

Biological Computing

Ph.D., Ph-Math D. Valery V. Gritsak-Groener***, Ph.D. Julia Gritsak-Groener**, Ph.D.
Hamid R. Arabnia***

*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

$$f: \{\text{Ala, Cys, . . . , Trp, Tyr}\} \longrightarrow \text{Codons.} \quad (1)$$

We have linear cell automata $C = (D, f, \text{end}, \text{end}2)$, where D is DNK-molecule, f is stand function of C , end is beginning information and $\text{end}2$ is finishing of information.

Simple directed graph, or *simple digraph* \mathfrak{S} is an ordered pair:

$$\mathfrak{S} = (V(\mathfrak{S}), E(\mathfrak{S})), \quad E(\mathfrak{S}) \subseteq V(\mathfrak{S}) \amalg V(\mathfrak{S}), \quad (2)$$

where $V(\mathfrak{S})$ is a nonempty set called the set of vertices of \mathfrak{S} ; $E(\mathfrak{S})$ is a set disjoint union from $V(\mathfrak{S})$, called the set of *arrows* of \mathfrak{S} . Before, if $e = \langle v_1, v_2 \rangle \in E(\mathfrak{S}) \Rightarrow e^\# = \langle v_2, v_1 \rangle \notin E(\mathfrak{S})$, $v_1, v_2 \in V(\mathfrak{S})$. If $e = \langle v_1, v_2 \rangle$ arrow of \mathfrak{S} , $v_1 \stackrel{\text{def}}{=} \partial^+ e$ is called the tail (or initial) of e , and $v_2 \stackrel{\text{def}}{=} \partial^- e$ are called the spike (or terminal) of e . A digraph Δ is an ordered pair

$$\Delta = (V(\Delta), E(\Delta)), \quad (3)$$

where $E(\Delta) = E_1(\mathfrak{S}_1) \cup \dots \cup E_n(\mathfrak{S}_n)$ and $E_i(\mathfrak{S}_i)$, $i = [1, n]$ is arrows of simple digraphs $\mathfrak{S}_i = (V(\Delta), E_i(\mathfrak{S}_i))$. Suppose $\Delta = (V(\Delta), E(\Delta))$ is a digraph. If $e = \langle v_1, v_2 \rangle \in E(\Delta)$, 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 . $fl(v_1)$ (or $\delta^+ v_1$) denotes the set of inneighbors v_1 of in Δ . Similarly, $st(v_1)$ (or $\delta^- v_1$) denotes the set of outneighbors v_1 of in Δ . An arrow having the same ends is called a loop of Δ . A *diwalk* joining the vertex v_1 to the vertex v_{n+1} in a digraph Δ is an alternating sequence L^\rightarrow :

$$v_1 e_1 v_2 e_2 v_3 \dots e_n v_{n+1} \quad (4)$$

with e_i incident out of v_i and incident into v_{i+1} . A directed walk L^\rightarrow (4) is called a *diloop* if $v_1 = v_{n+1}$. A digraph Δ (3) is called a *diforest* if Δ not contains a diloop. Moreover, a digraph

$$\Lambda = (V(\Lambda), E(\Lambda)), \quad (5)$$

is called a *straightedge* if $V(\Lambda) = [0, 1, 2, \dots, n]$ and $E(\Lambda) \subseteq \{(i-1, i)\}$, $i \in [1, n]$. Also, a *linear digraph* is called a digraph $\Gamma = (V, E)$ such that there exists a surjective map

$$\mathfrak{C}: \Gamma \longrightarrow \Lambda,$$

where Λ is a straightedge and a restriction of the map on diwalk L^\rightarrow is injective map, where L^\rightarrow is a subgraph Γ . Finally, a *vertex noun* of $v \in V$ is called $\mathfrak{C}(v)$. The others detailed see [5].

Let $A = (a_1, \dots, a_i, \dots) \neq \emptyset$ be a set, whose elements will be called *codons*. We say that triple $\mathfrak{J} = (A, \zeta, U)$ has *cell universe*, if there is given an injective map

$$\zeta: \{\#, A\} \longrightarrow U \cup U^2,$$

where U is non-empty set, $A = (a_1, \dots, a_i, \dots) \neq \emptyset$, and m is maximal arity of codons for A . The set U is called a *vertices cell* (*v-cell*). A *cell digraph* Γ is an ordered pair

$$\Gamma = (V, E), \quad V = A \setminus \{\#\}, \quad E \subset V \amalg V \quad (6)$$

where V is a nonempty set called the set of *vertices* of Γ ; E is a subset disjoint union from V , called the set of arrows of Γ if the following conditions hold:

- (1) $\forall x \in V, (x, x) \notin E$;
- (2) $\forall x, y \in V, (x, y) \in E \Rightarrow (y, x) \notin E$.

Before, if $e = \langle v_1, v_2 \rangle \in E \Rightarrow e^* = \langle v_2, v_1 \rangle \notin E$, $v_1, v_2 \in V$. If $e = \langle v_1, v_2 \rangle$ arrow of Γ , $v_1 \stackrel{def}{=} \delta^+ e$ is called the *initial* of e , and $v_2 \stackrel{def}{=} \delta^- e$ are called the *terminal* of e .

Theorem 1. *Cell digraph* Γ of DNK-RNK -Automata is a linear diforest. The proof's detailed is given in [6]

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

$$\psi: E \longrightarrow (\Xi, \Xi \times \Xi).$$

The set Ξ is called an *arrows cell* (*a-cell*). V^h and V^y are called an *input* and an *output*.
Sekstant

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

is called *2-Band Cellular Automata* (2CA). Further, the word $S = a_1 a_2 \dots a_s \#$ is induces 2LCA (7), if $a_1 \in V^h, a_s \in V^y, \zeta\{\#\} \longrightarrow U \cup U^2, \zeta\{a_i\} \longrightarrow U \cup U^2, \forall (a_i, a_{i+1}) \in E, \psi(a_i, a_{i+1}) \longrightarrow (\Xi, \Xi \times \Xi)$. 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). Futhermore, an information of DNA-band takes to RNA-band under the structure of cell digraph Γ .

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* \mathfrak{J} is isomorffc to the linear cellular automata *g-LCA*. Let N^g and N^0 rules of cell transform of \mathfrak{J} and *g-LCA*, St^g and St^0 cell states \mathfrak{J} and *g-LCA*, then *g-LCA* has $N^g \leq 2N^0 + 1, St^g \leq 2St^0$.

2. Logical Realization of *g-LCA* \mathfrak{J}

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 “ \vee ”, which is called the *disjunction*.
- (c) The logical sign “ \neg ”, which is called the *negation*.
- (d) The logical sign “ \odot ”, which is called the *distiguition*
- (e) The logical sign “ \square ”, 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* \mathfrak{J} letters $A = (a_1, \dots, a_i, \dots) \neq \emptyset$ 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

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

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

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

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

$$S_1 \circ S_2 = a_1 a_2 \dots a_s b_1 b_2 \dots b_r, \quad (9)$$

And designate of g-word S that

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

Theorem 3. g-LCA \mathfrak{J} 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 \mathfrak{J} .

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

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

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, \dots , the cell b_{r-1} on b_r , see Fig. 1.

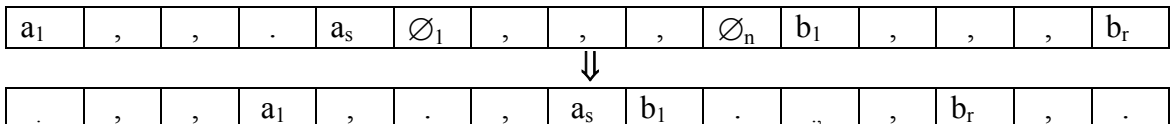


Fig. 1

The detailed \mathfrak{J} - algorithms of the processing $S_1 \circ S_\diamond^n \circ S_2 \longrightarrow S_1 \circ S_2$ can be found in [6].

Lemma 3.2. Let S be a g-word. We shall denote by $\vee S$ obtained by writing, from left to right, the sign “ \vee ”, the g-word S. Let $S_1 = a_1 \dots a_s$ and $S_2 = b_1 \dots b_r$ be g-words.

Then there is an \mathfrak{J} -algorithm of the processing $S_1, S_2 \longrightarrow \vee(S_1 \circ S_2)$, where we shall denote by $\vee(S_1 \circ S_2)$, which is called a *disjunctions* of S_1 and S_2 , is a g-word, see Fig. 2.

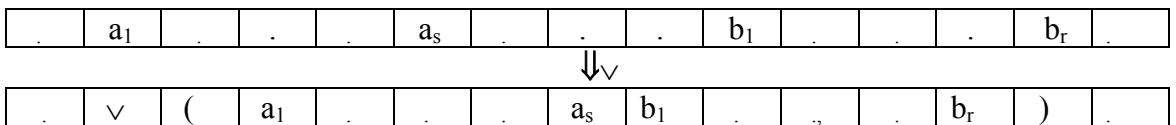


Fig. 2

The detailed \mathfrak{J} - algorithms of the processing $S_1, S_2 \longrightarrow \vee(S_1 \circ S_2)$ can be found in [6].

Lemma 3.3. Let $S = a_1 \dots a_s$ 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 \mathfrak{J} -algorithm of the processing $S \longrightarrow \neg(S)$, see Fig. 3.

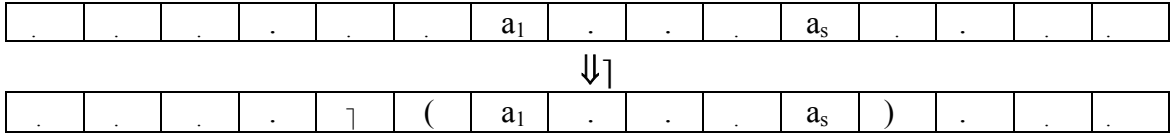


Fig. 3

The detailed \mathfrak{J} - algorithms of the processing $S \longrightarrow \neg(S)$ can be found in [6].

Lemma 3.4. Let $S = a_1 \dots a \dots a \dots a_s$ be a g-word and “a” be a letter. We shall denote by $\odot_a S$ the g-word constructed as follows: form the g-word $\odot S$, link each occurrence of a in S to the \odot written on the left of S, and then replace a everywhere it occurs by the sign \square , which is called a *distiguitions* of S. Then there is an \mathfrak{J} -algorithm of the processing $S \longrightarrow \odot_a(S) = a_1 \dots \square \dots \square \dots a_s$, see Fig. 4.

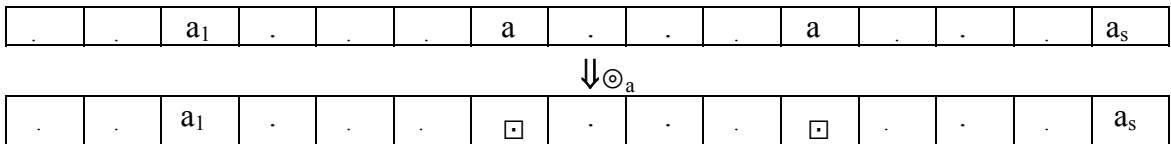


Fig. 4

The detailed \mathfrak{J} - algorithms of the processing $S \longrightarrow \odot(a)(S)$ can be found in [8].

Lemma 3.5. Let $S_1 = a_1 \dots a_s$ and $S_2 = b_1 \dots b_r$ are g-words. We shall denote by $=(S_1 \circ S_2)$ obtained by writing in left of $S_1 \circ S_2$ the sign “ $=$ ”, 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 $=\neg(S_1 \circ S_2)$ obtained by writing in left the signs “ $=\neg$ ”. Then there is an \mathfrak{J} -algorithm of the processing $S_1, S_2 \longrightarrow =(S_1 \circ S_2)$ or $=\neg(S_1 \circ S_2)$, see Fig. 5.

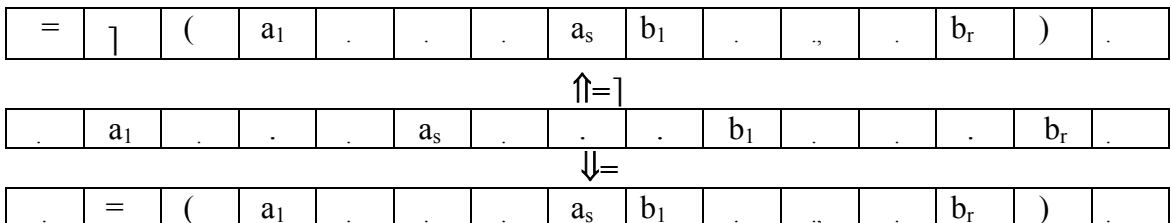


Fig. 5

The detailed \mathfrak{J} - algorithms of the processing $S_1, S_2 \longrightarrow =(S_1 \circ S_2)$ or $=\neg(S_1 \circ S_2)$ can be found in [7], [8].

Lemma 3.6. Let $S_1 = a_1 \dots a_s$ and $S_2 = b_1 \dots b_r$ 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 *includations* of S_1 to S_2 . Then there is an \mathfrak{J} -algorithm of the processing $S_1, S_2 \longrightarrow \subset(S_1 \circ S_2)$, see Fig. 6.

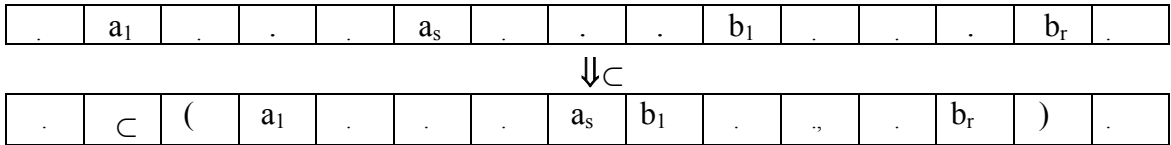


Fig.6

The detailed \mathfrak{J} - algorithms of the processing $S_1, S_2 \longrightarrow C(S_1 \circ S_2)$ can be found in [7], [8].

Lemma 3.7. Let $S = a_1 \dots a_s$ is g-word and a is 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 *belongutions* of a to S. Then there is an \mathfrak{J} -algorithm of the processing $a, S \longrightarrow \in(a \circ S)$, see Fig. 7.

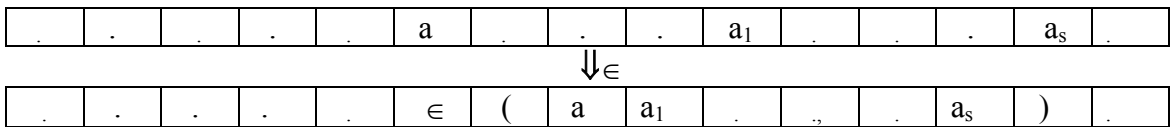


Fig.6

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

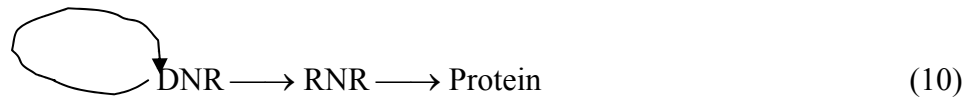
Corollary 3.1. We have

$$N^g \leq 27, St^g \leq 10. \tag{11}$$

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



we have g-LCA universal design.

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

Theorem 5. Let in the theory G of LCA is a sequences $S_1 \dots S_n$ 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 $S_1 \vee S_2$.

- (3) There is g-word S_1 of the second genus, preceding S , and a c-letter a such that S is $\odot(a)S_1$.
- (4) There is in the sequences a g-word S_1 of the second genus, preceding S , such that S is $\lrcorner S_1$.
- (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 S is $i(S_1 S_2)$.

Тоді, підпоследовність $S_1 \dots S_i$, $i \leq n$, is a *construct* of S_i .

Proof in [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 $w: A \longrightarrow N$, 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 $S = a_1 a_2 \dots a_s$, we put $w(S) = \sum_{i=1}^s w(a_i)$, and $w(\varepsilon) = 0$, $w(\#) = 2$. $w(S)$ is called the *mass* of the g-word S . We

denote S^\square the g-word obtained by deleting the states \square in S with the left shift on the remote places. If $S_1 = S' \circ S_2 \circ S''$, 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 \forall 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, \dots, S^m in the sequence such that it be founds in the worm S_i before S , and a sign φ mass m such that $S = \varphi S^1 \dots S^m$.

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 \forall 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, \dots, S_m are m a worm and if φ is a sign of mass m , then the g-word $S = \varphi S_1 \dots S_m$ 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 $\varphi S_1 \dots S_m$, where S_1, \dots, S_m are worms and φ 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 “ \odot ”, S^\square is identical with the iteration $\varphi S_1 \dots S_m$ and its are g-relations.

(χ) S begins with a linguistification sign “ θ ”, S^\square is identical with the iteration $\theta S_1 \dots S_m$ 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 “ \vee ” or a “ \lceil ”, S^\square is identical with the iteration $\vee S_1 \dots S_m$ (or $\lceil S_1 \dots S_m$) and its are g-relations.

(ε) S begins with the iteration sign “ σ ”, S^\square is identical with the iteration $\sigma S_1 \dots S_m$ and its are g-terms.

$\forall S_i$ are a 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 $\lceil S$ is g-relation in G .

Proof in [2] and [8].

Lemma 7.2. If S_1 and S_2 are g-relations in the theory G of g-LCA, then $\vee S_1 S_2$ 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

$\odot_a(S)$ is a g-term in G .

Proof in [2] and [8].

Lemma 7.4. If S_1, \dots, S_m are g-terms in the theory G of g-LCA, and if “ σ ” is an iteration of mass m in G , then $\sigma S_1 \dots S_m$ 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

1. V.V.Gritsak (V.V. Gritsak–Groener). Proc. Nat. Acad. Sci. Ukraine SSR, ser. A, N6(1990)68.
2. V.V.Gritsak (V.V.Gritsak–Groener), Bulletin Schevtshenko Kyjiv National University, ser. phys.-math. science, N1(1993)5.

3. Valery V.Gritsak–Groener, Julia Gritsak–Groener, Hamid R. Arabnia, Superpower Computational Creation for Biological Computing, Proceedings of the 2005 International Conference on Scientific Computing, Las Vegas, Nevada, USA June 20-23, 2005, pp.184-190.
4. Valery V. Gritsak (Gritsak–Groener), Julia P. Okolita, A Categorical Model of Neurosystem, Proceedings of the 1996 International Conference on Parallel and Distributed Processing Techniques and Applications, Sunnyvale Hilton, California, USA, August 9-11, 1996, pp.1610-1613.
5. Valery V.Gritsak-Groener, Julia Gritsak-Groener, S. Iglin, ARTS COMBINATORIA. NTU “HPI”, Charkiv, second edit., 2006.
6. Wilhelm Groener (V.V.Gritsak–Groener), Fundamental of Mathematical Cybernetics. ZhDTU, Zhytomyr, 2004.
7. Valery V.Gritsak-Groener, Hamid R. Arabnia, Modern Mathematical Biology, 2006.
8. Valery V.Gritsak–Groener, Fundamental of Mathematical Cybernetics, vol 2. ZhDTU, Zhytomyr, 2006.
9. V. V.Gritsak (Valery V.Gritsak–Groener), Computational Model of Transfer RNA Molecules. Preprint 87IMBG-14, Institute of Molecular Biology and Genetics of National Academy Science of Ukraine, Kyjiv,1987.