Acta Cybernetica **vol** (year) pp–pp.

On Implementing Relational Databases on DNA Strands

István Katsányi*

Abstract

This work describes the theoretical bases of the implementation of relational databases in test tubes, using an abstract model of molecular computing. It specifies the representation of relations and the execution program of the relational algebra (RA) operations. We investigate the possibilities of practical usage of the proposed model as well as the bounds of it.

Key words: Molecular computing, theory of computing, relational database.

1 Introduction

In the last decade molecular biology has become the fastest growing discipline in the world. Some of the results are widely known, let us only mention the major breakthroughs in the Human Genome Project and nanotechnology. The progress made possible the birth of a new branch of science, that is called molecular computing (or DNA computing). Leonard M. Adleman published a paper [1] in 1994, which later become the foundation–stone of this new subject. In his article Adleman demonstrates how can a classic NP–complete problem: the problem of searching for a Hamiltonian path in a directed graph can be solved in polynomial time using the techniques of molecular biology and DNA strands. He outlines the great opportunity laying in the large computing power and the extremely compact data storage. In a test tube there can be performed as much as 10^{16} operations in a second. That is much more than current supercomputers can execute. In a litre of water DNA strands can encode 10^8 terabytes, and we can perform associative searches on the data in constant time.

In the past years many papers dealing with the computing power of DNA were published. However, only a few article studied the possibility of data storage and processing (see e.g. [2], [3]). Recently two papers ([4], [5]) described methods that yielded in an operation that closely resembles to the join operation of relational algebra. In spite of this no one has extensively studied the potentialities of the usage of molecular computing in the field of RA. By this work we would like to show the (at least theoretical) possibility of the use of the results of molecular computing in this area. In the *Preliminaries* section we introduce John Reif's RDNA model of biomolecular computing. We define

*Eötvös Loránd University, Department of General Computer Science, 1117 Budapest, Pázmány Péter sétány 1/C, e-mail: `kacsa@ludens.elte.hu`

a possible representation of relations using that model in the *Representation* section, and show how to implement the RA operations in this model in the *Operations* section. We close the paper by conclusions and references.

2 Preliminaries

As common in formal language theory, we denote the free monoid generated by a finite set X by X^*. We call X as *alphabet*, elements of X as *letters*, and elements of X^* as *words*. The symbol ϵ means the empty word. The length of a word $u \in X^*$ will be denoted by $|u|$. The cardinality of a set S will also be denoted by $|S|$.

We describe briefly the *RDNA* model introduced by J. H. Reif in [4]. For motivations, connections to molecular biology and complete definitions please refer to the original article. The operations of the model are abstractions of the well understood recombinant DNA operations and basic molecular biology operations. The structural properties of DNA are represented in a structure called *complex*.

We use an alphabet consisting $n \geq 1$ pairs of letters that said to be complementary: $\Sigma = \{a_1, a_2, \ldots a_n, a'_1, a'_2, \ldots a'_n\}$, where a_i and a'_i form a complementary pair for all $i \in [1, n]$. By a *linear string* we mean a word from Σ^*, and by a *loop string* we mean any possible circular rotation of a word from Σ^*. The set of circular rotations of word $u \in \Sigma^*$ is $\{u_2 u_1 \in \Sigma^* \mid u_1 u_2 = u\}$. A loop string can be represented by any particular instance of the rotated words. For a linear string u define $G(u)$ as a directed graph of $|u|$ vertices and $|u| - 1$ edges, such that the graph consists of one (nonrepeating) directed path, where the consecutive edges are labelled by the consecutive letters of u, from first to last. For a loop string u define $G(u)$ as a directed graph of $|u|$ vertices and $|u|$ edges, such that the graph consists of one directed loop, where the consecutive edges are labelled by the consecutive letters of u. For a set S of linear or loop strings over Σ define $G(S)$ as the union of the disjoint directed graphs $G(u)$ for each $u \in S$. Hence $G(S)$ consists of $|S|$ disjoint directed paths or loops.

Define a *labelled pairing* of the edges of $G(S)$ to be a set μ of unordered pairs of distinct directed edges of $G(S)$ such that (1) μ pairs the starting and ending vertices of the edges as well as the edges itself, (2) no edge appears more than once in μ, and (3) each pair of edges in μ have complementary labels and point to the opposite direction. We define a *complex over* S to be the pair (S, μ), where μ is a labelled pairing of $G(S)$.

The complex (S, μ) has a naturally defined graph $G(S, \mu)$ derived from the graph $G(S)$ by merging together the vertices i, i' as well as the vertices j, j' for all labelled pairs $((i, j), (j', i'))$ in μ, so the resulting graph has edges in both directions between the two merged nodes. Note that three nodes may be merged into one, for example j and j' would be merged with j'' if the pair $((j, k), (k', j))$ is in μ in addition to the pair $((i, j), (j', i'))$.

The complex (S, μ) is a *linear complex* if $\mu = \emptyset$ and S is a set of linear strings. The complex $(S, \emptyset)$ is a *single linear complex* if S contains one linear string only, so the graph $G(S)$ is a single directed path.

A complex may be used to model both the information content and also the three–dimensional structure of single– or double–stranded DNA, including hybridization and secondary structure. The use of complexes allows modelling the effect of various recombinant DNA operations, and thus providing rigorous definitions of recombinant DNA operations.

Operations of the RDNA model

We use a slightly different notation for the operations than Reif uses. We also extend the list of operations by the operations *Prepare*, *Assign* and *Amplify*. By a test tube we mean a multiset of *connected* complexes, where a complex (S, μ) is said to be connected, if its graph $G(S, \mu)$ is connected. We call two sets of complexes equivalent, if the union of the graphs of the complexes in each set are isomorphic. The allowed operations (also called instructions) are the following:

1. *Prepare.* The operation $T := Prepare(S)$ prepares the test tube T containing the linear complex $(S, \emptyset)$ from the set of linear strings $S \subset \Sigma^*$.
2. *Assign.* We use the usual := operator for assigning values for test tubes.
3. *Merge.* After the operation $Merge(T_1, T_2)$ the test tube T_1 becomes the union of multisets T_1 and T_2.
4. *Copy.* $T' := Copy(T)$ produces a copy of the test tube T containing only linear complexes.
5. *Amplify.* By using $Amplify(T, n)$ each complex of the multiset T is replaced by at least n identical copies of it, hence the volume of T is multiplied by at least n.
6. *Detect.* $Detect(T)$ returns true if T is not the empty multiset, false otherwise.
7. *Select.* Operations $T' := Select_{=}(T, n)$ and $T' := Select_{\neq}(T, n)$ separate the contents of T by the size of the complexes. The size of a complex is the number of nodes in its graph. T' will contain those complexes of T, whose size are equal, (or in case of $Select_{\neq}$ not equal) to n.
8. *Separate.* Operations $T' := Separate_{incl}(T, u)$ and $T' := Separate_{excl}(T, u)$ separate the contents of T by the content of the complexes in it, where u is a word over Σ. The operation may only be applied on a test tube of linear complexes. T' will contain those complexes $(S, \emptyset)$ of T, where there exists (or in case of $Separate_{excl}$ does not exist) a word $v \in S$ such that u is a subword of v.
9. *Cleave.* The operations $Cleave_{before}(T, \sigma)$ and $Cleave_{after}(T, \sigma)$ cuts every path of the complexes of T before (resp. after) the edges labelled with $\sigma \in \Sigma$. If the created complex is not connected, it is replaced by connected complexes equivalent to it.

10. *Anneal.* The $Anneal(T)$ operation changes the test tube T nondeterministically. Each complex (S, μ) of T is replaced by a complex of form (S, μ'), where μ' is a superset of μ, and there is no μ'' labelled pairing of $G(S)$ such that $\mu'' \supsetneq \mu'$.
11. *Ligate.* By the use of the operation $Ligate(T)$ every complex (S, μ) in T is replaced by a complex (S', μ'), which has the same graph as (S, μ), except that all vertices that are paired to the same node in μ are merged. That means that any two paths in $G(S)$ that have ending and beginning nodes that are paired to the same (third) node are concatenated.
12. *Denature.* The $Denature(T)$ operation replaces every complex (S, μ) in T by the set of connected complexes $\{(\{\alpha\}, \emptyset) \mid \alpha \in S\}$.

3 Representation

In this section we show a method, by which arbitrary finite relations can be represented in test tubes using the devices introduced in the previous section.

Each relation is represented by a unique test tube containing complexes over a common alphabet. The tubes contain mainly linear complexes. During performing certain operations there may occur other structures as well, but these are eliminated by the end of the operation.

Suppose that we have m not necessarily different sets $A_1, \ldots, A_m$, by which the bases of n relations $R_1, \ldots, R_n$ are defined. Each relation R_i consists of k_i components: $R_i \subseteq A_{f_{i,1}} \times \cdots \times A_{f_{i,k_i}}$ $(i = 1, \ldots, n, k_i \geq 1, f_{i,j} \in [1, m]$ for all $j \in [1, k_i])$. We will also call components of a relation as *columns*, and the indexes $f_{i,j}$ as *labels* of columns. For technical reasons we only allow relations of different base sets: for all $i \in [1, n]$, $j, k \in [1, k_i]$ such that $j \neq k$ the non–equality $f_{i,j} \neq f_{i,k}$ must hold. We may easily overcome this limitation by introducing new base sets.

We only deal with finite relations, so each A_i must be a finite set $(i = 1, \ldots, m)$. Since they are finite, we can encode them over a common alphabet X (e.g. the set of bits). Let these encodings denoted by the injective mappings $e_i : A_i \mapsto X^{l_i}$, where l_i is the letters needed to uniquely encode the elements of A_i in the alphabet X $(i \in [1, m])$. Please note that the length of $e_i(r)$ is always exactly l_i, independently of $r \in A_i$.

By the use of these mapping, we may define injective mappings from R_i to words over X. For each $i = 1, \ldots, n$ define $h_i : R_i \mapsto X^{L_i}$, where $L_i = \sum_{j=1}^{k_i} l_{f_{i,j}}$, and for all $r = (r_1, \ldots, r_{k_i}) \in R_i$, $h_i(r)$ is defined as the concatenation of the mappings of the components of r: $h_i(r) = e_{fi,1}(r_1) \cdots e_{f_{i,k_i}}(r_{k_i})$.

An element $r \in R_i$ $(i \in [1, n])$ is represented by a single linear complex over an alphabet Σ of length L_i: each letter is a triplet, where the first components are letters of X, such that the concatenation of the first components gives the word $h_i(r)$. The second components contain the index of the base set whose element is partially encoded in the letter, and the last element is the position of the encoded letter within the encoding of the referred base set. Hence we define the alphabet

$$\Sigma = \{(x, j, k), (x, j, k)' \mid x \in X, j \in [1, m], k \in [1, l_j])\}$$

where each letter has its primed version as complementary pair and vice versa. Since the alphabet Σ is fixed, we have to define all relations occurring in the computation in advance. This problem can be eliminated if we use a suitable encoding of Σ. Now let us formally define for every $i \in [1, n]$ the injective mapping $g_i : X^{L_i} \mapsto \Sigma^{L_i}$ in the above manner: for every $x_1 \dots x_{L_i} \in X^{L_i}$

$$\begin{aligned} g_i(x_1 \dots x_{L_i}) = & (x_1, f_{i,1}, 1)(x_2, f_{i,1}, 2) \cdots (x_{l_{f_{i,1}}}, f_{i,1}, l_{f_{i,1}}) \cdot \\ & (x_{l_{f_{i,1}}+1}, f_{i,2}, 1)) \cdots (x_{l_{f_{i,1}}+l_{f_{i,2}}}, f_{i,2}, l_{f_{i,2}})) \cdot \\ & \vdots \\ & (x_{L_i - l_{f_{i,k_i}}+1}, f_{i,k_i}, 1) \cdots (x_{L_i}, f_{i,k_i}, l_{f_{i,k_i}}). \end{aligned}$$

Define for every $i \in [1, n]$ the mapping $f_i = g_i \circ h_i$. Of course this mapping is also injective. An element of a relation $r \in R_i$ is represented by the single linear complex $(\{f_i(r)\}, \emptyset)$, and R_i is represented by the test tube containing the linear complex $(\{f_i(r) \mid r \in R_i\}, \emptyset)$ $(i \in [1, n])$.

Let us look at an example. We have one relation, $R_1 \subseteq A_1 \times A_2$, where $A_1 = \{i \mid 0 \le i \le 99\}$ and $A_2 = \{i \mid 0 \le i \le 9\}$. A possible encoding of the sets A_1 and A_2 is the decimal representation, we need two digits for A_1 and one digit for A_2. We get: $X = \{0, 1, \dots, 9\}$, $l_1 = 2$, $l_2 = 1$. $h_1((a, b)) = a'b'$ where $a' \in X^2$, $b' \in X$, a' and b' are the decimal representation of a and b containing leading zeroes if necessary. The complex alphabet is the following:

$$\begin{aligned} \Sigma = \{ & (x, j, k), (x, j, k)' \mid x \in [0, 9], j \in [1, 2], (j = 1 \Longrightarrow k \in [1, 2], \\ & j = 2 \Longrightarrow k = 1)\}. \end{aligned}$$

If $R_1 = \{(2, 3), (85, 0)\}$, then $h_1(R_1) = \{(02, 3), (85, 0)\}$, and

$$f_1(R_1) = \{(0, 1, 1)(2, 1, 2)(3, 2, 1), \quad (8, 1, 1)(5, 1, 2)(0, 2, 1)\}.$$

The test tube of R_1 contains two single linear complexes, each containing one single string from $f_1(R_1)$.

Although the introduced representation is redundant, not too simple and have some limitations, we choose it because it is robust and allows simple implementation of the RA operations.

4 Operations

In this section we give methods for creating test tubes containing given relations as well as creating tubes from existing ones as a result of a RA operation. We give a molecular program for Union, Selection, Cartesian product, Projection and Difference. The other RA operations can be expressed by these ones. In all our examples the test tube T_i will denote the tube that contains the representation of relation R_i ($i \in [1, n]$). We will use auxiliary tubes, too. These will be denoted by indexed S symbols.

Set up

After we fixed the originating relations and all computation by which we want to define new relations from the existing ones, we fix the alphabet Σ and the mappings defined in the previous section. The test tubes that represent the original relations can be set up by subsequent uses of the *Prepare* operation. The realization of this process in laboratory can be very expensive and time consuming for relations of many elements. An alternative method for creating large databases of DNA strands can be found in [3]. A third way can be starting from a naturally existent set of DNA strands and transform them to the form required in our model using biomolecular operations only.

Union

The execution program for creating the union R_k of two relations R_i and R_j is very simple ($i, j, k \in [1, n]$):

For $\boldsymbol{R_k := R_i \cup R_j}$ do:

$S_1 := Copy(T_i)$
$S_2 := Copy(T_j)$
$Merge(S_1, S_2)$
$T_k := S_1$

We simply make copies of the tubes representing R_i and R_j, merge them together to form T_k.

Selection

We give different programs for the selection operation σ depending on the selection condition. First, suppose that the condition is that a given column equals to a constant value or formally: in the relation R_i the column labelled j is equal to $u \in A_j$, where $i \in [1, n]$, $j \in [1, m]$. Let the letters $x_1, \ldots, x_{l_j} \in X$ be determined by the equation $e_i(u) = x_1 \cdots x_{l_j}$.

For $\boldsymbol{R_k := \sigma_{c_j = u} R_i}$ do:

$S_1 := Copy(T_i)$
$T_k := Separate_{incl}(S_1, (x_1, j, 1) \ldots (x_{l_j}, j, l_j))$

The program is based on a single *Separate* instruction that selects from a copy of the original tube those strings, that contain the (possible long) encoding of word u over alphabet Σ as subword. However, if for technical reasons we allow separations of short sequences only, we may take advantage of our redundant representation and perform the separation step by step, letter by letter, getting T_k after l_j *Separate* instructions:

For $R_k := \sigma_{c_j=u} R_i$ do:

$S_1 := Copy(T_i)$
$S_2 := Separate_{incl}(S_1, (x_1, j, 1))$
$S_3 := Separate_{incl}(S_2, (x_2, j, 2))$
$\vdots$
$S_{l_j} := Separate_{incl}(S_{l_j-1}, (x_{l_j-1}, j, l_j - 1))$
$T_k := Separate_{incl}(S_{l_j}, (x_{l_j}, j, l_j))$

Next, we show how to deal with selections where the condition is that two columns are equal. Select those elements of R_i whose columns labelled with a and b are equal. Let us suppose, that the two columns have a representation over X of equal length, that is suppose $l_a = l_b$.

For $R_k := \sigma_{c_a=c_b} R_i$ do:

$S_1 := Copy(T_i)$
For $j = 1, 2, \ldots l_a$ do
    $S_2 := \emptyset$
    For each $x \in X$ do
        $S_3 := Separate_{incl}(S_1, (x, a, j))$
        $S_4 := Separate_{incl}(S_3, (x, b, j))$
        $Merge(S_2, S_4)$
    end do
    $S_1 := S_2$
    end do
$T_k := S_1$

The inner loop separates those complexes of S, whose jth letter are equal both in column a and column b in the representation over X, since it is the union of such words, where the jth letter equal to an $x \in X$ in both columns for all $x \in X$. Having done this separation for all letters of the columns we get T_k, using a total of $2l_a\,|X|$ *Separate*, $l_a\,|X|$ *Merge*, $2l_a$ *Assign* and one *Copy* instructions.

If the condition contains $\neq$ instead of $=$, we have to modify slightly the former algorithms to give the union of those complexes that differ in at least one position from the given constant value, or the value of the other column. If the condition contains the logical operator *and*, then we model it by sucessive selection. We model *or* by merging the resulting tubes of the constituent selections. The operation *not* can always be avoided using the former operators.

It is also possible to model selection operations that contain simple arithmetic expressions in their conditions. There are several methods, by which we can perform calculations on DNA molecules, see e.g. [6], [7] and [8]. However, dealing with comparative relations < and > is not settled yet, to handle them is an open problem as of today.

Cartesian product

For creating the Cartesian product of two test tubes T_i and T_j we „stick" the proper ends of the strings in the tubes ($i, j \in [1, n]$). Because the result can have much more element, than the original relations, the test tubes must be amplified by a factor n, which is no smaller than the number of complexes in any of the two test tubes. An upper bound for n is of course $\max\{|R_i|, |R_j|\}$ as well as $\max\{|X|^{l_i}, |X|^{l_j}\}$. Please note that in our representation all column labels of a relation must be unique, so *before* executing the program creating the product of two tubes, we must *relabel* one of each pair of the columns that would have equal label in the product. This is especially important, if we take the Cartesian product of a relation with itself. The definition of relabelling and the molecular program for it is shown in the end of this section.

For $R_k := R_i \times R_j$ do:

$S_1 := Copy(T_i)$
$S_2 := Copy(T_j)$
$Amplify(S_1, n)$
$Amplify(S_2, n)$
$S_3 := Prepare(\{(x, f_{j,1}, 1)'(y, f_{i,k_i}, l_{f_{i,k_i}})' \mid x, y \in X\})$
$Merge(S_1, S_2)$
$Merge(S_1, S_3)$
$Anneal(S_1)$
$Ligate(S_1)$
$Denature(S_1)$
$S_1 := Select_{=}(S_1, L_i + L_j)$
$T_k := S_1$

We prepare a tube S_3 containing complexes of size two: the complements of any possible first symbol of R_j represented over the alphabet Σ followed by complements of any possible last symbol of R_i represented over the alphabet Σ. After merging this tube with the amplified copies of T_i and T_j, these complexes can anneal to the last letters of R_i and to the first letters of R_j. If both edges are annealed, the annealed complexes of R_i and R_j are ligated: they are stuck together, and remain stuck even when after *Denature*, the complex of S_3 breaks off. After these operations complexes of size $L_i + L_j$ form the elements of T_k representing the relation $R_k = R_i \times R_j$.

Projection

First let us show how can we project a relation R_i into a single column labelled j ($i \in [1, n], j \in [1, m]$):

For $R_k := \Pi_{c_j} R_i$ **do:**

$S_1 := Copy(T_i)$
for each $x \in X$ do
  $Cleave_{before}(S_1, (x, j, 1))$
  $Cleave_{after}(S_1, (x, j, l_{f_{i,j}}))$
end do
for each $x \in X$ do
  $S_1 := Separate_{incl}(S_1, (x, j, 1))$
end do
$T_k := S_1$

We cleave (cut) the complexes *before* any possible first letter of the ith column represented in the alphabet Σ, and then cleave *after* any possible last letter. By that we cleave each complex into three parts: the parts (complexes) that contain any (let say, first) letter of the ith column will constitute the tube containing the projection.

When we want to project into more than one column, then in addition to cutting the unnecessary ends of the strings as in the previous case, we have to erase some inner „gaps" as well: substrings that do not belong to columns in the projection list, but laying between them. We will show a molecular program by which we can erase one gap. For modelling the general case of the projection we must call this procedure for all inner gaps, and than must cut the needless ends using a procedure very similar to the former one. Let us now look at the program that erases the gap between the ath and bth column in the tube representing the relation R_i $(i \in [1, n], a, b \in [1, k_i], a < b)$:

For $\boldsymbol{S_0 := EraseGap(T_i, i, a, b)}$ **do:**

$S_1 := Prepare(\{(x, f_{i,1}, 1)'(y, f_{i,k_i}, l_{f_{i,k_i}})' \mid x, y \in X\})$
$S_2 := Prepare(\{(x, f_{i,b}, 1)'(y, f_{i,a}, l_{f_{i,a}})' \mid x, y \in X\})$
$S_0 := Copy(T_1)$
$Merge(S_0, S_1)$
$Anneal(S_0)$
for each $x \in X$
  $Cleavage_{after}(S_0, (x, f_{i,a}, l_{f_{i,a}}))$
  $Cleavage_{before}(S_0, (x, f_{i,b}, 1))$
end do
$S_0 := Select_{=}(S_0, \sum_{j=1}^{a} l_{f_{i,j}} + \sum_{j=b}^{k_i} l_{f_{i,j}} + 2)$
$Merge(S_0, S_2)$
$Anneal(S_0)$
$Ligate(S_0)$
$Denature(S_0)$
$S_0 := Select_{=}(S_0, \sum_{j=1}^{a} l_{f_{i,j}} + \sum_{j=b}^{k_i} l_{f_{i,j}})$

We create the tube S_1, whose complexes can anneal to the first and last letter of *any* complex in T_i. Using this tube we can achive that the strings of T_i form loops, so that we can cut them with the possibility not to confuse the separated beginnings and endings. After creating the rings we cut the unneeded columns and stick the broken parts together with the help of the tube S_2, whose complexes can anneal to the first letter of the bth column and to the last letter of the ath column. We may now select the result.

Difference

The last RA operation we examine is the set difference. For creating the test tube T_k that represents the difference of the relations represented in the tubes T_i and T_j we may use the following molecular program, which has a precondition that relations R_i and R_j has the same base sets ($i, j, k \in [1, n]$):

For $R_k := R_i \setminus R_j$ do:

$S_1 := Copy(T_i)$
$S_2 := Copy(T_j)$
$S_3 := Prepare(\{(x, a, b)' \mid x \in X, a \in [1, m], b \in [1, l_a]\})$
$Amplify(S_3, |R_j|)$
$Merge(S_2, S_3)$
$Anneal(S_2)$
$Ligate(S_2)$
$Denature(S_2)$
$S_4 := Select_=(S_2, L_j)$
$S_5 := \emptyset$
For each $x \in X$ do
  $S_6 := Separate_{incl}(S_2, (x, f_{i,1}, 1)')$
  $Merge(S_5, S_6)$
end do
$Merge(S_1, S_5)$
$Anneal(S_1)$
$S_7 := Select = (S_1, L_i)$
$S_8 := \emptyset$
For each $x \in X$ do
  $S_9 := Separate_{incl}(S_7, (x, f_{i,1}, 1))$
  $Merge(S_8, S_9)$
end do
$T_k := S_9$

After creating copies of T_i and T_j we create a tube S_3, that contains complexes of size one: any possible primed letter of Σ. Note that this tube does not depend on T_i or T_j, it can be used for calculating other differences as well. After merging the suitably amplified tube of S_3 with the copy of T_j, the short complexes are permitted to anneal to the complexes of S_2, hence after the *Anneal* and *Ligate* operations each complex of S_2 will be annealed (or paired) to an equally long complementary complex. After *Denature* these pairs of complexes break apart, and after some selection and separation we get the tube S_5, that contain strings that are exactly the complements of the strings in T_j. When we merge this tube with S_1, those complexes that appear both in T_i and T_j will form pairs after annealing, and those complexes that appear only in one of the tubes T_i, or T_j remain in their linear structure. From the result we only have to separate the linear complexes of T_i.

Relabelling

We say that the relation $R_j \subseteq A_{f_{j,1}} \times \cdots \times A_{f_{j,k_j}}$ is a *relabelling* of $R_i \subseteq A_{f_{i,1}} \times \cdots \times A_{f_{i,k_i}}$ if $k_i = k_j$, $R_i = R_j$ and for each $k \in [1, k_i]$ either $f_{i,k} = f_{j,k}$, or $f_{i,k} \neq f_{j,k}$, but $A_{f_{i,k}} = A_{f_{j,k}}$ and $e_{f_{i,k}} = e_{f_{j,k}}$. Hence the relabelled relation has the same base sets as the original relation, it has the same value, too, but some of its components may have a different label, but it does not affect the representation of that component over the alphabet X. Of course the representation of the two relations over the alphabet Σ will be different.

For our purposes it is enough show that relabelling where all of the columns are relabelled can be done in our model. An easy way of doing this is based on the fact that such relabelling can be expressed by relational operations:

$$R_j = \Pi_{f_{j,1},\ldots,f_{j,k_i}} \sigma_{f_{i,1}=f_{j,1} \wedge \ldots \wedge f_{i,k_i}=f_{j,k_i}} R_i \times A_{f_{j,1}} \times \cdots \times A_{f_{j,k_j}}.$$

The tube representing $A_{f_{j,1}} \times \cdots \times A_{f_{j,k_j}}$ can effectively be prepared, in spite of the fact that this tube contains an exponential number of strings. After preparing the tube we may perform the marked operations as stated before. Another way for relabelling is to form a ring of each string (similarly to the *EraseGap* operation), cut each loop before and after the letter we want to replace, then bind the broken loops again inserting the substituting letter. After doing this for all letters of the columns to relabel, we are ready.

5 Conclusions

In this work we showed that building a relational database in test tubes using DNA strands is possible. The proof is based on the assertion that the RDNA model of biomolecular computing is indeed a sound model of biomolecular operations. It seems to be a correct model, because it is based on the basic structure of DNA strands and on the well understood operations on test tubes. However, by the time of writing no real laboratory experiments justified neither the model nor any application based on the model. During laboratory realization it may turn out, that alternate versions of the mentioned operations proves to be more efficient or reliable, it depends on the used laboratory techniques.

It is not unlikely in the not too far future, that we can make complex queries on artificially created or naturally existent DNA databases. Utilizing the enormous storage capacity of DNA and the possibility of associative searches on the strands it may be possible to efficiently work with databases of size much greater than that is manageable on conventional computer architectures.

References

[1] Leonard M. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266:1021–1024, November 11, 1994.

[2] Eric B. Baum. Building an associative memory vastly larger than the brain. *Science*, 268:583–585, April 28, 1995.

[3] John H. Reif, T. H. LaBean, M. Pirrug, V. S. Rana, B. Guo, C. Kingsford, and G. S. Wickham. Experimental construction of a very large scale DNA database with associatice search capability. In *DNA Computing, 7th international Workshop on DNA-Based Computers, DNA 2001, Tampa, U.S.A., 10–13 June 2001*, pages 241–250. University of South Florida, 2001.

[4] John H. Reif. Parallel molecular computation: Models and simulations. *Algorithmica*, 1998. Special issue on Computational Biology. See also [9].

[5] Masanori Arita, Masami Hagiya, and Akira Suyama. Joining and rotating data with molecules. In *IEEE International Conference on Evolutionary Computation*, pages 243–248, Indiana University, Purdue University, Indianapolis, Illinois, April 13–16, 1997.

[6] Pierluigi Frisco. Parallel arithmetic with splicing. *Romanian Journal of Information Science and Technology (ROMJIST)*, 3(2):113–128, 2000.

[7] Eric B. Baum and Dan Boneh. Running dynamic programming algorithms on a DNA computer. In *Proceedings of the Second Annual Meeting on DNA Based Computers, held at Princeton University, June 10-12, 1996.* [10].

[8] Leonard M. Adleman, Paul W. K. Rothemund, Sam Roweis, and Erik Winfree. On applying molecular computation to the data encryption standard. In *Proceedings of the Second Annual Meeting on DNA Based Computers, held at Princeton University, June 10-12, 1996.* [10].

[9] John H. Reif. Parallel molecular computation: Models and simulations. In *Proceedings of the Seventh Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA95), Santa Barbara, June 1995*, pages 213–223. Association for Computing Machinery, June 1995. See also [4].

[10] American Mathematical Society. *Proceedings of the Second Annual Meeting on DNA Based Computers, held at Princeton University, June 10-12, 1996.*, DIMACS: Series in Discrete Mathematics and Theoretical Computer Science., 1996.