

CODIFICATION OF ACYCLIC ISOPRENOID STRUCTURES USING  
CONTEXT-FREE GRAMMARS AND PUSHDOWN AUTOMATA

Mariana Barasch,<sup>a</sup> S. Marcus<sup>a</sup> and A.T. Balaban<sup>b,\*</sup>

<sup>a</sup> Faculty of Mathematics, The University, Bucharest, Romania

<sup>b</sup> Department of Organic Chemistry, The Polytechnic,  
Bucharest, Romania

\* To whom correspondence should be addressed

( Received: April 1981)

*Abstract.* Starting from an idea of Read for coding tree-like graphs, acyclic isoprenoid structures are coded using a binary notation system. Context - free grammars are used for generating either regular (head-to-tail) or standard acyclic polyisoprenoid structures (head-to-tail, tail-to-head, tail-to-tail, head-to-head). It is shown that the characteristics of these codes are the following : for regular acyclic polyisoprenoid structures the code detects nine errors and may correct four errors ; for standard acyclic polyisoprenoid structures the code detects a single error. By using deterministic pushdown automata, one may check the presence or absence of errors, and in the former case one may correct in part the errors.

## 1. Introduction

Associated with the generation of isoprenoid structures using web grammars<sup>1</sup> there emerges the problem of generating the corresponding codes which are necessary for a simpler computer representation of such graphs. The coding problem for graphs is that of specifying an algorithm for associating with every graph a *code*, that is, a linear string of symbols, in such a way that two graphs are isomorphic if, and only if, they have the same code. On the basis of an elaborated coding algorithm we have constructed a context-free grammar which associates the corresponding code, if for each production of the web grammar used for obtaining a structure of the required type we apply the corresponding production of the context-free grammar, generator of codes.

It should be mentioned that we are aware of the limitations of the present system which consciously encompasses only acyclic isoprenoid structures. However, we consider it as a demonstration of the usefulness of grammars for chemistry. More general and ambitious coding systems have been developed, such as the correspondence between chemical names or formulas and grammars,<sup>2</sup> or linear notation systems like the Wiswesser system<sup>3</sup>, or topological coding systems like the Morgan algorithm used by the Chemical Abstracts Service.<sup>4</sup> For a literature review of such general coding systems and for a novel approach improving considerably the Morgan algorithm, cf.ref.<sup>5</sup>

## 2. Coding and decoding of acyclic isoprenoid structures

During the coding algorithm every vertex will have a *tag* associated with it, where a tag is a string of symbols from some 2-symbol set. In the present application we use the set  $\{0,1\}$  so that every tag is a binary integer.

Let us call the *essential vertex* of an acyclic standard isoprenoid structure the first vertex with degree 3 met when going along the given graph (one imagines a traveller following a path along the graph, starting with one of its endpoints). This vertex exists in any structure of the required type, beginning with the trivial one (one isoprene unit). The algorithm intends to label the essential vertex of the structure and then to take this tag as binary representation of the graph. We take into account that any structure of the required type has only vertices whose degrees belong to the set  $\{1,2,3\}$ .

### 2.1. Coding algorithm

The Coding algorithm is based on an idea of Read.<sup>6</sup>

- Step 1. Choose a crossing sense for the acyclic graph ( $\uparrow, \downarrow, \rightarrow, \leftarrow$ ) and take as essential vertex the first vertex with degree 3 met when going along the graph in the chosen sense.
- Step 2. Note all endpoints (with degree 1) and the vertices to which they are adjacent. Call these latter vertices *roots*. Label the endpoints with 10.

- Step 3. If the root has degree 2 we obtain its tag thus : write down the tag of its adjacent endpoint and place a '1' before it and a '0' after it.
- Delete all the present endpoints and their incident edges.
- Step 4. If the root has degree 3 being different from the essential vertex and having two adjacent endpoints, we obtain its tag thus : juxtapose in non-descending order the tags of its adjacent endpoints and place a '1' before this binary integer and a '0' after it.
- Delete all the present endpoints and their incident edges.
- If the root has a single endpoint repeat step 2 and 3 till obtaining a second endpoint and apply what was written in the previous step.
- Step 5. If the root is just the essential vertex and has three adjacent endpoints construct its tag thus : juxtapose in non-descending order the tags of its adjacent endpoints and place a '1' before this binary integer and a '0' after it.
- Delete all the present endpoints and their incident edges. Otherwise, if the resulting graph has more than four vertices repeat steps 2, 3 and 4 till obtaining the situation shown in step 5.

Step 6. The tag of the essential vertex will be the binary representation of the acyclic graph and this is determined in a unique way by the essential vertex chosen.

Example

Take the following structure whose binary representation will be studied depending on the essential vertex chosen. We will label by N the essential vertex - it is unique with reference to the chosen sense (direction). In turn we consider all the directions of going along the graph and indicate the vertex of degree 1 which represents the starting point in going along the graph. The four directions of going along will be illustrated in Figures 1,2,3 and 4 in conformity with the starting endpoint. The given structure :

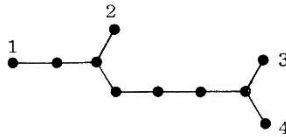
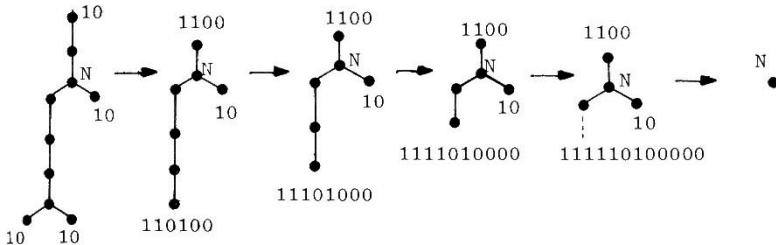
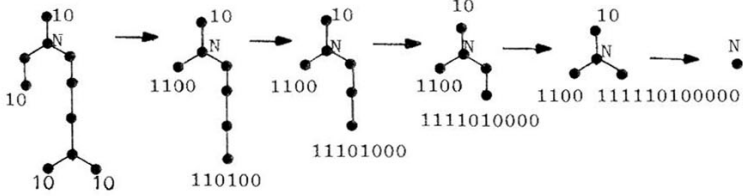


Figure 1



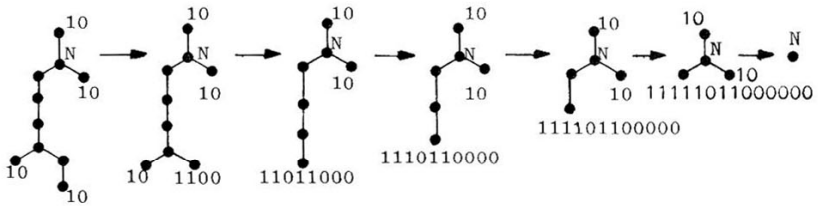
The code of the structure (1) is : 11011001111101000000

Figure 2



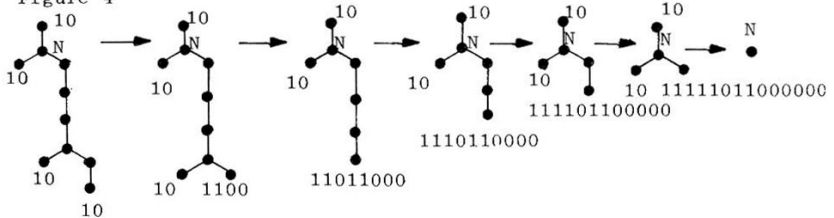
The code of the structure (2) is : 11011001111101000000

Figure 3



The code of the structure (3) is : 11010111110110000000

Figure 4



The code of the structure (4) is : 11010111110110000000

Comments

a) The sense of going along determines in a unique way the essential vertex. However there exist paths which have the same essential vertex (figures 1 and 2, and respectively, figures 3 and 4).

b) The choice of the essential vertex determines in a unique way the code of the structure (structures (1) and (2), and respectively, structures (3) and (4) have the same code).

c) We observe that by means of the algorithm mentioned above for the chosen graph, we obtained two distinct codes corresponding to two different choices of the essential vertex.

This would mean that the coding algorithm does not give a unique binary representation for a given graph since this would depend by the choice of the essential vertex. Actually, the four graphs presented in Figures 1-4 are isomorphic, but a chemical distinction may be made between cases 1 or 2 on one hand, and 3 and 4 on the other hand, namely that we determine the direction of going along the graph by starting from the endpoint which is closest to a given functional group which may be present in the compound represented by the given graph (a functional group contains atoms different from carbon and hydrogen, or multiple C = C bonds). From a chemical viewpoint, the variants (1) and (2) of the given graph represent an acyclic isoprenoid structure composed of two isoprene units linked "head-to-tail" (HT), while the variants (3) and (4) correspond to an acyclic isoprenoid structure with "tail-to-head" (TH) linking. Thus, from now on we will refer to the notion of unicity of a code for a chemical structure, not for a graph, i.e., the direction of going along the given graph may be prescribed by the presence and priority of functional groups.

### 2.2. Decoding

The above algorithm gives a unique code for any standard acyclic isoprenoid structure, since it prescribes exactly

what must be done at each stage and does not depend on any labelling of the vertices. To show that two non-isomorphic structures cannot have the same code we now describe a decoding algorithm, whereby the structure can be recovered from the code.

It is easily checked that the two symbols "1" and "0" used above behave exactly like the parentheses "(" and ")" respectively. Any tag is incorporated into parentheses, thus avoiding any dispute as to where one tag ends and another begins. The decoding procedure is most easily appreciated if we write the code with parentheses instead of 1's and 0's. Thus for the structure (1) we have :

(( )(( ))(((( ( ) ( ) ) ) ) ) ) )

as the code that is to be decoded. All we do is to see which left and right parentheses go together, regarding them as being the extreme left- and right-hand portions of circles or of other closed curves, and fill in the rest of this curve. Thus, the representation with parentheses above becomes the drawing presented in Fig.5.

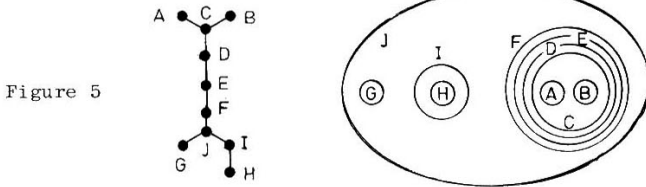


Figure 5

Each closed curve corresponds to a vertex of the structure, and the relation of adjacency is that of immediate inclusion of one circle into another.

This method of drawing circles is equivalent to following a more formal algorithm, using the original "0", "1" symbols.



Decoding algorithm

- Step 1. Associate a label (a number, letter, or other symbol) which each "1" appearing in the code. (This is purely for reference; each "1" corresponds to a vertex of the structure).
- Step 2. Scan the code from left to right until a configuration "110" is found. Write down the pair of labels associated with the two 1's in the configuration, then delete the second "1" and the "0".
- Step 3. If the resulting string is "10", the vertex associated with 1 in the configuration represents the essential vertex; then go to Step 4. Otherwise, if the resulting string has more than two symbols repeat from Step 2.
- Step 4. The pairs of labels which have been written down in Step 2 will specify the edges of the structure. From these the structure can be reconstructed.

Thus the decoding of the structure from Fig.5 gives the following result :

```

Code :      1 10 1 1 0 0 1 1 1 1 0 1 0 0 0 0 0
           :  :  :  :  :  :  :  :  :  :  :  :
Vertices:  J G I H   F E D C A B
Bond                           Stepwise deletion of 10 and vertex
G-J      ↘ 1110 0 111 ...
H-I      JIH   FED ...
J-I      110111 ...
          JIFED ...
C-A      1111110100 ...
          JFEDCA B
B-C      111111000 ...
etc.     JFEDCB
    
```

The justification for this is that the sequence "10" gives the smallest size circle, and therefore represents a vertex of degree 1. It is adjacent only to the vertex represented by the circle, say Z, which contains it. If it is the left-most of the circles contained in Z, this will produce the configuration "110". If it is not, then it will eventually become so when the circles, also in Z, which lie to its left have been deleted. The sole adjacency of this vertex is recorded before it is deleted from the code. We end up with "10". The label associated with this "1" represents the essential vertex of the structure.

We will denote "head-to-tail" (regular) and "standard" linking by HT and S, respectively : S indicates head-to-tail, head-to-head, tail-to-head, or tail-to-tail linking. <sup>1,7</sup>

### 3. Context-free grammar for codification of standard (S) and regular (HT) acyclic isoprenoid structures

The context-free grammars for codification of standard acyclic isoprenoid structures were obtained by mimicking - in terms of codes - each production of the constructed web grammars, ignoring the labels of vertices used by them. As any grammar represents a generative mechanism, the codification of the structure which appears in each rule of the considered web grammar could not be done totally in conformity with the coding algorithm. The modification consists in a criterion for ordering the tags of endpoints for constructing the tag of the root. Since during the algorithm some tags of endpoints are still unknown (namely, they are not expressed by "0" and "1") they

could not be arranged in a non-decreasing order. For this reason their incorporation for obtaining the tag of the root will be effected respecting the position : if this is a vertex of degree three, different from the essential vertex, one respects the position left or right relatively to the essential vertex ; if this is the essential vertex; one respects the position left, up or right. Thus, after the codification process given by the context-free grammars, we shall obtain a unique code for each structure of the required type with the exception of the order of those two and respectively three configurations from the same circle (parentheses). We obtain the unique code by arranging the configurations which appear in the same circle in a non-decreasing order. Owing to the generative character of the grammars, we know the direction of going along the graph and hence the essential vertex of each obtained structure.

If we use for each rule of the considered web grammar its corresponding production of the context - free grammar, we obtain in parallel with the generation of the structure also its code.





*3.1. The codification of acyclic regular {HT} isoprenoid structures using a context - free grammar*

Let the context - free grammar be  $G^{HT} = (V_N, V_T, P, S)$ ; where

$$V_N = \{S, B, D\} \quad , \quad V_T = \{0, 1\}$$

The productions used by  $G^{HT}$  are given by Table 1.

Table 1

No. of $G^{HT}$	Production of $G^{HT}$ using "0" and "1"	Production of $G^{HT}$ using curves	The corresponding production of $G_W^{HT}$
1	$S \rightarrow 1B1100B0$	$S \rightarrow$ 	$\pi_1$
2	$B \rightarrow 1D0$	$B \rightarrow$ 	$\pi_2$
3	$D \rightarrow 111BB000$	$D \rightarrow$ 	$\pi_3$
4	$B \rightarrow 10$	$B \rightarrow$ 	$\pi_4$

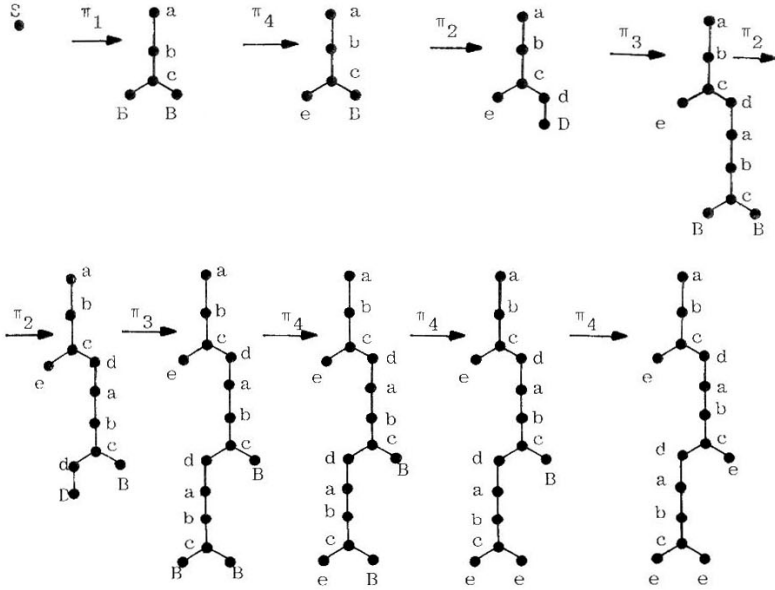
Comments

The productions  $\pi_2$  and  $\pi_3$  of  $G_W^{HT}$ , and respectively 2 and 3 of  $G^{HT}$  may be merged into a single rule. We did not do this because we intended to follow more easily the process of concatenation of isoprene units.

In the productions, the nonterminals B and D represent labels of the endpoints of the structure which appear in the corresponding rule of the considered web grammar.

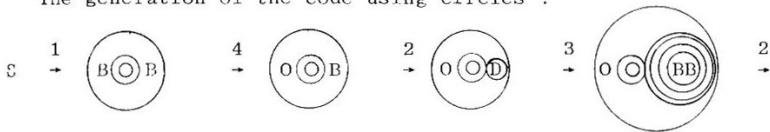
Example

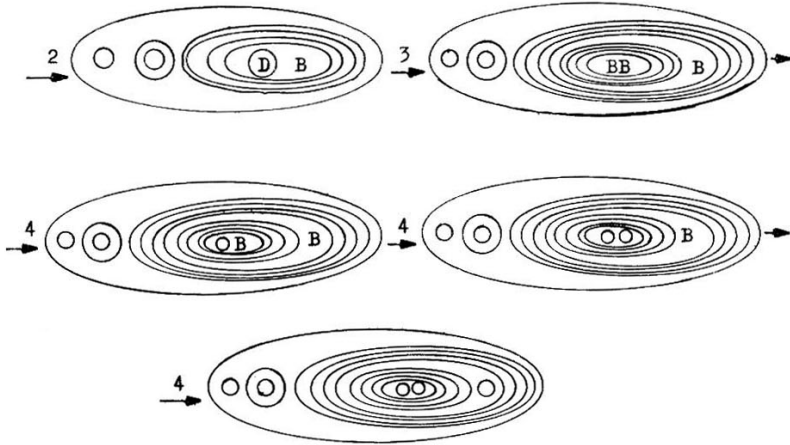
We will generate an acyclic HT isoprenoid structure and its code.



$S \xrightarrow{1} 1B1100E0 \xrightarrow{4} 11011100E0 \xrightarrow{2} 110111001D00 \xrightarrow{3}$   
 $\xrightarrow{3} 110110011111BB00000 \xrightarrow{2} 110110011111E0B000000 \xrightarrow{3}$   
 $\xrightarrow{3} 1101100111111111BB00000B00000 \xrightarrow{4}$   
 $\xrightarrow{4} 1101100111111111110B00000B00000 \xrightarrow{4}$   
 $\xrightarrow{4} 110110011111111110100000B00000 \xrightarrow{4}$   
 $\xrightarrow{4} 1101100111111111101000001000000$

The generation of the code using circles :



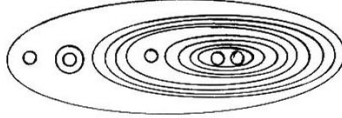


The generation of the code using parentheses

$$\begin{array}{l}
 1 \quad 4 \quad 2 \quad 3 \\
 S \rightarrow (B(()B) \rightarrow ((()())B) \rightarrow ((()()) (D)) \rightarrow \\
 3 \quad 2 \quad 3 \\
 \rightarrow (((()())(((BB)))))) \rightarrow (((()())((((D) B)))))) \rightarrow \\
 3 \quad 4 \\
 \rightarrow (((()())((((((( BB ))) B)))))) \rightarrow \\
 4 \quad 4 \\
 \rightarrow (((()())(((((((( B ))) B)))))) \rightarrow \\
 4 \quad 4 \\
 \rightarrow (( () ) (((((((((( () ))))) B)))))) \rightarrow \\
 4 \\
 \rightarrow (( () ) (((((((((( () ))))) ())))))
 \end{array}$$

We obtain the unique code after arranging the configurations which appear in each circle of the code above in non-descending order. Thus, we have the representation of Figure 6 which gives us the unique code 110110011111011111010000000000 of the above HT acyclic isoprenoid structure with three isoprene units.

Figure 6



3.2. The codification of acyclic standard isoprenoid structures using a context-free grammar

Let us take the context-free grammar  $G^S = (V_N, V_T, P, S)$ , where

$$V_N = \{S, A, B, D\} \quad , \quad V_T = \{0, 1\}$$

The productions used by  $G^S$  are given by Table 2

Table 2

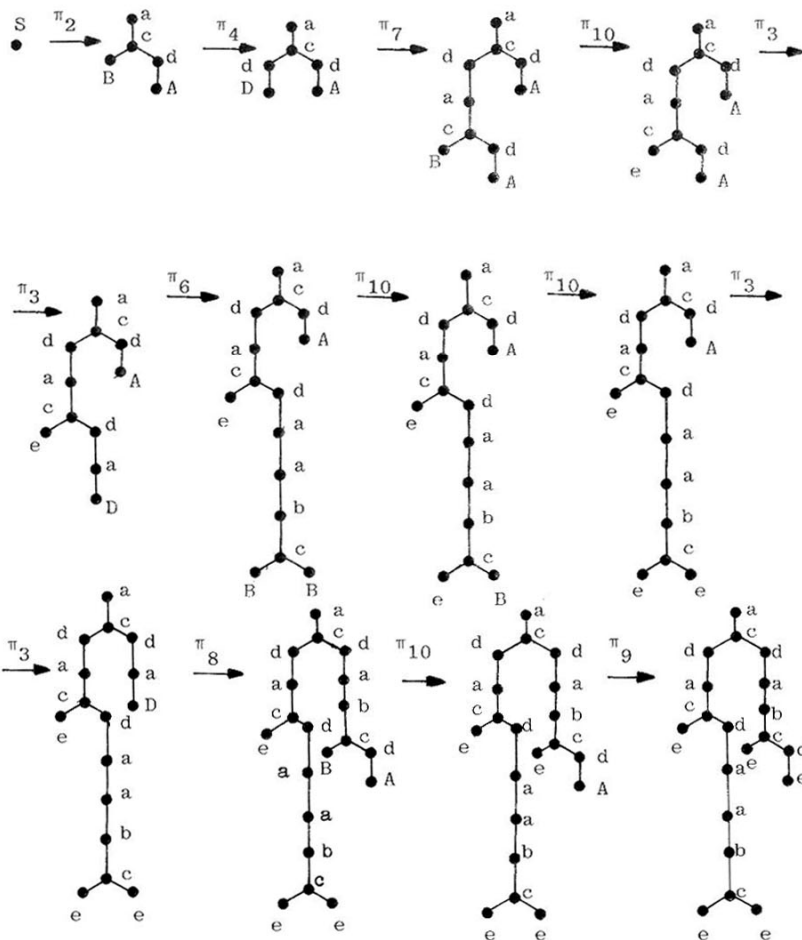
No.	Production of $G^S$ using "1" and "0"	Production of $G^S$ using curves	The corresponding production of $G_W^S$
1	$S \rightarrow 1B1100B0$	$S \rightarrow$	$\pi_1$
2	$S \rightarrow 1B101A00$	$S \rightarrow$	$\pi_2$
3	$A \rightarrow 1D0$	$A \rightarrow$	$\pi_3$
4	$B \rightarrow 1D0$	$B \rightarrow$	$\pi_4$
5	$D \rightarrow 111BB000$	$D \rightarrow$	$\pi_5, \pi_6$
6	$D \rightarrow 11B1A000$	$D \rightarrow$	$\pi_7, \pi_8$
7	$A \rightarrow 10$	$A \rightarrow$	$\pi_9$
8	$B \rightarrow 10$	$B \rightarrow$	$\pi_{10}$

Comments

In the productions, the nonterminals A, B, D represent labels of the endpoints of the structure which appears in the corresponding rule of the considered web grammar.

Example

We will generate a standard acyclic isoprenoid structure and its code.





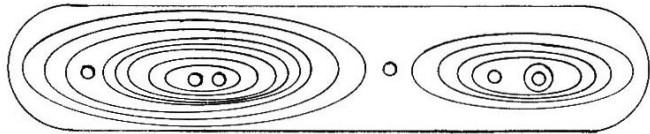
```

S  2 → 1B101A00 → 4 11D0101A00 → 6 1111B1A0000101A00 → 8
    8 → 1111101A0000101A00 → 3 11111011D00000101A00 → 5
    5 → 11111011111BB00000000101A00 → 8
    8 → 1111101111110B00000000101A00 → 8
    8 → 111110111111010000000000101A00 → 3
    3 → 1111101111110100000000001011D000 → 6
    6 → 111110111111010000000000101111B1A000000 → 8
    8 → 111110111111010000000000101111101A000000 → 7
    7 → 11111011111101000000000010111110110000000

```

The generation of the code using circles leads to the final form presented in Fig.7.

Figure 7



The generation of the code using parentheses :

```

S  2 → (B()(A)) → 3 ((D)()(A)) → 6 (((B(A)))) ( ) (A) → 8
    8 → (((((A)))) ( ) (A)) → 4 (((((D)))) ( ) (A)) → 5
    5 → ((((((PB)))))))( ) (A) → 8 (((((((B)))))))( ) (A)
    8 → ((((( )(((( ( ) )))))))) ( ) (A) → 4
    4 → ((((( ) (((((( ) ( ))) )))))( ) (D)) → 6

```



### 3.3. Characteristics, (error-detecting and error-correcting ability) of the codes

Using the context-free grammars  $G^{HT}$  and  $G^S$  we generated the codes (binary integers) of the HT and respectively  $S$  acyclic isoprenoid structures. Let us call these binary integers the *words of HT* and respectively *S code*.

The *length of the word* is the number of symbols, "0" and "1" in our case, which appear in the word.

Because of the type of codification (for each vertex of the structure we put into the word a "1" and a "0") for any word of the HT or  $S$  code, the number of 1 symbols is equal to the number of 0 symbols. Also, taking into consideration that any studied structure has as its number of vertices a multiple of 5, we conclude that the length of any word of HT or  $S$  code is a multiple of 10. The words with the same length are the binary representations of the structures composed of the same number of isoprene units.

We denote the *distance between two words  $x$  and  $y$*  of the same length by  $d(x,y)$ ; this is the number of the places of these words where there are different symbols. The name of distance is justified if we observe that the properties of the distance between two points are fulfilled :

- (i)  $d(x,y) = 0$  if and only if  $x$  coincides with  $y$  ;
- (ii)  $d(x,y) = d(y,x)$ , the distance between two words does not depend on the order in which the words are considered ;

(iii) whatever the words  $x, y, z$ , we have the following relation :

$$d(x, y) \leq d(x, z) + d(z, y)$$

In fact,  $d(x, y)$  is the well-known Hamel distance used in the algebraic theory of binary codes. (See, for instance, /8/).

Given a binary code with words of the same length, the problem is to detect the eventual errors and to correct them. By an *error* we mean the wrong receiving of a symbol (for example, instead of a "0" emitted, one receives "1" and vice-versa).

A code as considered above *detects a simple* {unique} *error*, whatever this unique error is, if its appearance determines the transformation of a word of the code into a word which does not belong to the code. Using the notion of distance, the characterization of codes which detect a unique error is given by :

*Proposition 1.* A code detects a unique error if and only if there exist two words belonging to it, so that the distance between them is greater than 1.

This proposition can be generalized if we introduce the following definition : A code *detects n errors* if any chain of at most  $n$  errors determines the transformation of the words of the code into words which do not belong to the code. Thus, we have the following result :

*Proposition 2.* A code detects  $n$  errors if and only if there are two words belonging to it, so that the distance between them is greater than  $n$ .

A code corrects any unique error if such an error determines the transformation of a word belonging to the code into a word which does not belong to the code, but which is related to a certain and unique word of the code, much more closely than to any other word of the code. This unique word will be in fact the correct word which is to replace the wrong one.

*Proposition 3.* A code corrects any unique error if and only if the distance between two arbitrary words of the code is greater than 2.

A code corrects  $n$  errors if any chain of at most  $n$  errors transforms a word of the code into a word which does not belong to the code, but which is related to a certain and unique word of the code more closely than any other word of the code. The generalization of the above proposition is given by :

*Proposition 4.* A code corrects  $n$  errors if and only if the distance between two arbitrary words of the code is greater than  $2n$ .

We note by  $C_{n.10}^{HT(S)}$  the set of words which represent the unique binary representations of the HT (S) acyclic isoprenoid structures with  $n$  isoprene units. Let us call  $C_{n.10}^{HT(S)}(C_{n.10})$  the code of HT (S) acyclic isoprenoid structures with  $n$  isoprene units. Taking into account the observations mentioned above, it results that all the words of  $C_{n.10}^{HT(S)}(C_{n.10})$  code have the length equal with  $n.10$ , for any  $n \geq 1$ ,  $n$  natural. Let us call the code of acyclic HT(S) isoprenoid structures, denoted by  $C^{HT(S)}$ , the set formed by the union of all  $C_{n.10}^{HT(S)}(C_{n.10})$  sets, where  $n \geq 1$ , namely

$$C^{HT} = \bigcup_{n \geq 1} C_{n.10}^{HT} \quad ; \quad C^S = \bigcup_{n \geq 1} C_{n.10}^S$$

It is easy to observe that  $C^{HT} \subset C^S$ , because the set of acyclic HT isoprenoid structures is a subset of the set of acyclic S isoprenoid structures.

In the following we shall study the properties of detecting and correcting errors of those two codes,  $C^{HT}$  and  $C^S$ . Let us consider the codes  $C_{n.10}^{HT}$ ,  $n \geq 3$ , as the problem raised makes no sense for  $C_{1.10}^{HT}$  and  $C_{2.10}^{HT}$  which contain a single word.

From the viewpoint of a chemical language, the notion of *identical* does not correspond to that met in drawing. Thus, the structures of Fig.10 are considered identical and have the same code.

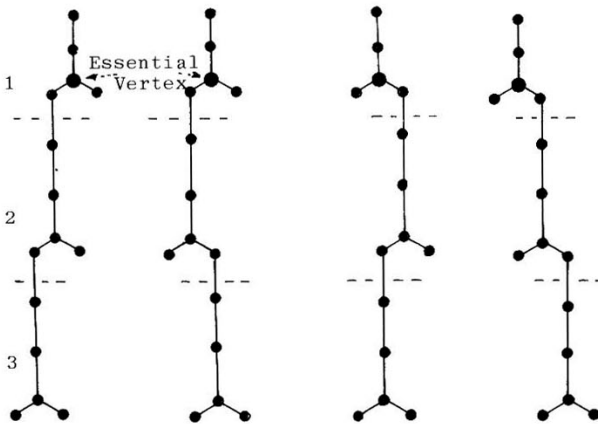


Figure 10

Each of the above structures represents three isoprene units with HT linking, two units being situated on the same part of the essential vertex (either on the left or on the right) and the last of those two units may be situated on the left or on the right of the vertex of degree 3 from the second isoprene unit (the isoprene units have been numbered in the sense of generating the structure, namely from top to bottom).

Two structures are *identical* if they have the same distribution of the isoprene units along those two ramifications which start from the essential vertex of the structure excluding the positioning of the branch on the left or on the right hand of the essential vertex. This is also valid for the distribution of the isoprene units along each branch in comparison with the preceding vertex of degree 3.

The notion of *level  $i$*  of an acyclic isoprenoid structure is given by the Figure 11.

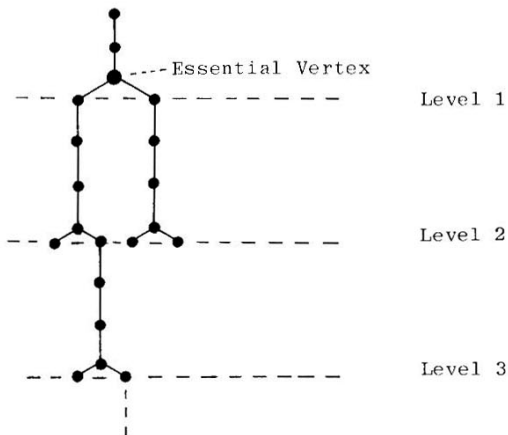


Figure 11

We observe that the distance between the binary representation of two HT acyclic isoprenoid structures with  $n$  isoprene units ( $n > 3$ ) is minimum when those structures are identical. An exception is constituted by two structures differing by a single isoprene unit which is translated above or below with a single level; this may happen at any level of the same branch relative to the essential vertex. Let us call such structures, *structures of type X*.

For simplification, in the examples of Figure 12, we will note with  $x_i$  the coding of the structure labelled with  $i$ .

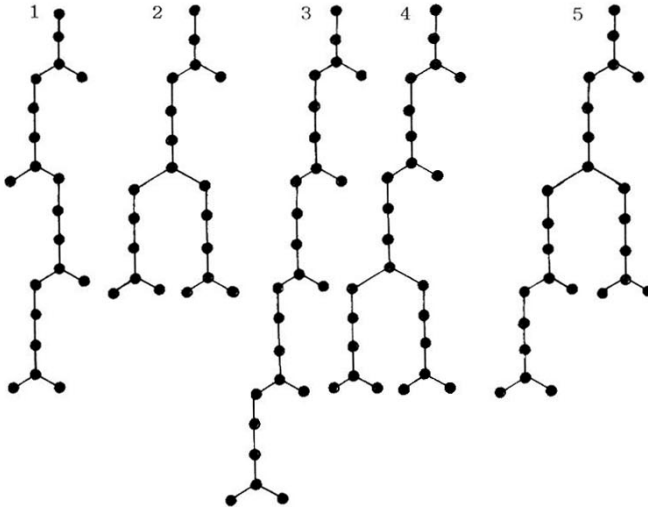


Figure 12

$x_1 = 1101100111111011111011111010000000000000$

$x_2 = 1101100111111111101000001111101000000000$

$x_1, x_2 \in C_{4.10}^{\text{HT}} ; d(x_1, x_2) = 10$



$x_3 = 11011001111110111110111110111110100000000000000000$

$x_4 = 11011001111110111111110100000111110100000000000000$

$x_5 = 11011001111111111010000011111011111010000000000000$

$$x_3, x_4, x_5 \in C_{5.10}^{HT} \quad ; \quad d(x_3, x_4) = 10$$

$$d(x_4, x_5) = 10$$

The argumentation of this observation consists in the fact that the difference between the two structures affects only one from the three tags by means of which we obtain the tag of the essential vertex. Furthermore, in this tag the modification takes place on a compact zone (area), namely in the first part of the tag of the first vertex which appears above the level with the smallest number of order relatively to which we can regard the translation of the isoprene unit. This vertex, denoted by X, appears on one branch which starts from the essential vertex. Let us determine the code of X in two structures, a and b (Fig.13) in conformity with the coding algorithm.

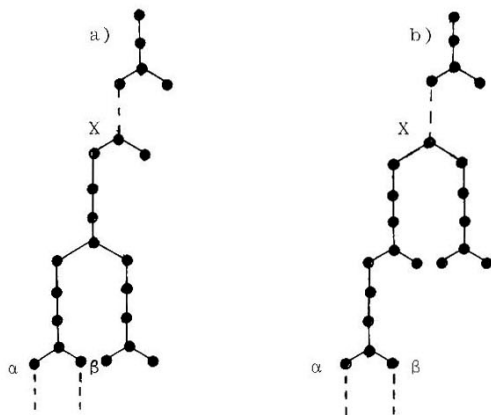


Figure 13

Noting with  $X_a, X_b$  the tag of vertex X of the structures a and respectively b and with  $\alpha$  and  $\beta$  the tags of vertices specified in the figure, we have :

$$X_a = 11011111111101000001111\alpha\beta000000000$$

$$X_b = 1111110100000111101111\alpha\beta000000000$$

We obtain the distance between the tags

$$d(X_a, X_b) = 10$$

which coincides with the distance between the binary representations of the structures a and b .

Structures of type X appear in any set of acyclic HT isoprenoid structures with n isoprene units,  $n \geq 4$ . Thus, any code  $C_{n,10}^{HT}$ ,  $n \geq 4$  contains at the least two words whose distance is 10.

In the same way we can prove that the distance between the binary representations of two structures which differ only by a translation of an isoprene unit upwards or downwards with two levels, the translation taking place on the same branch relatively to the essential vertex, is 12, etc. When the translation of an isoprene unit takes place from a branch to the other, this affects two from those three tags by means of which we formed the tag of the essential vertex and therefore the distance between the binary representations of the structures will be greater.

In conclusion, the minimum distance between two arbitrary words of a  $C_{n,10}^{HT}$  code,  $n \geq 4$  being 10, in conformity with Propositions 2 and 4 it results that any such code detects nine errors and is able to correct four errors. A  $C_{3,10}^{HT}$

code contains two words which constitute the representations of the two structures of Fig.14.

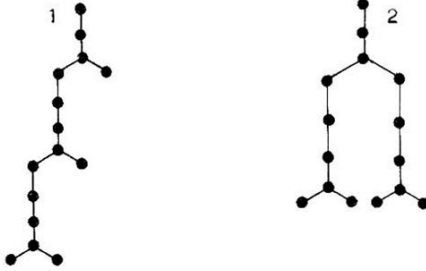


Figure 14

$$x_1 = 110110011111011111010000000000$$

$$x_2 = 111001111101000001111101000000$$

$$d(x_1, x_2) = 14$$

Because the distance between those two words of the code is 14, in conformity with Propositions 2 and 4, it results

that  $C_{3.10}^{HT}$  detects 13 errors and is able to correct 6 errors.

Thus, if we consider the  $C_{(3)}^{HT} = \bigcup_{n \geq 3} C_{n.10}^{HT}$  subset of the  $C^{HT}$

we can say the following :

The code of acyclic HT isoprenoid structures with n isoprene units,  $n \geq 3$  is able to detect 9 errors ( $9 = \min \{9,13\}$ ) and to correct 4 errors ( $4 = \min \{4,6\}$ ).

In the case of acyclic S isoprenoid structures it makes sense to study the problem for codes  $C_{n.10}^S$ ,  $n \geq 2$ , because  $C_{1.10}^S$  contains only one word. Let us have the  $C_{2.10}^S$  code whose words are the binary representations of the structures presented in Figure 15.

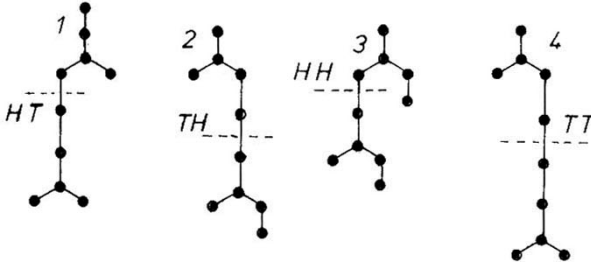


Figure 15

$x_1 = 11011001111101000000$   
 $x_2 = 1101011111101100000000$   
 $x_3 = 1101100111110110000000$   
 $x_4 = 1101011111110100000000$

$d(x_1, x_2) = 6$   
 $d(x_1, x_3) = 2$   
 $d(x_1, x_4) = 6$

$d(x_2, x_3) = 6$   
 $d(x_2, x_4) = 2$   
 $d(x_3, x_4) = 4$

In the case of acyclic  $S$  isoprenoid structures with two isoprene units the distance between the binary representation is minimum when a HH linking is replaced by a HT one or vice-versa and when a TH linking is replaced by a TT one or vice-versa. The same observation is valid also for the distance between the binary representations of the standard structures with  $n$  isoprene units,  $n > 2$ , which differ only by the type of concatenation of a single isoprene unit situated on the last level of the same branch relatively to the essential vertex, namely HH linking instead of HT linking or vice-versa, or TH linking instead of TT linking or vice-versa. Since each the  $S_{n,10}$  codes,  $n \geq 3$  contains at least two words as binary repre-

representations of such structures, it results that the minimum distance between two arbitrary words of any  $C_{n,10}^S$  code,  $n \geq 2$  is 2. Therefore,  $C_{n,10}^S$ ,  $n \geq 2$  is able to detect a unique error, but is not able to correct any error (see Propositions 1 and 3). Thus, if we consider the  $C_{(2)}^S = \bigcup_{n \geq 2} C_{n,10}^S$  subset of the  $C^S$ , we can affirm the following :

The code of acyclic  $S$  isoprenoid structures with  $n$  isoprene units,  $n \geq 2$  is able to detect a unique error.

#### 4, Deterministic pushdown automata

A *pushdown automaton* is a natural model for syntactic analyzers of context-free languages, whose infinite storage consists of one pushdown list. We shall represent a *pushdown list* as a string of symbols with the topmost symbol written either on the left or on the right depending on which convention is most convenient for the situation at hand. From now on, we shall assume that the topmost symbol on the pushdown list is the rightmost symbol of the string representation the pushdown list. A pushdown automaton contains three basic components, illustrated in Figure 16.

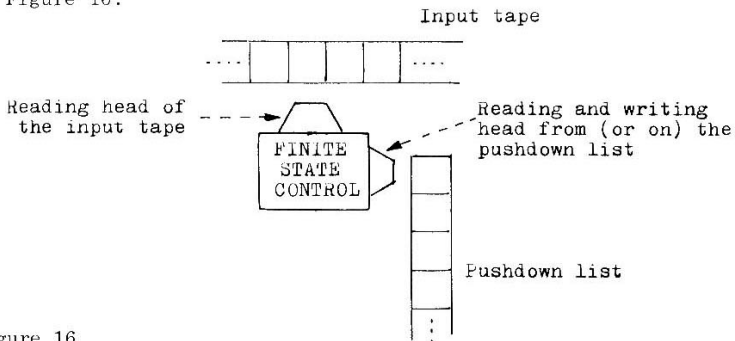


Figure 16

(i) an input tape divided into cells (squares); one square can contain a symbol from an *input alphabet*  $I$ . The number of the filled squares is finite ;

(ii) a *finite state control* which is able to be in a certain state belonging to a finite set of *states*  $S$ . The finite state control has a "reading head" for the input tape and another head for reading and writing the informations from, respectively on, the pushdown list ;

(iii) a *pushdown list* which contains symbols from a *finite alphabet*  $Z$ .

The working of such automata in a certain moment depends on : the symbol situated on the input tape which has been read in the preceding moment ; the state of the finite state control; the topmost symbol of the pushdown list.

The input tape can shift in any moment with one square towards the right passing before the reading head, or can remain stationary. In the last case we consider that the finite state control reads the *null word*, denoted  $\epsilon$ . When a symbol belonging to  $I \cup \{ \epsilon \}$  is read, the finite state control performs two actions :

1. it passes into another state
2. it writes into the pushdown list, replacing the topmost symbol with a word  $r \in Z^*$  (this word can be the null one, which actually means the deletion of the contents of the topmost symbol). Of course, the introduction of a word of length  $k$ ,  $k > 0$  determines

a shifting downwards of the contents of the pushdown list with  $k$  squares.

The pushdown automaton begins its process from an *initial state*  $s_0$ , having  $z_0 \in Z$  as the topmost symbol of the pushdown list. This symbol is the *start symbol*. The recognition of a word (string) situated on the input tape takes place after its last symbol has been read and the pushdown list has become empty.

This notion is formulated more formally as follows :

*Definition*. A deterministic pushdown automaton (DPDA for short) is a 6-tuple

$$P = (I, S, Z, f, s_0, z_0)$$

where

$I$  is a finite *input alphabet* ;

$S$  is a finite set of state symbols representing the possible *states* of the finite state control ;

$Z$  is a finite *alphabet of pushdown list symbols* ;

$f$  is a *mapping* from  $S \times (I \cup \{\epsilon\}) \times Z$  to  $S \times Z^*$  ;

$s_0 \in S$  is the *initial state* of the finite state control ;

$z_0 \in Z$  is the symbol that appears initially on the pushdown list, the *start symbol*.

A *configuration* of  $P$  is a triple  $\delta = (s, p, q) \in S \times I^* \times Z^*$ .

The configuration  $\delta_1 = (s, ip, zq)$ , where  $i \in I \cup \{\epsilon\}$ ,  $z \in Z$ , precedes directly the configuration  $\delta_2 = (s', p, q_1q)$  if and only if,  $f(s, i, z) = (s', q_1)$ . This will be denoted by  $\delta_1 \vdash \delta_2$ .

The configuration  $\delta$  precedes the configuration  $\delta'$ , if and only if there exists a finite string of configurations  $\delta = \delta_0, \delta_1, \dots, \delta_n = \delta'$ , so that  $\delta_i \vdash \delta_{i+1}$ ,  $i=0, \dots, n-1$ . This will be denoted by  $\delta \vdash^* \delta'$ .

*Observations*

1. The configuration  $(s, p, \varepsilon)$  cannot precede any other configuration, because the mapping  $f$  is not defined for such configuration. For this reason, at the beginning in the pushdown list there exists the start symbol  $z_0 \in Z$ .
2. The configuration  $(s, p, q)$  indicates that the automaton is in state  $s$  and that in the following moments the word  $p$ , situated on the input tape, will be introduced and that in the pushdown list there exists the word  $q$  written from bottom upwards, the rightmost symbol appearing in the topmost square.
3. The relation  $\delta_1 \vdash \delta_2$  indicates that the automaton is in state  $s$ , that the topmost symbol of the pushdown list is  $z$ , and that the input symbol which must be introduced is  $i$ . We may have  $i = \varepsilon$  when the single possible evolution leading to  $i$  is to pass in state  $s'$  and to replace  $z$  by the word  $q_1$ .

The language recognized by a deterministic pushdown automaton  $P$ , denoted  $L(P)$ , is defined by

$$L(P) = \{ p \mid \exists s \in S, (s_0, p, z_0) \vdash^* (s, \varepsilon, \varepsilon) \}$$



4.1. *Deterministic pushdown automaton for checking the equality between the number of 1's and 0's of the binary representation of a HT or S acyclic isoprenoid structure*

It was shown in the preceding paper that any binary representation of a HT or S acyclic isoprenoid structure has the number of symbols 1 equal with that of symbols 0.

The following deterministic pushdown automaton is able to check the correctness of any codification from this point of view.

$$P = (I, S, Z, f, f_0, z_0)$$

where  $I = \{0,1\}$  ,  $S = \{s_0\}$  ,  $Z = \{1, z_0\}$  and the mapping  $f$  is given by Table 3

Table 3

$f(s_0, i, z)$	1	$z_0$
0	$(s_0, \epsilon)$	
1	$(s_0, 1)$	$(s_0, 1)$
$\epsilon$		$(s_0, \epsilon)$

The unfilled places indicate that for those configurations, the mapping  $f$  is not defined, namely the  $i$  symbol cannot be applied to the input of the automaton when we have state  $s_0$  and  $z_0$  as the topmost symbol for the pushdown list.

The language recognized by this DPDA is the set of all words (strings) with "0" and "1" symbols and which fulfills the following conditions :

- (i) the leftmost symbol of the string is "1"
- (ii) the number of 1's is equal with the number of 0's.

These represent a part of those conditions which must be fulfilled by any correct binary representation of a HT or S acyclic isoprenoid structure. The rules which determine the working of the DPDA described above, are :

1. Whenever, when going along the input string a "1" symbol is met, it is introduced in the pushdown list.
2. Whenever, when going along the input string a "0" symbol is met, the topmost "1" symbol of the pushdown is deleted.

We can obtain the decoding of the input string, thus :

- we associate to the DPDA defined above, an output vector  $V$ , whose dimension (length) is smaller than the input string with 2

- we associate to each symbol "1" from the input string a distinct label (number, letter or any other symbol)

- whenever we are in the case  $i=0$  and  $z=1$ , before applying the rule  $f(s_0, 0, 1) = (s_0, \epsilon)$  we write down in the output vector  $V$  the labels of those two symbols 1 which appear (if they exist) at the top of the pushdown list. It does not matter in which order these labels are written down in the  $V$  vector. Each of these two labels will determine an edge of the given structure. Because of the structure of the isoprene unit and the admitted types of concatenation, any acyclic HT or S isoprenoid structure with  $n$  isoprene units has  $5n - 1$  edges.

This explains the length  $10n - 2$  chosen for vector V. If the analysis of the input string ends successfully, vector V will contain the edges of the structure, whose coding is the input string. By means of the vector V the structure can be re-constructed.

To make an easier presentation of the following examples we number the situations in which the mapping f is defined, thus :

1.  $f(s_o, 1, z_o) = (s_o, 1z_o)$
2.  $f(s_o, 1, 1) = (s_o, 11)$
3.  $f(s_o, 0, 1) = (s_o, \epsilon)$
4.  $f(s_o, \epsilon, z_o) = (s_o, \epsilon)$

*Examples of tests by means of the constructed DPDA*

1) Let the input string be 11011001111011000000, in which each 1 symbol will be labelled with a letter from the alphabet. The length of the output vector V is 18.

$$\begin{array}{l}
 (s_o, 11011001111011000000, z_o) \xrightarrow{1} (s_o, 1011001111011000000, 1z_o) \xrightarrow{2} \\
 \text{AB CD EFGH IJ} \qquad \qquad \qquad \text{B CD EFGH IJ} \qquad \qquad \qquad \text{A} \\
 \xrightarrow{2} (s_o, 011001111011000000, 11z_o) \xrightarrow{3} (s_o, 11001111011000000, 1z_o) \xrightarrow{2} \\
 \text{CD EFGH IJ} \qquad \text{BA} \qquad \qquad \text{CD EFGH IJ} \qquad \qquad \text{A} \\
 \xrightarrow{2} (s_o, 1001111011000000, 11z_o) \xrightarrow{2} (s_o, 001111011000000, 111z_o) \xrightarrow{3} \\
 \text{D EFGH IJ} \qquad \text{CA} \qquad \qquad \text{EFGH IJ} \qquad \text{DCA} \qquad \text{DC} \\
 \xrightarrow{3} (s_o, 01111011000000, 11z_o) \xrightarrow{3} (s_o, 1111011000000, 1z_o) \xrightarrow{2} \\
 \text{DC} \qquad \text{EFGH IJ} \qquad \text{CA} \qquad \text{CA} \qquad \text{EFGH IJ} \qquad \text{A} \\
 \xrightarrow{2} (s_o, 111011000000, 11z_o) \xrightarrow{2} (s_o, 11011000000, 111z_o) \xrightarrow{2} \\
 \text{FGH IJ} \qquad \text{EA} \qquad \qquad \text{GH IJ} \qquad \text{FEA}
 \end{array}$$

$$\begin{array}{l}
 \begin{array}{c} \overline{2} \\ \text{H IJ} \end{array} (s_o, 1011000000, 1111z_o) \begin{array}{c} \overline{2} \\ \text{GFEA} \end{array} (s_o, 011000000, 11111z_o) \begin{array}{c} \overline{3} \\ \text{IJ HGFEA HG} \end{array} \\
 \begin{array}{c} \overline{3} \\ \text{HG} \end{array} (s_o, 11000000, 1111z_o) \begin{array}{c} \overline{2} \\ \text{IJ GFEA} \end{array} (s_o, 1000000, 11111z_o) \begin{array}{c} \overline{2} \\ \text{J IGFEA} \end{array} \\
 \begin{array}{c} \overline{2} \\ \text{JIGFEA} \end{array} (s_o, 000000, 111111z_o) \begin{array}{c} \overline{3} \\ \text{JI} \end{array} (s_o, 00000, 11111z_o) \begin{array}{c} \overline{3} \\ \text{IGFEA IG} \end{array} \\
 \begin{array}{c} \overline{3} \\ \text{IG} \end{array} (s_o, 0000, 1111z_o) \begin{array}{c} \overline{3} \\ \text{GFEA GF} \end{array} (s_o, 000, 111z_o) \begin{array}{c} \overline{3} \\ \text{FEA FE} \end{array} \\
 \begin{array}{c} \overline{3} \\ \text{FE} \end{array} (s_o, 00, 11z_o) \begin{array}{c} \overline{3} \\ \text{EA} \end{array} (s_o, 0, 1z_o) \begin{array}{c} \overline{3} \\ \text{EA} \end{array} (s_o, \epsilon, z_o) \begin{array}{c} \overline{4} \\ \text{A} \end{array} \\
 \begin{array}{c} \overline{4} \\ \text{A} \end{array} (s_o, \epsilon, \epsilon)
 \end{array}$$

Hence the input string is accepted (recognized) by the DPDA. When under the sign  $\overline{3}$  there appear two labels, this means that before the execution of the rule numbered 3, these labels will be put in the output vector V from left to right. Labels do not appear under the sign  $\overline{3}$  when in the pushdown list there exists a single 1 symbol (a situation met always before the last but one stage, if the input string is accepted by the DPDA) whose label is associated to the essential vertex of the structure. In our case, the output vector V has the following content

B	A	D	C	C	A	H	G	J	I	I	G	G	F	F	E	E	A
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

and the label of the essential vertex is A.

For reconstructing the structure, we take pairs of vertices from left to right (or vice-versa) of the output vector V. Each element of a pair will represent an edge of the structure.

Thus, we have the structure of Fig.17.

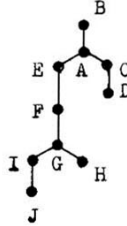


Figure 17

formed of two isoprene units with HH linking, the concatenation taking place between the vertices E and F (the labels associated to the vertices are only for reference, they are not specific for the structure).

2) Let the input string be 11011001111111000000.

$$\begin{aligned}
 & (s_o, 11011001111111000000, z_o) \stackrel{1}{\vdash} (s_o, 1011001111111000000, 1z_o) \stackrel{2}{\vdash} \\
 & \quad \quad \quad \text{AB CD EFGHIJK} \quad \quad \quad \text{B CD EFGHIJK} \quad \quad \quad \text{A} \\
 & \stackrel{2}{\vdash} (s_o, 011001111111000000, 11z_o) \stackrel{3}{\vdash} (s_o, 11001111111000000, 1z_o) \stackrel{2}{\vdash} \\
 & \quad \quad \quad \text{CD EFGHIJK} \quad \quad \quad \text{BA BA CD EFGHIJK} \quad \quad \quad \text{A} \\
 & \stackrel{2}{\vdash} (s_o, 1001111111000000, 11z_o) \stackrel{2}{\vdash} (s_o, 001111111000000, 111z_o) \stackrel{3}{\vdash} \\
 & \quad \quad \quad \text{D EFGHIJK} \quad \quad \quad \text{CA} \quad \quad \quad \text{EFGHIJK} \quad \quad \quad \text{DCA} \quad \quad \quad \text{DC} \\
 & \stackrel{3}{\vdash} (s_o, 01111111000000, 11z_o) \stackrel{3}{\vdash} (s_o, 1111111000000, 1z_o) \stackrel{2}{\vdash} \\
 & \quad \quad \quad \text{DC EFGHIJK} \quad \quad \quad \text{CA CA EFGHIJK} \quad \quad \quad \text{A} \\
 & \stackrel{2}{\vdash} (s_o, 111111000000, 11z_o) \stackrel{2}{\vdash} (s_o, 11111000000, 111z_o) \stackrel{2}{\vdash} \\
 & \quad \quad \quad \text{FGHIJK} \quad \quad \quad \text{EA} \quad \quad \quad \text{GHIJK} \quad \quad \quad \text{FEA} \\
 & \stackrel{2}{\vdash} (s_o, 1111000000, 1111z_o) \stackrel{2}{\vdash} (s_o, 111000000, 11111z_o) \stackrel{2}{\vdash} \\
 & \quad \quad \quad \text{HIJK} \quad \quad \quad \text{GFEA} \quad \quad \quad \text{IJK} \quad \quad \quad \text{HGFEA} \\
 & \stackrel{2}{\vdash} (s_o, 11000000, 111111z_o) \stackrel{2}{\vdash} (s_o, 1000000, 1111111z_o) \stackrel{2}{\vdash} \\
 & \quad \quad \quad \text{JK} \quad \quad \quad \text{IEGF EA} \quad \quad \quad \text{K} \quad \quad \quad \text{JIHGFEA}
 \end{aligned}$$

$$\begin{array}{c}
 \frac{2}{\text{JI}} (s_0, 000000, 1111111z_0) \frac{3}{\text{KJ}} (s_0, 000000, 1111111z_0) \frac{3}{\text{JI}} \\
 \text{KJIHGFEA} \quad \text{KJ} \quad \text{JIHGFEA} \quad \text{JI} \\
 \\
 \frac{3}{\text{JI}} (s_0, 0000, 111111z_0) \frac{3}{\text{IH}} (s_0, 000, 11111z_0) \frac{3}{\text{HG}} \\
 \text{IHGFEA} \quad \text{IH} \quad \text{HGFEA} \quad \text{HG} \\
 \\
 \frac{3}{\text{HG}} (s_0, 00, 1111z_0) \frac{3}{\text{GF}} (s_0, 0, 111z_0) \frac{3}{\text{EA}} (s_0, \epsilon, 11z_0) \\
 \text{HG} \quad \text{GFEA} \quad \text{GF} \quad \text{FEA} \quad \text{FE} \quad \text{EA}
 \end{array}$$

Hence the input string is not accepted by the DPDA.

Thus, the information recorded in the output vector V would not be taken into consideration.

*Comments*

As the  $C^S$  code of the acyclic standard (S) isoprenoid structures is a subset of  $L(P)$ , the language recognized by the constructed DPDA, there exist strings accepted by the automaton which however are not binary representations of the required type. Thus, given a string of symbols from the set  $\{0,1\}$ , as the binary representation of a structure, we check that this corresponds to an acyclic S isoprenoid structure thus :

- (i) check if the binary representation is accepted by the constructed DPDA ;
- (ii) reconstruct the structure corresponding to the given string using the decoding algorithm or the output vector V ;
- (iii) relabel the obtained structure according to the algorithm of labelling<sup>1</sup> and start to analyse the structure using the reduction rules of the web grammar  $G_W^S$ . If the analysis ends successfully,

then the given binary representation belongs to the  $C^S$  code. Otherwise, the given string is not a binary representation of a structure of the required type.

## References

- <sup>1</sup> A.T.Balaban, M.Barasch and S.Marcus, *Math.Chem.*, 8, 193(1980).
- <sup>2</sup> L.N.Kanal, W.E.Underwood, E.I.Burkart, G.S.Mancill and K.Rankin, "Grammars and Translators for Chemical Structure Formulas, Line Formulas and Nomenclature", Computer Science Technical Report Series, TR 787, Univ.of Maryland, July 1979.
- <sup>3</sup> W.J.Wiswesser, "A Line-Formula Chemical Notation", Crowell, New York, 1954 ; E.G.Smith and P.A.Baker, "The Wiswesser Line-Formula Chemical Notation (WLN)", 3rd Ed.: Chemical Information Management ; Cherry Hill, New Jersey, 1976 .
- <sup>4</sup> H.L.Morgan, *J.Chem.Docum.* 5, 107 (1965) ; A.L.Goodson, *J.Chem.Inf.Comput.Sci.*, 20, 167 (1980) .
- <sup>5</sup> A.T.Balaban, O.Mekenyan and D.Bonchev , to appear .
- <sup>6</sup> R.C.Read, "The Coding of Trees and Tree-like Graphs", preprint, University of the West Indies, Jamaica, 1969 .
- <sup>7</sup> A.T.Balaban, M.Barasch and S.Marcus, *Math.Chem.*, 5, 239 (1979) ; 8, 215 (1980) .
- <sup>8</sup> S.Marcus, E.Nicolau and S.Stati, "Introduzione alla linguistica matematica", Casa editrice Ricardo Patron, Bologna, 1971.