexes are ligated by the enzyme. This gives us a very good mismatch protection at the ends of each duplex. To improve the reliability, the end-tag can be used for PCR amplification of the output. The PCR amplified output can be sequenced to also verify the operations performed. Finally, the number of series-parallel operations may be limited due to the steric constraints (branching-out) that three-arm junctions introduce.

There are 4^{2n} different sequences possible for a $2n$ length DNA strand, if random encodings are used. An advantage of using our fixed encoding scheme is that we need to synthesize only 2^n DNA sequences per operation [4], which is a very small subset of 4^{2n}. The increase in demand of DNA strands is also linear in the number of operations rather than exponential ($m \cdot 2^n$ instead of $2^m \cdot 2^n$). Furthermore, since all possible representations of the operands, for each of the operations, are added to the solution in the beginning, all possible outputs are produced in the solution. We can then simply add a given input to the solution, which will bind to its complementary sequence, and then extract the unique output. The same solution can then be reused with different inputs, for sequential extraction of the corresponding outputs. Thus the vessel serves as a black-box (representing some function to be performed), which takes in different input values to produce the corresponding outputs, without having to perform the operation again. Moreover, the fact that all the intermediate results are present in the final output, prevents the loss of any information about the computation, and can be used for implementing reversible logic gates [3].

There are some other advantages to our approach. The use of dinucleotides as single bit units lets us represent information at a much higher bit density than any previous methodology, and brings it very close to the theoretical limit of a bit per base. [5] Our methodology also makes possible a succinct representation of basic computational operators. The use of different bases for encoding the bits and the operation order permits series as well as series-parallel operation sequences in one test tube, producing a unique result, and simplifies the extraction process. The unique tags help keep the operation sequence in order. They also avoid the *fan-out* problem where a randomized sequence of molecules can produce an exponentially increasing number of product strands with the progression of the operation sequence. We use 2^n molecules for each step, which interact with 2^n molecules from the previous step to generate the same number of molecules for the next step.

Generation of DNA molecules carrying sticky-ends as the output of each operation, in the reaction vessel, is another nice feature of our scheme. This output is well suited to serve as the input for the next operation which enables us to accomplish the generic input/output semantics of an electronic computer. This is an important first step in solving problems using DNA computing where the DNA molecules have to go through multiple rounds of computation. Moreover, in most of the prior approaches where DNA was used for computing, the encoding was application specific and based on the particular application in mind. One of the most important advantages of our proposed mechanism is that it makes both encoding and computation more general (uniform) and application independent. As our examples show, almost any basic computational operation can be carried out on our “DNA computer”. The addition of a memory is an important issue and we plan to incorporate that into our model. See [13] for a solution to this problem. They propose a *sticker* based model which has a random access memory.

In conclusion, we present a new approach for computing with DNA. The ability to perform complex bit operations in solution might help us learn more about the nature of computation and lead to the development of better DNA based computers, capable of solving a wide range of complex problems.

[4] All the strands can be synthesized in a single combinatorial synthesis cycle using techniques of photolithography and nucleic acid chemistry [4, 11].

[5] Although, in principle mono-nucleotides can be used as single bits, there may be some complications to using such strands for performing different operations. For example, a long sequence of a mono-nucleotide can form unwanted triplexes.

Acknowledgements

We thank R. Jakubiak, Prof. E. T. Kool, H. S. Malik, Prof. B. L. Miller, Prof. M. Ogihara and Prof. A. Ray for their insightful comments and helpful discussions.

References

[1] Leonard M. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266:1021–1024, November 1994.

[2] Leonard M. Adleman. On the potential of molecular computing. *Science*, 268(5210):483–484, 1995.

[3] C. H. Bennet and R. Landauer. The fundamental physical limits of computation. *Scientific American*, 253:48, 1985.

[4] S. P. A. Fodor and et al. Light-directed, spatially addressable parallel chemical synthesis. *Science*, 251:767, 1991.

[5] David K. Gifford. On the path to computation with DNA. *Science*, 266:993–994, November 1994.

[6] Frank Guarnieri, Makiko Fliss, and Carter Bancroft. Making DNA add. *Science*, 273(5272):220–223, July 1996.

[7] M. K. Herrlein and et al. A covalent lock for self-assembled oligonucleotide conjugates. *Journal American Chemical Society*, 117:10151, 1995.

[8] Richard J. Lipton. DNA solution of hard computational problems. *Science*, 268:542–545, April 1995.

[9] Mitsunori Ogihara and Animesh Ray. Simulating boolean circuits on a DNA computer. In *1st Annual Conference on Computational Molecular Biology*, pages 326–331, 1997.

[10] John S. Oliver. Computation with DNA-matrix multiplication. In *2nd Annual Meeting on DNA Based Computers*, June 1996.

[11] A. C. Pease and et al. Light-generated oligonucleotide arrays for rapid dna sequence analysis. *Proc. National Academy of Science, USA*, 91:5022, 1994.

[12] Robert Pool. A boom in plans for DNA computing. *Science*, 268:498–499, April 1995.

[13] Sam Roweis, Erik Winfree, Richard Burgoyne, Nickolas V. Chelyapov, Myron F. Goodman, Paul W. K. Rothemund, and Leonard M. Adleman. A sticker based architecture for DNA computation. In *2nd Annual Meeting on DNA Based Computers*, May 1996.

[14] Harvey Rubin. Looking for the dna killer app. *Nature Structural Biology*, 3(8):656–658, August 1996.

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

[16] R. B. Wallace and et. al. Hybridization of synthetic oligodeoxyribonucleotides to ϕ-x 174 dna: The effect of single base pair mismatch. *Nucleic Acid Research*, 6:3543, 1979.

[17] Erik Winfree, Xiaoping Yang, and Nadrian C. Seeman. Universal computation via self-assembly of DNA: Some theory and experiments. In *2nd Annual Meeting on DNA Based Computers*, May 1996.