Journal of Algorithms and Computation

journal homepage: http://jac.ut.ac.ir

A new indexed approach to render the attractors of Kleinian groups

Alessandro Rosa*[1]

[1]Software Developer, Brindisi, Italy .

ABSTRACT

One widespread procedure to render the attractor of Kleinian groups, appearing in the renown book [8], wants huge memory resources to compute and store the results. We present a new faster and lighter version that drops the original array and pulls out group elements from integers.

ARTICLE INFO

Article history:
Received 30, June 2017
Received in revised form 18, November 2017
Accepted 3 December 2017
Available online 23, December 2017

Keyword: Kleinian groups, converge uniformly, limit cycles.

AMS subject Classification: 05C72.

1 Introduction: some definitions

Let K be a group of one-to-one relations. One model binds the elements of K to strings of symbols: letters showing up in two cases and distinguishing the group elements from their inverses: 'a' (lower case) and 'A' (upper case).
The *generating set* G, the smallest subgroup of K, includes relations tagged with single letters that collect into the *alphabet* of K. All entries in K come from the combination, termed *multiplication*[1], of elements in G. Multiplication corresponds lexically to concatenation of letters into the so-called *word*. Words resemble to algorithms and they enjoy both symbolic (code) and operative (run) features. There are finite ($bbbbaBAbA$) or infinite[2] ($\overline{b}bbbaBAbA$) words and the reading order, left-to-right (LR) or right-to-left

*alessandro.a.rosa@gmail.com
[1]For example $a \circ b$, but the operator $\circ$ is often omitted for sake of brevity.
[2]The overline symbol marks the period, like for numbers.

(RL), drives the letters/generators picking. Thus the *word value*, the last orbit element, may change depending on that order. Given $W = abA$ in RL, the *orbit* is the sequence $z_1 = A(z_0), z_2 = b(z_1), z_3 = a(z_2), \ldots$. Subwords, returning the identity map I, are said *crash word* and they provoke the *cancellation* of letters and produce *reduced words*. Given $W = aBba$, we find one cancellation, bB, so that W reduces to aa.
Words in K converge uniformly to limit cycles,[3] collectively defined as the *attractor*. Generators and words show up in a twofold (lexical and geometric) nature: as symbol/point and as concatenation/orbit respectively. Such a duality extends to groups, in terms of words/attractor.[4]

2 Basic setup

Working with attractors wants a sufficient degree of freedom and to consider all words in the group. So we step back to the abstraction of strings and symbols, because we need a 'malleable' setup to work with: *any concatenation of letters with length* $d < +\infty$. According to the theory of enumerative combinatorics, it is represented by a m-branched tree (see fig. 1), where m is the alphabet size.

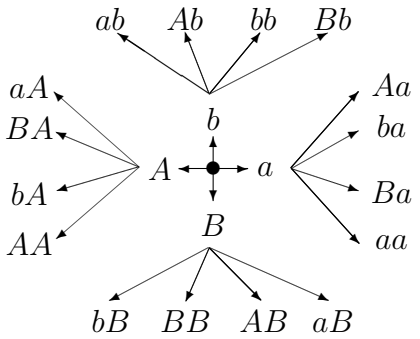


Figure 1: **Original tree**. Enumeration of all possible concatenations of symbols up to depth 2.

Luckily, the theory of combinatorial groups[5] helps to set strong links between tree graphs and groups: generators and words deal with concepts of *node*, *path*, *depth*, *root*, *leaf*, *parent*, *child*. Group generation rules can be resumed by the *presentation* tool. We account two versions: the *Cayley multiplication table* including all multiplicative pairs of elements for *finitely generated groups*[6] G; and the *group presentation* $\langle R|S\rangle$: a compact list of generators, said *relators* R, and of *relations* S [6]. The tree shows as the easiest graphical expedient to explain how groups are generated through a presentation. Fig. 1

[3]Every K is a *convergence group*; see [4], pp. 334–340.
[4]Alternatively defined the 'limit set'.
[5]Pioneered by Sir Arthur Cayley during 1850s. Refer to [2].
[6]Equipped with a finite number of generators and of relations between them.

shows the *original tree*, related to the simplest presentation, where $S \equiv \emptyset$. We will work with trees of bounded depth $d < +\infty$.

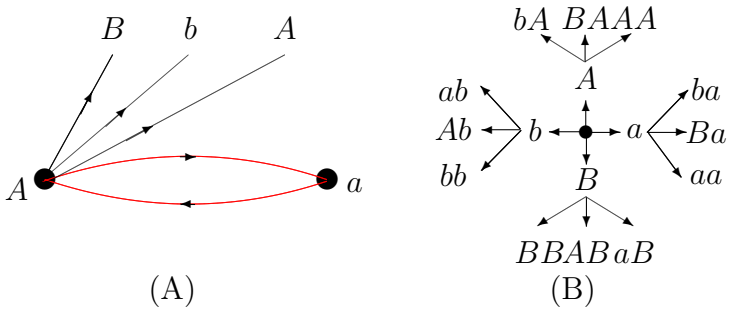


Figure 2: **Pruning**. (A) New nodes are pruned if relating to crash words (in red). (B) The resulting pruned tree.

3 Once-punctured torus groups

Let K be a Kleinian group, a discrete group of orientation preserving conformal maps M. In recent times, this topic gained more interest from the popular audience as it was dragged by the caravan of fractals, due to the similarities with Julia sets.[7] Let M be a linear fractional map $(mz + n)/(pz + q)$ in one complex variable $z \in \mathbb{C}$. We are interested into the quasi-Fuchsian subfamily of K and we will work with 4 generators a, b, A, B: the topological model is the once-punctured torus and it shows as the free product $K = G * H$, $G = \{a, A\}, H = \{b, B\}$. The presentation is $\langle x, X | xX = I \rangle$ or the Cayley table 1, because Kleinian groups are finitely generated. K is also free because only trivial relations appear. We will discuss the role of Cayley table later in section 5. This presentation prunes the original tree in fig. 1 from nodes related to strings with crash words *Aa, aA, Bb, bB* (fig. 2/A at p. 55), squeezing into the identity map I that sends points forth and back, $z_1 = A(z_0), z_0 = a(z_1)$, and arresting the tree growth along the branch. The goal is to have no reduced words and get the pruned tree in fig. 2/B.

4 The revised deterministic approach

The problem of rendering the attractor[8] of K has been studied thoroughly, in terms of *automatic groups*,[9] only in [7, 8], as far as the author knows.

[7] Gaston Julia was the first to set this analogy in 1918, while studying the iterations of functions in one complex variable. Refer to [1]. Also see [8] as introduction to Kleinian groups.

[8] The renderings of attractors in this article have been computed through author's software 'Circles': http://alessandrorosa.altervista.org/circles/

[9] Any finitely generated group equipped with a finite state automata. Refer to [3], p. 356.

	I	**a**	**b**	**A**	**B**
I	I	a	b	A	B
a	a	a	b	I	B
b	b	a	b	A	I
A	A	I	b	A	B
B	B	a	I	A	B

Table 1: **Cayley multiplication table for once-punctured torus groups.**

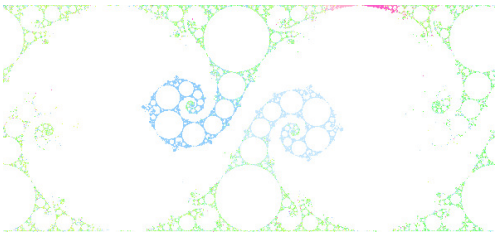


Figure 3: **Probabilistic rendering.** Attractor for the parameter $\mu = -0.097 + 1.838i$ in the Maskit T1,1 embedding, rendered via a 'boosted-up' modification. Only 1.048.576 words has been required to enhance sharp details.

Two approaches have been developed. One is *probabilistic*, not relying on tree model and working on one only word/orbit, which gets longer and longer as generators are appended through random picks, given a probability law.[10] The second is *deterministic*, where words obey to the combinatorial tree model. The larger the depth, the longer the word, the finer the rendering: this is the base meaning of this approach, running millions of words to get fine quality pictures.[11] The original implementation implements a huge array of unique words - the *dictionary*[12] – and requires expensive resources, in terms of memory allocation. Our version only intends to save resources, not to return finer quality renderings.

We are going to explain a two-stages strategy revisiting the pruned tree of letters. Numbers are the synthesis of two entities: *symbol* and *value*. Consider the set $\{10, 11, 12, \ldots, 19\}$ in terms of symbols: the elements come from appending one digit on the right of '1', like a new branch of a tree (see fig. 4/A). *It makes sense to review numbers as paths.* Given the mapping $[a \to 0, A \to 1, b \to 2, B \to 3]$, we replace letters with digits in the tree at p. 55. Starting from the root at depth 0, corresponding to I (empty string), we move to the left branch at depth 1 and get the *subword* '1'; we route to one next branch and get '31' at depth 2. Each number represents one only chain of digits and so we pull out the

[10] A sketch is available in [8], p. 152. This algorithm can be boosted up via commutators, similarly to a technique for the deterministic approach, discussed in [8], pp. 181, 248.

[11] Authors of [8] had to pull additional manipulations (so-called 'repetends') out of the hat, to catch up sharp renderings, because orbits run slower as they get closer to parabolic points.

[12] See p. 114 of [8] suggested cardinality $10E7$ or more, refer to caption of fig. 6.

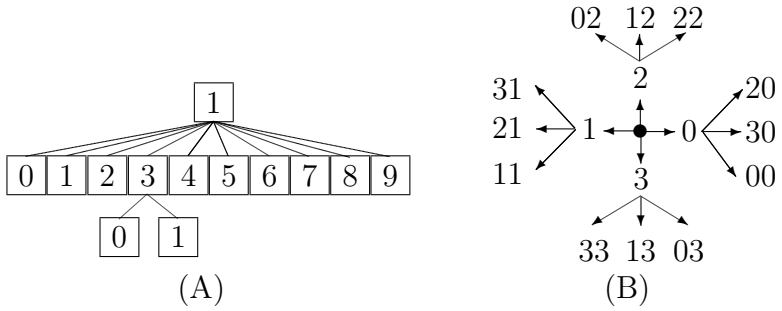


Figure 4: **First stage**. (A) Numerical paths: the mid row shows numbers composition from 10 to 19. The bottom row shows one next step. (B) Digital version of the lexical pruned tree.

digital tree in fig. 4/B.
The master plan is to move from symbols to digits and finally to indexes (numbers). Now notice that words include digits from 0 to $m-1$ (here $m=4$) and nodes bind to base-m numbers, turning into indexes in base-10 (prefixed by '♯' in fig. 5). Indexes count the appearing order of nodes, during the whole tree growth. The notation $31_{4\leftarrow}$ indicates that 31 is written in base-4 and read in RL order: it amounts to $4_{10\leftarrow}$. For example, $Ba \Rightarrow 21_{4\leftarrow} \Rightarrow 5_{10\leftarrow}$. We finally get the *indexed tree* in fig. 5/A), whose nodes bind to base-10 indexes.

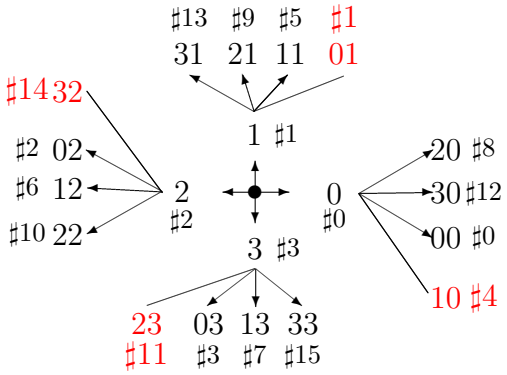


Figure 5: **Second stage**. Indexed version of the digital tree: red nodes mark crash words.

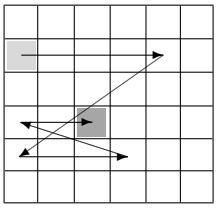
We do not need words to be stored: they are already 'coded' inside integers!

5 Cayley Multiplication Tables

Cayley tables portray all multiplicative combinations between pairs of elements of G and they are homologue to *state transition tables*, supported by a *finite state automaton* (FSA): in fact word runs [5] behave like dynamical systems, whose 'states' match to values

inside cells. Cayley tables prune the original tree, depending on whether the final state is of crash kind (indexed with 0) or not.

	0	1	2	3	4
0	0	1	2	3	4
1	1	1	2	0	4
2	2	1	2	3	0
3	3	0	2	3	4
4	4	1	0	3	4



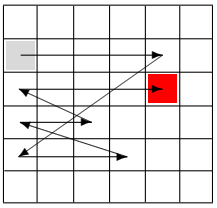
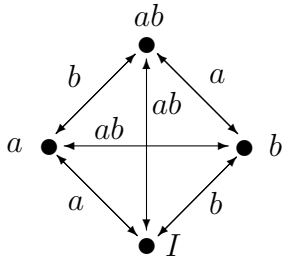


Table 2: **Zig-zagging for words of once-punctured torus groups**. On the left, the indexed version of Cayley table 1. At the center, the succession of states related to abA, that is, '123' (RL). On the right, the crash path of $AabA$.

The Cayley table for once-punctured torus groups is simple to run: just the recognition of the crash words $\{aA, Aa, bB, Bb\}$. But there are groups equipped with more complicate presentations, demanding a generalized management: *take on each string from the original tree and run it along the Cayley table*. We will illustrate it in two examples.
The once-punctured torus groups offer a comfortable start. The indexes, originally ranging in $[0-9]$, will be incremented by 1 to avoid collisions with the crash state. Let the indexed table 2 at p. 58 and $W = abA$ (RL), which turns into '123'. For algorithmic reasons, the table scan begins from the neutral state of the identity element at row 0: $WI \equiv W$. Reading the first symbol '3', we move to column 3, with state value '3': in fact, in a dynamical system, each state rules the value of the next one in progression. We read the second symbol '2', we move to column 2 with state 2. Again, we place at row 2, we read the third and last symbol '1', so we move to column 1, with (final) regular state value 1.
The second example concerns the *Klein-four group*, equipped with less obvious presentation: $\langle a, b | a^2 = b^2 = (ab)^2 = I \rangle$. Despite of its name, it has nothing to do with Kleinian groups, but we want to show that crash states do not necessarily relate to pairs of inverse maps.

	I	a	b	ab
I	I	a	b	ab
a	a	I	ab	b
b	b	ab	I	a
ab	ab	b	a	I



	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

Table 3: **Klein Four-Group**. From left to right, multiplication table, tree model and indexed version of the left table.

We notice one entry tagged with 'ab', a compound word. We apply the translation $[a \to 1, b \to 2, ab \to 3]$ to preserve the one-to-one symbols match and obtain the indexed

version (table 3, on the right), helping to rework the indexed words. Let the indexed $W =$ '123' (RL), or $(a)(b)(ab)$.[13] Reading W leads to the zero index at row 1 / column 1, so '123' is a crash word! There exist even more complicate groups, with multiple crash states.[14]

6 Implementation

Disclaimer. We will show Javascript pseudo-code implementing the indexed scan. Being an higher level language, Javascript runs reasonably slower than C++. The fastest algorithms want bare codings: just the essential computations and possibly no external function calls. Our web environment[15] has broader goals and does not follow such indications; our benchmarks just attest faster speeds, not the fastest possible.
We count all nodes in the original tree and return the indexed word through the routine `get_RL_word`. The goal is to return strings whose length is equal to the node depth. This routine does not just run as ordinary base conversion: tests showed that there could be base-10 indexes not retrieving words of required depth and thus bugging the final rendering. So we put an extra `if`-statement assuring that the tree is transversed all the way back to depth 1. Our code is tuned to 9 generators at most. If more are required, the string `out` shall be replaced by an array that stores indexes (even with multiple digits) separately: the string object – although being again an array – allows to save just one symbol per indexed entry.

RL-path scanner

```
function get_RL_word( _n, _gens_n, _depth )
{
// _gens_n : is the number of symbols, i.e. of generators
var _rem = 0, _quot = _n, out = "" ;
while( true )
{
// remainder incremented by 1 to match our indexing
// rule: 0 for identity, other digits for generators index
_rem = ( _quot % _gens_n ) + 1 ;
_quot = ( _quot / _gens_n ) >> 0 ; // integer division
//it stops only when depth 1 is reached
if ( _quot < _gens_n && _depth <= 1 ) return _rem + '' + out ;
out = _rem + '' + out ; // string concatenation
_depth-- ;
}
}
```

[13] We will explain further why we split it into tokens.
[14] Including commutators of order 2, $(abAB)^2 = I$, for example: see [8], p. 359.
[15] See footnote 8.

We give the code below to check an indexed word run through any Cayley table. It is simply a multi-dimensional array reading, returning 0 if a crash word is met, otherwise returns 1.

Check word run

```
// indexed Cayley table for once-punctured torus groups
var _idx_cayley_table = [ [ 0, 1, 2, 3, 4 ],
[ 1, 1, 2, 0, 4 ],
[ 2, 1, 2, 3, 0 ],
[ 3, 0, 2, 3, 4 ],
[ 4, 1, 0, 3, 4 ] ] ;

function check_word_run( _idx_word, _cayley_table )
{

// we start from row 0, by convention
var _idx = -1, _ret = 1, _row = 0 ;

/* get the input word, split indexes into tokens, convert'em all
into numbers and reverse for RL order */

_idx_word=_idx_word.split( "" ) ;
_idx_word=_idx_word.map(function(_i){return parseInt(_i,10 );}) ;
_idx_word = _idx_word.reverse(); //RL reading order

for( var _i = 0 ; _i < _idx_word.length ; _i++ )
{
_idx = _idx_word[_i], _row = _idx_cayley_table[_row][_idx] ;
if ( _row == 0 ) { _ret = 0; break ; } // crash state is met
}

return _idx == -1 ? 0 : _ret ;
}
```

All nodes will be scanned according to the tree in fig. 5 and keep track of both depth and run for each node. Given an m-branched tree, let $\lambda = m^d$ be the number of nodes at bounded depth $d \leq D < +\infty$. The attractor can be rendered in '*limit set*' or '*tiling*' mode, whether points are drawn for $\lambda = m^D$ leaves only or for all $\sigma = \sum_{d=1}^{D} \lambda$ nodes respectively. We finally pseudo-code the nested loops below, to render the attractor in three steps: (1) a loop feeding the fixed points to start the orbits;[16] (2) a loop to get the *indexed word* from each node; (3) a loop to read and compute and draw the word values.

Index search algorithm

```
var _gens_n = 4, _rl_word = "", _max_depth = 4, _fp = null ;
/*assume that we already collected the fixed points of the Mobius
```

[16] See table in [8], p. 135.

```
transformations of K into the array _input_fixed_pts and that we have
a multi-dimensional array storing the current Cayley Table*/
// feed fixed points
for( var _p = 0 ; _p < _input_fixed_pts.length ; _p++ )
{
_nodes_n = Math.pow( _gens_n, _d ) ;
// n is the index of each node belonging to depth _d
for( var _n = 0 ; _n < _nodes_n ; _n++ )
{
_fp = _input_fixed_pts[_p] ;
_rl_word = get_RL_word( _n, _gens_n, _d ) ;
// pseudo-code (loop):
// 1.0 read _rl_word
// 2.0 check_word_run( _rl_word, _cayley_table )
// 2.1 if crash state is met, continue to the next iteration
// 2.2 otherwise, for each digit in the _rl_word:
// 2.2.1 get the related Mobius transformation M_n.
// 2.2.2 apply _fp = M_n(_fp)
// 3.0 draw _fp on the screen, according to
'tiling' (any word)
'limit set' (only words whose length = depth) mode
}
}
```

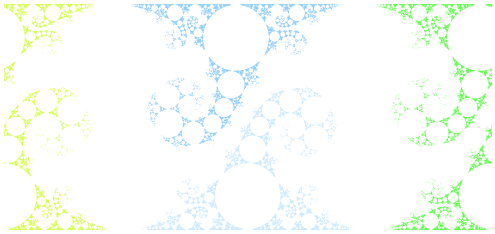


Figure 6: **Index-search approach.** Same attractor as in picture 3, rendered through a tree of depth 14, counting about 350 millions of words. In terms of original deterministic approach, the dictionary would weight more then 4 Gigabytes.

7 Conclusions

The benefits of this version account to: 1) words are pulled out from integers, not by tree transversion[17]; 2) save memory resources required by the dictionary; 3) very easy

[17]See *breadth-first* and *depth-first* implementations, p. 115 and 148–151 of [8] respectively.

implementation; 4) quick extension to arbitrary Cayley tables.

References

[1] Alexander D.S., Iavernaro F., Rosa A., *Early days in complex dynamics*, AMS, 2011.

[2] Cayley A., *On the theory of groups, as depending on the symbolic equation* $\theta^n - 1 = 0$, Philosophical Magazine, Vol. 7 (1854), pp. 40–47.

[3] Epstein D., Cannon J., Holt D., Levy S., Paterson M., Thurston W., *Word Processing in Groups*, Boston, MA, Jones and Bartlett Publishers, 1992.

[4] Gehring F., Martin G., *Discrete quasiconformal groups I*, Proceedings of the London Mathematical Society, 55 (1987), pp. 331–358.

[5] Hopcroft J.E., Ullman J.D., *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, 1979.

[6] Lyndon R.C., Schupp P.E., *Combinatorial Group Theory*, Springer-Verlag, 1977.

[7] McShane G., Parker J.R., Redfern I., *Drawing limit sets of Kleinian groups using finite state automata*, Experiment. Math. Volume 3, Issue 2 (1994), pp. 153–170.

[8] Mumford D., Series C., Wright D., *Indra's pearls: The Vision of Felix Klein*, Cambridge University Press, 2002 (reprinted in 2015).

[17] This article is dedicated to the memory of the Iranian mathematician and Fields medallist Maryam Mirzakhani (1977 – 2017). The author did not know her in person, either he is not involved in her field of interest. But he admired her working on top mathematical problems with extraordinary pleasure and joy, played with success in parallel with her role as mother and wife.