geometric edges since there are at most two orbits of oriented edges.

Suppose the stabilizer of a geometric edge was non-trivial. Then the squares of its elements would be trivial as they stabilize an oriented edge. Since the group is torsion free, this cannot happen. **q.e.d.**

2.4.4 Grushko's Theorem

Theorem 2.75. *If a free product $A * B$ has a set of n generators, then it has a separated set of n generators, i.e., a generating set contained in $A \cup B$. In particular,*

$$\mathrm{rk}\,(A * B) = \mathrm{rk}\,(A) + \mathrm{rk}\,(B)\,.$$

We give a version of Stallings' proof based on [Stal88]. We will, however, avoid the use of two-complexes and work with folds (also invented by Stallings). This makes the proof more combinatorial. It actually yields an algorithm for the construction of a separated generating set.

Proof. Let us start with an arbitrary set of n generators for $A * B$. Recall that the elements are essentially words over the alphabet $A \cup B - 1$. We will devise an algorithm that takes this generating set as an input and has a separated generating set of at most equal size as its output.

Our main bookkeeping device is a graph with edge labels in $A \cup B$. Here is the precise data structure that we use:

Definition 2.76. An A, B-labeling is a connected graph Γ together with a map $\phi : \overrightarrow{\mathcal{E}}_\Gamma \to A \cup B - 1$ satisfying

$$\phi[\overrightarrow{e}^{\mathrm{op}}] = \phi[\overrightarrow{e}]^{-1}.$$

The values of ϕ are called labels. We color all edges red that have their label in A. The remaining edges are colored blue. A vertex

in Γ is monochromatic if all edges in its link have the same color. A path in Γ is called monochromatic if it contains edges of one color only.

Since any directed edge "reads" an element of $A * B$, any directed path "reads" the product. We call a path null-homotopic if it reads the trivial element in $A * B$.

Clearly, we have a homomorphism

$$\phi : \pi_1(\Gamma, v) \to A * B$$

for any vertex $v \in \Gamma$.

An A, B-labeling is generating if the induced homomorphism is surjective.

The start for our algorithm is a rose:

Observation 2.77. *We can realize any finite generating set for $A * B$ by a generating A, B-labeling modeled on a rose with subdivided loops.*

The goal of the algorithm is a rose too:

Observation 2.78. *Any generating $A * B$-labeling modeled on a rose with undivided loops, i.e., a graph with only one vertex, represents a separated generating set for $A * B$.*

Our algorithm modifies A, B-labelings. The goal is to reduce the number of vertices. Eventually, we have a graph with only one vertex, which represents a separated generating set.

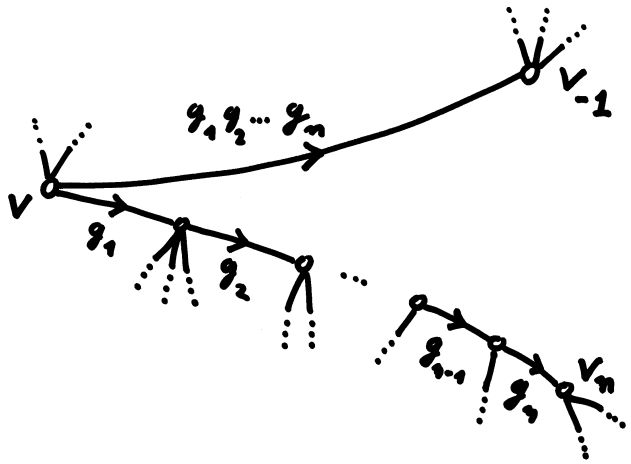
For the reduction step, assume our labeling Γ has at least two vertices. Then Γ contains path that is not a loop. This path can read any element of $A * B$, but as $\phi : \pi_1(\Gamma, v) \to A * B$ is surjective, we can append a loop so that we obtain a null-homotopic path that connects two different vertices.

The next step in the reduction is to come up with a monochromatic, null-homotopic path connecting two different

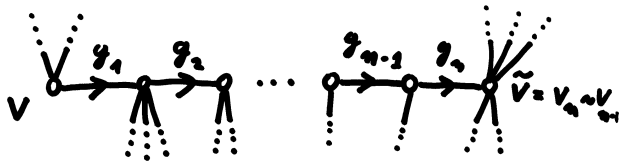
vertices. For those, who do not care about a construction: Among all null-homotopic non-loops, chose a shortest one and prove that this must be a monochromatic path.

A more constructive approach runs like this: We already have a null-homotopic non-loop. First remove from this path all monochromatic, null-homotopic loops. This will shorten the path and we will still have a null-homotopic non-loop. The edges in this path come in runs of edges of the same color. Since we remove null-homotopic, monochromatic loops, each of these runs is either not null-homotopic or not a loop. Each run gives an element in A or B. The product of these has to be trivial (the whole path is null-homotopic), but by the definition of runs, the factors are taken alternatingly from A and B. An alternating product of non-trivial elements, however, cannot be trivial in the free product $A * B$. Hence one of the runs reads the identity element. Then, this run is not a loop. Hence we found a monochromatic, null-homotopic path connecting two different vertices.

Note that our path has length ≥ 2 since we do not allow for trivial edge labels. Since it reads the trivial element, the first edge reads the inverse of the rest. Therefore, adjusting the orientations, we get the following picture:



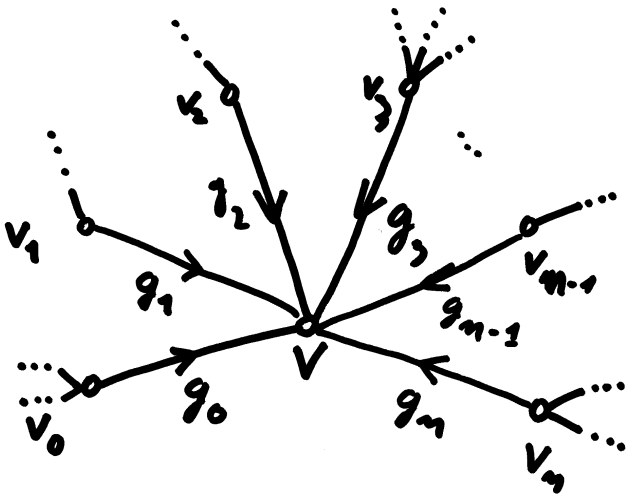
Now, we perform a fold, i.e., we replace the picture above by



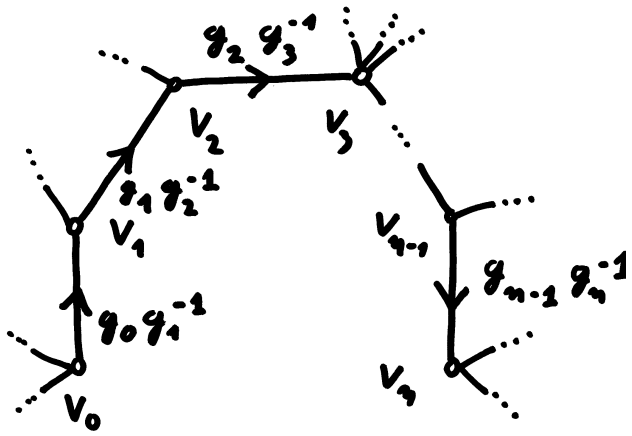
Clearly, the fundamental group of Γ did not change: we removed one edge and reduced the number of vertices by one. Moreover, the induced homorphism ϕ is still surjective: for every closed loop before the fold we can find a closed loop after the fold reading the same group element. So we still have a generating A, B-labeling. But the number of vertices dropped by one. Keep going, until there is only one vertex left. This completes the proof.

Wait a minute. Pictures can be so misleading, and in this prove, there is a serious gap: We want to remove the initial edge of our path. The justification is that we do not need it as the complementary segment of the path already reads the same group element. But what if this segment passes through the initial segment? Then deleting it would cut the path – ouch.

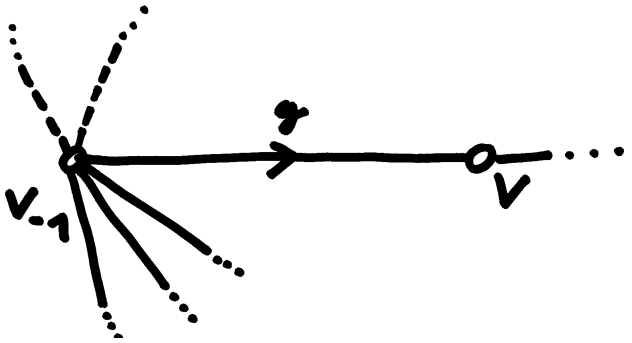
Hm. Here is a patch. First let us observe that there is another process that reduces the number of vertices in Γ. Suppose v is a monochromatic vertex, then we can remove v. To do this, we replace



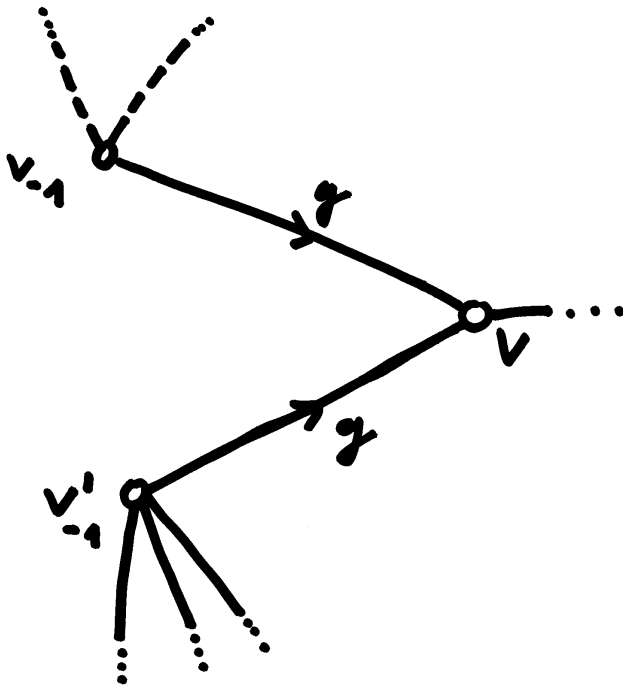
by



So let us have a closer look at the initial edge e of our monochromatic path. Let v_{-1} be the first and v be the second vertex in the path. Our plan is to do an unfold along e first to split v_{-1} into two vertices, one of which keeps the connections to blue edges whereas the other one stays connected to the red edges. So we replace the picture



by



Note that we increased the number of vertices by one. On the other hand, we have a path starting at the new vertex v'_{-1} which is monochromatic and null-homotopic and whose first edge is not used a second time. Hence we can immediately follow the unfold by a fold which restores the number of vertices. So what was the progress?

By the unfold, the vertex v became monochromatic. Hence we can get rid of it, thereby reducing the number of vertices. So we did one step back and two steps forward, and this does complete the proof. **q.e.d.**

Remark 2.79. There is a topological interpretation of this proof, which explains why we call paths that read the trivial element "null-homotopic": Realize A and B as a fundamental groups of base pointed spaces. Then, by van Kampen's Theorem, their wedge X has fundamental group $A * B$. A generating set can be realized as a map from the n-rose Y to X. A path in Y is that maps to a null-homotopic loop in X is what we called null-homotopic.

One of Stallings proofs for Grushko's Theorem takes place in this setting: The preimage of the wedge-point is probably not

connected. So we want to reduce the number of its components. This is done by glueing two cells onto Y without changing the homotopy type. This procedure corresponds to the folds above: we just collapsed the two cell immediately after the gluing. The patch is only needed because we insist on this immediate return to a graph. For this reason, the topological proof is, in fact, shorter and more elegant.

2.5 The Hanna Neumann Conjecture

Definition 2.80. A group G has the finite intersection property if the intersection of any two finitely generated subgroups in G is finitely generated.

Theorem 2.81 (Howson [Hows54]). *Free groups enjoy the finite intersection property.*

By now, there are many proofs of this theorem. The given here is stolen from [Shor90].

Definition 2.82. Let G be a group with finite generating set Σ. A subgroup $H \leq G$ is quasi-convex with respect to Σ if there is a constant $R \geq 0$ such that every geodesic path in the Cayley graph $\Gamma_\Sigma(G)$ joining two points in H lies in an R-neighbourhood of H. That is, every point on such a path has distance $\leq R$ to at least one point in $H \subset \Gamma$.

Example 2.83. Any finitely generated subgroup H of a free group is quasi-convex with respect to the standard generators: Let B be a ball in the Cayley tree Γ centered at 1 containing all generators of H. The union HB is connected and hence a subtree. Any geodesic joining two point of H in Γ actually lies in HB. The constant R, therefore can be chosen to be the radius of B.

Proposition 2.84. *Quasi-convex subgroups are finitely generated.*

Proof. Let G, H, Σ, and R be as in the definition (2.82), and let B be the open ball in $\Gamma = \Gamma_\Sigma(G)$ of radius $R+1$. It is easy to see that

$$X = HB \subseteq \Gamma$$

is connected and that

$$\Xi := \{h \in H \,|\, hB \cap B \neq \emptyset\}$$

is finte. By (1.52) this finite set generates H. **q.e.d.**

Proposition 2.85. *The intersection of two quasi-convex subgroups is quasi-convex. More precisely, let G be a group with finite generating set Σ and let A and B be two subgroups that are both quasi-convex with respect to Σ. Then $A \cap B$ is quasi-convex with respect to Σ.*

Proof. Let R_A and R_B be the quasi-convexity constants for the two subgroups. Let $\mathcal{P}_R$ be the set of paths in $\Gamma = \Gamma_\Sigma(G)$ starting at 1 of length R. For any such path p let $\overline{p}$ denote its end point – note that this is a vertex in Γ and therefore an element of the group G. Consider the finite set

$$\Pi := \{(p,q) \in \mathcal{P}_{R_A} \times \mathcal{P}_{R_B} \,|\, \overline{p}a = \overline{q}b \text{ for some } a \in A,\, b \in B\}$$

For any pair $(p,q) \in \Pi$, pick two paths $\gamma_{(p,q)}$ (with $\overline{\gamma_{(p,q)}} \in A$) and $\delta_{(p,q)}$ (with $\overline{\delta_{(p,q)}} \in B$) such that $\overline{p\gamma_{(p,q)}} = \overline{q\delta_{(p,q)}}$. Let R be the maximum length that occurs as a path of the form $p\gamma_{(p,q)}$. We will show that any point that has distance $\leq R_A$ to A and distance $\leq R_B$ to B must have distance $\leq R$ to $A \cap B$. From this, quasi-convexity follows since the statement of course applies to points on geodesic paths joining points in $A \cap B$.

So let $g \in G$ be in the R_A-neighbourhood of A and in the R_B-neighbourhood of B. Then there is a path p of length $\leq R_A$ from g to some element of A. Similarly there is a short path q

connecting g to B. Hence $(p,q) \in \Pi$. Observe that $\overline{gp} \in A$ whence $\overline{gp\gamma_{(p,q)}} \in A$. Similarly, $\overline{gq\delta_{(p,q)}} \in B$, but because $\overline{p\gamma_{(p,q)}} = \overline{q\delta_{(p,q)}}$ it follows that $\overline{gp\gamma_{(p,q)}} \in A \cap B$. Hence g has distance $\leq R$ to $A \cap B$. **q.e.d.**

Proof of (2.81). By (2.83), finitely generated subgroups of free groups are quasi-convex. By (2.85), the intersection of two of these is quasi-convex itself. Finally, by (2.83), it the intersection is finitely generated. **q.e.d.**

Remark 2.86. Let G and H be two finitely generated subgroups of the free group F. As G and H are free groups of finte rank, one might ask how the rank of $G \cap H$ relates to the ranks of G and H. Hanna Neumann [Neum55] showed:

$$\mathrm{rk}\,(G \cap H) - 1 \leq 2\,(\mathrm{rk}\,(G) - 1)\,(\mathrm{rk}\,(H) - 1)$$

She asked whether

$$\mathrm{rk}\,(G \cap H) - 1 \leq (\mathrm{rk}\,(G) - 1)\,(\mathrm{rk}\,(H) - 1)$$

This problem is still open and know as the Hanna Neumann conjecture.

Exercise 2.87. Show that $F_2 \times C_\infty$ does not have the finite intersection property. That is, find two finitely generated subgroups whose intersection is not finitely generated.

2.6 Equations in Free Groups and the Conjugacy Problem

Reduced word provide normal forms for elements in F_n. Hence it is easy to decide if two words in the free generators represent the same group element. We can phrase this as

Observation 2.88. *Free groups have a solvable word problem.*

This is only one of many algorithmic problems one might study for a group with a fixed generating set:

Definition 2.89. The word problem for a group $G = \langle \Sigma \rangle$ is to decide algorithmically for any two words in $\Sigma \uplus \Sigma^{-1}$ whether they represent the same group element.

The conjugacy problem is to decide algorithmically whether two given words w_1 and w_2 represent conjugated group elements, i.e., if the equation

$$X w_1 X^{-1} = w_2$$

has a solution in the ambient group.

The subgroup membership problem is to decide algorithmically whether a given word represents an element of the subgroup generated by a finite list of given elements.

The subgroup conjugacy problem is to decide algorithmically for two finite sets of words if the two subgroups they generate are conjugate.

Theorem 2.90. *The conjugacy problem in F_n is solvable. More precisely, if the equation*

$$X u X^{-1} = w$$

has at least one solution in F_n, then there is a solution v such that

$$|v| < \frac{|u| + |w|}{2}.$$

Proof. Suppose v is a solution of minimal length. So we have $vuv^{-1} = w$. Assuming that the word v, u, and w are reduced, we observe that cancellations on the left had side occur only on the boundaries of u. We have to keep track of these cancellations.

First, there might be cancellations on both sides of u. In this case, we are undoing a conjugation. Since we cannot conjugate something nontrivial into the empty word, the number of those cancellations is $< \frac{|u|}{2}$.

Afterwards, cancellations can only occur on one side of u', where u is whatever is left of u. Note that for each letter that cancels in front of u', a copy of this letter appears in the back. Hence it does not make sense do have $|u'|$ cancellations or more – we could spare these superflous letters.

The total number of cancellations, therefore, is $< \frac{|u|-|u'|}{2} + |u'| \leq |u|$.

On the other hand, we know that after all cancellations are done, the right hand side equalls w. Hence

$$2|v| + |u| < |w| + 2|u|$$

and the claim follows. **q.e.d.**

Corollary 2.91. *Different generators of F_n are not conjugate.*

Exercise 2.92. Find an efficient algorithm to solve the conjugacy problem in finitely generated free groups.

Exercise 2.93. Modify the graphs-and-folds technique used in proving Grushko's Theorem to devise an algorithm that does the following:

> The input it takes is a finite set $\{g_1, \ldots, g_r\}$ of elements in F_n given as reduced words in the standard generators.
>
> The output is a list of free generators $\{h_1, \ldots, h_s\}$ for the subgroup $\langle g_1, \ldots, g_r \rangle$ generated by the g_i.

Exercise 2.94. Find an algorithm that solves the subgroup membership problem for finitely generated free groups.

Remark 2.95. The subgroup conjugacy problem is related to the recognition problem for groups: Suppose we had a machine the could tell us if the standard generator x_1 is has a conjugate in $\langle g_1, \ldots, g_r \rangle$, then we could run the test on all the generators and see if the normal closure of $\langle g_1, \ldots, g_r \rangle$ is all of the free group. Given this machine, we had an easy way of deciding if a finite presentation acutally presents the trivial group. This problem, however is undecidable.

Conjecture 2.96. *The subgroup conjugacy problem is unsolvable for non-abelian free groups.*

For free groups, there has been a lot of research about decidability questions. We list the most famous results:

Theorem 2.97 (Makanin [Maka82]). *There is an algorithm that, given an equation in a free group, decides whether the equation has a solution.*

Theorem 2.98 (Razborov [Razb84]). *There is an algorithm that, given a system of equations in a free group, decides whether it has a solution.*

In both cases, the basis for the algorithm generalizes what we did for the conjugacy problem: Find an a priory bound on the length of a minimal solution.

The ultimate theorem along those lines is a recent solution to Tarski's problem. To state the problem (and the solution) we need to understand the "elementary theory" of a free group.

Definition 2.99. Let F_n be a free group of rank n. Consider statements of first order logic over an alphabet containing a multiplication operation, the identity relation, infinitely many variables, and one constant symbol for any of the n free generators of F_n. We can interpret those statements over F_n in an obvious way.

The elementary theory of F_n is the set of all true statements.

The elementary theory encoded everything that can be said about F_n with "finite linguistic means", i.e., you are not allowed to use the langugage of sets or phrases like "and so forth". Obviously, it would be nice if the elementary theories of free groups were decidable. This would generalize the theorems of Makanin and

Razborov to a large extend – e.g., we could deal with systems of equations and inequalities.

By means of the standard inclusions

$$F_2 \leq F_3 \subset F_4 \leq \cdots$$

we can interpret any statement over a free group on n generators as a statement about all free groups of higher rank, as well. Tarski asked whether there is a statement that has different truth values when interpreted in different free groups.

Both problems, decidability of elementary theories and Tarski's problem, have positive solutions. The priority for these results, which turn out to be strongly related, is still unsettled. O. Kharlampovich and A. Myasnikov have their proofs spread out in [KM98a], [KM98b], [KM98c], [KM00a], [KM00b], and [KM00c]; Z. Sela has presented his account in [Se01a], [Se01b], [Se01c], [Se01d], [Se01e], and [Se01f]. Both proofs are several hundred pages each. The upshot is:

Theorem 2.100 (Kharlampovic-Myasnikov, Sela). *The elementary theories of all non-abelian free groups of finite rank coincide and are decidable.*

Remark 2.101. It is also possible to describe the set of all solutions to a system of equations (or more generally the set of all solutions to an open sentence) by means of "parametrized words". This also comes out of the heavy machinery used to solve Tarski's problem.

3 Grigorchuk's First Group

The First Grigorchuk Group $\mathcal{G}_1$ is a group of automorphisms of the infinite binary rooted tree. So let us consider the automorphism group of this gadget first.

Let $T_2^\bullet$ be the rooted binary tree without terminal vertices. Any vertex in $T_2^\bullet$ can be reached from the root by a minimal path. Along this path, you have to make binary decisions whether you want to go left or right. Hence the vertices in $T_2^\bullet$ are finite word over $\{-1, 1\}$ with the empty word as the root. The vertices come in levels indexed by natural numbers: the root is the unique vertex at level 0, its children are at level 1, and so on. The level of a vertex is its path-metric distance to the root. Let $T_2^\bullet(n)$ denote the sub-tree spanned by the vertices of level $\leq n$.

let $\mathcal{T}_2^\bullet := \mathrm{Aut}(T_2^\bullet)$ denote the automorphism group of $T_2^\bullet$. Note that any automorphism preserves the root (as this is the only vertex of valency 2). Hence the sets $T_2^\bullet(n)$ are invariant under automorphism, and we have, for any n, a canonical homomorphism

$$\pi_n : \mathcal{T}_2^\bullet \to \mathrm{Aut}\left(T_2^\bullet(n)\right).$$

In fact, $\mathcal{T}_2^\bullet$ is easily seen to be the inverse limit of the system

$$\mathrm{Aut}\left(T_2^\bullet(0)\right) \leftarrow \mathrm{Aut}\left(T_2^\bullet(1)\right) \leftarrow \mathrm{Aut}\left(T_2^\bullet(2)\right) \leftarrow \cdots.$$

It follows that $\mathcal{T}_2^\bullet$ is pro-finite. So it is a compact topological group.

Let us have a look at the kernels of these homomorphism. We define

$$\mathcal{T}^{(n)} := \ker\left(\pi_n : \mathcal{T}^\bullet \to \mathrm{Aut}\left(T_2^\bullet(n)\right)\right).$$

All of these subgroups are normal. The first one deserves our utmost attention: Note that $T_2^\bullet$ contains two copies of itself as subtrees – the vertices at level 1 serve as roots for these subtrees. Let us call these subtrees the left subtree T^{l} and the

right subtree T^{r}. The subgroup $\mathcal{T}_2^\bullet$ is the group of automorphisms taking T^{l} to T^{l} and T^{r} to T^{r}. Moreover, this subgroup is clearly isomorphic to the square of $\mathcal{T}_2^\bullet$:

$$\mathcal{T}_2^\bullet \times T_2^\bullet \cong \mathcal{T}^{(1)}.$$

On the other hand, the short exact sequence

$$\mathcal{T}^{(1)} \hookrightarrow \mathcal{T}^\bullet \twoheadrightarrow C_2$$

splits since the swap $\sigma \in \mathcal{T}^\bullet$ has order two. This is the automorphism that interchanges the left and right subtree. The formal definition makes use of the representation of vertices as words over $\{\pm 1\}$: The swap σ just flips the sign in the first slot. Hence

$$\mathcal{T}^\bullet \cong \mathcal{T}^{(1)} \rtimes C_2$$

Putting things together, we obtain a strange isomorphism:

$$\mathcal{T}_2^\bullet \cong (\mathcal{T}_2^\bullet \times T_2^\bullet) \rtimes C_2 = \mathcal{T}_2^\bullet \wr C_2$$

This allows us to define automorphisms recursively. As an example, consider the tree automorphism φ defined by

$$\varphi = (1, \varphi)\sigma.$$

This equation has a unique solution. Indeed, the right hand side tells us first how φ acts on level 1 vertices. Then we can plug this information back into the right hand side. Now the right hand side is defined on all vertices up to level 2. We can continue in this fashion. Similarly we have

Observation 3.1. *Let φ_i be variables with values in $\mathcal{T}_2^\bullet$, w_i and u_i given words in these variables and some fixed given automorphisms (like the swap), and ε_i given elements of $\{1, \sigma\}$. Then any system of equations*

$$\begin{aligned} \varphi_1 &= (w_1, u_1)\varepsilon_1 \\ &\vdots \\ \varphi_n &= (w_n, u_n)\varepsilon_n \end{aligned}$$

has a unique solution. **q.e.d.**

Exercise 3.2. Show that the φ defined by

$$\varphi = (1, \varphi)\sigma$$

has infinite order.

Now we can define cool automorphisms!

Definition 3.3. Let the automorphisms β, γ, and δ be defined by

$$\begin{aligned} \beta &= (\sigma, \gamma) \\ \gamma &= (\sigma, \delta) \\ \delta &= (1, \beta) \end{aligned}$$

The First Grigorchuk Group $\mathcal{G}_1$ is the subgroup of $\mathcal{T}^\bullet$ generated by $\{\sigma, \beta, \gamma, \delta\}$:

$$\mathcal{G}_1 := \langle \sigma, \beta, \gamma, \delta \rangle .$$

Our observation (3.1) is quite powerful. In fact, we can derive a complete multiplication table for the generators β, γ, and δ as follows:

Proposition 3.4. *The set $\{1, \beta, \gamma, \delta\}$ is a subgroup of $\mathcal{G}_1$ isomorphic to Klein's Vierergruppe $C_2 \times C_2$.*

Proof. From the defining equations, we get

$$\begin{aligned} \beta^2 &= (1, \gamma^2) \\ \gamma^2 &= (1, \delta^2) \\ \delta^2 &= (1, \beta^2) \end{aligned}$$

Regarding this as a system of equations in β^2, γ^2, and δ^2, we conclude that the unique solution is

$$\beta^2 = \gamma^2 = \delta^2 = 1.$$

We turn this trick into a method, i.e., we use it twice. So let us write down a system of equations in the products of length 3. We find:

$$\begin{aligned} \beta\gamma\delta &= (1, \gamma\delta\beta) \\ \gamma\delta\beta &= (1, \delta\beta\gamma) \\ \delta\beta\gamma &= (1, \beta\gamma\delta) \end{aligned}$$

and it follows that

$$\beta\gamma\delta = \gamma\delta\beta = \delta\beta\gamma = 1.$$

So we established that the defining relations of Klein's Vierergruppe hold. Hence we only have to check that β, γ, and δ are non-trivial. But this follows from the swaps that occur in the defining set of equations. **q.e.d.**

Let us call an automorphism of $T_2^\bullet$ is recursive if it can be defined by a finite system of equations. If you invert all the equations, you see that the inverse of a recursive automorphism is recursive, too. Moreover, it is easy to see that products of recursive automorphisms are recursive. Hence the recursive automorphisms form a group. We shall study a subgroup of this momentarily.

3.1 Automaton Groups and the Word Problem

Definition 3.5. A simplistic finite state automaton over the alphabet $\mathcal{A}$ is a finite directed graph with a distinguished start vertex whose edges and vertices are labeled by elements of $\mathcal{A}$ such that for any vertex v and any letter $a \in \mathcal{A}$ there is precisely one edge starting at v labeled with a.

Remark 3.6. There is an obvious way to use a finite state automaton over $\mathcal{A}$ to define a transformation $\mathcal{A}^* \to \mathcal{A}^*$. Given a sequence of letters, there is a unique directed path starting at the start

vertex that reads this sequence of letters. The output is given by reading the vertex labels along this path, starting with the vertex after the start vertex.

Definition 3.7. A sophisticated finite state automaton over the alphabet $\mathcal{A}$ is a finite directed graph with a distinguished start vertex together with to labelings. The vertices are labeled by elements of $\mathrm{Perm}(\mathcal{A})$ and the edges carry labels taken from the alphabet $\mathcal{A}$ such that for any vertex v and any letter $a \in \mathcal{A}$ there is precisely one edge starting at v labeled with a.

Remark 3.8. There is also an obvious way to use a sophisticated finite state automaton over $\mathcal{A}$ to define a transformation $\mathcal{A}^* \to \mathcal{A}^*$. Given a sequence of letters, take the unique directed path starting at the start vertex that reads this sequence of letters. The output is given by applying the permutations you read along this path to the letters in your sequence.

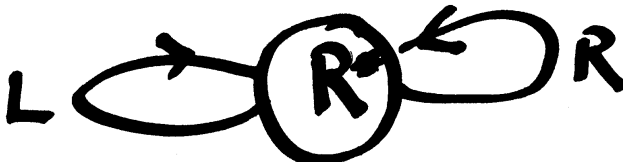
Exercise 3.9. Show that a transformation $\mathcal{A}^* \to \mathcal{A}^*$ can be defined by a simplistic finite state automaton if and only if it can be realized by a sophisticated finite state automaton.

Remark 3.10. Maybe, one should introduce the even more convenient notion of a finite state automaton deluxe where the vertices carry labels in $\mathrm{Maps}(\mathcal{A}, \mathcal{A})$. This generalizes both notions: For simplistic automata, use constant maps; and for sophisticated automata, use permutations. However, it does not add to the computational power of these devices.

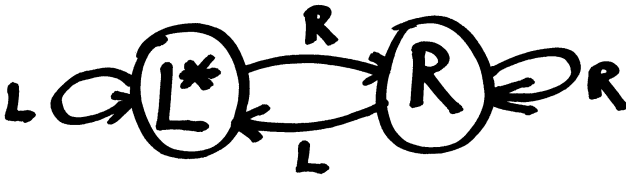
Definition 3.11. A vertex v in a finite state automaton is accessible if there is a directed path from the start vertex to v. An automaton is sophomoric if it has inaccessible vertices.

From now on, all automata will be simplistic but not sophomoric. Let us have a look at some small automata over the alphabet with two letters $\{\mathbf{L}, \mathbf{R}\}$.

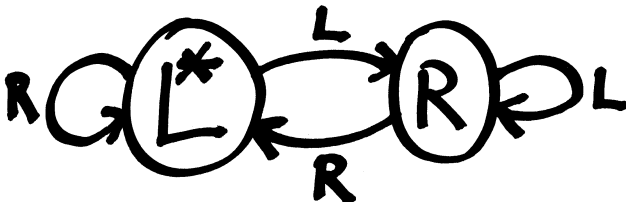
Example 3.12. Here is an automaton, that maps everything to a string of Rs of the same length.



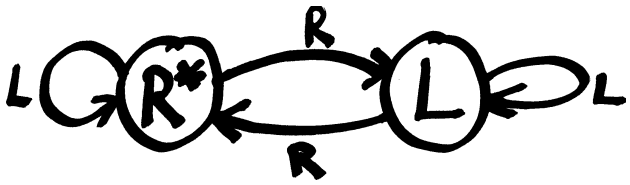
Example 3.13. The identity can be realized with two vertices:



Example 3.14. And here is the swap:



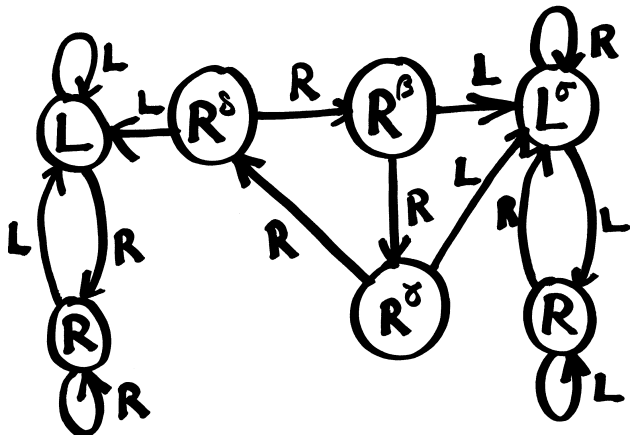
Example 3.15. The twist automorphism



needs two states but has a fairly complicated dynamic.

The picture shows that this automorphism is just a twisted swap whence the name!

Example 3.16. Here is a picture that displays all the generators of $\mathcal{G}_1$ at once. You just have to pick the right vertex as start.



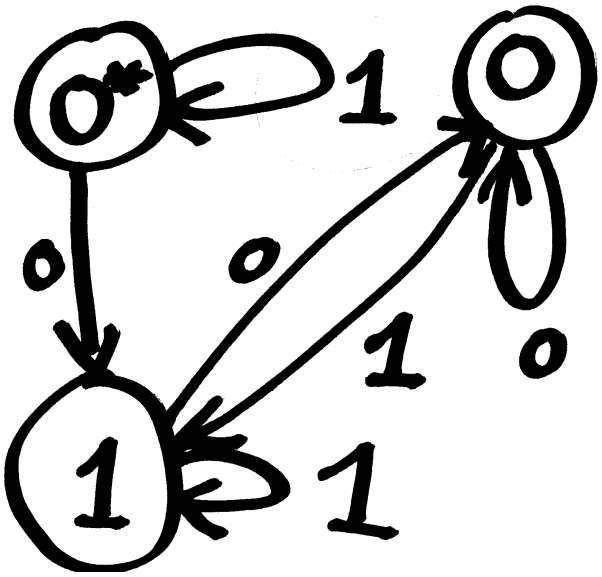
Definition 3.17. A map $\varphi : \mathcal{A}^* \to \mathcal{A}^*$ is finitary if it can be realized by a finite state automaton.

Observation 3.18. *Of course, bijective transformations $\varphi : \mathcal{A}^* \to \mathcal{A}^*$ are tree-automorphisms of a rooted tree where each vertex has precisely $|\mathcal{A}|$ children. Hence we can speak of finitary tree-automorphisms.*

Moreover, any finitary automorphism is recursive. You construct the defining equations from the automation A as follows: For each vertex introduce a variable, and the defining equation will have the children of this vertex in the pair followed by a swap of the identity, depending on the local labels. Rule: the child that prints R *is in the right slot. As the preceding sentence is incomprehensible for those who are not in the know, let us consider an example. Here is the system for the twist (3.15) where t corresponds to the start at* R *and y corresponds to the start at* L*.*

$$\begin{aligned} x &= (y, x)\sigma \\ y &= (y, x) \end{aligned}$$

Exercise 3.19. This is an automaton over $\{0, 1\}$. What does it do? Is the transformation invertible? Is it of infinite order? If the transformation is invertible, find a recursive definition as small as possible.



We will show that finitary bijections form a group. Obviously, we have to construct automata for inverses and products. So let us start with inverses.

Key Idea 3.20. *The basic idea of inverting an automaton is this: if you are at the start vertex and the input sequence gives you a letter, you do not have to look at the edge labels but at the labels of the neighboring vertices. If there is precisely one with that label, you know where to go. While you are on the way, you print out the label of the edge. So we would like to construct the inverse of an automaton just by exchanging the label of an oriented edge with the label of its terminal vertex.*

However, this simple minded procedure might not be allowed! Have a look at example (3.15). The trouble comes from the fact that some vertices have edges with different label pointing to them.

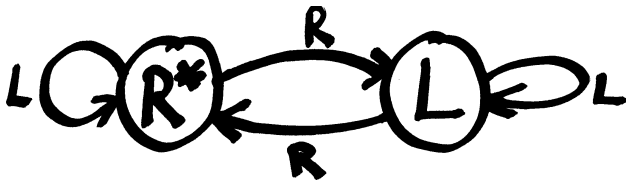
Definition 3.21. A finite state automaton is tidy if every vertex has all its incoming edges given identical labels.

The main tool for inverting an automaton is the blow-up construction:

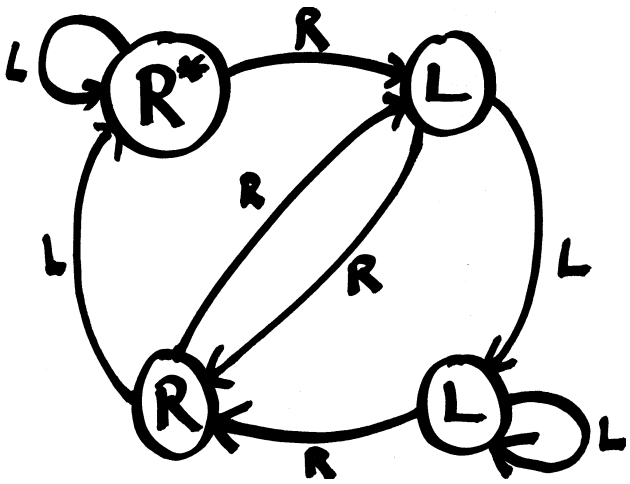
Proposition 3.22 (Blow-Up). *For every finite state automaton, there is an equivalent tidy finite state automaton.*

Proof. We use a cover. So let A be a finite state automaton over the alphabet $\mathcal{A}$. We define a new automaton B on the vertex set $\mathcal{V}_A \times \mathcal{A}$. For an edge $\overrightarrow{e}$ in A with label a pointing from S to T, we glue in $|\mathcal{A}|$ edges in B. They point from the vertices $(S, -)$ to the vertex (T, a).

It is obvious that the result B is tidy and equivalent to A. To understand this, let us give the pictures for the twist-automaton. Here is the untidy version again:



And this is the way to tidy it up:



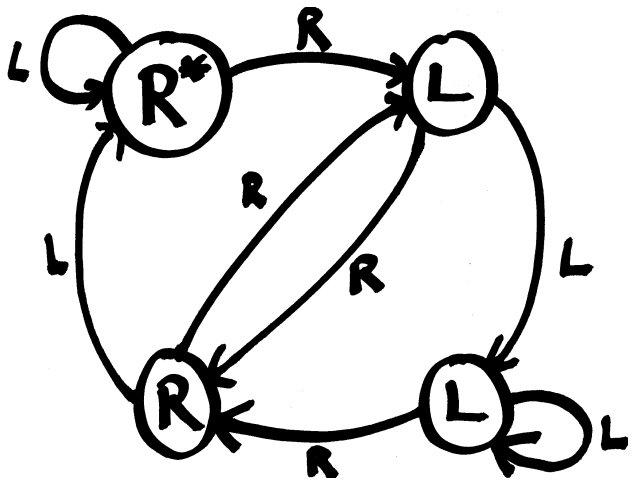
Play and see!! **q.e.d.**

For tidy automata, finding the inverse is no problem.

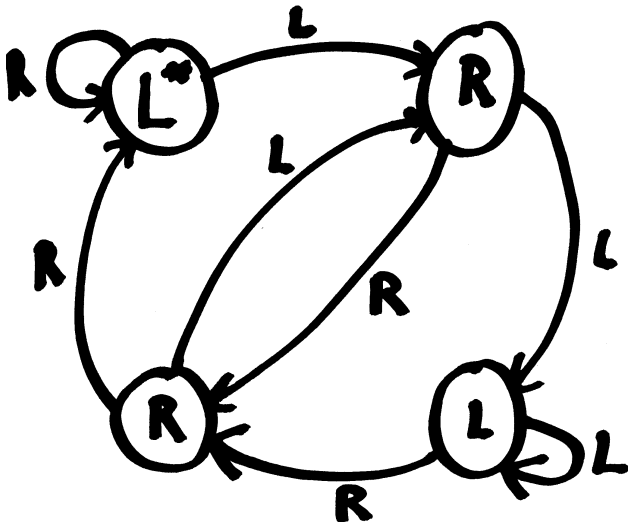
Proposition 3.23. *Let A be a tidy finite state automaton. Define a graph B with edge and vertex labels on the same vertex set by exchanging the label of each oriented edge with the label of its*

terminal vertex. The result is a finite state automaton if and only if A realizes an invertible transformation. In this case B is tidy and realizes the inverse.

Proof. Look at the picture for the twist-automorphism. Here is the tidy twist:



And here is the inverse (the untwist):



No play with it, and see how the key idea (3.20) works. **q.e.d.**

Corollary 3.24. *It is decidable whether a finite state automaton defines an invertible transformation, and if it does an automaton realizing the inverse can be constructed effectively.*

Proposition 3.25. *Suppose the transformations $\varphi : \mathcal{A}^* \to \mathcal{A}^*$ and $\psi : \mathcal{A}^* \to \mathcal{A}^*$ are finitary, i.e., they can both be realized by finite state automata A and B. Then the composition $\psi\varphi$ (second factor acts first on the input!) is finitary, too. Moreover, an automaton realizing the product can be effectively constructed from A and B.*

Proof. The product automaton is constructed on the vertex set $A \times B$. The start vertex is the pair of start vertices. From (S_1, T_1) we have an edge labeled with a to (S_2, T_2) if there is an a-edge from S_1 to S_2 and an edge from T_1 to T_2 that has the same label as v_2. The point is, that this label is the output of A at this stage and therefore guides the computational path in B. **q.e.d.**

Corollary 3.26. *The set of all those bijections that can be realized by finite state automata forms a group.*

Definition 3.27. An automata group over $\mathcal{A}$ is a group of bijective finitary transformations $\mathcal{A}^* \to \mathcal{A}^*$.

Example 3.28. From (3.16) it follows that $\mathcal{G}_1$ is an automata group.

The following lemma introduces an idea that is ubiquitous in the study of finite state automata. The key observation is that a path in a finite graph has to contain a loop if it becomes too long.

Lemma 3.29. *If two finite state automata A and B over the same alphabet are inequivalent, then there is an input of length $\leq |A| \times |B|$ for which their outputs differ.*

Let us first note an immediate consequence.

Corollary 3.30. *There is an algorithm that, taking two finite state automata A_1, and A_2 over the same alphabet is its input, decides if these automata define the same transformation.*

Proof. Let us assume the shortest input sequence that proves the two automata to be inequivalent has length $> |A| \times |B|$. Follow the computational paths in A and B for this sequence. As there are only $|A| \times |B|$ many pairs of states $(S,T) \in A \times B$, one of these pairs is visited twice. Then, however, the part of the input sequence between the two times can be cut out without affecting the rest of the computation. **q.e.d.**

Exercise 3.31. The decision procedure for equivalence of finite state automata based on (3.29) is exponential since you have to test all input sequences of length $\leq |A| \times |B|$. Find an e f f e c t i v e algorithm that decides if two automata are equivalent.

Let us fix some consequences that pertain to Grigorchuk's Group.

Corollary 3.32. *The word problem for finitely generated automata groups is solvable.*

Corollary 3.33. *The word problem in $\mathcal{G}_1$ is solvable.*

Remark 3.34. This solution to the word problem closely parallels the solution to the word problem in finitely generated linear groups. The idea is as follows: Given a word in the generators, just multiply the corresponding matrices and check if the result is the identity matrix. Of course, this presupposes that you can multiply matrices. The problem is that, say, complex numbers do not have finite representations.

To overcome this problem, observe that finitely many matrices have only finitely many coefficients. So all computations really take place in a finite extension of the prime field. Now, there is a little lemma to be proved that says you can always do this extension in two steps: (a) pass to a purely transcendental extension. This field obviously has a computationally effective arithmetic. (b) move on to an algebraic extension of finite degree – this can be done since finitely generated algebraic extensions are

finite. Those extensions can be represented as matrix algebras over their base field. Hence they, too, are computationally effective.