How programming was used to determine factorizations of self-similar groups associated to the first Grigorchuk group

Beata BAJORSKA-HARAPIŃSKA

Abstract. One of the problems we have encountered in [1] was to determine all generators of a group needed for the proof of one of the main theorems. Although the final proof of this theorem is pure algebraic it would be hard to discover without tons of calculations which we decided to perform using *Mathematica*. The generators of the group in question can be treated as vertices of an infinite tree, however some vertices correspond to the same generator. Thus the point was to find the smallest subset of vertices of the tree corresponding to all (distinct) generators of the group. We have used various tree-search methods (some crucial observations allowed to choose a proper one) and eventually it turned out that all vertices we need could be taken from the single ray of the tree. In this paper we present the way of finding this ray.

Keywords: infinite iterated wreath products, the first Grigorchuk group, Gray code.

2010 Mathematics Subject Classification: 20-04.

1. Introduction and the background of the problem

We first formulate the problem which was solved with a little help of computers. All the definitions will be given only in the case we considered. For the general case and some properties see the papers we cite. All the other results are taken from [1].

By [3] every element in the group $\wr_{i=1}^{\infty} \mathbb{Z}_2$, that is the infinite iterated wreath product of an infinite number of copies of $\mathbb{Z}_2$, can be uniquely presented as an infinite sequence, called an **array**, namely

$$g = [g_1, g_2(x_1), g_3(x_1, x_2), \ldots], \tag{1}$$

B. Bajorska-Harapińska
Institute of Mathematics, Silesian University of Technology, Kaszubska 23, 44-100 Gliwice, Poland,
e-mail: beata.bajorska@polsl.pl

where $g_1 \in \mathbb{Z}_2$ and for $n \geqslant 2$, $g_n \in P_n := \mathbb{Z}_2[x_1, \ldots, x_{n-1}]/\langle x_1^2 - x_1, \ldots, x_{n-1}^2 - x_{n-1}\rangle$, that is g_n is a polynomial that can be written as a sum of distinct monomials of the form $x_{i_1} \cdots x_{i_k}$ (note that indices are distinct, that is $i_j \neq i_s$ for $j \neq s$).

The product of $g = [g_1, g_2(x_1), \ldots]$ and $h = [h_1, h_2(x_1), \ldots]$ is given by the rule

$$gh = [g_1 + h_1, g_2(x_1) + h_2(x_1 + g_1), g_3(x_1, x_2) + h_3(x_1 + g_1, x_2 + g_2(x_1)), \ldots], \quad (2)$$

with addition on n-th component performed in P_n, meaning that we first add polynomials in the usual way, then reduce all the powers of variables to the first one (that is we take $x_i^p = x_i$ for each natural p and $i \in \{1, \ldots, n-1\}$) and at last we simplify the coefficients modulo 2. The identity element in the group $\wr_{i=1}^{\infty} \mathbb{Z}_2$ is

$$1 := [0, 0, 0, \ldots]. \quad (3)$$

For example, let $g = h = [0, x_1 + 1, x_1(x_2 + 1), \ldots, x_1 \cdots x_n(x_{n+1} + 1), \ldots]$. Then from formula (2) we get that

– the first component of gh is $0 + 0 = 0$,

– the second component is $(x_1 + 1) + ((x_1 + 0) + 1) = 0$, since $2x_1 + 2 \pmod 2 = 0$,

– the third is $x_1(x_2+1) + (x_1+0)((x_2+x_1+1)+1) = 0$, as we have $2x_1x_2 + x_1^2 + 3x_1 \to 2x_1x_2 + x_1 + 3x_1 = 2x_1x_2 + 4x_1 \pmod 2 = 0$,

– from formula (10) below we conclude that each component of gh is 0,

which means that gh actually equals the identity element defined by (3).

We have used the following general recursive construction [4]:

1. *Base cases*: the set Z_0 and the group

$$H_0 := \langle Z_0 \rangle = \{k_1^{\varepsilon_1} \cdots k_n^{\varepsilon_1},\ k_i \in Z_0,\ \varepsilon_i \in \{\pm 1\},\ n \in \mathbb{N}\}, \quad (4)$$

2. *Recursion rules*: for every natural n

$$Z_n := \{z^{(h)},\ z \in Z_0,\ h \in H_{n-1}\}, \qquad H_n := \langle Z_n \rangle, \quad (5)$$

where for every $z = [z_1, z_2(x_1), z_3(x_1, x_2), \ldots]$, $h = [h_1, h_2(x_1), h_3(x_1, x_2), \ldots]$ we have

$$\begin{aligned} z^{(h)} := [&z_1,\ z_2(x_1 + h_1),\ z_3(x_1 + h_1, x_2 + h_2(x_1)), \\ &z_4(x_1 + h_1, x_2 + h_2(x_1), x_3 + h_3(x_1, x_2)), \ldots]. \end{aligned} \quad (6)$$

The set Z_0 in our case was made from the generators of so-called the first Grigorchuk group [2], which are denoted by a, b, c, d (d is actually not necessarily needed, but convenient). In [1] we established the representations of these elements in the form (1), namely

$$a = [1, 0, 0, \ldots] \quad (7)$$

and arrays for b, c, d are constructed as follows:

1. the first component is 0,
2. for $n \geqslant 2$ the n-th component is either 0 or of the form

$$\alpha_n(x_1, \ldots, x_{n-1}) := x_1 x_2 \cdots x_{n-2}(x_{n-1} + 1), \text{ for } n \geqslant 3, \quad \alpha_2(x_1) := x_1 + 1 \quad (8)$$

and 0 appears every three components starting from 4th for b, 3rd for c and 2nd for d, that is (arguments of α_i are omitted for short)

$$\begin{aligned} b &= [0, \alpha_2, \alpha_3, 0, \alpha_5, \alpha_6, 0, \alpha_8, \alpha_9, 0, \alpha_{11}, \alpha_{12}, \ldots], \\ c &= [0, \alpha_2, 0, \alpha_4, \alpha_5, 0, \alpha_7, \alpha_8, 0, \alpha_{10}, \alpha_{11}, 0, \ldots], \\ d &= [0, 0, \alpha_3, \alpha_4, 0, \alpha_6, \alpha_7, 0, \alpha_9, \alpha_{10}, 0, \alpha_{12}, \ldots]. \end{aligned} \tag{9}$$

Moreover, polynomials α_n have the following property: for $n \geqslant 3, \varepsilon_i \in \{0,1\}$, $i = 2, ..., n-1$ and for every $(x_1, \ldots, x_{n-1}) \in \mathbb{Z}_2^{n-1}$ we have

$$\alpha_n(x_1, x_2 + \varepsilon_2\alpha_2(x_1), \ldots, x_{n-1} + \varepsilon_{n-1}\alpha_{n-1}(x_1, \ldots, x_{n-2})) = \alpha_n(x_1, \ldots, x_{n-1}) \tag{10}$$

This was the first crucial observation. It means that the result of the action of h on z defined by (6) is an array which probably has not a very complicated structure so there was a hope to discover and describe it in a sensible way.

Now, to define Z_0, instead of each generator g of G given by (7) and (9) we take two: g_e (with even components the same as in g and 0 elsewhere) and g_o (with odd components the same as in g and 0 elsewhere). For a nothing interesting happens as $a_e = [0, 0, 0, \ldots]$ is the identity element by (3) (not important while generating a group) and $a_o = a$. For elements defined in (9) it follows that zero and nonzero components appear in cycles of 6 (except for the first component, which is always 0) and hence we have

$$\begin{aligned} b_e =& [0, \alpha_2, 0, 0, 0, \alpha_6, 0, \alpha_8, 0, 0, 0, \alpha_{12}, 0, \ldots], \\ b_o =& [0, 0, \alpha_3, 0, \alpha_5, 0, 0, 0, \alpha_9, 0, \alpha_{11}, 0, 0, \ldots], \\ c_e =& [0, \alpha_2, 0, \alpha_4, 0, 0, 0, \alpha_8, 0, \alpha_{10}, 0, 0, 0, \ldots], \\ c_o =& [0, 0, 0, 0, \alpha_5, 0, \alpha_7, 0, 0, 0, \alpha_{11}, 0, \alpha_{13}, \ldots], \\ d_e =& [0, 0, 0, \alpha_4.0, \alpha_6, 0, 0, 0, \alpha_{10}, 0, \alpha_{12}, 0, \ldots], \\ d_o =& [0, 0, \alpha_3, 0, 0, 0, \alpha_7, 0, \alpha_9, 0, 0, 0, \alpha_{13}, \ldots]. \end{aligned} \tag{11}$$

Since from (2), (10) and (11) we get $d_o = b_o c_o$, $d_e = b_e c_e$, elements d_o, d_e are not necessarily needed in Z_0 (because they can be obtained from other elements), so

$$Z_0 := \{a, b_o, b_e, c_o, c_e\}. \tag{12}$$

The form of elements in the group H_0 given by (4) which was more convenient for further considerations was obtained due to the algebraic properties – it turned out that each element of H_0 can be written in the form

$$t_1 a t_2 \cdots a t_n, \tag{13}$$

(without possibly t_1 or t_n) where each t_i, $i = 1, \ldots n$ is an element in the set

$$T := \{b_o, c_o, d_o, b_e, c_e, d_e, b_o b_e, b_o c_e, b_o d_e, c_o b_e, c_o c_e, c_o d_e, d_o b_e, d_o c_e, d_o d_e\}. \tag{14}$$

In view of formula (2) and (10), the nonzero components of each element in T of the form $q_o w_e$ are exactly nonzero components of q_o and w_e, e.g. from (11) we have

$$\begin{aligned} b_o c_e &= [0, \alpha_2, \alpha_3, \alpha_4, \alpha_5, 0, 0, \alpha_8, \alpha_9, \alpha_{10}, \alpha_{11}, 0, \ldots], \\ c_0 b_e &= [0, \alpha_2, 0, 0, \alpha_5, \alpha_6, \alpha_7, \alpha_8, 0, 0, \alpha_{11}, \alpha_{12}, \ldots]. \end{aligned} \tag{15}$$

To discover some properties of H_1 we needed the set of generators Z_1 (see formulas (5)). The form of elements in Z_1 defined in (6) was not obvious, however. Because of formula (16) (given in the next section) there is a correspondence between Z_1 and the rooted tree of words X^* on 16-element alphabet $X = T \cup \{a\}$ – more precisely, for every element $z \in Z_0$ (defined by (12)) an element of the form $z^{(q_s q_{s-1} \cdots q_1)} \in Z_1$, $q_i \in T \cup \{a\}$ corresponds to the vertex $q_1 q_2 \cdots q_s$ of the tree X^*. However, it follows from definitions and the other observations that this is not 1-1 correspondence – different vertices can correspond to the same element, e.g. aaa and a correspond to the same element since by formula (2) we have $aa = 1$. So our aim was to find the smallest set of vertices of the tree X^* providing all elements in Z_1. The idea to discover it was to use a *brute force* method, namely to evaluate enough elements of the form (6) to see some regularities. We tried to do it by hand at first. But although we used functions instead of polynomials and observed some general properties which made the calculations shorter, evaluating all elements of the form $z^{(at_1at_2at_3)}$ still did not give enough regularity. So it seemed to be too much work to do this way.

Fortunately, because of the regularity in the form of arrays in question we were able to bypass the infinity in definition (6) and therefore the programming was possible.

2. The solution

Step 0. Basic algebraic observations which make the calculation of $z^{(h)}$ defined by (6) as short as possible (for the proofs see [1]):

Obs.1 *Each component of every element in Z_1 is either 0 or a polynomial having value 1 at exactly one point (whence a monomial). More precisely, for every $z \in Z_0$, $h \in H_0$ the n-th component of $z^{(h)}$ is*

1° *0 if and only if the n-th component of z is 0,*

2° *of the form $\alpha_n(x_1 + y_1, \ldots, x_{n-1} + y_{n-1})$, $y_i \in \{0, 1\}$ (where α_i is defined by (8)) otherwise.*

That was the crucial observation which allowed to bypass the infinity. Namely, instead of examining the whole array it was enough to consider only one (nonzero) component, n-th say, with n large enough to catch the regularities.

Obs.2 *For every $z \in Z_0$ and every $g, h \in H_0$ we have*

$$\left(z^{(gh)}\right) = \left(z^{(h)}\right)^{(g)}. \tag{16}$$

That was another crucial observation as the calculations became recursive.

Obs.3 *For every $h \in H_0$ we have* $\quad a^{(h)} = a.$

That means that we do not have to perform the calculations for a.

Obs.4 *Elements $z \in Z_1$ and $z^{(a)} \in Z_1$ differ only by the factor containing x_1. More precisely, if in z we have the factor $x_1 + y_1$, then in $z^{(a)}$ we have $x_1 + 1 - y_1$ for $y_1 \in \{0, 1\}$. Moreover, actions on $z \in Z_1$ by elements in T do not change x_1.*

That means that we know how the results of actions by a look like.

Obs.5 *For every $z \in Z_0$ and every $t \in T$ we have* $\quad z^{(t)} = z$.

That means that the first nontrivial action is by a.

Summarizing: each element (array) in Z_1 (except for $a^{(h)}$, which always equals a) is of the form

$$[0, \varepsilon_2\alpha_2(x_1+y_1), \varepsilon_3\alpha_3(x_1+y_1, x_2+y_2), \varepsilon_4\alpha_4(x_1+y_1, x_2+y_2, x_3+y_3), ...] \tag{17}$$

with $\varepsilon_i, y_i \in \{0,1\}$. So its n-th component is either 0 or $\alpha_n(x_1+y_1, ..., x_{n-1}+y_{n-1})$. Therefore instead of examining the whole array it is enough to consider its n-th component (assuming nonzero), let's call it k, for n large enough and examine the result of the action defined by (6) by a and some $t \in T$ performed on k recursively by turns. Our aim was to discover what are the possible values for y_i and how to obtain the element in Z_1 defined by a given sequence (y_i) using a single element in H_0.

Step 1. Basics: the function which allows to evaluate $(s+1)$-th (assuming nonzero) component of an element in Z_1 of the form $z^{(h)}$, where s is the length of h.

Procedure 1. ***Input:*** *An array $h = [h_1, h_2(x_1), \ldots, h_s(x_1, \ldots, x_{s-1})]$ of the form (1) (truncated to s components) and a polynomial $k(x_1, \ldots, x_s)$ over $\mathbb{Z}_2$ in variables $x_1, \ldots, x_s$ (being $s+1$-th nonzero component of each generator in Z_1).*

Output: *A polynomial being $(s+1)$-th component in the array of the form (6), that is $k(x_1+h_1, x_2+h_2(x_1), \ldots, x_s+h_s(x_1, \ldots, x_{s-1}))$.*

Step 1. *For each $i = 1, 2, \ldots, s$ in k substitute $x_i \rightarrow x_i + h_i(x_1, \ldots, x_{i-1})$,*

Step 2. *For each $i = 1, 2, \ldots, s$ and every $p \geqslant 2$ in the result of Step 1. substitute $x_i^p \rightarrow x_i$,*

Step 3. *Simplify the polynomial resulted from Step 2. and reduce its coefficients modulo 2.*

Note that actually there is no lower bound on the number of component to be evaluated, so it is possible to use this procedure also for q-th component of an element in Z_1 for every $q \leqslant s$.

Step 2. Breadth-first search: we wanted to check what are the possible forms of a fixed (nonzero) component of elements in Z_1. Note that by (17), if the s-th component is nonzero, it is of the form $k = \alpha_s(x_1+y_1, \ldots, x_{s-1}+y_{s-1})$ whence we can get maximally 2^{s-1} distinct monomials on s-th component. Moreover, by Obs.3, Obs.5 and formula (13) to get every possible element in Z_1 it was enough to consider only actions by elements of H_0 in the form $t_i a t_{i-1} a \cdots t_2 a t_1$ or $a t_i a t_{i-1} \cdots a t_1$, $t_s \in T$. Finally, by Obs.2 the action is recursive. The first action on k, by t_1, is trivial (meaning we get k as a result), then we act on k by a (the result is given in Obs.3, so there is still no need to use Procedure 1), then on $k^{(a)}$ we act by t_2, then on $(k^{(a)})^{(t_2)}$ by a again and so on up to t_n or a. So we denote

$$k_n := \begin{cases} k & n = 1, \\ k_{n-1}^{(a)} & n \geqslant 2 \text{ and even}, \\ k_{n-1}^{(t_{(n+1)/2})} & n \geqslant 3 \text{ and odd}. \end{cases} \tag{18}$$

The procedure giving all the necessary information is the following:

Procedure 2. ***Input:*** *The polynomial $k = x_1x_2\cdots x_{s-1}(x_s+1)$, the list T given by (14) and the array a (all arrays truncated to s components), and the maximal length n of elements in H_0 we act by.*

Output: *For each $i = 1, 2, \dots, n$ the list of all possible k_i given by formula (18) and all distinct k_i together with the number and form of all distinct elements obtained up to the moment, that is k_q, $q \leqslant i$.*

```
Step 1. Set $\mathbf{k_1} := k$, $\mathbf{listGen_1} := \{k\}$,
Step 2. For each $i = 2, 3, \dots, n$ do the following
    1° Calculate all $\mathbf{k_i}$ using Procedure 1 on every element of $\mathbf{listGen_{i-1}}$,
    2° Determine the list $\mathbf{newGen_i}$ of all $\mathbf{k_i}$ which are not in $\mathbf{listGen_{i-1}}$,
    3° Define the list $\mathbf{listGen_i} = \mathbf{listGen_{i-1}} \cup \mathbf{newGen_i}$,
Step 3. For each $i = 1, 2, \dots, n$ display $\mathbf{listGen_i}$ and $\mathbf{newGen_i}$ together with the
        number of their elements.
```

For the beginning we decided to carry on calculations for $s = 4$ just to check how it works. We had called Procedure 2 increasing n up to 16 and observing that we got exactly one new element each time we increased n by 1. Then we have increased s and eventually we decided to carry on the calculations for $s = 13$ (because of the earlier attempts this looked pretty promising to be large enough). We have evaluated elements this way up to $n = 24$ using Procedure 2. Again, we got exactly one new element each time (i.e. for each i the list *newGen*$_i$ contains one element only). Moreover, studying the list of all k_i we observed that it is possible to get this result using a and c_ob_e only.

Step 3. Depth-first search: We wanted to confirm the last observation so we evaluated the sequence of generators in Z_1 recursively acting on k by a and $t = c_ob_e$ taken in turns (i.e. treated as loops), using the following

Procedure 3. ***Input:*** *The polynomial $k = x_1x_2\cdots x_{s-1}(x_s+1)$, the arrays $t = c_ob_e$ (given by (15)) and a (both truncated to s components), and the number n standing for the half of the maximal length of elements in H_0 we act by (i.e. the number of loops).*

Output: *The list of all $\mathbf{k_i}$, $i \leqslant 2n$ and the number of distinct $\mathbf{k_i}$ obtained from k by acting by a and t only.*

```
Step 1. Set $\mathbf{gen_1} := k$, $\mathbf{listGen_1} := \{k\}$,
Step 2. For each $i = 2, 3, \dots, 2n$ do the following
    1° Calculate $\mathbf{gen_i} = k_i$ using $t$ for odd $i$ and $a$ for even $i$,
    2° Determine the list $\mathbf{listGen_i}$ of all distinct $\mathbf{gen_q}$, $q \leqslant i$,
Step 3. Display the list $\{\mathbf{gen_i},\ i = 1, 2, \dots, 2n\}$ and the number of elements on
        $\mathbf{listGen_{2n}}$.
```

We have used the procedure increasing n successively. Unfortunately, it turned out that only the first 32 elements out of 34 resulted from Procedure 3 for $n = 17$ are distinct. So we needed another element $u \in T$ giving new elements in Z_1. Examining the existing generators and the element t we started to suspect that the problem might lay in the fact that 6-th component of t is 0, which prevents the factor x_6 in k from changing. Since also 7-th component of t is 0, the same situation might have took place in the future. We chose $u = b_oc_e \in T$ as its 6-th and 7-th components are nonzero. Evaluating k_{33} from k_{32} using u gave a new element, fortunately.

Therefore we extended the previous procedure to the following:

Procedure 4. ***Input:*** *The polynomial* $k = x_1x_2 \cdots x_{s-1}(x_s+1)$, *the arrays* $a, t = c_ob_e$ *and* $u = b_oc_e$ *(all truncated to* s *components) and the number* n *standing for the half of the maximal length of elements in* H_0 *we act by (=the number of loops).*

Output: *The list of all* $\mathbf{k_i}$, $i \leqslant 2n$ *and the number of distinct* $\mathbf{k_i}$ *obtained from* k *by acting by* a *and* t *only except for* k_{33}, *which we obtain using* u.

```
Step 1. Set $\mathbf{gen_1} := k$, $\mathbf{listGen_1} := \{k\}$,
Step 2. For each $i = 2, 3, \ldots, 2n$ do the following
    1° Calculate the element $\mathbf{gen_i} = k_i$ using the array $a$ for all even $i$ and the
       array $t$ for all odd $i$ except for 33; for $i = 33$ (if appears) use $u$,
    2° Determine the list $\mathbf{listGen_i}$ of all distinct $\mathbf{gen_q}$, $q \leqslant i$,
Step 3. Display the list $\{\mathbf{gen_i},\ i = 1, \ldots, 2n\}$ and the number of elements on
        $\mathbf{listGen_{2n}}$.
```

As we expected, Procedure 4 gave distinct elements (1 at each recursive step) up to 64-th. Thus we got all possible elements which could have been obtained by changing only the first 6 factors in k, that is $x_1, \ldots, x_6$. Then u was needed again to change x_7.

Step 4. Choosing the ray: We have suspected that a problem would occur when would there be time to change x_9 (which was expected to take place after evaluation all elements obtained by changing the first 8 factors in k, i.e. for element number $2^7 + 1 = 257$), as 9-th component of u is 0. But fortunately 9-th component of t is nonzero, so t might have been used instead of u in that case. To check our suspicion we used a new procedure evaluating elements k_i in cycles – firstly, using a and t alternatively 15 times (i.e. making 15 loops), then a and u (1 loop) then again a, t alternatively 15 times (15 loops). In the proper moment we have used t instead of u (which made it 31 loops of a, t in a row) and again a and u was used. Note that it was enough to determine how many times we have to use t and u in a row, so we **eventually** defined the following list

```
seq={
 {t, 15}, {u, 1}, {t, 15}, {u, 1}, {t, 15}, {u, 1}, {t, 15}, {u, 1},
 {t, 15}, {u, 1}, {t, 15}, {u, 1}, {t, 15}, {u, 1}, {t, 31}, {u, 1},
 {t, 15}, {u, 1}, {t, 15}, {u, 1}, {t, 15}, {u, 1}, {t, 15}, {u, 1},
 {t, 15}, {u, 1}, {t, 15}, {u, 1}, {t, 31}, {u, 1}, {t, 15}, {u, 1},
 {t, 15}, {u, 1}, {t, 15}, {u, 1}, {t, 15}, {u, 1}, {t, 15}, {u, 1},
 {t, 15}, {u, 1}, {t, 31}, {u, 1}, {t, 15}, {u, 1}, {t, 15}, {u, 1},
 {t, 15}, {u, 1}, {t, 15}, {u, 1}, {t, 15}, {u, 1}, {t, 15}, {u, 1},
 {t, 15}, {u, 1}, {t, 15}
}
```

where {`t,15`} means that we use a and t alternatively 15 times in a row on k. This allowed to evaluate all k_i up to k_{1057} constructed in the above described manner. Of course the list `seq` could have been defined by some procedure, note however that we did not start with such a long list (in case we were totally wrong) but we have extended it successively (adding 3-4 elements) up to the above form.

The procedure evaluating k_i chosen in the manner determined by the list `seq` and confirming our suspicion on the method of creating the list of k_i is the following

Procedure 5. ***Input:*** *The list* **seq**, *the polynomial* $k = x_1 x_2 \cdots x_{s-1}(x_s + 1)$, *the arrays* a, $t = c_o b_e$ *and* $u = b_o c_e$ *(all truncated to* s *components).*

Output: *the list of* $\mathbf{k_i}$ *obtained from* k *by using* a *and elements defined by* **seq** *alternatively.*

Step 1. *Using the list* **seq** *define the list* **seqEx** *which is the explicit list of elements we act by.*
Step 2. *Evaluate* $\mathbf{k_i}$ *using only elements from* **seqEx**.
Step 3. *Display the length of* **seqEx** *and the number of distinct* $\mathbf{k_i}$ *obtained by* **seqEx**.

As we suspected, both numbers given as a result of Procedure 5 were the same which means that we have found the method of generating elements in Z_1.

Step 5. Getting the proof: examining the list resulting from Procedure 5 we have found some regularities. Recall that each k_i on this list is of the form

$$(x_1 + y_1)(x_2 + y_2) \cdots (x_{12} + y_{12})(x_{13} + 1 - y_{13}), \quad y_j \in \{0, 1\}$$

(as all the calculations starting from Step 2. were performed for $s = 13$). As we were trying to observe the general rules for creating the list of k_i we first made the following general observation on the rule of changing the first 5 factors of k_i. Namely, they were changing in some order up to 32-th element (giving all the possibilities) and then in a reversed one. Then again in the original order and again in a reversed one and so on. So as to the other factors of k_i, they remained unchanged for each consecutive 31 elements and changed each 32 elements. For instance, we have obtained the following elements from Procedure 5

```
1) x[1]x[2]x[3]x[4]x[5]x[6]x[7]x[8]x[9]x[10]x[11]x[12](1+x[13])
...
32) x[1]x[2]x[3]x[4](1+x[5])x[6]x[7]x[8]x[9]x[10]x[11]x[12](1+x[13])
33) x[1]x[2]x[3]x[4](1+x[5])(1+x[6])x[7]x[8]x[9]x[10]x[11]x[12](1+x[13])
...
64) x[1]x[2]x[3]x[4]x[5](1+x[6])x[7]x[8]x[9]x[10]x[11]x[12](1+x[13])
65) x[1]x[2]x[3]x[4]x[5](1+x[6])(1+x[7])x[8]x[9]x[10]x[11]x[12](1+x[13])
...
96) x[1]x[2]x[3]x[4](1+x[5])(1+x[6])(1+x[7])x[8]x[9]x[10]x[11]x[12](1+x[13])
97) x[1]x[2]x[3]x[4](1+x[5])x[6](1+x[7])x[8]x[9]x[10]x[11]x[12](1+x[13])
```

To see further regularities connected with the remaining factors in k_i we prepared a table (given below) in which we encoded information on the form of k_i. We have divided the sequence $(k_i)_{i \leqslant 1056}$ in sections of 32 elements. For j between 1 and 5 we have only marked whether this is the "original" list or the "reversed" one. For the remaining elements we have encoded each factor $x_j + y_j$ by y_j only. For instance:

– Elements 1)–32) resulted from Procedure 5 (that is $k_1 - k_{32}$) form section 1 in the table; the first 5 factors in these elements change in some order, which we called “original”, giving all possible elements that could have been obtained (see formula (17)), and all y_j, $j \geqslant 6$ are 0,

– Elements 33)–64) (which are $k_{33} - k_{64}$) form section 2; now the first 5 factors in these elements change in the reversed order, $y_6 = 1$ and the remaining y_j are still 0,

– Elements 65)–96) (i.e. $k_{65} - k_{96}$) form section 3; the first 5 factors in these elements again change in the original order, $y_6 = y_7 = 1$ and the remaining y_j are still 0,

and so on.

section	index i (of k_i)		y_j						
	min	max	j=1–5	j=6	j=7	j=8	j=9	j=10	j=11
1	1	32	(orig)	0	0	0	0	0	0
2	33	64	(rev)	1	0	0	0	0	0
3	65	96	(orig)	1	1	0	0	0	0
4	97	128	(rev)	0	1	0	0	0	0
5	129	160	(orig)	0	1	1	0	0	0
6	161	192	(rev)	1	1	1	0	0	0
7	193	224	(orig)	1	0	1	0	0	0
8	225	256	(rev)	0	0	1	0	0	0
9	257	288	(orig)	0	0	1	1	0	0
10	289	320	(rev)	1	0	1	1	0	0
11	321	352	(orig)	1	1	1	1	0	0
12	353	384	(rev)	0	1	1	1	0	0
13	385	416	(orig)	0	1	0	1	0	0
14	417	448	(rev)	1	1	0	1	0	0
15	449	480	(orig)	1	0	0	1	0	0
16	481	512	(rev)	0	0	0	1	0	0
17	513	544	(orig)	0	0	0	1	1	0
18	545	576	(rev)	1	0	0	1	1	0
19	577	608	(orig)	1	1	0	1	1	0
20	609	640	(rev)	0	1	0	1	1	0
21	641	672	(orig)	0	1	1	1	1	0
22	673	704	(rev)	1	1	1	1	1	0
23	705	736	(orig)	1	0	1	1	1	0
24	737	768	(rev)	0	0	1	1	1	0
25	769	800	(orig)	0	0	1	0	1	0
26	801	832	(rev)	1	0	1	0	1	0
27	833	864	(orig)	1	1	1	0	1	0
28	865	896	(rev)	0	1	1	0	1	0
29	897	928	(orig)	0	1	0	0	1	0
30	929	960	(rev)	1	1	0	0	1	0
31	961	992	(orig)	1	0	0	0	1	0
32	993	1024	(rev)	0	0	0	0	1	0
33	1025	1056	(orig)	0	0	0	0	1	1

The next step was to examine 5-bit sequences made from $y_6 - y_{10}$ for each section. The first observation was that $y_6 - y_{10}$ change from section to section exactly in the order we called original (i.e. the one defined for $y_1 - y_5$ within odd-numbered sections) so again we have obtained all possible 0-1 sequences. We have suspected that this will work similarly for $y_{11} - y_{15}$ (i.e. they remain unchanged within sections of consecutive 1024 elements and change from section to section in the order we called original) and so on. Still, it was not obvious how to prove it in general. But eventually, we understood what we were looking at – actually the 5-bit sequences made from $y_6 - y_{10}$ almost formed the Gray code list, but the sequences were written in the inverse order (i.e. with the least significant bit at the beginning). So it was enough to reverse the notation, shift the numbering (to start from 0) and that solved the problem as we can encode every natural number with the Gray code. Moreover, we have noticed a new (at least to us) method to create (define) the n-bit Gray code list G_n, namely G_n is the list of elements $g_0, \dots g_{2^n-1}$ created due to the following rules

1. $g_0 = 0...0$ (n bits)
2. for every integer $0 \leqslant k \leqslant 2^{n-1} - 1$ the element g_{2k+1} is obtained from g_{2k} by changing the last bit,
3. for every integer $1 \leqslant k \leqslant 2^{n-1}$ for g_{2k} we encounter two situations:
   a. the last bit of g_{2k-1} is 1 — then to get g_{2k} we change 2nd bit from the right in g_{2k-1},
   b. the last m bits in g_{2k-1} are 0 and $m+1$-st (from the right) bit is 1 — then to get g_{2k} we change $m+2$-nd bit (from the right) in g_{2k-1}.

Note that the standard method to get G_n is the recursive one: once we have G_{n-1} we write it in the reverse order. Next, we add the prefix 0 to every element of the original list, the prefix 1 to every element of the reversed one and concatenate. Since the list for 1-bit code is $G_1 = (0, 1)$ the list for 2-bit code is $G_2 = (00, 01, 11, 10)$ (that means that $0 = 00_G$, $1 = 01_G$, $2 = 11_G$ and $3 = 10_G$), next $G_3 = (000, 001, 011, 010, 110, 111, 101, 100)$ and so on.

3. Conclusions

There are not too many general methods for examining automorphisms of the rooted trees and their properties, so the *brute force* approach we have described can be useful sometimes. Note however, that the programming was possible because of the regularity of elements we examined. On the other hand, there are still many questions and problems in this area concerning automorphisms that have pretty regular form, so some procedures (or even all of them after small changes) we have described in the previous section can be used in future investigations.

Let us also mention that one immediate profit we have gained was the idea for the proof of our second main theorem in [1], which also concerned the construction described here but with a different set Z_0. Namely, instead of each generator g defined by (7) or (9) we took three, with each three components to be the same as in g and 0 elsewhere, starting from 1-st, 2-nd and 3-rd. For a that gave identity or a, as before.

Since components of b, c, d are regular in cycles of 3, from each of them we got either identity or elements with every three nonzero components starting either from 2nd or 3rd or 4th, that is those elements were

$$a_{3\mathbb{N}-1} := [0, \alpha_2, 0, 0, \alpha_5, 0, 0, \alpha_8, 0, 0, \alpha_{11}, 0, 0, \ldots],$$

$$a_{3\mathbb{N}} := [0, 0, \alpha_3, 0, 0, \alpha_6, 0, 0, \alpha_9, 0, 0, \alpha_{12}, 0, \ldots],$$

$$a_{3\mathbb{N}+1} := [0, 0, 0, \alpha_4, 0, 0, \alpha_7, 0, 0, \alpha_{10}, 0, 0, \alpha_{13}, \ldots].$$

and finally $Z_0 = \{a, a_{3\mathbb{N}-1},\ a_{3\mathbb{N}}, a_{3\mathbb{N}+1}\}$. It turned out that we could also define some set T such that elements of H_0 were in the form $t_1 a \ldots a t_n,\ t_i \in T$ and all observations remained valid whence elements of Z_1 could have also been defined using the Gray code.

4. *Mathematica* code

Below we give a code (with comments) in *Mathematica* for all the procedures we described in Section 2.

Procedure 1

```
gen[k_, h_] := Module[{m, subst, poly, polyMod}, m = Length[h];
      subst = Table[x[j] -> (x[j] + h[[j]]), {j, 1, m}];
       (* new argument *)
      poly = Expand[(k /. subst)];
       (* substitution *)
      polyMod = PolynomialMod[(poly /. x[j_]^_ -> x[j]), 2];
       (* reduction    *)
  Return[Simplify[polyMod]]     (* factor form  *)
  ];
```

Procedure 2

```
allGen[n_] := (
  step[1] = listGen[1] = {k};
  step[m_ /; m <= n] := step[m] = If[EvenQ[m],
     gen[listGen[m - 1], a],
     Table[gen[listGen[m - 1], T[[i]]], {i, 15}]
      ];     (* m-th step of generating *)
  newGen[m_]:=newGen[m]=Complement[Flatten[step[m]],listGen[m-1]];
    (* new generators that appear on m-th step *)
  listGen[m_] := listGen[m] = Join[listGen[m - 1], newGen[m]];
    (* list of distinct generators after m steps *)
  Print["Step 1 (the basic generator): ";
 Do[
  Print["Step ",i, " (acting on ",
   If[EvenQ[i],
     "each generator from the previous list by a): ",
     "generators from the previous step by n-th element of T): "]
    ];
  Do[Print[j, ") ", step[i][[j]]], {j, Length[step[i]]}];
  Print["List of new generators: ", newGen[i]];
  Print["Distinct generators obtained so far: ",Length[listGen[i]]];
  Print["List : ", listGen[i]]
 ,{i, 2, n}]
 )
```

Procedure 3

```
procGen[n_] := (
Module[{m},
 gens[1] = k;
 gens[m_]:=gens[m]=If[EvenQ[m], gen[gens[m-1],a], gen[gens[m-1],t]];
      (* a for even, t for odd *)
 Do[Print[i, ") ", gens[i]], {i, 1, 2 n}]
 ];
 listGen = DeleteDuplicates[Table[gens[i], {i, 1, 2 n}]];
    (* distinct generators' list *)
 Print["Number of distinct generators: ", Length[listGen]]
)
```

Procedure 4

```
proc2Gen[n_] := (
   Module[{m},
      gens[1] = k; (* 1st part of the list *)
      gens[m_ /; m < 33] :=
         gens[m]=If[EvenQ[m], gen[gens[m-1],a], gen[gens[m-1],t]];
      gens[33]=gen[gens[32], s]; (* 2nd part of the list *)
      gens[m_ /; m > 33] :=
         gens[m]=If[EvenQ[m], gen[gens[m-1],a], gen[gens[m-1],t]];
      Do[Print[i, ") ", gens[i]], {i, 1, 2 n}]
   ];
   listGen = DeleteDuplicates[Table[gens[i], {i, 1, 2 n}]];
      (* distinct generators' list *)
   Print["Number of distinct generators: ", Length[listGen]])
```

Procedure 5

```
generators[seq_] := (
  seqEx = {};
  Do[For[q=1,q <= i[[2]], q++,
      AppendTo[seqEx,a ];  AppendTo[seqEx,i[[1]]]],
      {i, seq}];  (* explicit form of seq including a's  *)
  listGen[1] = k;
  listGen[m_ /;m<= Length[seqEx]+1]:=
  listGen[m]=gen[listGen[m-1], seqEx[[m-1]]];
  distGen=DeleteDuplicates[Table[listGen[i], {i,Length[seqEx]+1}]];
   Print["Number of generators obtained by seq: ", Length[seqEx]+1];
   Print["Number of distinct generators: ", Length[distGen]])
```

Acknowledgements. The author gratefully acknowledges the many helpful suggestions and remarks of the Referee. Also the author would like to thank Damian Słota and Maciej Harapiński for their help in writing *Mathematica* code.

Bibliography

1. Bajorska B.: *Factorizations of self similar groups associated to the first Grigorchuk group.* Comm. Algebra **44** (2016), 5004–5026.
2. Grigorchuk R.I.: *On Burnside's problem on periodic groups.* Funkcjonal. Anal. i Prilozhen **14**, no.1 (1980), 53–54 (in Russian). English translation: Functional Anal. Appl. **14**, no.1 (1980), 41–43.
3. Kaloujnine L.: *Über eine Verallgemeinerung der p-Sylow-gruppen symmetrischer Gruppen.* Acta Math. Acad. Sci. Hungar. **2** (1951), 197–221 (in Russian, German summary).
4. Sushchanskiĭ V.I.: *Wreath products and periodic factorable groups.* Mat. Sb. **180**, no. 8 (1989), 1073–1091 (in Russian). English translation: Math. USSR-Sb. **67**, no.2 (1990), 535–553.