Computability of Bass-Serre structures

Christian Perfect Andrew Duncan

September 14, 2011

Introduction

Grzegorczyk [Grz53] introduced a hierarchy of classes of recursive functions delineated by the number of times unbounded recursion is used in the construction of their members. Following Rabin [Rab60], Cannonito [Can66] defined a notion of computability for groups in the context of this hierarchy. Cannonito and Gatterdam [CG73] [Gat73] showed that, when taking a free product with amalgamation or an HNN extension of group(s) represented by functions computable at a particular level of the hierarchy, one creates a group with corresponding functions computable at most one level higher.

We describe the Grzegorczyk hierarchy and present a clear notation for defining functions within it, along with constructions of many useful elementary functions.

Section 1 defines the notation used in this paper. Section 2 defines the Grzegorczyk hierarchy of computable functions and gives definitions and positions on the hierarchy for many useful functions. Section 3 repeats Serre's formulation of graphs, and Section 4 gives conditions for a tree to be computable at a certain level of the Grzegorczyk hierarchy.

In section 5 we restate [Can66]'s definition of an $\mathcal{E}^n$-computable group and $\mathcal{E}^3$-computable index of the free group on countably many generators.

Section 6 consists of a statement and proof of the main result of this paper – that the fundamental group of a graph of $\mathcal{E}^n$ groups is $\mathcal{E}^{n+1}$-computable, generalising the results of [CG73]. Additionally, we show that the fundamental group of a graph of finitely-generated $\mathcal{E}^n$-computable groups is also $\mathcal{E}^n$-computable, and give an example of a graph of $\mathcal{E}^3$-computable groups whose fundamental group is at least $\mathcal{E}^4$-computable.

Finally, Section 7 defines computability for Stallings' pregroups [Sta71] in the context of the Grzegorczyk hierarchy, generalising the results above.

1 Building blocks

1.1 Notation

When defining expressions or new notation, $E \equiv E' := D$ means I will write E or E' to mean D.

1.2 Constructing functions

Let $\mathbb{Z}_{\geq 0}$ denote the non-negative integers. A function $f(x_1, \ldots, x_n)$ is a map,

$$f : \mathbb{Z}_{\geq 0}^n \to \mathbb{Z}_{\geq 0},$$

taking $n \geq 0$ parameters and resulting in a single value.

Functions in general can take any number of parameters, so it is clearly impractical to write out definitions of each operation for each possible arity. When the arity of a function f doesn't matter, I will write it as $f(\mathbf{x})$, where $\mathbf{x}$ represents an arbitrarily big vector. When a function has to take at least a specific number of parameters $y_1, \ldots, y_n$, and potentially more, I will write something of the form $f(\mathbf{x}, y_1, \ldots, y_n)$

1.2.1 The primitive functions

We will begin by defining the *primitive functions*, which are functions of one of the following three types:

- *The zero function.* The zero function is a nullary function which returns the number 0. For brevity I will write 0 to mean the zero function.
- *The successor function.* The successor function $s(x) := x + 1$ is a unary function which adds 1 to its input.
- *The projection functions.* For every $n > 0$ and every $1 \leq i \leq n$ there is a projection function $P_n^i(x_1, \ldots, x_n) := x_i$. P_n^i is an n-ary function which returns its i^{th} parameter.

There are two operations we shall use to create new functions from these primitives.

1.2.2 Composition

A function h is constructed by *composition* of functions f and g if $h = f \circ g$ where

$$f \circ g(\mathbf{x}) \equiv f(g(\mathbf{x})) := C(f, g)(\mathbf{x}).$$

In general, given an n-ary function f and m-ary functions $g_1, \ldots, g_n$, the composition operator C constructs a new m-ary function

$$f(g_1(\mathbf{x}), \ldots, g_n(\mathbf{x})) := C(f, g_1, \ldots, g_n)(\mathbf{x}).$$

We will omit the generalised parameters for most function definitions, like so:

$$\begin{aligned} f \circ g &:= C(f, g), \\ f(g_1, \ldots, g_n) &:= C(f, g_1, \ldots, g_n). \end{aligned}$$

The mixing of functions of different arities can be dealt with systematically. For every n-ary function f, and every $m > n$, we can define an m-ary function f_m,

$$f_m(x_1, \ldots, x_n, \ldots, x_m) := C(f, P^1_m, \ldots, P^n_m,$$

such that, for all $x_1, \ldots, x_m$, $f_m(x_1, \ldots, x_n, \ldots, x_m) \equiv f(x_1, \ldots, x_n)$.

Variable substitution can also be dealt with systematically:

$$f(x_1, \ldots, y, \ldots, x_n) := f(x_1, \ldots, y(x_1, \ldots, x_n), \ldots, x_n) = C(f, P^1_n, \ldots, y, \ldots, P^n_n).$$

1.2.3 Recursion

A function h is constructed by *recursion* from functions f and g if

$$\begin{aligned} h(\mathbf{x}, 0) &:= f(\mathbf{x}), \\ h(\mathbf{x}, n+1) &:= g(n, h(\mathbf{x}, n), \mathbf{x}), \end{aligned}$$

in which case we write $h = R(f, g)$.

Note that the parameter of recursion n can be made to be in any position by composition with the projection functions. If we have a function $h(x_1, \ldots, x_m, n)$, we can define a new function,

$$f'(x_1, \ldots, x_k, n, x_{k+1}, \ldots, x_m) := C(f, P^1_{m+1}, \ldots, P^k_{m+1}, P^{k+2}_{m+1}, \ldots, P^m_{m+1}, P^{k+1}_{m+1}),$$

so that

$$f'(x_1, \ldots, x_k, n, x_{k+1}, \ldots, x_m) := f(x_1, \ldots, x_m, n).$$

1.3 Further notation

1.3.1 Bound variables

A *bound variable* is one which is used to define a function, not passed as a parameter into it. For instance, in the porjection function P_n^i, n and i are bound variables. Bound variables do not contribute to the arity of a function.

1.3.2 Iteration

A function which evaluates n iterations of a function f, where n is a bound variable, can be constructed by composing f with itself $n-1$ times.

$$f^{(n)}(\mathbf{x}) := \underbrace{C(f, C(f, C(f, \dots C(f, f))}_{n-1 \text{ compositions}})(\mathbf{x}).$$

1.3.3 Infix operators

When using infix notation, binary operators are written between their operands.

For a binary function f_*, representing the operation $*$, we will use $x*y$ to denote $f_*(x, y)$, whenever the former notation is more conventional.

$$x * y := f_*(x, y).$$

1.3.4 Sets

A set $S \subseteq \mathbb{Z}_{\geq 0}^d$ is defined by its characteristic function χ_S.

$$\chi_S(\mathbf{x}) := \begin{cases} 1 & \mathbf{x} \in S, \\ 0 & \mathbf{x} \notin S. \end{cases}$$

1.3.5 Relations

If $R \subseteq \mathbb{Z}_{\geq 0}^d$ is a relation, R is defined by χ_R. In addition, for binary relations I will write

$$xRy := \chi_R(x, y).$$

1.3.6 Logic

The concept of falsity will be represented by the number 0, and all other values will represent truth. With this in mind, predicates $P(\mathbf{x})$ can be written out as ordinary functions once the logical operations have been defined.

2 The Grzegorczyk hierarchy

A class $\mathcal{C}$ is *closed under finite composition* if, whenever $f, g_1, \ldots, g_m$ all belong to $\mathcal{C}$, then $C(f, g_1, \ldots, g_m)$ also belongs to $\mathcal{C}$. Therefore a function defined by finitely many compositions of elements of $\mathcal{C}$ is also an element of $\mathcal{C}$.

A function f is said to be *bounded* by another function g if, for all $\mathbf{x}$, $f(\mathbf{x}) < g(\mathbf{x})$.

A class $\mathcal{C}$ is *closed under bounded recursion* if, whenever f, g, h belong to $\mathcal{C}$ and $R(f, g)(\mathbf{x})$ is bounded by $h(\mathbf{x})$, then $R(f, g)$ also belongs to $\mathcal{C}$.

The *Grzegorczyk hierarchy* is an infinite nested hierarchy of classes $\mathcal{E}^n$ of functions which are each closed under finite composition and bounded recursion.

$\mathcal{E}^0$ consists of just the primitive functions, and whatever else can be obtained by finitely many applications of the composition operator and bounded recursion.

$\mathcal{E}^{n+1}$, for $n \geq 0$, is the smallest class which

- contains all of $\mathcal{E}^n$;
- contains every function obtained by a single unbounded recursion on $\mathcal{E}^n$ functions;
- is closed under finite composition and bounded recursion.

2.1 The stockpile of functions

Below I will give explicit definitions for the operations of elementary arithmetic, plus some other useful things, using the notation defined above. I have tried to use definitions which, though not the most obvious, place functions as low down on the hierarchy as possible.

2.2 $\mathcal{E}^0$ functions

Add a constant n to x by iterating the successor function n times on x:

$$+_n(x) \equiv x + n := s^{(n)}(x). \tag{1}$$

Note that n is a bound variable, that is, there is a different function $+_n$ for each n.

A nullary function returning any single constant can be constructed by composing $+_n$ with z

$$c_n() := 0 + n. \tag{2}$$

From now on I will just write the number n to mean the function c_n, when appropriate.

Decrement by 1:

$$\begin{aligned} dec(0) &:= 0, \\ dec(n+1) &:= n. \end{aligned} \tag{3}$$

Proper subtraction:

$$\begin{aligned} x \dot{-} 0 &:= x, \\ x \dot{-} (y+1) &:= dec(x \dot{-} y). \end{aligned} \tag{4}$$

$x \dot{-} y$ is bounded by $P_2^1 \equiv x$, so belongs to $\mathcal{E}^0$.

Signature:

$$\begin{aligned} sg(0) &:= 0, \\ sg(x+1) &:= 1. \end{aligned} \tag{5}$$

And reverse signature:

$$\begin{aligned} \overline{sg}(0) &:= 1, \\ \overline{sg}(x+1) &:= 0. \end{aligned} \tag{6}$$

Logical AND:

$$\begin{aligned} x \wedge 0 &:= 0, \\ x \wedge (n+1) &:= x. \end{aligned} \tag{7}$$

$\wedge$ is bounded by $P_2^1 \equiv x$, so is in $\mathcal{E}^0$.

Logical OR:

$$\begin{aligned} x \vee 0 &:= x, \\ x \vee (n+1) &:= 1. \end{aligned} \tag{8}$$

$\vee$ is bounded by $s \circ P_2^1 \equiv x + 1$, so is in $\mathcal{E}^0$.

Logical NOT:

$$\neg x := \overline{sg}(x). \tag{9}$$

Ordering:

$$\begin{aligned} x > y &:= x \dot{-} y, \\ x < y &:= y \dot{-} x, \\ x \geq y &:= \neg(y > x), \\ x \leq y &:= \neg(x > y). \end{aligned} \tag{10}$$

Equality:

$$x = y := (x \leq y) \wedge (y \leq x) \tag{11}$$

The smallest of two numbers:

$$\min(x, y) := x \dot{-} (x \dot{-} y). \tag{12}$$

Remainder when dividing x by y:

$$\begin{aligned} 0 \bmod y &:= 0, \\ (x+1) \bmod y &:= rm' \left(y \dot{-} \left((x \bmod y) + 1 \right), x \bmod y \right). \end{aligned} \tag{13}$$

$$\begin{aligned} rm'(0, y) &:= 0, \\ rm'(x+1, y) &:= y + 1. \end{aligned}$$

rm' is bounded by $y + 1$ so belongs to $\mathcal{E}^0$. $x \bmod y$ is defined by recursion on $\mathcal{E}^0$ functions and is bounded by $y - 1$, so belongs to $\mathcal{E}^0$.

2.3 $\mathcal{E}^1$ functions

Adding two numbers together is achieved by a recursion on the successor function.

$$\begin{aligned} x + 0 &:= x, \\ x + (y+1) &:= (x + y) + 1. \end{aligned} \tag{14}$$

Because every function in $\mathcal{E}^0$ is bounded by a function of the form $x + n$, where n is a bound variable, addition must belong to $\mathcal{E}^1$ and not $\mathcal{E}^0$.

We can define another version of $+_n$ where n is unbound:

$$+_n(y) \equiv f_+(n, y) := n + y. \tag{15}$$

This version is in $\mathcal{E}^1$. This is really an abuse of notation, but it will be useful in a little while.

The biggest of two numbers:

$$\max(x, y) := x + (y \dot{-} x). \tag{16}$$

Note that $max(x, y)$ is bounded by one of P_2^1 or P_2^2, but I don't think there's a way of defining it without using addition, so it has to be in $\mathcal{E}^1$.

Absolute difference:

$$|x - y| := (x \dot{-} y) + (y \dot{-} x). \tag{17}$$

Any constant multiple $n \cdot x$ is produced by n iterations of addition:

$$n \cdot x \equiv 0 + n + \cdots + n := +_x^{(n)}(0). \tag{18}$$

So we can also get any positive- and constant-coefficient linear polynomial $a_0 + a_1x_1 + a_2x_2 + \ldots a_nx_n$ by finite compositions.

Remember that the functions we are considering have domain $\mathbb{Z}^d_{\geq 0}$, so computations involving subtraction can go wrong. The key point is that proper subtraction is not associative. For example, $(4+3)\dot{-}5 \neq 4 + (3\dot{-}5)$. For polynomials with negative coefficients, computing all the positive terms and then subtracting the negative terms using proper subtraction will give the closest match to the actual value.

Quotient:

$$\begin{aligned} \left\lfloor \frac{0}{y} \right\rfloor &:= 0, \\ \left\lfloor \frac{x+1}{y} \right\rfloor &:= \left\lfloor \frac{x}{y} \right\rfloor + \overline{sg}\left((x+1) \bmod y\right). \end{aligned} \tag{19}$$

Does x divide y:

$$x \mid y := \overline{sg}(x \bmod y). \tag{20}$$

Bounded minimisation: given a predicate $P(\mathbf{x}, i)$ and a bound y this returns the least i strictly less than y such that $P(\mathbf{x}, i)$ is true. If there is no such i, y is returned.

$$\begin{aligned} \min_{i<0} P(\mathbf{x}, i) &:= 0, \\ \min_{i<y+1} P(\mathbf{x}, i) &:= \left(\min_{i<y} P(\mathbf{x}, i)\right) + sg\left((\min_{i<y} P(\mathbf{x}, y) = y) \wedge (P(\mathbf{x}, y) = 0)\right). \end{aligned} \tag{21}$$

Bounded maximisation:

$$\begin{aligned} \max_{i<y} P(\mathbf{x}, i) &:= y\dot{-}\min_{i<y} P'(\mathbf{x}, y, i), \\ P'(\mathbf{x}, y, i) &:= P(\mathbf{x}, y\dot{-}i). \end{aligned} \tag{22}$$

If P is an $\mathcal{E}^1$ function then so are bounded minimisations and maximisations over P, because they are defined by a recursion over $\mathcal{E}^1$ functions, bounded by y.

Lemma 2.1. Let $A = \{a_0, a_1, \ldots\} \subseteq \mathbb{Z}_{\geq 0}$ be a monotonically increasing integer sequence. Define a function $\sigma_A : \mathbb{Z}_{\geq 0} \to A$ such that $\sigma_A(n) = a_n$.

If σ_A is $\mathcal{E}^n$-computable, $n \geq 1$, then its inverse $\sigma_A^{-1} : a_n \mapsto n$ is also $\mathcal{E}^n$-computable.

Proof. Since A is monotonically increasing, $a_n \geq a_0 + n \geq n$. So we can construct σ_A^{-1} through bounded minimisation on σ_A.

$$\sigma_A^{-1}(n) := \min_{i<n+1} \sigma_A(i) = n$$

Constructed this way, σ_A^{-1} is $\mathcal{E}^n$-computable since the function performing the bounded minimisation is $\mathcal{E}^1$-computable and the equality predicate is $\mathcal{E}^0$-computable. □

2.4 $\mathcal{E}^2$ functions

Multiplication is an unbounded recursion on the addition operation:

$$\begin{aligned} x \cdot 0 &:= 0, \\ x \cdot (y+1) &:= (x \cdot y) + x. \end{aligned} \tag{23}$$

We can get x^n, when n is constant, by n compositions of multiplication by x, and hence any (positive-coefficient) polynomial by finite compositions of addition and multiplication. So polynomials belong to $\mathcal{E}^2$.

Series sum:

$$\begin{aligned} S_f(\mathbf{x}, 0) &\equiv \sum_{i<0} f(\mathbf{x}, i) := 0, \\ S_f(\mathbf{x}, y+1) &\equiv \sum_{i<y+1} f(\mathbf{x}, i) := f(\mathbf{x}, y) + \sum_{i<y} f(\mathbf{x}, i). \end{aligned} \tag{24}$$

Note that the level of the sum on the hierarchy really depends on the level of $f(\mathbf{x}, i)$. However, we can say that if f is in $\mathcal{E}^2$ then $S_f(\mathbf{x}, y)$ is also in $\mathcal{E}^2$ because it entails $y - 1$ additions. If y is a bound variable and f is in $\mathcal{E}^1$, then we could even place the sum in $\mathcal{E}^1$ by constructing the sum through y compositions of addition.

Bounded existential quantifier:

$$\exists_{i<y} P(\mathbf{x}, i) := \sum_{i<y} P(\mathbf{x}, i). \tag{25}$$

Definition by cases: if $P_i(\mathbf{x})$, $0 \leq i \leq k$ is a finite collection of mutually exclusive, exhaustive predicates, then we can define

$$f(\mathbf{x}) := \begin{cases} g_0(\mathbf{x}) & \text{if } P_0(\mathbf{x}), \\ \vdots & \vdots \\ g_k(\mathbf{x}) & \text{if } P_k(\mathbf{x}). \end{cases}$$

In our language, this is:

$$f(\mathbf{x}) := \sum_{i<k+1} g_i(\mathbf{x}) \cdot sg(P_i(\mathbf{x})), \tag{26}$$

where k is a bound variable. If all the P_i and g_i are in $\mathcal{E}^2$, then so is f, because we are just doing $k+1$ multiplications and additions.

The integer square root function $\lfloor x^{\frac{1}{2}} \rfloor$ computes the least integer n such that $n^2 < x$.

$$\lfloor x^{\frac{1}{2}} \rfloor := \min_{n<x} (n \cdot n < x)$$

2.5 $\mathcal{E}^3$ functions

$\mathcal{E}^3$ is where just about all the really useful operations are.

First of all, we can perform exponentiation by unbounded recursion on the multiplication operation:

$$\begin{aligned} x^0 &:= 1, \\ x^{y+1} &:= x \cdot x^y. \end{aligned} \tag{27}$$

Factorial:

$$\begin{aligned} 0! &:= 1, \\ (x+1)! &:= (x+1) \cdot x!. \end{aligned} \tag{28}$$

Series product:

$$\begin{aligned} P_f(\mathbf{x}, 0) \equiv \prod_{i<0} f(\mathbf{x}, i) &:= 1, \\ P_f(\mathbf{x}, y+1) \equiv \prod_{i<y+1} f(\mathbf{x}, i) &:= f(\mathbf{x}, y+1) \cdot \prod_{i<y} f(\mathbf{x}, i). \end{aligned} \tag{29}$$

If f is in $\mathcal{E}^3$ then so is the product $P_f(\mathbf{x}, y)$, as it consists of $y-1$ multiplications. If y is a bound variable, then we can define the product using y compositions of multiplication instead of the recursion, so if f is in $\mathcal{E}^2$ then the series product is too.

Bounded universal quantifier:

$$\forall_{i<y} P(\mathbf{x}, i) := \prod_{i<y} P(\mathbf{x}, i). \tag{30}$$

Primality:

$$x \text{ prime} := (x > 1) \wedge \forall_{z<x} \left((z < 2) \vee \neg (z \mid x) \right). \tag{31}$$

The first prime after x:

$$\text{nextprime}(x) := \min_{i \leq x!+1} (i \text{ prime}) \wedge (i > x). \tag{32}$$

Note that this is an exceptionally inefficient way of finding primes, in terms of number of computations performed.

The n^{th} prime, starting at $n = 0$:

$$\begin{aligned} p_0 \equiv p(0) &:= 2, \\ p_{n+1} \equiv p(n+1) &:= \text{nextprime}(p_n). \end{aligned} \tag{33}$$

Exponent of x in the decomposition of y:

$$[y]_x := \max_{i<y} \left(x^i \mid y\right). \tag{34}$$

2.6 Lists

We will be working with n-tuples, or lists, extensively. All of the list operations can be defined as $\mathcal{E}^3$ functions on numbers by way of the Gödel encoding.

Gödel encoding of an n-tuple:

$$\mathbf{x} \equiv [x_0, \ldots, x_{n-1}] := \prod_{i<n} p_i^{x_i+1}. \tag{35}$$

The $+1$ in the exponent is so we can reliably determine the length of a list. The empty list should be encoded as 1: then it will have length 0. The bold $\mathbf{x}$ is shorthand so I don't have to write out the brackets formation. It shouldn't be confused with the use of $\mathbf{x}$ previously to mean an arbitrary number of parameters in a function application. An encoded list $\mathbf{x}$ is just a single number.

Value of n^{th} position in an encoded list:

$$(\mathbf{x})_i := [\mathbf{x}]_{p_i} - 1. \tag{36}$$

Length of an encoded list:

$$|\mathbf{x}| := \min_{i<\mathbf{x}} \left([\mathbf{x}]_{p_i} = 0\right). \tag{37}$$

The index of the first occurrence of a given value in an encoded list:

$$\text{find}(\mathbf{x}, y) := \min_{i<|\mathbf{x}|} (\mathbf{x})_i = y. \tag{38}$$

Note that $\text{find}(\mathbf{x}, y)$ returns $|\mathbf{x}|$ if y does not occur in the list.

Whether an encoded list contains a given value:

$$y \in \mathbf{x} := \text{find}(\mathbf{x}, y) < |\mathbf{x}|. \tag{39}$$

The largest element of an encoded list:

$$\text{biggest}(\mathbf{x}) := \text{biggest}'(\mathbf{x}, |\mathbf{x}| - 1), \tag{40}$$

$$\begin{aligned} \text{biggest}'(\mathbf{x}, 0) &:= (\mathbf{x})_0, \\ \text{biggest}'(\mathbf{x}, n+1) &:= \max\left((\mathbf{x})_{n+1}, \text{biggest}'(\mathbf{x}, n)\right). \end{aligned}$$

To obtain the sub-list of $\mathbf{x}$ starting at position s and ending at position e:

$$\mathbf{x}[s \ldots e] \equiv \text{slice}(\mathbf{x}, s, e) := \prod_{i=s'}^{e'} (p_{i-s'})^{(\mathbf{x})_i + 1}, \tag{41}$$

$$\begin{aligned} e' &:= \max(0, \min(e, |\mathbf{x}| - 1)), \\ s' &:= \max(0, \min(s, e')). \end{aligned}$$

To append a single value to the end of an encoded list:

$$\text{splice}(\mathbf{x}, n) := \mathbf{x} \cdot p_{|\mathbf{x}|}^{n+1}. \tag{42}$$

To concatenate two encoded lists:

$$\mathbf{x} +\!\!+ \mathbf{y} := \text{splice}(\mathbf{x}, (\mathbf{y})_0) +\!\!+ \mathbf{y}[1 \ldots |\mathbf{y}| - 1]. \tag{43}$$

$\mathbf{x} +\!\!+ \mathbf{y}$ is bounded by $p_{|\mathbf{x}|+|\mathbf{y}|}^{(|\mathbf{x}|+|\mathbf{y}|)(\text{biggest}(\mathbf{x})+\text{biggest}(\mathbf{y}))}$, so is $\mathcal{E}^3$-computable.

3 Graphs

Definition 3.1. A graph is a set of vertices V and a set of directed edges $E = E^+ \cup E^-$, with two associated maps. The first,

$$e \to (\iota(e), \tau(e)),$$

associates each edge with its initial and terminal vertices. The second,

$$e \to \bar{e},$$

is an involution on the edges, which must satisfy the conditions that $\bar{\bar{e}} = e$ and $\iota(\bar{e}) = \tau(e)$, $\forall e \in E$. We will write e_{ij} to mean the edge with $\iota(e_{ij}) = i$ and $\tau(e_{ij}) = j$. For any pair of edges e and $\bar{e}$, one is 'positive' and belongs to E^+, and one is 'negative' and belongs to E^-.

Definition 3.2. A *path* in a graph is a sequence of edges $(e_1, e_2, \ldots, e_n)$ such

that $\tau(e_i) = \iota(e_{i+1})$, $1 \leq i < n$. A path is *reduced* if it does not contain an edge followed by its inverse, that is, if $e_{i+1} \neq \bar{e_i}$, $1 \leq i < n$.

Definition 3.3. A *circuit* is a path $(e_1, \ldots, e_n)$ with $\iota(e_1) = \tau(e_n)$.

Definition 3.4. A graph is *connected* if there exists a path between each pair of its vertices.

4 Computable Trees

Definition 4.1. A tree is a connected, non-empty graph with no reduced circuits.

A *rooted* tree has a particular vertex v_0 said to be at the 'top'. The children of a vertex v are those vertices adjacent to v and further from the root than v. In an *ordered* tree, the set of children of each vertex is ordered, so there is a least, or *leftmost*, child, and so on.

Definition 4.2. An *ordered recursive tree* is a rooted, ordered tree obtained by a recursive process which begins with the root node and adds vertices below existing ones, in effect enumerating the edges.

We can assign a labelling to an ordered recursive tree by giving vertices increasing integer labels corresponding to the order in which they were added to the tree, beginning with 0 for the root. This way, the sequence of labels encountered on a path heading downwards is always increasing.

If we say that the positive edges point downwards, every positive edge can also be given a unique labelling by saying that the edge e_{ij} has label $max(i, j)$. For the purposes of the computation, each negative edge will have the same label as its positive counterpart.

Definition 4.3. Suppose we have a tree $T = (V, E)$ with root vertex v_0. T is *$\mathcal{E}^n$-computable* if there is an $\mathcal{E}^n$-decidable labelling

$$i : V \to \mathbb{Z}_{\geq 0},$$

(we can assume $i(v_0) = 0$) and an $\mathcal{E}^n$-computable 'parent function'

$$\phi_T : i(V) \to i(V),$$

which gives the label of the parent of a vertex. For completeness, we also require that $\phi_T(v_0) = i(v_0) = 0$.

Definition 4.4. A finite ordered recursive tree can be entirely described in a canonical way by a *depth-first walk*. Note that every node has a single parent and a finite ordered set of children. The process on 'visiting' a vertex v_i goes as follows: for each child node v_j, travel along e_{ij} and 'visit' v_j, then travel backwards along $\overline{e_{ij}}$. The sequence of labelled edges travelled along by beginning this process at v_0 is the depth-first walk of the tree.

Note that the root vertex's label will not appear in this sequence because it has no positive edge leading into it.

Because an edge and its inverse have the same label, the first occurrence of a label in the sequence represents the positive edge, and the second occurrence represents the trip back up the negative edge.

The depth-first walk can be encoded as a Gödel number by the method in Section 2.6. We will define a function $\phi_{\mathbf{t}}(n)$ which takes an encoded depth-first walk $\mathbf{t}$ of a tree and gives the label of the parent of the vertex with label n.

4.1 Some functions to work with encoded trees

Let $\mathbf{t}$ be an encoded depth-first walk of a tree.

To find the first and second occurrences of the edge with label n in the walk:

$$\begin{aligned} \mathrm{fst}(\mathbf{t}, n) &:= \mathrm{find}(\mathbf{t}, n), \\ \mathrm{snd}(\mathbf{t}, n) &:= \mathrm{find}\left(\mathbf{t}[\mathrm{fst}(\mathbf{t}, n) + 1 \ldots |\mathbf{t}| - 1], n\right). \end{aligned} \tag{44}$$

The function find is $\mathcal{E}^3$-computable, and snd is bounded by $|\mathbf{t}|$ so is also $\mathcal{E}^3$-computable.

To get the label of the n^{th} child of the root:

$$\begin{aligned} \mathrm{child}(\mathbf{t}, 0) &:= (\mathbf{t})_0, \\ \mathrm{child}(\mathbf{t}, n+1) &:= (\mathbf{t})_{\mathrm{snd}(\mathbf{t}, \mathrm{child}(\mathbf{t}, n))+1}. \end{aligned} \tag{45}$$

child is constructed from $\mathcal{E}^3$ functions and is bounded by the number of vertices in the tree. As the tree is finite, child is $\mathcal{E}^3$-computable.

The depth-first walk of the subtree descended from a vertex v_n is the subsequence found between the two occurrences of n in $\mathbf{t}$.

To find the subtree descended from the n^{th} child node of the root:

$$\mathrm{subtree}(\mathbf{t}, n) := \mathbf{t}[\mathrm{fst}(\mathrm{child}(\mathbf{t}, n)) + 1 \ldots \mathrm{snd}(\mathrm{child}(\mathbf{t}, n)) - 1]. \tag{46}$$

subtree is constructed from $\mathcal{E}^3$ operations so is $\mathcal{E}^3$-computable.

The number of children of the root node:

$$\mathrm{kids}(\mathbf{t}) := \min_{i<\mathbf{t}}(\mathrm{child}(\mathbf{t}, i) = 0). \tag{47}$$

kids is constructed by bounded minimisation on $\mathcal{E}^3$ operations so is $\mathcal{E}^3$-computable.

Whether the root has a child with label n:

$$\text{haschild}(\mathbf{t}, n) := (\min i < \text{kids}(\mathbf{t})\text{child}(\mathbf{t}, i) = n) < \text{kids}(\mathbf{t}). \tag{48}$$

Finally, the parent function $\phi_{\mathbf{t}}$. The idea is basically to do the depth-first walk, remembering which vertex you just came from. If you reach a subtree which doesn't contain n then turn back, or if you reach the required vertex then return the label of the vertex you just came from.

$$\begin{aligned} \phi_{\mathbf{t}}(0) &:= 0, \\ \phi_{\mathbf{t}}(n+1) &:= \text{p}'(\mathbf{t}, n+1, 0). \end{aligned} \tag{49}$$

$$\text{p}'(\mathbf{t}, n, p) := \begin{cases} 0 & n \notin \mathbf{t}, \\ p & \text{haschild}(\mathbf{t}, n), \\ \text{p}'(\text{subtree}(\mathbf{t}, z), n, \text{child}(\mathbf{t}, z)) & \text{otherwise.} \end{cases}$$

$$z := \min_{i < kids(\mathbf{t})} (n \in subtree(\mathbf{t}, i)).$$

We will also need to know if the tree contains an edge e_{ij}:

$$e_{ij} \in \mathbf{t} \Leftrightarrow (\phi_{\mathbf{t}}(i) = j \vee \phi_{\mathbf{t}}(j) = i) \wedge i \neq j \tag{50}$$

Lemma 4.5. A finite ordered tree is $\mathcal{E}^3$ computable.

Proof. As the tree is finite, of course its labelling is an $\mathcal{E}^3$-decidable subset of $\mathbb{Z}_{\geq 0}$, and $\phi_{\mathbf{t}}$ as defined above is $\mathcal{E}^3$-computable. □

5 Computable Groups

Following [Can66], we say a group G is $\mathcal{E}^n$-computable if it has a triple of $\mathcal{E}^n$-computable functions (i, m, j), such that

- $i : G \to \mathbb{Z}_{\geq 0}$ is an injection of G onto an $\mathcal{E}^n$-decidable subset of $\mathbb{Z}_{\geq 0}$;
- $m : i(G) \times i(G) \to i(G)$ computes the product of two group elements, i.e. $m\,(i(g_1), i(g_2)) = i(g_1 g_2)$, $\forall g_1, g_2 \in G$;
- $j : i(G) \to i(G)$ computes the inverse of any element of G.

Note that i itself is not computable but its image has computable characteristic function.

Lemma 5.1. [CG73, 3.1] A free group $F = \langle a_1, \dots \rangle$ on finitely or countably many generators is $\mathcal{E}^3$-computable.

The proof of the above lemma is not important for the purposes of this paper, but the index defined on the free group will be used later in this paper. The index $i(w)$ of a word $w = a_{i_0}^{\alpha_0} \dots a_{i_r}^{\alpha_r}$ is

$$i(w) = \prod_{k=0}^{r} p_k e^{J(i_k, \alpha_k)}.$$

$J(x, y)$ is an $\mathcal{E}^3$-computable integer pairing function $J : \mathbb{Z}_{\geq 0} \times \mathbb{Z} \to \mathbb{Z}_{\geq 0}$ defined in [CG73].

Definition 5.2. [CG73, 2.3] Let $A \subset \mathbb{Z}_{\geq 0}$. Then the class of functions $\mathcal{E}^n(A)$ for $n \geq 2$ (with domain $\mathbb{Z}_{\geq 0}^k$ for aribtrary k and range $\mathbb{Z}_{\geq 0}$) is the smallest class of functions, closed under composition and bounded recursion, containing all of $\mathcal{E}^n$ in addition to

$$c_A(x) := \begin{cases} 0 & \text{if } x \in A, \\ 1 & \text{if } x \notin A. \end{cases} \tag{51}$$

Lemma 5.3. Let $A \subset \mathbb{Z}_{\geq 0}$. If c_A is $\mathcal{E}^m$-computable, then any $\mathcal{E}^n(A)$-computable function, $n \leq m$, is $\mathcal{E}^m$-computable.

Definition 5.4. The *word problem* $\mathrm{WP}(G)$, associated with a group G with generating set X, is the set of all words $w \in X^*$ such that $w =_G 1$. That is, it is the set of all words representing the trivial element of G.

Theorem 5.5. *[CG73, 3.2] For every countable group G there exists $a \subset \mathbb{Z}_{\geq 0}$ such that G is $\mathcal{E}^3(A)$-computable. In particular, the word problem of G is one such A.*

Corollary 5.6. *When $n \geq 3$, WP(G) is $\mathcal{E}^n$-decidable if and only if G is $\mathcal{E}^n$-computable. Also, WP(G) is not $\mathcal{E}^n$-decidable if and only if G is not $\mathcal{E}^n$-computable.*

Definition 5.7. Let H be a subgroup of an $\mathcal{E}^n$-computable group G. H is *$\mathcal{E}^m$-decidable*, with m necessarily larger than or equal to n, if the characteristic function of $i(H) \subseteq i(G)$ is $\mathcal{E}^m$-computable.

Definition 5.8. A homomorphism $\phi : G_1 \to G_2$ is $\mathcal{E}^n$-computable if there exists an $\mathcal{E}^n$-computable function $\hat{\phi} : i_1(G_1) \to i_2(G_2)$ such that

$$\hat{\phi}(i_1(g)) = i_2(\phi(g)), \forall g \in G_1,$$

6 Computability of Bass-Serre groups

Let $\Gamma = (V, E)$ be a connected graph.

Definition 6.1. Associate with each vertex v a vertex group $G_v = \langle X_v \mid R_v \rangle$, with the X_v all pairwise disjoint. Call the set of all vertex groups $\mathbf{G}$.

For each edge $e \in E$ associate two isomorphic *edge groups* $A_e \leq G_{\iota(e)}$ and $B_e \leq G_{\tau(e)}$, with $A_e = B_{\bar{e}}$, and an isomorphism $\phi_e : A_e \to B_e$ satisfiying $\phi_e^{-1} = \phi_{\bar{e}}$.

$(\mathbf{G}, \Gamma)$ is called a *graph of groups*.

Definition 6.2. Let T be a spanning tree of Γ (a subtree of Γ containing every vertex), with a root vertex v_0. Then the *fundamental group* of $(\mathbf{G}, \Gamma)$, with respect to T and v_0, is the group $G = \pi_1(\mathbf{G}, \Gamma, T, v_0)$, defined as follows:

The generators of G are

$$X = \left(\bigcup_{v \in V} X_v \right) \cup \left(\bigcup_{e \in E} \{e\} \right),$$

and the relations are

$$R = \left(\bigcup_{v \in V} R_v \right) \cup \{e^{-1} a e = \phi_e(a), \forall a \in A_e, \forall e \in E\} \cup \{e^{-1} = \bar{e}, \forall e \in E\} \cup \{e = 1, \forall e \in T\}.$$

G can also be defined as the free product of all the vertex groups and the free group on the graph's edges minus the spanning tree, with the appropriate edge groups amalgamated.

$$G = \pi_1(\mathbf{G}, \Gamma, T, v_0) = (*_{v \in V} G_v) \underset{\phi_e}{*} F(E(\Gamma \setminus T)) \tag{52}$$

Definition 6.3. A word $w \in X^*$ is *admissible* if it can be factored in the form

$$w = a_0 e_1 a_1 e_2 \dots a_{n-1} e_n a_n$$

where $(e_1, \dots, e_n)$ is a circuit in Γ with vertex sequence $(v_0, \dots, v_n)$, such that $v_0 = v_n$, $v_i = \iota(e_{i+1}) = \tau(e_i)$, $0 < i < n$, and $a_i \in G_{v_i}$, $i = 0, \dots, n$. The sequence $(a_0, e_1, \dots, a_{n-1}, e_n, a_n)$ is an *admissible sequence* for w.

It will be more convenient from now on to refer to G_{v_i} by G_i.

Lemma 6.4. (Higgins?) Let $w \in X^*$ be an admissible word, and suppose $w =_G 1$. Then either:

- $n = 0$ and $a_0 = 1$ in G_{v_0}, or
- $n > 0$ and there exists $0 < i < n$ such that $e_{i+1} = \bar{e_i}$ and $a_i \in B_{e_i}$.

CP Can't find the relevant lemma in Higgins. Proof is very simple - should I just write it out?

Definition 6.5. Let $\phi_T(n)$ be the parent function for T. The path in T from

v_i to v_j can be found by freely reducing the path $\psi(i,j)$, defined by:

$$\begin{aligned}\psi(0,0) &:= \epsilon,\\ \psi(i,0) &:= (e_{i,p_T(i)}) \cdot \psi(p_T(i)),\\ \psi(0,i) &:= \psi(i,0)^{-1},\\ \psi(i,j) &:= \psi(i,0)\psi(0,j).\end{aligned} \tag{53}$$

The free reduction of $\psi(i,j)$ will be implicit from now on.

What we really want is a function $\pi : G \to X^*$ such that $\pi(g)$ is an admissible word which represents $g \in G$ and whose circuit begins and ends at v_0.

If g belongs to some G_{v_i}, then $\pi(g) = \psi(0,i) \cdot g \cdot \psi(i,0)$. Otherwise, if $g = e_{ij}$, some edge letter, then $\pi(e_{ij}) = \psi(0,i) \cdot e_{ij} \cdot \psi(j,0)$.

Lemma 6.6. Let $w = x_1 \dots x_n$ be a word in X^*, representing the group element $g \in G$. Then by replacing each generator x_i by $\pi(x_i)$ and freely reducing the resulting word, we obtain an admissible word starting at v_0 which is equivalent to w. Call this word $\pi(w)$.

Proof. The new word is equivalent to w because the only letters we are adding are edge letters from the spanning tree, which are all trivial in the presentation of G. Also, it is admissible since each x_i in w is replaced by an admissible word with circuit starting and ending at v_0. □

Theorem 6.7. *Suppose we have $G = \pi_1(\mathbf{G}, \Gamma, T, v_0)$, where Γ has ν vertices. Suppose all the G_i are finitely generated $\mathcal{E}^n$-computable groups for $n \geq 3$. Assume all the edge groups are $\mathcal{E}^n$-decidable, and all the isomorphisms ϕ_e are $\mathcal{E}^n$-computable. Then G is $\mathcal{E}^{n+1}$-computable.*

Proof. We will show that the word problem of G is $\mathcal{E}^{n+1}$-decidable. We can assume that the generating sets of all the vertex groups G_v are disjoint, and that there is an $\mathcal{E}^n$-computable structure (i_v, m_v, j_v) associated with each G_v.

Encode the depth-first walk of the spanning tree T as a Gödel number $\mathbf{t}$ per Section 4.1. Then the vertex-parent function $\phi_{\mathbf{t}}$ is $\mathcal{E}^3$-computable. From now on each vertex group can be referred to by the integer label of the vertex it belongs to.

To define $i(G)$, we will begin by using the first ν^2 numbers to represent potential edges. For $0 \leq a, b \leq \nu - 1$ define $i(e_{a,b}) := a \cdot \nu + b$.

Each vertex group G_i has a generating set X_i. For each $x_{i,j} \in X_i$, define $i(x_{i,j}) := \nu^2 + J(i,j)$.

The rest of $i(G)$ works the same way as the standard free group index described in [CG73, 3.1].

Given a word w, we wish to decide whether $w =_G 1$. First, compute $w' = \pi(w)$. We now need to split w' into an admissible sequence $(a_0, e_1, \dots, e_n, a_n)$.

To do this, we must decide for each letter in w' either which vertex group it is from or if it is an edge letter. Assign to each letter w_i of w' a code as follows: If w_i belongs to G_i, then its code is i. If it is an edge letter, then its code is $\text{biggest}(\mathbf{t}) + 1$. We can then define an $\mathcal{E}^n$-decidable equivalence relation $\approx$ on the indices of the elements of G, whose equivalence classes correspond to the vertex groups plus one more for each edge letter.

As $\approx$ has finitely many equivalence classes, and they are all $\mathcal{E}^n$-decidable, $\approx$ is $\mathcal{E}^n$-decidable.

Each letter w_i in w can then be replaced by $\pi(w_i)$. When working on the encoded version of w, this process is $\mathcal{E}^n$-computable, since $\phi_{\mathbf{t}}$ is computable, and we only want to go as far as producing a list of generators, not a single element of $i(G)$. By concatenating all of the $\pi(w_i)$, we will create an admissible word equivalent to w.

The next task is to split w' into 'syllables', or contiguous subwords. A syllable is either a single edge letter or a word from one of the vertex groups.

From now on, let $\mathbf{w}$ be an encoded admissible word. We will explain how to construct an encoded admissible sequence for $\mathbf{w}$.

Define a function which gives the position of the start of the syllable to which $(\mathbf{w})_i$ belongs:

$$\begin{aligned} \text{backtrack}(\mathbf{w}, 0) &:= 0, \\ \text{backtrack}(\mathbf{w}, i+1) &:= \begin{cases} i+1 & (\mathbf{w})_{i+1} \leq \nu^2, \\ i+1 & (\mathbf{w})_i \not\approx (\mathbf{w})_{i+1}, \\ \text{backtrack}(\mathbf{w}, i) & (\mathbf{w})_i \approx (\mathbf{w})_{i+1}. \end{cases} \end{aligned} \tag{54}$$

The function backtrack is constructed from $\mathcal{E}^n$ operations and is bounded by $|\mathbf{w}|$, so is $\mathcal{E}^n$-computable.

Now, the start of the n^{th} syllable is given by:

$$\begin{aligned} \text{start}(\mathbf{w}, 0) &:= 0, \\ \text{start}(\mathbf{w}, n+1) &:= \min_{\text{start}(\mathbf{w},n)+1 \leq i < |\mathbf{w}|} (\text{backtrack}(\mathbf{w}, i) \neq \text{start}(\mathbf{w}, n)). \end{aligned} \tag{55}$$

The number of syllables can be computed like so:

$$\text{numsyllables}(\mathbf{w}) = \min_{i<|\mathbf{w}|} (\text{start}(\mathbf{w}, i) = |\mathbf{w}|. \tag{56}$$

And now the n^{th} syllable itself can be found:

$$\text{syllable}(\mathbf{w}, n) = \mathbf{w}[\text{start}(\mathbf{w}, n) \ldots \text{start}(\mathbf{w}, n+1) - 1]. \tag{57}$$

The function syllable is constructed from $\mathcal{E}^n$-computable functions and is bounded by $\mathbf{w}$ so is $\mathcal{E}^n$-computable. AJD For a non-edge syllable you obtain a syllable

consisting of a sequence $i(x_1), \ldots, i(x_n)$, where the x_i are in some fixed X_v. The exact procedure for testing to see if this belongs to an edge group needs more explanation.

We can then use the conditions of Lemma 6.4 to determine if $w' = (a_0, e_1, \ldots, e_n, a_n)$, encoded as $\mathbf{w}'$, is trivial.

If $n = 0$, then $w' = 1 \Leftrightarrow a_0 = 1$, which is an $\mathcal{E}^n$-decidable question.

If $n > 0$, then we need to find a sequence of the form $e^{-1}A_e e$ or $eB_e e^{-1}$. The first of these is given by

$$\min_{i<\text{numsyllables}(\mathbf{w}')-2} \begin{aligned}&((\text{syllable}(\mathbf{w}', i) = e \in E)\wedge \\ &(\text{syllable}(\mathbf{w}', i+1) \in B_e)\wedge \\ &(\text{syllable}(\mathbf{w}', i+2) = \text{syllable}(\mathbf{w}', i)^{-1}))\end{aligned} \tag{58}$$

(and the same the other way round for A_e.) Note that since the first syllable must be an edge letter, we can say the last syllable is the inverse of the first by checking its length is 1, then computing its inverse. We don't need to know how to compute the whole multiplication table to do this.

If the result of that calculation is numsyllables$(\mathbf{w}') - 2$ then w is not trivial. Otherwise, we can replace the found sequence $eB_e e^{-1}$ with $\phi_e(\text{syllable}(\mathbf{w}', i+1)$, and try again. The new word is still admissible and has fewer syllables than the original one, so repeated applications of this process will eventually lead to a word of one syllable or a negative answer. Because ϕ_e might increase the index of the word, this recursion raises us up a level on the Grzegorczyk hierarchy.

So the word problem $WP(G)$ is $\mathcal{E}^{n+1}$-decidable, and hence G is $\mathcal{E}^{n+1}$-computable by Corollary 5.6. □

Theorems [CG73, 4.6] and [CG73, 5.3] follow as corollaries of the above result, as free products with amalgamation and HNN extensions can be considered as the fundamental groups of graphs with one edge connecting two vertices and one vertex, respectively.

Lemma 6.8. The nth prime $p_n \approx n \log n$.

Lemma 6.9. $\exp(x, y) := x^y \in \mathcal{E}^3$.

Lemma 6.10. The Gödel encoding of any word of length n on a finite alphabet is bounded above by $p_n^{J(n,n)}$, an $\mathcal{E}^3$ function.

Corollary 6.11. *Same assumptions as Theorem 6.7, but also assume that the edge groups are finitely generated. So each edge homomorphism $\phi_{e_{ij}}$ sends a finitely generated subgroup $E_{ij} \leq G_i$ to another finitely generated subgroup $E_{ji} \leq G_j$.*

Then $G = \pi_1(\mathbf{G}, \Gamma, T, v_0)$ is $\mathcal{E}^n$-computable.

Proof. In the algorithm from Theorem 6.7, the potential for unbounded recursion comes from applying the edge homomorphisms repeatedly. We will show

that in this case the result of applying the edge homomorphisms repeatedly is bounded by an $\mathcal{E}^3$-computable function.

Let w be a word on G of length n. An upper bound on the number of syllables in w is n, so let's say there are n applications of edge homomorphisms.

A homomorphism $\phi_{e_{ij}}$ rewrites generators $x \in X_i$ as words $\phi_{e_{ij}}(x) \in X_j^*$. As $G - i$ is finitely generated, there is a word $\phi_{e_{ij}}(x)$ of maximum, finite, length k_{ij}. So a word $w_i \in E_{ij}$ of length m is rewritten to a word $w_j \in E_j$ of length at most $k_{ij}m$.

Because the graph is finite, we can find a greatest $k \in \{k_{ij}\}$. So after reducing all the syllables (assuming that can be done for the input word), the resulting word in G_0 has length at most $k^n n = L$.

Hence the index of the resulting word is bounded by

$$p_{k^n n}^{J(k^n n, k^n n)},$$

an $\mathcal{E}^3$ function, by Lemma 6.10. So recursion on the edge homomorphisms is bounded by an $\mathcal{E}^n$ function, and so G is $\mathcal{E}^n$-computable.

□

Corollary 6.12. *There exists a graph of $\mathcal{E}^3$-computable groups whose fundamental group is $\mathcal{E}^4$-computable.*

Proof. Let $G_0 = \langle x, y \rangle$ and $G_1 = \langle a, b \rangle$ be two copies of the free group on two generators. Let $(\mathbf{G}, \Gamma)$ be the graph of groups with $\mathbf{G} = \{G_0, G_1\}$ and Γ the graph with a single edge e from v_0 to v_1; so $\Gamma = T = (\{v_0, v_1\}, \{e, \bar{e}\})$.

Let $E_{01} = \langle \{x^{-n} y x^n, n \in \mathbb{N}\} \rangle$, $E_{10} = \langle \{a^{-n} b a^n, n \in \mathbb{N}\} \rangle$ be the (infinitely generated) identified subgroups of G_0 and G_1, respectively. In fact the given generators of the edge subgroups freely generate these subgroups. This means that a homomorphism can be defined merely by defining the images of the generators.

We begin by defining the homomorphism ϕ_e for the generators of G_0 with even powers of x:

$$\phi_e(x^{-2n} y x^{2n}) := a^{(-n)!} b a^{(n)!}, \forall n \in \mathbb{Z}_{\geq 0}.$$

Next we define a function to decide membership of the set of factorial numbers, $\{d!, d \in \mathbb{Z}_{\geq 0}\}$.

$$\begin{aligned} \mathrm{fi}(n) &:= \mathrm{fi}'(n, n). \\ \mathrm{fi}'(n, 0) &:= 0, \\ \mathrm{fi}'(n, d+1) &:= \begin{cases} d+1, & (d+1)! = n, \\ \mathrm{fi}'(n, d), & (d+1)! \neq n. \end{cases} \end{aligned} \tag{59}$$

Helpfully, fi also acts as the inverse of the factorial operation, since it returns

the integer d such that $n = d!$, when such a d exists.

The function $\mathrm{fi}(n)$ is $\mathcal{E}^3$-computable since it is bounded by n and is constructed from $\mathcal{E}^3$-computable functions.

In order for ϕ_e to be an isomorphism, we need to identify the remaining generators of G_0 (those with odd powers of x) with the remaining generators of G_1 (those with non-factorial powers of x).

To do that, we need a bijection between $\mathbb{Z}_{\geq 0}$ and the the non-factorial numbers.

Define

$$\begin{aligned} \mathrm{nf}(1) &:= 3, \\ \mathrm{nf}(n+1) &:= \begin{cases} \mathrm{nf}(n)+1, & \neg(\mathrm{fi}(\mathrm{nf}(n)+1)), \\ \mathrm{nf}(n)+2, & \mathrm{fi}(\mathrm{nf}(n)+1). \end{cases} \end{aligned} \tag{60}$$

Again, $\mathrm{nf}(n)$ is $\mathcal{E}^3$-computable. Its inverse is also $\mathcal{E}^3$-computable since $\mathrm{nf}^{-1}(n) = \min_{i<n}(\mathrm{nf}(i) = n)$.

CP Monotonically increasing integer sequences which are $\mathcal{E}^n$-computable, $n \geq 1$, always have $\mathcal{E}^n$-computable inverses. Is that worth mentioning somewhere?

Define

$$\phi_e(x^{-(2n-1)}yx^{2n-1}) := a^{-\mathrm{nf}(n)}ba^{\mathrm{nf}(n)}, \forall n \in \mathbb{Z}_{\geq 0}.$$

Now we can define $\phi_{\bar{e}}$ to be the inverse of ϕ_e.

$$\phi_{\bar{e}}(a^{-n}ba^n) := \begin{cases} x^{-2\cdot\mathrm{fi}(n)}yx^{2\cdot\mathrm{fi}(n)}, & \mathrm{fi}(n), \\ x^{-\mathrm{nf}^{-1}(n)}yx^{\mathrm{nf}^{-1}(n)}, & \neg\,\mathrm{fi}(n). \end{cases}$$

Note that since G_0 and G_1 are $\mathcal{E}^3$-computable, the edge isomorphism is $\mathcal{E}^3$-computable in both directions.

Construct the index for $G = G_0 \underset{\phi_e}{*} G_1$ by following the method in 6.7.

Consider a word w of the form

$$w = ea^{-1}e^{-1}\dots x^{-1}ea^{-1}e^{-1}x^{-2n}yx^{2n}eae^{-1}x\dots eae^{-1}.$$

In order to decide if this word is trivial, it must be simplified to something of the form $x^{-m}yx^m$. Start with the central subword $x^{-2n}yx^{2n}$. Each conjugation by $eae^{-1}x$ causes the number m to be raised to the factorial of itself. Since there can be arbitrarily many such conjugations, the final value of m will not be bounded by any $\mathcal{E}^3$ function.

As m depends on the input, the length of the simplified word can only be computed by unbounded recursion on an $\mathcal{E}^3$ function, so is $\mathcal{E}^4$-computable and not $\mathcal{E}^3$-computable.

Hence, the word problem of G is $\mathcal{E}^4$-computable, and by Corollary 5.6, G is $\mathcal{E}^4$-computable. □

7 Stallings' pregroups and their universal groups

We now turn to the notion of pregroups in the sense of Stallings, [Sta71], [Sta87]. A *pregroup* $P = (P, D, m, \varepsilon, {}^{-1})$ consists of a set P, a distinguished element ε, a subset $D \subset P \times P$, equipped with a partial multiplication $m : D \to P$, $(a, b) \mapsto ab$, and an involution (or *inversion*) $P \to P$, $a \mapsto a^{-1}$, satisfying the following axioms for all $a, b, c, d \in P$. (By "ab is defined" we mean to say that $(a, b) \in D$ and $m(a, b) = ab$.)

(P1) $a\varepsilon$ and εa are defined and $a\varepsilon = \varepsilon a = a$;

(P2) $a^{-1}a$ and aa^{-1} are defined and $a^{-1}a = aa^{-1} = \varepsilon$;

(P3) if ab is defined, then so is $b^{-1}a^{-1}$, and $(ab)^{-1} = b^{-1}a^{-1}$;

(P4) if ab and bc are defined, then $(ab)c$ is defined if and only if $a(bc)$ is defined, in which case
$$(ab)c = a(bc);$$

(P5) if ab, bc, and cd are all defined then either abc or bcd is defined.

It is shown in [Hoa88] that (P3) follows from (P1), (P2), and (P4), hence can be omitted. From now on, when we write $ab = c \in P$, we mean that a, b and c are in P, $(a, b) \in D$ and $m(a, b) = c$.

The *universal group* $\mathbf{U}(P)$ of the pregroup P can be defined as the quotient monoid

$$\mathbf{U}(P) = P^* / \{\, ab = c \mid m(a, b) = c \,\},$$

where P^* denotes the free monoid on P. Note that in $\mathbf{U}(P)$ the identity element $\varepsilon \in P$ is identified with the empty word $1 \in P^*$. The elements of $\mathbf{U}(P)$ may therefore represented by finite sequences $(a_1, \ldots, a_n)$ of elements of P such that $a_i a_{i+1}$ is not defined in P, for $1 \leq i < n$, and $a_i \neq \varepsilon$, for all i. (The last condition is necessary only to exclude the sequence (ε).) Such sequences are called *P-reduced* sequences. Since every element in $\mathbf{U}(P)$ has an inverse, it is clear that $\mathbf{U}(P)$ forms a group.

If Σ is any set, then the disjoint union $P = \{\varepsilon\} \cup \Sigma \cup \overline{\Sigma}$ where $\overline{\Sigma}$ is a copy of Σ yields a pregroup with involution given by $\overline{\varepsilon} = \varepsilon$, $\overline{\overline{a}} = a$, for all $a \in \Sigma$, such that $p\overline{p} = \varepsilon$, for all $p \in P$. In this case the universal group $\mathbf{U}(P)$ is nothing but the free group $F(\Sigma)$.

The universal property of $\mathbf{U}(P)$ holds trivially, namely the canonical morphism of pregroups $P \to \mathbf{U}(P)$ defines the left-adjoint functor to the forgetful functor from groups to pregroups.

7.1 Stallings' Theorem

Stallings [Sta71] showed that composition of the inclusion map $P \to P^*$ with the standard quotient map $P^* \to \mathbf{U}(P)$ is injective, where P^* is the free monoid on P. His construction involves defining an equivalence relation, which we shall now describe, on the set of P-reduced words. First define the binary relation $\sim$ on the set P-reduced sequences of elements of P by

$$(a_1, \ldots, a_i, a_{i+1}, \ldots, a_n) \sim (a_1, \ldots, a_i c, c^{-1} a_{i+1}, \ldots, a_n),$$

provided $(a_i, c), (c^{-1}, a_{i+1}) \in D$. Then Stallings' equivalence relation $\approx$ is the transitive closure of $\sim$.

Canonical examples of pregroups arise from free products of groups with almagamation and HNN-extensions (see [DDM10] for details).

The following is the principal result on the universal groups of pregroups.

Theorem 7.1 (Stallings [Sta71])**.** *Let P be a pregroup. Then:*

1) Every element of $\mathbf{U}(P)$ can be represented by a P-reduced sequence;

2) any two P-reduced sequences representing the same element are $\approx$ equivalent, in particular they have the same length;

3) P embeds into $\mathbf{U}(P)$.

If $\mathbf{a} = (a_1, \ldots, a_i, a_{i+1}, \ldots, a_n)$ is a word in P^* and $a_i a_{i+1} = b \in P$ then we say that the word $(a_1, \ldots, b, \ldots, a_n)$ is obtained from $\mathbf{a}$ by *elementary P-reduction.* We also say that the empty word 1 is obtained from the word (ε), of length one, by elementary P-reduction. If $\mathbf{b}$ is obtained from a word $\mathbf{a}$ by a finite sequence of elementary P-reductions then we say that $\mathbf{b}$ is obtained from $\mathbf{a}$ by *P-reduction* and that $\mathbf{b}$ is a *P-reduction* of $\mathbf{a}$. (Note that, unless $\mathbf{b} = 1$, such a reduction can always be carried out without ever replacing and occurrence of ε with 1.) In particular the following lemma follows from statement 2 of Theorem 7.1.

Lemma 7.2. Let $\mathbf{c} = (c_1, \ldots, c_m)$ and $\mathbf{d} = (d_1, \ldots, d_n)$ be P-reduced words. Then $\mathbf{c} = \mathbf{d}$ in $\mathbf{U}(P)$ if and only if $n = m$ and 1 is a P-reduction of the word $(d_m^{-1}, \ldots, d_1^{-1}, c_1, \ldots, c_m)$.

If $\mathbf{a} \in P^*$ and both $\mathbf{b}$ and $\mathbf{c}$ are P-reduced, P-reductions of $\mathbf{a}$ then it follows from Theorem 7.1 that $\mathbf{b} \approx \mathbf{c}$.

Rimilinger [Rim87a] (see also Rimilinger [Rim87b] and Hoare [Hoa88]) has shown that,

- given any graph of groups with fundamental group G, a pregroup P may be constructed such that $\mathbf{U}(P) = G$; and conversely that
- given a pregroup P wich satisfies a condition called "finite height" then a graph of groups with fundamental group $\mathbf{U}(P)$ may be constructed.

7.2 Pregroups and the Grzegorczyk hierarchy

Let $P = (P, D, m, \varepsilon, {}^{-1})$ be a pregroup and let i be an injective function $i : P \longrightarrow \mathbb{Z}_{\geq 0}$. (In this case P must be countable.) Define

- $D' = \{(x, y) \in \mathbb{Z}_{\geq 0} \times \mathbb{Z}_{\geq 0} : x = i(p), y = i(q), (p, q) \in D\}$,
- the function $m' : D' \longrightarrow \mathbb{Z}_{\geq 0}$, such that $m'(i(p), i(q)) = i(m(p, q))$, for all $(p, q) \in D$, and
- the function j given by $j(i(p)) = i(p^{-1})$, for all $p \in P$.

Then P is said to have *index* (i, m', j).

Definition 7.3. The pregroup P is $\mathcal{E}^n(A)$-*computable*, with respect to index (i, m', j), if

1. $i(P)$ is $\mathcal{E}^n(A)$-decidable;
2. D' is $\mathcal{E}^n(A)$-decidable;
3. m' is $\mathcal{E}^n(A)$-computable and
4. j is $\mathcal{E}^n(A)$-computable.

AJD Should results be stated for $\mathcal{E}^n(A)$ or just $\mathcal{E}^n$? From now on I'll drop A.

Theorem 7.4. *Let P be a $\mathcal{E}^n$-computable pregroup. Then the universal group $\mathbf{U}(P)$ of P is $\mathcal{E}^{n+1}$-computable.*

In order to prove this theorem we first introduce some functions and then use these to construct an index for $\mathbf{U}(P)$.

7.3 Functions for pregroups

In this section we shall use the notation and constructions of Section 2.6. As we shall mostly be working with encoded lists we use $\mathbf{a}$ to represent both an encoded list $\mathbf{a}$ or an arbitrary non-negative integer $\mathbf{a}$, as appropriate to the context. Let $i : P \longrightarrow \mathbb{Z}_{\geq 0}$ be an injective function and assume that $i(P)$ is $\mathcal{E}^n$-decidable. We shall assume that $i(\varepsilon) = 1$. Elements of P^* are regarded as finite sequences of elements of P. Define an indexing function $i_1 : P^* \longrightarrow \mathbb{Z}_{\geq 0}$ by

$$i_1(a_0, \ldots, a_m) = [i(a_0), \ldots, i(a_m)]. \tag{61}$$

Then i_1 is injective; in particular, by definition of the Gödel number of the empty sequence, $i_1(1) = 1$, while, on the other hand, $i_1(\varepsilon) = 4$.

The function i_1 is $\mathcal{E}^n$-decidable: the characteristic function of $i_1(P^*)$ is the function c such that

$$c(\mathbf{a}) = \bigwedge_{s=0}^{|\mathbf{a}|-1} (\mathbf{a})_s \in i(P).$$

We also define functions corresponding to multiplication in P^* and to the extension of the inversion operation on P to P^*. That is we define $m_1 : \mathbb{Z}_{\geq 0} \times \mathbb{Z}_{\geq 0} \longrightarrow \mathbb{Z}_{\geq 0}$ and $j_1 : \mathbb{Z}_{\geq 0} \longrightarrow \mathbb{Z}_{\geq 0}$ by

$$m_1(i_1(\mathbf{u}), i_1(\mathbf{v})) = i_1(\mathbf{u}) +\!\!+ i_1(\mathbf{v})) \tag{62}$$

and

$$j_1(i_1(u_0, \ldots, u_m)) = i_1(j(u_m), \ldots, j(u_0)). \tag{63}$$

Then m_1 and j_1 are $\mathcal{E}^n$-computable functions, $m_1((i_1(\mathbf{u}), i_1(\mathbf{v})) = i_1(\mathbf{uv})$ and $j_1(i_1(u_0, \ldots, u_m)) = i_1(u_m^{-1}, \ldots, u_0^{-1})$, as required.

Next we define functions to work with P-reduction of encoded words. Define $\mathrm{Dfind} : \mathbb{Z}_{\geq 0} \longrightarrow \mathbb{Z}_{\geq 0}$ by

$$\mathrm{Dfind}(\mathbf{a}) = \min_y(y < |\mathbf{a}| \wedge ((\mathbf{a})_y, (\mathbf{a})_{y+1}) \in D'). \tag{64}$$

As D' is $\mathcal{E}^n$-decidable and the minimisation is bounded this is an $\mathcal{E}^n$-computable function.

Next define the predicate Preduced on $\mathbb{Z}_{\geq 0}$ by

$$\mathrm{Preduced}(\mathbf{a}) = \mathrm{find}(\mathbf{a}, i(\varepsilon)) = |\mathbf{a}| \wedge \mathrm{Dfind}(\mathbf{a}) = |\mathbf{a}|. \tag{65}$$

Again this is $\mathcal{E}^n$-computable and we have $\mathrm{Preduced}(i_1(\mathbf{u})) = 1$ if and only if $\mathbf{u}$ is a P-reduced word in P^*.

Now we construct a function to perform P-reduction on an encoded word. This function will perform elementary P-reduction at the leftmost position where it is possible in a given word. As pointed out in Section 7.1 the order in which P-reductions are performed does not affect the equivalence class of the P-reduced word that is produced, so there will be no need to consider any other possible sequences of P-reduction.

Define a function $\mathrm{leftm} : \mathbb{Z}_{\geq 0} \longrightarrow \mathbb{Z}_{\geq 0}$ by

$$\mathrm{leftm}(\mathbf{a}) = \begin{cases} 1, & \text{if } \mathbf{a} = [1], \text{ or } [\varepsilon] \\ \mathbf{a}[0 \cdots \mathrm{Dfind}(\mathbf{a}) - 1] +\!\!+ \\ m'((\mathbf{a})_{\mathrm{Dfind}(\mathbf{a})}, (\mathbf{a})_{\mathrm{Dfind}(\mathbf{a})+1}) +\!\!+ \\ \mathbf{a}[\mathrm{Dfind}(\mathbf{a}) + 2 \cdots |\mathbf{a}| - 1], & \text{otherwise.} \end{cases} \tag{66}$$

This is an $\mathcal{E}^n$-computable function.

Now define $\mathrm{Predn} : \mathbb{Z}_{\geq 0} \times \mathbb{Z}_{\geq 0} \longrightarrow \mathbb{Z}_{\geq 0}$ by

$$\begin{aligned} \mathrm{Predn}(\mathbf{a}, 0) &= \mathbf{a}, \\ \mathrm{Predn}(\mathbf{a}, n+1) &= \mathrm{leftm}(\mathrm{Predn}(\mathbf{a}, n)). \end{aligned} \tag{67}$$

As Predn is defined by primitive recursion using $\mathcal{E}^n$-computable functions it is $\mathcal{E}^{n+1}$-computable. As the value of $m'(x, y)$ is not necessarily bounded by a $\mathcal{E}^n$-computable function the recursion is not in general bounded.

Finally define Preduce : $\mathbb{Z}_{\geq 0} \longrightarrow \mathbb{Z}_{\geq 0}$ by

$$\text{Preduce}(\mathbf{a}) = \text{Predn}(\mathbf{a}, |\mathbf{a}|). \tag{68}$$

Now if $\mathbf{u} \in P^*$ and $\mathbf{v}$ is a P-reduced word obtained from $\mathbf{u}$ by P-reduction, where reduction is always made at the leftmost possible position, then $\text{Preduce}(i_1(\mathbf{u}) = i_1(\mathbf{v})$.

In order to recognize when two encoded words represent elements of the same $\approx$ equivalence class we define the relation Interleaven $\subseteq \mathbb{Z}_{\geq 0} \times \mathbb{Z}_{\geq 0}$ defined as

$$\begin{aligned} \text{Interleaven}(\mathbf{a}, \mathbf{b}) = \text{Preduced}(\mathbf{a}) \wedge \text{Preduced}(\mathbf{b}) \\ \wedge |\mathbf{a}| = |\mathbf{b}| \wedge \text{Preduce}(m_1(j_1(\mathbf{b}), \mathbf{a})) = 1. \end{aligned} \tag{69}$$

This is an $\mathcal{E}^{n+1}$-decidable relation and from Theorem 7.1 and Lemma 7.2 we have, for P-reduced words $\mathbf{u}$ and $\mathbf{v}$, $\mathbf{u} = \mathbf{v} \in \mathbf{U}(P)$ if and only if $|\mathbf{u}| = |\mathbf{v}|$ and $\mathbf{v}^{-1}\mathbf{u}$ P-reduces to the empty word, if and only if Interleaven$(i_1(\mathbf{u}), i_1(\mathbf{v}))$. We shall write $m \approx n$, for integers m, n such that Interleaven(m, n).

7.4 Proof of Theorem 7.4

We define a function $i_2 : \mathbb{Z}_{\geq 0} \longrightarrow \mathbb{Z}_{\geq 0}$ which will be used to construct an index for $\mathbf{U}(P)$.

$$i_2(\mathbf{a}) = \min_y (y < \mathbf{a} \wedge y \in i_1(P) \wedge \text{Preduced}(y) \wedge y \approx \mathbf{a}). \tag{70}$$

Now define functions $i_G : P^* \longrightarrow \mathbb{Z}_{\geq 0}$, $m_G : i_G(P^*) \times i_G(P^*) \longrightarrow \mathbb{Z}_{\geq 0}$ and $j_G : i_G(P^*) \longrightarrow \mathbb{Z}_{\geq 0}$ by

$$i_G(\mathbf{u}) = i_2 i_1(\mathbf{u}), \tag{71}$$

$$m_G(\mathbf{a}, \mathbf{b}) = i_2(\text{Preduce}(m_1(\mathbf{a}, \mathbf{b})) \text{ and} \tag{72}$$

$$j_G(\mathbf{a}) = i_2 j_1(\mathbf{a}), \tag{73}$$

for all for all P-reduced words $\mathbf{u} \in P^*$ and $\mathbf{a}$, $\mathbf{b} \in i_G(P^*)$. Theorem 7.4 then follows from the next Proposition.

Proposition 7.5. *The group $G = \mathbf{U}(P)$ has $\mathcal{E}^{n+1}$ index (i_G, m_G, j_G).*

Proof. The group G consists of equivalence classes of P-reduced words under the relation $\approx$. If $\mathbf{u}$ and $\mathbf{v}$ are P-reduced words with $\mathbf{u} \approx \mathbf{v}$ then $i_G(\mathbf{u}) = i_2 i_1(\mathbf{u}) = i_2 i_1(\mathbf{v}) = i_G(\mathbf{v})$, by construction. Therefore i_G is a well defined function on G. Moreover if $\mathbf{u}$ and $\mathbf{v}$ are P-reduced words such that $i_G(\mathbf{u}) = i_G(\mathbf{v})$ then both $\mathbf{u}$ and $\mathbf{v}$ are equivalent to some $\mathbf{w}$, so i_G is injective. Let R be the set of P-reduced words and let S be the image $i_G(R)$ of R under i_G. Then, for $\mathbf{x} \in \mathbb{Z}_{\geq 0}$,

$$c_B(\mathbf{x}) = \mathbf{x} \in i_1(P_1) \wedge \text{Preduced}(\mathbf{x}) \wedge i_2(\mathbf{x}) = \mathbf{x},$$

so c_B is $\mathcal{E}^{n+1}$-computable. Hence $i_G(R) = i_G(G)$ is $\mathcal{E}^{n+1}$-decidable.

As i_2 is $\mathcal{E}^{n+1}$-computable, so are m_G and j_G. It therefore remains to show that, for all $\mathbf{u}, \mathbf{v} \in R$,

$$m_G(i_G(\mathbf{u}), i_G(\mathbf{v})) = i_G(\mathbf{u} \cdot \mathbf{v}) \text{ and } j_G(i_G(\mathbf{u})) = i_G(\mathbf{u}^{-1}),$$

where $\mathbf{u} \cdot \mathbf{v}$ denotes the product of $\mathbf{u}$ and $\mathbf{v}$ in G.

Let $\mathbf{u}_0$ and $\mathbf{v}_0$ be P-reduced words such that $i_G(\mathbf{u}_0) = i_2(i_1(\mathbf{u}_0)) = i_1(\mathbf{u}_0) = i_G(\mathbf{u})$ and similarly $i_2(i_1(\mathbf{v}_0)) = i_1(\mathbf{v}_0) = i_G(\mathbf{v})$. Let $\mathbf{a} = i_1(\mathbf{u}_0)$ and $\mathbf{b} = i_1(\mathbf{v}_0)$; so $i_2(\mathbf{a}) = \mathbf{a}$ and $i_2(\mathbf{b}) = \mathbf{b}$. The product $\mathbf{u} \cdot \mathbf{v}$ is the $\approx$ equivalence class of the P-reduction of the concatenation of $\mathbf{u}$ and $\mathbf{v}$. Therefore $i_G(\mathbf{u} \cdot \mathbf{v}) = i_2(\text{Preduce}(m_1(i_1(\mathbf{u}), i_1(\mathbf{v}))))$. Since $\mathbf{u} =_G \mathbf{u}_0$ and $\mathbf{v} =_G \mathbf{v}_0$, we have $\mathbf{u} \cdot \mathbf{v} =_G \mathbf{u}_0 \cdot \mathbf{v}_0$, so $\text{Preduce}(m_1(i_1(\mathbf{u}), i_1(\mathbf{v})) = \text{Preduce}(m_1(\mathbf{a}, \mathbf{b}))$. Hence $i_G(\mathbf{u} \cdot \mathbf{v}) = i_2(\text{Preduce}(m_1(\mathbf{a}, \mathbf{b}))) = m_G(\mathbf{a}, \mathbf{b}) = m_G((i_G(\mathbf{u}), i_G(\mathbf{v}))$, as required.

A similar argument shows that $j_G(i_G(\mathbf{u})) = i_G(\mathbf{u}^{-1})$. □

7.5 Questions

1. Under which conditions is $\mathbf{U}(P)$ $\mathcal{E}^n$-computable?
2. How does the index found in this section compare with that found for Bass-Serre groups in earlier sections?
3. ....

References

[Can66] Frank B. Cannonito. Hierarchies of computable groups and the word problem. *J. Symbolic Logic*, 31:376–392, 1966.

[CG73] F. B. Cannonito and R. W. Gatterdam. The computability of group constructions. I. In *Word problems: decision problems and the Burnside problem in group theory (Conf., Univ. California, Irvine, Calif. 1969; dedicated to Hanna Neumann)*, pages 365–400. Studies in Logic and the Foundations of Math., Vol. 71. North-Holland, Amsterdam, 1973.

[DDM10] V. Diekert, A. J. Duncan, and A. G. Miasnikov. Geodesic rewriting systems and pregroups. In O. Bogopolski, I. Bumagin, O. Kharlampovich, and E. Ventura, editors, *Combinatorial and Geometric Group Theory, Dortmund and Carleton Conferences*, Trends in Mathematics, pages 55–91, Basel, 2010. Birkhäuser.

[Gat73] R. W. Gatterdam. The computability of group constructions. II. *Bull. Austral. Math. Soc.*, 8:27–60, 1973.

[Grz53] Andrzej Grzegorczyk. Some classes of recursive functions. *Rozprawy Mat.*, 4:46, 1953.

[Hoa88] A. H. M. Hoare. Pregroups and length functions. *Math. Proc. Cambridge Philos. Soc.*, 104(1):21–30, 1988.

[Rab60] Michael O. Rabin. Computable algebra, general theory and theory of computable fields. *Trans. Amer. Math. Soc.*, 95:341–360, 1960.

[Rim87a] Frank Rimlinger. Pregroups and Bass-Serre theory. *Mem. Amer. Math. Soc.*, 65(361):viii+73, 1987.

[Rim87b] Frank Rimlinger. A subgroup theorem for pregroups. In *Combinatorial group theory and topology (Alta, Utah, 1984)*, volume 111 of *Ann. of Math. Stud.*, pages 163–174. Princeton Univ. Press, Princeton, NJ, 1987.

[Sta71] John Stallings. *Group theory and three-dimensional manifolds*. Yale University Press, New Haven, Conn., 1971. A James K. Whittemore Lecture in Mathematics given at Yale University, 1969, Yale Mathematical Monographs, 4.

[Sta87] John R. Stallings. Adian groups and pregroups. In *Essays in group theory*, volume 8 of *Math. Sci. Res. Inst. Publ.*, pages 321–342. Springer, New York, 1987.