Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science 171 (2007) 209–222
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

A Model in κ for DNA Addition

Xian Xu,[1,2] Xiaoju Dong[3] and Yuxi Fu[3]

BASICS
Department of Computer Science and Technology
Shanghai Jiao Tong University
1954 Hua Shan Road(200030), Shanghai, P.R. China

Abstract

DNA computing is a hot research topic in recent years. Formalization and verification using theories(π-calculus, bioambients, κ-calculus and etc.) in Computer Science attract attention because it can help prove and predict to a certian degree various kinds of biological processes. Combining these two aspects, formal methods can be used to verify algorithms in DNA computing, including basic arithmetic operations if they are to be included in a DNA chip. In this paper, we first introduce a newly-designed algorithm for solving binary addition with DNA, which contributes to a unit in DNA computer processor, and then formalize the algorithm in κ-calculus(a formal method well suited for describing protein interactions) to show the correctness of it in a sense, and a sensible example is provided. Finally, some discussion on the described model is made, in addition to a few possible future improvement directions.

Keywords: DNA Computing, Addition, Modeling, κ calculus, biological computing

1 Introduction

Since the DNA's self-contained properties are utilized, by Adleman [1], to attack those hard problems in Computer Science, such as the Hamilton path problem, DNA computing has been increasingly regarded as a promising area for solving more difficult problems, experimenting the idea to construct a DNA computer(some work like [4] has been reported), which enjoys the appealing characteristics like vast parallelism.

To implement a DNA computer, researchers harness the basic properties and principles of DNA behavior. The infrastructure of a computer is to be designed and implemented, in which process the elementary step is the implementation of ALU(Arithmetic Logic Unit), including the preliminary mathematics, concerning

[1] The author is supported by The National Distinguished Young Scientist Fund of NNSFC (60225012), BDCC (03DZ14025), The National Nature Science Foundation of China (60473006), MSRA, and The BoShiDian Research Fund (20010248033).

[2] Email: xuxian@sjtu.edu.cn

[3] Email: xjdong,yxfu@sjtu.edu.cn

doi:10.1016/j.entcs.2007.05.018

the addition, subtraction, multiplication, division and etc. Hereafter we will introduce a new way to realize the addition by DNA.

Beneath the algorithm of DNA lies the mechanism of biomolecular computing, the DNA self-assembly. It serves as the underlying driver to achieve various DNA applications. Eric Winfree and his colleagues have conducted a bundle of research on this subject, and [17,18] are some of their results.

On the other hand, Computer Science contributes to the development in biomolecular experiments and analysis as well. While biology is an experiment-based discipline, characterized by observation, analysis, summary, conjecture and hypothesis, but it is mostly a work based on approximation and speculation, not completely on strict mathematics. Therefore, providing a robust method to describe and analyze various results of biological phenomena, such as signal transduction and gene regulation, gradually reveals its significance. Due to the intimate relation to Computer Science, biology, especially the molecular biology, makes use of a variety of formal methods to help model and verify the correctness of experiments and postulation, and this field has been receiving more and more notice. Process algebras and other theories alike, because of their intrinsic expressibility, are appropriate for modeling different kinds of biomolecular processes. The methods include π-calculus [15,16], stochastic π-calculus [9,11,13], bioambient [6,14], brane-calculi [5], P-system [10], κ calculus [7] and etc. They are capable of effectively describing a certain kind of biological processes, forming a network of methods for specific processes, from which researchers can choose according to their demand. Moreover, tools corresponding to the methods are partly available to execute the description and simulate the biological processes, for example the BioSPI [3], thus tracing the procedure and verifying the results.

In fact, the two aspects of the study between biomolecular disciplines and Computer Science are not clearly cut, blurred by mutual benefiting. For example, the advantage of high parallelism in DNA computing offers its efficiency in aid to solve hard problems such as NPC problems, and reversely, the theories in Computer Science, especially those mathematical ones based on computer automation can be used to boost the analysis and comprehension of a variety of biological processes, particularly on the molecular level, of which the major form is to model biological procedures with formal methods to verify and predict certain properties or events. These two aspects actually interact with each other in several ways, prompting both to develop further on the whole. Below we try to do something related to the scenario described above.

As a new DNA algorithm for addition was put forward by the BioX lab [12], we attempt to model the algorithm with the calculus κ by Vincent Danos [7]. As follows, we will first get the readers to be familiar with the two aspects, that is, the algorithm and the κ calculus. Then we put forth to the modeling, before some discussions on the description effect and the conclusion are given.

2 The κ Calculus

κ, or formal molecular biology [7], is a formal method designed to characterize the interaction among different kinds of proteins. Compared with other formal methods, such as π-calculus and bioambient, it features that behavior of proteins of distinctive kinds can be described in an interactive way, for instance, connection and disconnection on certain classes of domains on proteins, and the interaction network can be depicted in a structural and neat manner.

Below we give the syntax and semantics in a brief way to make the readers familiar with the method.

2.1 Syntax

The syntax of κ is given below.

$$S := 0 \mid A(\rho) \mid S, S \mid (x)(S)$$

The basic element of κ is solution, indicated by S, which can be seen as a system of proteins, including candidates for reactions. A solution can be one of the followings:

- 0 (empty solution). It means none.
- $A(\rho)$ (protein). It means a protein with the interface ρ. An interface contains the information of the domain sites of a protein, that is, the reaction capability of it.A protein may have a set of sites, each of which can have two kinds of states: *hidden*, *visible*. The former means it is kept from being seen, and the latter means the site can be reached. If a site is visible, it may be connected through certain channel, known as an edge connection, whose name is chosen from a name set $\mathcal{E}$ comprising $x, y, z \cdots$, then the site is said to be *bound*, and *free* otherwise.We use $A, B, C \cdots$ to denote proteins and $\rho, \sigma \cdots$ to denote interfaces. Moreover, the sites of a protein are ordered by natural numbers($\mathbb{N}$), and the total number of sites of a protein A is defined as $\mathfrak{s}(A)$.
  In fact an interface is a (partial) mapping from $\mathbb{N}$ to $\mathcal{E} \cup \{h, v\}$, with h for hidden and v for visible. And for convenience, for example, an interface ρ of $A(\mathfrak{s}(A) = 3)$ defined as {(1, h), (2, x), (3, v)}, can be denoted by $\rho = \overline{1} + 2^x + 3$, where the correspondence of symbols is easy to find.
- S, S (group). It means composition of solutions. Perhaps with proteins connected by a number of edges.
- $(x)(S)$ (new). It means the creation of an edge x between two proteins. And x is said to be bound in S, by which the free names and bound names of a solution can be precisely defined, similar to π-calculus. We omitted it.
  For instance, a solution may take the form(A and B are connected by edge x created for interaction between them):

$$S \overset{def}{=} (x)(A(1^x + \overline{2} + 3), C(\overline{1} + 2 + 3^x)), B(1 + 2^y)$$

2.2 Semantics

The semantics of κ is defined through the evolution relation, called growth relation in [7], on interfaces of proteins. It is shown below(Table 1). Note $\tilde{x}$ denotes a sequence of names, and it should be designed to avoid capturing of bound names. In the relation in Table 1, an interface can be switched from hidden (state) to visible (state) and vice versa, can be assigned an edge name for connection, and can form a larger growth from smaller ones. Then Table 2 defines the growth relation on

CREATE:	$\dfrac{x \in \tilde{x}}{\tilde{x} \vdash \imath \leq \imath^x}$		
HV-SWITCH:	$\dfrac{}{\tilde{x} \vdash \bar{\imath} \leq \imath}$	VH-SWITCH:	$\dfrac{}{\tilde{x} \vdash \imath \leq \bar{\imath}}$
REFLEX:	$\dfrac{\tilde{x} \cap fn(\rho) = \emptyset}{\tilde{x} \vdash \rho \leq \rho}$	SUM:	$\dfrac{\tilde{x} \vdash \rho \leq \sigma \quad \tilde{x} \vdash \rho' \leq \sigma'}{\tilde{x} \vdash \rho + \rho' \leq \sigma + \sigma'}$

Table 1
Interfaces evolution

solutions. In the relation in Table 2, a solution can evolve in accordance to that of the interfaces of the proteins in it, concerning the composition(GROUP) and the creation(SYNTH) reaction.

NIL:	$\dfrac{}{\tilde{x} \vdash 0 \leq 0}$
GROUP:	$\dfrac{S \leq T \quad \tilde{x} \vdash \rho \leq \sigma \quad \sigma \text{ is a partial interface on } A}{\tilde{x} \vdash S, A(\rho) \leq T, A(\sigma)}$
SYNTH:	$\dfrac{S \leq T \quad fn(\sigma) \subseteq \tilde{x} \quad \sigma \text{ is an (impartial) interface on } A}{\tilde{x} \vdash S \leq T, A(\rho)}$

Table 2
Solutions evolution

On the basis of the two tables(1,2), a reaction can be defined. Roughly speaking,

$$if\ \tilde{x} \vdash S \leq T,\ then\ we\ say\ S \longrightarrow (\tilde{x})T\ is\ a\ monotonic\ reaction.$$

By monotonic, we mean the reaction does not reduce proteins. Also in [7], several more conditions are required to ensure well behavior. The readers are referred to [7] for them. Furthermore, as in actual biological systems, interfaces may need renaming and extensions, the reaction defined above needs to be extended, but this does not seem complex and have little influence on our work here, so we omitted it.

Finally, we give an example of reaction (rules). Take the following one as an example, in which the two visible sites $2, 3$ on A, B, respectively, can react to form a connection (an edge) named x. Actually, this reaction can be obtained from the relation given in Table 1 and 2, therefore other kinds of reactions can be similarly gained.

$$A(1+2), B(\bar{1}+2+3) \longrightarrow (x)(A(1+2^x), B(\bar{1}+2+3^x))$$

3 Yet Another DNA Addition Algorithm

Even before the use of DNA's built-in computing capability to solve scientific problems, scientists had tried to compute with DNA, including the basic arithmetic operations, such as addition [8] and subtraction. Recently the researchers in BioX lab designed a new algorithm of binary number addition with DNA hoping to apply it in a larger project to construct a real DNA computer. Compared to the previous algorithms, it is described to be more simple, efficient. We will introduce it below before our model comes.

3.1 The Algorithm

Albeit the basic thought is analogous to some other algorithms, the algorithm here is aimed at the efficiency in building a DNA-driving device, so it has to save more time than space. Generally speaking, it is using brute force, that is, prepares all the possible results during the bit-computing and combines them to get a final result. From theoretical view, the time efficiency is satisfactory, linear in bio-steps[4]. And the number of DNA oligonucleotides needed is also linear in the number of the maximum bits of the two numbers.

To describe the algorithm, suppose the two binary numbers to be added is of length(in bits) n, and denoted by $A = A_{n-1}A_{n-2}\cdots A_2A_1A_0$ and $B = B_{n-1}B_{n-2}\cdots B_2B_1B_0$. For each bit, we decompose the addition process into two phases. Consider the $i'th$ bit$(0 \leqslant i \leqslant n-1)$,

- Add A_i to the carry bit from the last lower bit addition, denoted by C_i, and get the partial bit result H_i;
- Add H_i to B_i, and get the sum of the $i'th$ bit and its carry bit to the next higher bit.

Following this regulation, we can in advance write out all the possible bit addition on the $i'th$ bit, with some special care to the first and last bit. They are given in Table3.

In (i) and (ii) in Table 3, $H_i = A_i + C_i$. S_{i-1} is of no effect, just to record the sum of the last lower bit to distinguish the same carry bit, what is more, to help organize the complementing scene according to the result(C_{i+1}, S_i) in the (iii) and (iv) in Table 3, since the latter two must append the addition result in its tail(imagine a DNA string). So in fact, H_i is merely decided by A_i and C_i. By the way, we use 2 for convenience, in fact it can be any symbol other than those having been used within the tables.

In (iii) and (iv) in Table 3, H_i is added to B_i, and we get the addition result on the $i'th$ bit and the carry bit C_{i+1} to the next higher bit. Note S_i must be held, leading to the design of S_{i-1} in (i) and (ii).

And also note that when $i = 0$, $H_0 = A_0 + C_0$, and since $C_0 = 0$ obviously, we just need to keep A_0 for H_0, which will be reflected in the experiment. Moreover,

[4] for example, a complementing and ligating step

C_i	S_{i-1}	H_i
0	0	0
0	1	0
1	0	1
1	1	1

(i): $A_i = 0$

C_i	S_{i-1}	H_i
0	0	1
0	1	1
1	0	2
1	1	2

(ii): $A_i = 1$

H_i	C_{i+1}	S_i
0	0	0
1	0	1
2	1	0

(iii): $B_i = 0$

H_i	C_{i+1}	S_i
0	0	1
1	1	0
2	1	1

(iv): $B_i = 1$

Table 3
DNA addition on $i'th$ bit: $H_i = A_i + C_i$ and C_{i+1}, S_i is from $H_i + B_i$

when $i = n-1$, H_{n-1} and B_{n-1} are added, resulting in S_{n-1} and a carry bit, which we denote by C_n. Hence the whole addition result is $C_nS_{n-1}S_{n-2}\cdots S_2S_1S_0$, with the leftmost 0 removed.

3.2 Experimentation

From Table 3, if we regard each row of the tables as an ssDNA(single strand DNA) molecule with two corresponding sticky ends(separated by the double vertical line in each table in Table 3), one can actually imply that the addition algorithm is a continuous procedure of Watson-Crick complementing and ligating, starting from the H_0 strand, ending in the C_n, S_{n-1} strand. This process is easy to understand and implement because the algorithm has screened the hard part in its encoding, that is, using the brute-force approach.

The encoding is too large to present here, please refer to [12] for the whole encoding table. Two things need to be noticed. One is that because $C_0 = 0$ H_0 can not be 2, then the bottom row of the tables for B_0 can be discarded. The other is that in order to extract the result dsDNA(double strand DNA) comprising the whole computing procedure(also the addition result), PCR(Polymerase Chain Reaction) is used, so primers have to be added to the strand in some way. One easy way is to attach a left primer to H_0 and a right primer to S_{n-1}.

The computing begins with putting all the encoding ssDNA of each bit of the two binary numbers into reaction environment with ligase, the computing(self-assembly) then starts automatically. When it is time, use PCR to extract the result dsDNA, and try to sequence each $S_i(0 \leqslant i \leqslant n-1)$ and C_n.

Actually the experiment is much more complicated in operating, because it involves in vitro environment a number of techniques, each of which can contribute to the error rate, for example during the connecting and read-out, and this is discussed more in [12]. This renders the algorithm hard to scale up. And another challenge is that the computation procedure is badly continual, that is, it is nearly impossible to add three numbers by doing binary addition twice, and the experimenting complexity makes the first computing leave out a result incompatible for another computing. These disadvantages, among others, make the DNA addition algorithm described here a bit far from application in DNA computers(in fact other algorithm faces similar challenges).

More experiment details, including error handling in 'wet' environment, are beyond the scope of this paper, we below will focus on the modeling and verification of the algorithm mainly in theoretic respect. A brief comparison with other algorithms can be found in [12].

4 The κ Model for DNA Addition

Now we will get down to modeling the DNA addition algorithm with κ. First the overall thoughts are stated, then the elements of the model, that is, the typical proteins and reaction rules, are defined, and finally the model is described.

4.1 Basic Ideas

Aware of the reaction procedure in the addition algorithm, we can treat the process as somewhat a chain reaction of DNA complementing and ligating. Once the two binary numbers are given, the encodings are decided, and each row in Table 3 is represented as an ssDNA.

To be more concrete, the computing loops with the connection between the H_i in (i) or (ii) and H_i in (iii) or (iv), and the similar connection on $C_{i+1}, S_i(C_i, S_{i-1})$, alternatively. Some internal mechanism in this is that not until the last connection is finished, can the next connection be made, and the connection's choice is unique.

Thus, we can apply κ to describe the process. The rows (or ssDNA's) in Table 3 can be described as κ proteins(a little uneasy, but just think of it as an abstract element), each of which has two sites, one corresponding to H_i and the other to $C_{i+1}, S_i(C_i, S_{i-1})$, that is, the two parts separated by double vertical lines in each table in Table 3, and we refer to the two sites as left one and right one. Moreover, for each protein, the left site is initially visible but the right site is initially hidden. Then the computing proceeds with multiple steps, each of which operates like this: the left site of a protein is connected to the last protein's right site on some edge, thus enable(or activate) the right site(switching it to visible), then the similar process continues. It is easy to figure out that H_0 in (i) or (ii) in Table 3 should be set visible initially. And the change of state of the two primers can be used as an indicator to the end of the computing, when the addition result can be extracted from the connecting product of proteins.

4.2 The Definition

Here we give the formal definitions of proteins and reaction rules for modeling the DNA addition algorithm.

4.2.1 Proteins

The proteins here mean the abstract elements in κ, not the concrete ones in biological sense. As mentioned, we abstract each row in the tables in Table 3 as a protein, which has two sites, one for each of the ssDNA sticky ends.

For consistency, we define the proteins from the information in the tables in Table 3. For (i) in Table 3, define A_i^{0j} for row $j (j = 0, 1, 2, 3)$, in which 0 means that it is the case $A_i = 0 (i = 0, 1, 2, \cdots, n - 1)$. Similarly, we define A_i^{1j} for row j, B_i^{0j} for row j and B_i^{1j} for row j in (ii), (iii) and (iv) in Table 3 respectively.

For each protein, its interface is defined on the two 'ends', and we name them on the basis of the value they take. For example, protein A_i^{00} has the interface $(c_0 s_0 + \overline{h_0})$, which means that in this case the carry bit from last lower bit is 0, the sum of the last lower bit is 0, and the partial sum of this bit is 0. The site h_0 is hidden initially, as mentioned in the modeling idea above. Similarly, the rest proteins' interface can be defined.

We show the definition of proteins in Table 4.

$A_i^{00}(c_0 s_0 + \overline{h_0})$
$A_i^{01}(c_0 s_1 + \overline{h_0})$
$A_i^{02}(c_1 s_0 + \overline{h_1})$
$A_i^{03}(c_1 s_1 + \overline{h_1})$

(i): A_i^{0j}

$A_i^{10}(c_0 s_0 + \overline{h_1})$
$A_i^{11}(c_0 s_1 + \overline{h_1})$
$A_i^{12}(c_1 s_0 + \overline{h_2})$
$A_i^{13}(c_1 s_1 + \overline{h_2})$

(ii): A_i^{1j}

$B_i^{00}(h_0 + \overline{c_0 s_0})$
$B_i^{01}(h_1 + \overline{c_0 s_1})$
$B_i^{02}(h_2 + \overline{c_1 s_0})$

(iii): B_i^{0j}

$B_i^{10}(h_0 + \overline{c_0 s_1})$
$B_i^{11}(h_1 + \overline{c_1 s_0})$
$B_i^{12}(h_2 + \overline{c_1 s_1})$

(iv): B_i^{1j}

Table 4
Proteins definition for each row in Table 3

Now we consider the first and last bit. We would cancel the definition for A_0^{0j} and $A_0^{1j} (j = 0, 1, 2, 3)$ in Table 4, of course keep the rest intact. And we define them as $A_0^0(\overline{PrimerL} + h_0)$ and $A_0^1(\overline{PrimerL} + h_1)$, in which $PrimerL$ denotes the left primer to indicate the start of the computing and it is hidden until the connecting begins. To get a smooth ending, we define four more proteins $BEnd_n^{00}(c_0s_0 + \overline{PrimerR}), BEnd_n^{01}(c_0s_1 + \overline{PrimerR}), BEnd_n^{10}(c_1s_0 + \overline{PrimerR}), BEnd_n^{11}(c_1s_1 + \overline{PrimerR})$, in which, for example, $BEnd_n^{01}(c_0s_1 + \overline{PrimerR})$ means that the sum on the last bit is 1 and it carries none to the next higher bit, and $PrimerR$ is hidden until the computing ends, to control the cease of the whole computing.

Furthermore, because of the simplification on bit A_0, something alike can also be done to bit B_0. As the last row in (iii) and (iv) in Table 3 is impossible to arrive at for B_0, we can eliminate the corresponding proteins in our model, that is, the proteins B_0^{02} and B_0^{12}.

4.2.2 Rules

We divide the rules into several groups, each of which will be explained below.

- Activation. Consider the two sites of each protein. The connecting on the left site will activate the right one, switching it to be visible for future connection. We will give some examples. For example, the two rules below belong to this group. Note x is an edge, so is it in the other examples.

$$A_i^{00}(c_0s_0^x + \overline{h_0}) \longrightarrow A_i^{00}(c_0s_0^x + h_0)$$
$$B_i^{00}(h_0^x + \overline{c_0s_0}) \longrightarrow B_i^{00}(h_0^x + c_0s_0)$$

- Connection. After the right site of a protein is activated, it can react with a specific protein on its left site to form another connection. This proceeds with the steps of the addition. For example, the two rules below belong to this group.

$$A_i^{00}(c_0s_0^x + h_0), B_i^{00}(h_0 + \overline{c_0s_0}) \longrightarrow (y)(A_i^{00}(c_0s_0^x + h_0^y), B_i^{00}(h_0^y + \overline{c_0s_0}))$$
$$B_{i-1}^{00}(h_0^x + c_0s_0), A_i^{00}(c_0s_0 + \overline{h_0}) \longrightarrow (y)(B_{i-1}^{00}(h_0^x + c_0s_0^y), A_i^{00}(c_0s_0^y + \overline{h_0}))$$

- Beginning and ending. The beginning and ending of the addition need special handling. As mentioned, $PrimerL$ and $PrimerR$ are used to cope with these two conditions. For example, two of the rules in this group are listed below.

$$A_0^0(\overline{PrimerL} + h_0), B_0^{00}(h_0 + \overline{c_0s_0}) \longrightarrow$$
$$(y)(A_0^0(\overline{PrimerL} + h_0^y), B_0^{00}(h_0^y + \overline{c_0s_0}))$$
$$BEnd_n^{00}(c_0s_0 + \overline{PrimerR}), B_{n-1}^{00}(h_0^x + c_0s_0) \longrightarrow$$
$$(y)(BEnd_n^{00}(c_0s_0^y + \overline{PrimerR}), B_{n-1}^{00}(h_0^x + c_0s_0^y))$$

4.3 The Model

Since the complete model is fairly large, we only give above some typical samples from it for explanation. The entire model is given in the Appendix(available on the website [2]) of this paper.

Here we continue to describe the simulation process of the model and then give an example of addition computing. For convenience, we assume a simulation system is provided. The simulation can be described in several steps. Suppose two binary numbers to be added are $A = A_{n-1}A_{n-2}\cdots A_2A_1A_0, B = B_{n-1}B_{n-2}\cdots B_2B_1B_0$.

Then

(i) Put in the simulation system the encodings of A and B, that is, all the related proteins. For example, if $A_i = 1(i = 0, 1 \cdots, n-2, n-1)$, then put the proteins $A_i^{1j}(j = 0, 1, 2, 3)$, so it is with $B_i(i = 0, 1 \cdots, n-2, n-1)$. At the same time, put in A_0^0 if $A_0 = 0$ or A_0^1 if $A_0 = 1$, and $BEnd_n^{pq}(p, q = 0, 1)$ as well.

(ii) Set the system rules to those given above.

(iii) Run the system.

(iv) When the site $PrimerR$ on one of the proteins $BEnd_n^{pq}(p, q = 0, 1)$ is turned to be visible, the computing ends. Now from $PrimerR$ to $PrimerL$, read out all the different $c_i s_j$ sites on the sequence of proteins on the path, and string their subscripts j to obtain the result of the addition.

It can be readily verified that the simulation procedure is correct and produce the expected result with respect to the DNA addition algorithm described in [12]. To some extent, this can be shown in the computing example below.

For now it is not known that some effective tools exist, but it is said to be designed and implemented by some researchers, however, it seems still a long time before proper simulation tools are available.

4.3.1 A Sample Addition Computing

For an intuitive view, we give a computing of DNA addition as an example. Take two binary numbers of length $4(n = 4)$. For instance, $A = 1101, B = 0110$, in which the leftmost 0 of B is added to get consistency. Now we can obtain the encodings of the bits in the two numbers. For A, the corresponding proteins in the solution(the system) are shown in Table 5. For B, the corresponding proteins in the solution are shown in Table 6.

$$A_0^1(\overline{PrimerL} + h_1),$$

$$A_1^{00}(c_0s_0 + \overline{h_0}), A_1^{01}(c_0s_1 + \overline{h_0}), A_1^{02}(c_1s_0 + \overline{h_1}), A_1^{03}(c_1s_1 + \overline{h_1}),$$

$$A_2^{10}(c_0s_0 + \overline{h_1}), A_2^{11}(c_0s_1 + \overline{h_1}), A_2^{12}(c_1s_0 + \overline{h_2}), A_2^{13}(c_1s_1 + \overline{h_2},$$

$$A_3^{10}(c_0s_0 + \overline{h_1}), A_3^{11}(c_0s_1 + \overline{h_1}), A_3^{12}(c_1s_0 + \overline{h_2}), A_3^{13}(c_1s_1 + \overline{h_2}),$$

Table 5
Proteins for A

The run of the simulation computing is shown as follows(Figure 1). To be concise, we simply show the main route of reaction. The dots mean other proteins that do not take part in the current step. Note $y_i(i = 1, 2, 3, \cdots)$ are edge names, and the permutation of proteins does not matter.

Now it is time to sequence the proteins representing the result. We can find these proteins whose sites indicate the result, as shown below.

$$c_{\underline{1}}s_{\underline{0}}^{y8} \quad c_1s_{\underline{0}}^{y6} \quad c_0s_{\underline{1}}^{y4} \quad c_0s_{\underline{1}}^{y2}$$

$B_0^{00}(h_0+\overline{c_0s_0}), B_0^{01}(h_1+\overline{c_0s_1}), B_0^{02}(h_2+\overline{c_1s_0}),$
$B_1^{10}(h_0+\overline{c_0s_1}), B_1^{11}(h_1+\overline{c_1s_0}), B_1^{12}(h_2+\overline{c_1s_1}),$
$B_2^{10}(h_0+\overline{c_0s_1}), B_2^{11}(h_1+\overline{c_1s_0}), B_2^{12}(h_2+\overline{c_1s_1}),$
$B_3^{00}(h_0+\overline{c_0s_0}), B_3^{01}(h_1+\overline{c_0s_1}), B_3^{02}(h_2+\overline{c_1s_0}),$
$BEnd_4^{00}(c_0s_0+\overline{PrimerR}), BEnd_4^{01}(c_0s_1+\overline{PrimerR}),$
$BEnd_4^{10}(c_1s_0+\overline{PrimerR}), BEnd_4^{11}(c_1s_1+\overline{PrimerR})$

Table 6
Proteins for B

The sum of the two binary numbers can be obtained by stringing the underlined bits from left to right, that is the correct answer 10011. Thus, this example successfully shows that the model we designed is believed to be sound.

5 Discussion

Here we give some thoughts on the work described in this paper.

kappa advantage At first, we did not choose κ as the description method, instead we considered Bioambients for example, because it is also capable of describing molecules interaction. But finally κ is selected for its high effectiveness in describing the on-domain interaction among (protein) molecules. κ's abilities to model the process of connection, disconnection and domain state switch go well with the self-assembly characteristics of the DNA addition algorithm(in [12] and others), and the basic reaction rules are sufficient to deal with the reactions, mostly connection(complementing and ligating in biology), design the result extracting mechanism for easy ending phase, and support the simulation process by potential tools.

naming The names of proteins definition, including their sites, are difficult to design because of their large quantity and complexity in the names of ssDNA, such as C_i and H_i. We try to obey the convention of Table 3, but it is proved to be disastrous because the complicated collection of subscripts and superscripts become a mess. What is worse, the addition result is difficult to extricate from the ending solution(proterins string). Then the design of names of sites presented in this paper was worked out. It not only records the row number in the tables in Table 3 to clarify its temporary status during computing, but also retains all the necessary partial results(C_i, S_i, H_i) in the computing process, rendering it easier to extract the addition result.

brute force approach One disadvantage of the algorithm, successively the description model, is that it is much too large in scale, if the model size is measured by the number of proteins and rules in it. Provided that the addition algorithm is not modified to discard the brute force approach, it seems that the model is not likely to be made smaller greatly. So we think of the model here as a compact

$$\begin{aligned}
&A_0^1(\overline{PrimerL} + h_1), B_0^{01}(h_1 + \overline{c_0 s_1}), \cdots\cdots \\
&\to (y_1)(A_0^1(\overline{PrimerL} + h_1^{y_1}), B_0^{01}(h_1^{y_1} + \overline{c_0 s_1}), \cdots\cdots) \\
&\to (y_1)(A_0^1(PrimerL + h_1^{y_1}), B_0^{01}(h_1^{y_1} + \overline{c_0 s_1}), \cdots\cdots) \\
&\to (y_1)(B_0^{01}(h_1^{y_1} + c_0 s_1), A_1^{01}(c_0 s_1 + \overline{h_0}), \cdots\cdots) \\
&\to (y_1 y_2)(B_0^{01}(h_1^{y_1} + c_0 s_1^{y_2}), A_1^{01}(c_0 s_1^{y_2} + \overline{h_0}), \cdots\cdots) \\
&\to (y_1 y_2)(A_1^{01}(c_0 s_1^{y_2} + h_0), B_1^{10}(h_0 + \overline{c_0 s_1}), \cdots\cdots) \\
&\to (y_1 y_2 y_3)(A_1^{01}(\mathbf{c_0 s_1^{y_2}} + h_0^{y_3}), B_1^{10}(h_0^{y_3} + \overline{c_0 s_1}), \cdots\cdots) \\
&\to (y_1 y_2 y_3)(B_1^{10}(h_0^{y_3} + c_0 s_1), A_2^{11}(c_0 s_1 + \overline{h_1}), \cdots\cdots) \\
&\to (y_1 y_2 y_3 y_4)(B_1^{10}(h_0^{y_3} + c_0 s_1^{y_4}), A_2^{11}(c_0 s_1^{y4} + \overline{h_1}), \cdots\cdots) \\
&\to (y_1 y_2 y_3 y_4)(A_2^{11}(c_0 s_1^{y4} + h_1), B_2^{11}(h_1 + \overline{c_1 s_0}), \cdots\cdots) \\
&\to (y_1 y_2 y_3 y_4 y_5)(A_2^{11}(\mathbf{c_0 s_1^{y4}} + h_1^{y_5}), B_2^{11}(h_1^{y_5} + \overline{c_1 s_0}), \cdots\cdots) \\
&\to (y_1 y_2 y_3 y_4 y_5)(B_2^{11}(h_1^{y_5} + c_1 s_0), A_3^{12}(c_1 s_0 + \overline{h_2}), \cdots\cdots) \\
&\to (y_1 y_2 y_3 y_4 y_5 y_6)(B_2^{11}(h_1^{y_5} + c_1 s_0^{y_6}), A_3^{12}(c_1 s_0^{y_6} + \overline{h_2}), \cdots\cdots) \\
&\to (y_1 y_2 y_3 y_4 y_5 y_6)(A_3^{12}(c_1 s_0^{y_6} + h_2), B_3^{02}(h_2 + \overline{c_1 s_0}), \cdots\cdots) \\
&\to (y_1 y_2 y_3 y_4 y_5 y_6 y_7)(A_3^{12}(\mathbf{c_1 s_0^{y_6}} + h_2^{y_7}), B_3^{02}(h_2^{y_7} + \overline{c_1 s_0}), \cdots\cdots) \\
&\to (y_1 y_2 y_3 y_4 y_5 y_6 y_7)(B_3^{02}(h_2^{y_7} + c_1 s_0), BEnd_4^{10}(c_1 s_0 + \overline{PrimerR}), \cdots\cdots) \\
&\to (y_1 y_2 y_3 y_4 y_5 y_6 y_7 y_8)(B_3^{02}(h_2^{y_7} + c_1 s_0^{y_8}), BEnd_4^{10}(c_1 s_0^{y_8} + \overline{PrimerR}), \cdots\cdots) \\
&\to (y_1 y_2 y_3 y_4 y_5 y_6 y_7 y_8)(B_3^{02}(h_2^{y_7} + c_1 s_0^{y_8}), BEnd_4^{10}(\mathbf{c_1 s_0^{y_8}} + PrimerR), \cdots\cdots)
\end{aligned}$$

Fig. 1. An example addition computation

one in κ. As for the addition algorithm, the nature of it is brute force[5], which decides the complexity of the implementation, both in biological experiment and formal modeling, however the algorithm is indeed efficient in executing and can be finished in linear time to the bio-steps. This trading space for time methodology is appreciated in biological computing. On the other hand, if we plan to reduce the space complexity of the model, we probably have to redesign the addition algorithm and sacrifice some time in return for space savings. This can be an improvement direction, among others, that is, to find a possible equilibrium between time and size in model, meanwhile make the best of the vast parallelism

[5] a little different from traditional meaning, here we mean by 'brute force' the preparation of all the possible addition results on a bit of the addends

of DNA computing.

Another improvement possibility is that the formal method for modeling can be made easier to use and stronger in describing some kinds of molecules reactions. Although we use κ here, it does not mean that other methods can not be chosen. For instance, Bioambients, P-syetem and even π-calculus might be appropriate in describing certain aspects of the algorithm, and this is sensible. Thus whether we can make use of the merits of distinctive methods and design a new method suitable for modeling DNA computing may become an interesting question worth considering. Still improvements in κ itself are possible too, like designing higher-level object abstraction in solution definition, for example a protein complex as a whole object for interaction, to simplify the description, but this may need adjustment in the syntax and semantics.

As far as we are concerned, in comparison to other methods like π-calculus, κ is short of useful compilers and executors, which limits the application of it to some extent. Hence it become increasingly urgent to develop an effective simulation tool for κ. With such a tool, the first-step verification of the model can be conducted and the tool can be used to trace the proteins interaction and find possible clues for the proof of some mechanism or make possible predictions for some potential functions.

6 Conclusion

We presented a new algorithm for solving the arithmetic addition problem with DNA, as well as a formal method, κ, for modeling the protocol. Based on this, we defined a model in κ for the DNA addition algorithm, from proteins to reaction rules, in several categories and with detailed explanations of principles, in addition to some proper examples. The complete model can be found in the Appendix and we reflect some correctness of the model with an example addition in the model's dynamics. In the end, we discuss in depth some merits and demerits of the DNA addition algorithm and the description model, and meanwhile make some suggestions on future improvements for both of them. Besides, the demand for effective simulation tools is stated to emphasize its importance in the application of κ in describing biological computing.

Acknowledgement

We are much grateful to Dan Li and Huan Long for many helpful discussions on the topic of this paper. Also we appreciate the suggestion from the anonymous referees very much.

References

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

[2] Basics Studies in Computer Science: http://basics.sjtu.edu.cn.

[3] The BioSPI Project Homepage: http://www.wisdom.weizmann.ac.il/∼biospi/.

[4] Benenson, Y., B. Gil, U. Ben-Dor, R. Adar and E. Shapiro. *An Autonomous Molecular Computer for Logical Control of Gene Expression.* Nature, **429**:423-429, 2004. doi:10.1038/nature02551.

[5] Cardelli, L.. *Brane Calculi.* In Computational Methods in Systems Biology 2004 (V. Danos, V. Schachter, eds.), LNBI **3082**:257-278, 2005, Springer-Verlag(Berlin).

[6] Cardelli, L., and A. D. Gordon. *Mobile Ambients.* Theoretical Computer Science, Special Issue on Coordination(D. Le Métayer Editor), Vol **240**(1):177-213, June 2000.

[7] Danos, V., and C. Laneve. *Formal Molecular Biology.* In Theoretical Computer Science, Special issue: Computational systems biology, **325**(1):69-110, 2004.

[8] Guarnieri, F., M. Fliss and C. Bancroft. *Making DNA Add.* Science, **273**(5272):220-223, 1996.

[9] Kuttler, C., J. Niehren and R. Blossey. *Gene Regulation in the Pi Calculus: Simulating Cooperativity at the Lambda Switch.* Transactions on Computational Systems Biology, Special Issue on BioConcur 2004, and Lecture Notes in Computer Science, Lecture Notes in Bioinformatics, Springer Verlag, 2005.

[10] Păun, Gh.. *Computing with Membranes.* Journal of Computer and System Sciences, **61**(1):108-43 (2000)(and Turku Center for Computer Science-TUCS Report 208, November 1998, http://www.tucs.fi).

[11] Priami, C., A. Regev, W. Silverman and E. Shapiro. *Application of a Stochastic Name Passing Calculus to Representation and Simulation of Molecular Processes.* Information Processing Letters **80**:25-31, 2001.

[12] Qian, L., J. Zhao and et al. *DNA Add Using Linear Self-Assembly.* BioX Lab. Submitted. 2005.

[13] Regev, A.. *Representation and Simulation of Molecular Pathways in the Stochastic π-Calculus.* Presented at the 2nd Workshop on Computation of Biochemical Pathways and Genetic Networks, 2001.

[14] Regev, A., E.M. Panina, W. Silverman, L. Cardelli and E. Shapiro. *BioAmbients-An Abstraction for Biological Compartments.* Theoretical Computer Science, **325**:141-167, 2004.

[15] Regev, A., W. Silverman and E. Shapiro. *Representing Biomolecular Processes with Computer Process Algebra: π-Calculus Programs of Signal Transduction pathways.* American Association for Artificial Intelligence (http://www.aaai.org), 2000.

[16] Regev, A., W. Silverman and E. Shapiro. *Representation and Simulation of Biochemical Processes Using the Pi-Calculus Process Algebra.* Proceedings of the Pacific Symposium of Bio-computing 2001(PSB2001), **6**:459-470, 2001.

[17] Winfree, E.. *DNA Computing by Self-Assembly.* NAE's The Bridge, **33**(4):31-38, 2003.

[18] Winfree, E.. "Algorithmic Self-Assembly of DNA". PhD Thesis. Caltech, June 1998.