Biological Computing


*Germany, Ukraine, v_hrit3000@yahoo.com
** Ludwig-Maximilian University, Muenchen, Germany
*** University of Georgia, Georgia, USA
♣ Electronic mail address: v_hrit3000@yahoo.com

Abstracts

Biological computing theory has its roots in mathematical biology and mathematical computer sciences. Introduced by V.V. Gritsak-Groener [1], [9] in giving an example of superpower biological computational creation, which were among the first papers in this important area. The aim of this article is to describe complex logical interaction between the DNK, RNK-molecule that is transmitted through the medium of the cell automata.

We show (Theorem 1 and Theorem 2) that the structure DNA-RNA is equivalent to the structure Linear Cell Automata (g-LCA) and we show that g-LCA is Full Logical (Theorem 3 and Theorem 5). This article contains the main results:

Theorem 6-7 determines the needed and the sufficient conditions of equivalence LCA to universal logical construct (ULC).

1. Introduction

Biological computing theory has its roots in mathematical biology and mathematical computer sciences. Introduced by first author (see [1], [9]) in giving an example of superpower biological computational creation, which were among the first papers in this important area within computer sciences. The aim of this article is to describe complex logical interaction between the DNK, RNK-molecule that is transmitted through the medium of the cell automata.

We shall recall briefly the notion of biological computing.

A Linear Cell Automata (g-LCA), when it goes beyond the very elementary level of the General Cell Automata, makes considerable use of the results of DNA-Automata, as we remarked in the p.3. Let f be given by

\[ f: \{\text{Ala, Cys, \ldots, Trp, Tyr}\} \rightarrow \text{Codons}. \]  

We have linear cell automata \( C = (D, f, \text{end, end2}) \), where D is DNK-molecule, f is stand function of C, end is beginning information and end2 is finishing of information.

Simple directed graph, or simple digraph \( \mathcal{J} \) is an ordered pair:

\[ \mathcal{J} = (V(\mathcal{J}), E(\mathcal{J})), \quad E(\mathcal{J}) \subseteq V(\mathcal{J}) \cup V(\mathcal{J}), \]  

where \( V(\mathcal{J}) \) is a nonempty set called the set of vertices of \( \mathcal{J} \); \( E(\mathcal{J}) \) is a set disjoint union from \( V(\mathcal{J}) \), called the set of arrows of \( \mathcal{J} \). Before, if \( e = \langle v_1, v_2 \rangle \in E(\mathcal{J}) \Rightarrow e^\# = \langle v_2, v_1 \rangle \in E(\mathcal{J}), \) then \( v_1, v_2 \in V(\mathcal{J}). \) If \( e = \langle v_1, v_2 \rangle \) an arrow of \( \mathcal{J}, \) \( v_1 = \partial_1 e \) is called the tail (or initial) of \( e, \) and \( v_2 = \partial_2 e \) are called the spike (or terminal) of \( e. \) A digraph \( \Delta \) is an ordered pair

\[ \Delta = (V(\Delta), E(\Delta)), \]  

where
where $E(\Delta) = E_1(\mathcal{I}_1) \cup \ldots \cup E_n(\mathcal{I}_n)$ and $E(\mathcal{I}_i), i = [1, n]$ is arrows of simple digraphs \( \mathcal{I}_i = (V(\Delta), E(\mathcal{I}_i)) \). Suppose $\Delta = (V(\Delta), E(\Delta))$ is a digraph. If $e = <v_1, v_2> \in E(\Delta)$, $v_2$ is called an outneighbor of $v_1$, and $v_1$ inneighbor of $v_2$. $e$ is said to be incident out of $v_1$ and incident into $v_2$. $\ell(v_1)$ (or $\delta^+(v_1)$) denotes the set of inneighbors $v_1$ of in $\Delta$. Similarly, $st(v_1)$ (or $\delta^-(v_1)$) denotes the set of outneighbors $v_1$ of in $\Delta$. An arrow having the same ends is called a loop of $\Delta$. A $\texttt{diwalk}$ joining the vertex $v_1$ to the vertex $v_{n+1}$ in a digraph $\Delta$ is an alternating sequence $L \rightarrow v_1 e_1 v_2 e_2 v_3 \ldots e_n v_{n+1} \ (4)$ with $e_i$ incident out of $v_i$ and incident into $v_{i+1}$. A directed walk $L \rightarrow (4)$ is called a $\texttt{diloop}$ if $v_1 = v_{n+1}$. A digraph $\Delta (3)$ is called a $\texttt{diforest}$ if $\Delta$ not contains a dilooop. Moreover, a digraph

\[
\Lambda = (V(\Lambda), E(\Lambda)), \tag{5}
\]

is called a $\texttt{straightedge}$ if $V(\Lambda) = [0, 1, 2, \ldots, n]$ and $E(\Lambda) \subseteq \{(i-1, i), i \in [1, n]\}$. Also, a $\texttt{linear digraph}$ is called a digraph $\Gamma = (V, E)$ such that there exists a surjective map $\mathcal{C}: \Gamma \rightarrow \Lambda$, where $\Lambda$ is a straightedge and a restriction of the map on diwalk $L \rightarrow$ is injective map, where $L \rightarrow$ is a subgraph $\Gamma$. Finally, a vertex noun of $v \in V$ is called $\mathcal{C}(v)$. The others detailed see [5].

Let $\mathcal{A} = (a_1, \ldots, a_i, \ldots) \not\in \emptyset$ be a set, whose elements will be called codons. We say that triple $\mathcal{J} = (\mathcal{A}, \zeta, U)$ has cell universe, if there is given an injective map $\zeta: \{\#, \mathcal{A}\} \rightarrow U \cup U^2$, where $U$ is non-empty set, $\mathcal{A} = (a_1, \ldots, a_i, \ldots) \not\in \emptyset$, and $m$ is maximal arity of codons for $\mathcal{A}$. The set $U$ is called a vertices cell ($v$-cell). A cell digraph $\Gamma$ is an ordered pair

\[
\Gamma = (V, E), \ V = \mathcal{A} \setminus \{\#\}, E \subset \text{VIV} \tag{6}
\]

where $\mathcal{V}$ is a nonempty set called the set of vertices of $\Gamma$; $E$ is a subset disjoint union from $\mathcal{V}$, called the set of arrows of $\Gamma$ if the following conditions hold:

(1) $\forall x \in \mathcal{V}, (x, x) \notin E$;

(2) $\forall x, y \in \mathcal{V}, (x, y) \in E \Rightarrow (y, x) \notin E$.

Before, if $e = <v_1, v_2> \in E \Rightarrow e^\text{def} = <v_2, v_1> \notin E, v_1, v_2 \in \mathcal{V}$. If $e = <v_1, v_2>$ arrow of $\Gamma, v_1$ = $\partial^+ e$ is called the initial of $e$, and $v_2$ = $\partial^- e$ are called the terminal of $e$.

\textbf{Theorem 1. Cell digraph $\Gamma$ of DNK-RNK -Automata is a linear diforest. The proof's detailed is given in [6].}

Let $\Gamma = (V, E)$ is cell digraph (6), $V^h \subseteq V, \# \in V^y \subseteq V, \text{ and } D(\Gamma)$ is the set of all directed walk $L^\alpha(v_1, v_{n+1}) = v_1 e_1 v_2 e_2 v_3 \ldots e_n v_{n+1}$, where $v_1 \in V^h, v_{n+1} \in V^y$. There is given a map

\[
\psi: E \rightarrow (\Xi, \Xi \times \Xi).
\]
The set $\Xi$ is called an arrows cell (a-cell). $V^h$ and $V^v$ are called an input and an output.

Sekstant

$$CL = (\Gamma, \Xi, \psi, V^h, V^v, d)$$  (7)

is called 2-Band Cellular Automata (2CA). Further, the word $S = a_1 a_2 \ldots a_s \# \in V^h, a_s \in V^v, \psi(a_i, a_{i+1}) \in E,$ $\psi(a_i, a_{i+1}) \rightarrow (\Xi, \Xi \times \Xi).$ Here, CL is called 2-Band Linear Cellular Automata (2LCA).

Without loss of generality it can be assumed that 2LCA make up by the two cell bands, where DNA is the cell band (DNA-band) and RNA is the arrows cell band (RNA-band). Furthermore, an information of DNA-band takes to RNA-band under the structure of cell digraph $\Gamma.$

**Theorem 2.** 2-Band Linear Cellular Automata DNK-RNK-Automata is local isomorphic to the Linear Cellular Automata (LCA), where LCA has $N^g$ rules of cell transform and $St^g$ cell states, $\mu(N^g) < \infty$ and $\mu(St^g) < \infty.$ The proof is given in [7], [8].

**Corollary 2.1.** 2LCA DNK-RNK-Automata $\mathcal{I}$ is isomorphic to the linear cellular automata g-LCA. Let $N^g$ and $N^0$ rules of cell transform of $\mathcal{I}$ and g-LCA, $St^g$ and $St^0$ cell states $\mathcal{I}$ and g-LCA, then g-LCA has $N^g \leq 2N^0 + 1, St^g \leq 2St^0.$

**2. Logical Realization of g-LCA $\mathcal{I}$**

The formal symbol (fs) of a logical theory are the following:

(a) $A = \{a_1, a_2\}$ codon – letters (c-letters).
(b) The logical sign “$\lor$”, which is called the disjunction. 
(c) The logical sign “$\neg$”, which is called the negation.
(d) The logical sign “$\otimes$”, which is called the distinction.
(e) The logical sign “$\Box$”, which is called the replication.
(f) The specific sign “$=$”, which is called the equation.
(g) The specific sign “$\subset$”, which is called the includition.
(h) The specific sign “$\in$”, which is called the belongution.
(i) The specific sign “$($”, which is called the left bracket.
(j) The specific sign “$)$”, which is called the right bracket.

In g-LCA $\mathcal{I}$ letters $A = (a_1, \ldots, a_i, \ldots) \neq \emptyset$ are the codons, where $\mu(A) < \infty.$ $M(A)$ be the free monoid generated by $A.$ The elements of $M(A)$ are called g-words and are identified with finite sequences

$$S = a_1 a_2 \ldots a_s \#,$$  (8)

where $a_i \in A, i = [1, s], \# = \text{end or end2.}$ We recall that the length $l(S)$ of g-words (8) is $s.$ The composition in $M(A)$ will be written multiplicatively, so that
$$S_1 \circ S_2 = a_1 \ a_2 \ldots a_s \ b_1 \ b_2 \ldots b_r \ #,$$

is the sequence obtained by juxtaposition of $S_1 = a_1 \ a_2 \ldots a_s \#$ and $S_2 = b_1 \ b_2 \ldots b_r \#$. The 0-words $\varepsilon = \emptyset$ is the identity element of the monoid $M(A)$. Without loss of generality it can be designate that

$$S_1 \circ S_2 = a_1 \ a_2 \ldots a_s \ b_1 \ b_2 \ldots b_r,$$

(9)

And designate of g-word $S$ that

$$S = a_1 \ a_2 \ldots a_s.$$

**Theorem 3.** g-LCA $\mathcal{I}$ is linear cell automata if 30 cell states are:

- a) 20 non-empty codons of genetic code;
- b) 7 formal symbols;
- c) end-codon, end2-codon, $\emptyset$-cell,

and 8 rules of cell transform of $\mathcal{I}$.

**Proof.** The proofs of all statements in this theorem, including all lemmas can be found in [6], [8].

**Lemma 3.1.** Let g-LCA $\mathcal{I}$ is contained in the band the word $S = S_1 \circ S_\emptyset^s \circ S_2$, where $S_1 = a_1 \ldots a_s$, $S_2 = b_1 \ldots b_r$, $S_\emptyset^s = \emptyset_1, \ldots, \emptyset_n$, $\forall \emptyset_i$ is equivalence to $\emptyset$-cell. Then there is an $\mathcal{I}$-algorithm of the processing $S_1 \circ S_\emptyset^s \circ S_2 \longrightarrow S_1 \circ S_2$.

**The Algorithm Scheme.** We leave to the words $S_1, S_2 /\{\emptyset_n\}$ without change. Further we change a cell $\emptyset_n$ on $b_1$, the cell $b_1$ on $b_2$, . . ., the cell $b_r-1$ on $b_r$, see Fig. 1.

<table>
<thead>
<tr>
<th>a_1</th>
<th>,</th>
<th>,</th>
<th>,</th>
<th>a_s</th>
<th>$\emptyset_1$</th>
<th>,</th>
<th>,</th>
<th>,</th>
<th>$\emptyset_n$</th>
<th>b_1</th>
<th>,</th>
<th>,</th>
<th>,</th>
<th>b_r</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>,</td>
<td>,</td>
<td>,</td>
<td>a_s</td>
<td>b_1</td>
<td>,</td>
<td>,</td>
<td>,</td>
<td>b_r</td>
<td>,</td>
<td>,</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. 1

The detailed $\mathcal{I}$-algorithms of the processing $S_1 \circ S_\emptyset^s \circ S_2 \longrightarrow S_1 \circ S_2$ can be found in [6].

**Lemma 3.2.** Let $S$ be a g-word. We shall denote by $\lor S$ obtained by writing, from left to right, the sign “$\lor$”, the g-word $S$. Let $S_1 = a_1 \ldots a_s$ and $S_2 = b_1 \ldots b_r$ be g-words. Then there is an $\mathcal{I}$-algorithm of the processing $S_1, S_2 \longrightarrow \lor(S_1 \circ S_2)$, where we shall denote by $\lor(S_1 \circ S_2)$, which is called a disjunctions of $S_1$ and $S_2$, is a g-word, see Fig. 2.

| , | a_1 | , | , | a_s | , | , | b_1 | , | , | b_r |
|---|---|---|---|---|---|---|---|---|---|---|---|
| , | $\lor$ | ( | a_1 | , | , | a_s | b_1 | , | , | b_r | ) | , |

Fig. 2

The detailed $\mathcal{I}$-algorithms of the processing $S_1, S_2 \longrightarrow \lor(S_1 \circ S_2)$ can be found in [6].
Lemma 3.3. Let $S = a_1 \ldots a_s$ be a g-word. We shall denote by $\neg(S)$ obtained by writing, from left to right, the sign “$\neg$”, which is called a negations of S (It is read : not S). Then there is an $\mathcal{I}$-algorithm of the processing $S \rightarrow \neg(S)$, see Fig. 3.

```
. . . . . . a_1 . . . a_s . . . .
\downarrow
. . . . . . \neg (a_1 . . . a_s ) . . . .
```

Fig. 3

The detailed $\mathcal{I}$- algorithms of the processing $S \rightarrow \neg(S)$ can be found in [6].

Lemma 3.4. Let $S = a_1 \ldots a \ldots a \ldots a_s$ be a g-word and “a” be a letter. We shall denote by $\ominus aS$ the g-word constructed as follows: form the g-word $\ominus S$, link each occurrence of a in S to the $\ominus$ written on the left of S, and then replace a everywhere it occurs by the sign $\sqsubseteq$, which is called a distinctions of S. Then there is an $\mathcal{I}$-algorithm of the processing $S \rightarrow \ominus a(S) = a_1 \ldots \sqsubseteq \ldots \sqsubseteq \ldots a_s$, see Fig. 4.

```
. . a_1 . . . a . . . a . . . a_s
\downarrow \ominus a
. . a_1 . . . \sqsubseteq . . . \sqsubseteq . . . a_s
```

Fig. 4

The detailed $\mathcal{I}$- algorithms of the processing $S \rightarrow \ominus(a)(S)$ can be found in [8].

Lemma 3.5. Let $S_1 = a_1 \ldots a_s$ and $S_2 = b_1 \ldots b_r$ are g-words. We shall denote by $\equiv(S_1 \circ S_2)$ obtained by writing in left of $S_1 \circ S_2$ the sign “$\equiv$”, when $S_1$ coincidence $S_2$, which is called an equations of $S_1$ to $S_2$ (It is read: $S_1$ equation to $S_2$), for otherwise it denote $\equiv(S_1 \circ S_2)$ obtained by writing in left the signs “$\equiv$”. Then there is an $\mathcal{I}$-algorithm of the processing $S_1,S_2 \rightarrow \equiv(S_1 \circ S_2)$, see Fig. 5.

```
\equiv
. . a_1 . . . a_s . . . a . . . b_1 . . . b_r
\downarrow \equiv
. . a_1 . . . a_s . . . \sqsubseteq . . . \sqsubseteq . . . b_1 . . . b_r

\equiv
. . \equiv (a_1 . . . a_s b_1 . . . b_r)
```

Fig. 5

The detailed $\mathcal{I}$- algorithms of the processing $S_1,S_2 \rightarrow \equiv(S_1 \circ S_2)$ or $\equiv(S_1 \circ S_2)$ can be found in [7], [8].

Lemma 3.6. Let $S_1 = a_1 \ldots a_s$ and $S_2 = b_1 \ldots b_r$ are g-words. We shall denote by $\subset(S_1 \circ S_2)$ obtained by writing in left of $S_1 \circ S_2$ the sign “$\subset$”, when $S_1$ is a segment of $S_2$, which is called a inclusions of $S_1$ to $S_2$. Then there is an $\mathcal{I}$-algorithm of the processing $S_1,S_2 \rightarrow \subset(S_1 \circ S_2)$, see Fig. 6.
The detailed $\mathcal{I}$-algorithms of the processing $S_1, S_2 \rightarrow C(S_1 \circ S_2)$ can be found in [7], [8].

Lemma 3.7. Let $S = a_1 \ldots a_s$ be a $g$-word and $a$ be a $c$-letter. We shall denote by $\in(a \circ S)$ obtained by writing in left of $a \circ S$ the sign “$\in$”, when $a$ is a segment of $S$, which is called a \textit{belongutions} of $a$ to $S$. Then there is an $\mathcal{I}$-algorithm of the processing $a, S \rightarrow \in(a \circ S)$, see Fig. 7.

The detailed $\mathcal{I}$-algorithms of the processing $a, S \rightarrow \in(a \circ S)$ can be found in [7], [8].

Corollary 3.1. We have

\[ N^\# \leq 27, \, S^\# \leq 10. \tag{11} \]

Theorem 4. A linear cellular automata is universal computation.

Proof can be found in [7], [8].

3. $g$-LCA Universal Design

By [2] (detailed in [7], [8]), it follows that universal cellular automata is isomorphic $g$-LCA universal design. For all genetic information in system

\[ \text{DNR} \rightarrow \text{RNR} \rightarrow \text{Protein} \tag{10} \]

we have $g$-LCA universal design.

The specific signs “$=$”, “$\subset$”, and “$\in$”, of a theory $G$ are called an \textit{iteration}, and the others are called a \textit{linguisticotion}. With every the iteration is associated a natural number called its weight. A $g$-word is said to be of the \textit{first genus} if it begins with a linguistically sign, or with a “$\oplus$”, or if it consists of a $c$-letters; otherwise it is of the \textit{second genus}.

Theorem 5. Let in the theory $G$ of LCA is a sequences $S_1 \ldots S_n$ of $g$-words which has the following property: for each $g$-word $S$ of the sequence, one of the following conditions is satisfied:

1. $S$ is a $c$-letter.
2. There are two $g$-words $S_1$ and $S_2$ of the second genus, preceding $S$, such that $S$ is $S_1 \cup S_2$. 

\[ \dowc \]

Fig. 6
(3) There is g-word $S_1$ of the second genus, preceding $S$, and a c-letter $a$ such that $S$ is $\varnothing(a)S_1$.

(4) There is in the sequences a g-word $S_1$ of the second genus, preceding $S$, such that $S$ is $\varnothing S_1$.

(5) There is an iteration $i$ of weight 2 in $G$, and two g-words $S_1$ and $S_2$ of the first genus, preceding $S$, such that $i(S_1S_2)$.

The g-words of the first genus, which appear in the construct of $G$, are called g-terms in LCA. The g-words of the second genus, which appear in the construct of $G$, are called g-relations in LCA.

4. Classified Theorem of 2LCA

We are given a map $w: A \rightarrow N$, where the set $N$ is positive integers. $w(a_i)$ is called arity of the c-letter $a_i$. For each non-null the g-word $S = a_1a_2 \ldots a_n$, we put

$$w(S) = \sum_{i=1}^{n} w(a_i), \quad w(\varepsilon) = 0, \quad w(\#) = 2.$$ $w(S)$ is called the mass of the g-word $S$. We denote $S^\square$ the g-word obtained by deleting the states $\Box$ in $S$ with the left shift on the remote places. If $S_1 = S \circ S_2 \circ S^\prime$, the g-word $S_2$ is said to be a segment of $S_1$.

A worm is a sequence $S_i, i = [1, n]$, of g-words with the following property: for all g-word $S$ of sequence, one of the following two conditions is satisfied:

(a) $S$ is a sign of mass 0.
(b) $\exists m (m < n)$ g-words $S^1, \ldots, S^m$ in the sequence such that it be founds in the worm $S_i$ before $S$, and a sign $\varnothing$ mass $m$ such that $S = \varnothing S^1 \ldots S^m$.

A snake is a g-word $S$ of the following two conditions is satisfied:

(c) $l(S) = w(S) + 1$, where $l(S)$ is length of the g-word $S$.
(d) For all proper a segments $S_1$ of $S$, $w(S_1) \geq l(S_1)$.

**Theorem 6.** If a g-word $S$ is an iteration and a linguisticotion in the theory $G$, then $S$ is a snake.

**Proof.** The proofs of all statements in this theorem, including all lemmas can be found in [2], [8].

**Lemma 6.1.** If $S_1, \ldots, S_m$ are $m$ a worm and if $\varnothing$ is a sign of mass $m$, then the g-word $S = \varnothing S_1 \ldots S_m$ is a worm.

**Lemma 6.2.** A g-word is a worm iff it is a snake.

**Lemma 6.3.** $\forall$ a worm may be представлено g-LCA in exactly one way in the form $\varnothing S_1 \ldots S_m$, where $S_1, \ldots, S_m$ are worms and $\varnothing$ has mass $m$.

**Theorem 7.** Let a g-word $S$ be a snake.

For $S$ to be a g-term iff that one of the following conditions be satisfied:

(α) $S$ consists of a single c-letter.

(β) $S$ begins with “$\varnothing$”, $S^\square$ is identical with the iteration $\varnothing S_1 \ldots S_m$ and its are g-relations.
(χ) S begins with a linguisticsation sign “θ”, S is identical with the iteration 0S₁ . . . Sₘ and its are g-terms.
For S to be a g-relation iff that one of the following conditions be satisfied:
(δ) S begins with a “∨” or a “⌉”, S is identical with the iteration ∨S₁ . . . Sₘ (or ⌈S₁ . . . Sₘ) and its are g-relations.
(ε) S begins with the iteration sign “σ”, S is identical with the iteration σS₁ . . . Sₘ and its are g-terms.
∀ Sᵢ are g-words.

Proof. The Lemma 7.1 – 7.4 show that the conditions of theorem 7 are sufficient.

Lemma 7.1. If S is a g-relation in the theory G of g-LCA, then ⌈S is g-relation in G.
Proof in [2] and [8].

Lemma 7.2. If S₁ and S₂ are g-relations in the theory G of g-LCA, then ∨ S₁S₂ is a g-relation in G.
Proof in [2] and [8].

Lemma 7.3. If S is a g-relation in the theory G of g-LCA, and if a is a c-letter, then ⊚ₐ(S) is a g-term in G.
Proof in [2] and [8].

Lemma 7.4. If S₁, . . . , Sₘ are g-terms in the theory G of g-LCA, and if “σ” is an iteration of mass m in G, then σ S₁ . . . Sₘ is the g-relation in G.
Proof in [2] and [8].

Lemma 7.5. The conditions (α)-(ε) in theorem 7 are necessary conditions.
Then proof is trivial.

5. Hypothesis WILLIAM

Hypothesis WILLIAM. Is it true that the mechanism of transfer genetic information in system (10) is the classical recursive process?

Proposition. Hypothesis WILLIAM is said to be true if be realized the conditions theorems 6-7.

This will be the object of another paper.

LITERATURE


