An exploration of the word problem in various algebraic structures

Daniil Matveev, Kristen Dawson, John McBride

May 2024

1 Introduction

A common question explored by mathematicians has surrounded the idea of normal forms, or the irreducible forms of mathematical objects. Since the same structure may appear in different ways, it is helpful to know when two objects are in some sense equivalent. This question can appear in the form of determining if some function can be expressed as a polynomial equation (regular function). It also appears in an exciting example we have seen in learning Galois theory was the constructibility of the 17-gon,

$$16\cos\left(\frac{2\pi}{17}\right) = -1 + \sqrt{17} + \sqrt{34 - 2\sqrt{17}} + 2\sqrt{17 + 3\sqrt{17} - \sqrt{34 - 2\sqrt{17})} - 2\sqrt{34 + 2\sqrt{17}}.$$

In group theory, this decision problem is frequently present when determining if two group elements given by words formed from a finite list of generators are the same. This is commonly referred to as "the word problem", and has been studied by algebraists from the 20th century on. In general, this is an example of an undecidable problem. However, extensive work has been done to uncover the structures necessary to allow us to answer this question.

To establish the algorithmic undecidability of a problem, a formal definition of an algorithm is crucial. Section 2 introduces Turing machines, a cornerstone of computational models, to unveil the fundamental limitations of algorithms. Leveraging this theory, Section 3 demonstrates the undecidability of the word problem for semigroups in general. Section 4 replicates this result for groups, while introducing free groups as the first non-trivial example of a structure with a solvable word problem. We will then explore Coxeter groups, a particular group for which this question becomes decideable, before moving to a linear-time algorithm for a particular type of Coxeter group. Code for animations and implementations can be found here.

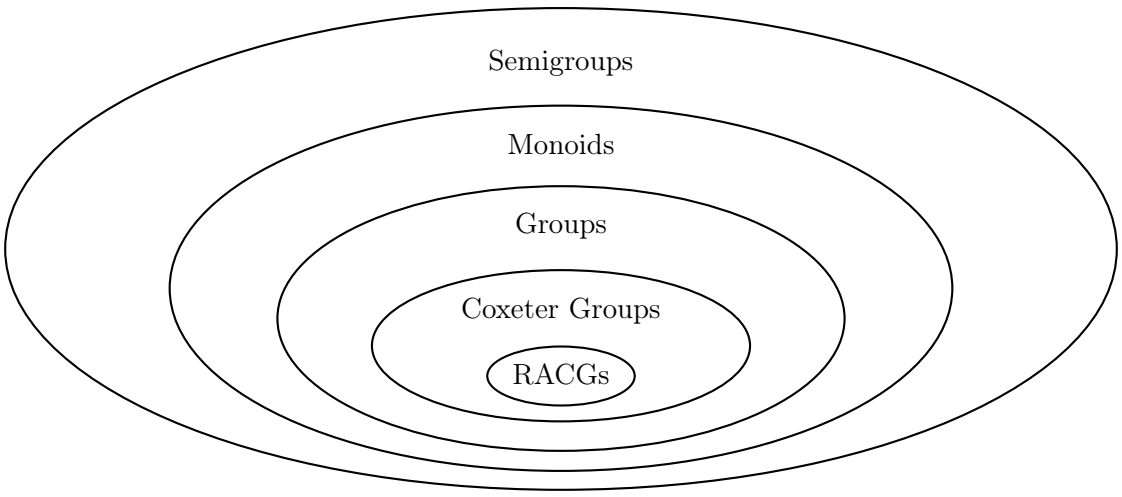


Fig. 1

2 Turing machines and halting problem

2.1 Introduction to Turing machines

A Turing machine (TM) is an abstract machine M that operates on an infinite tape T. At each moment the head of TM points to some cell in the tape. Each cell T_i is filled with some letter from a finite *tape alphabet* $\Gamma = \{\varnothing, a_1, a_2, \dots a_n\}$ such that $|\{i : T_i \neq \varnothing\}| < \infty$. The head has a state - letter from a finite *state alphabet* $Q = \{q_0, q_T, q_1, \dots, q_m\}$, where q_0 is the *initial state* and q_T is the *final (accept/termination) state.*

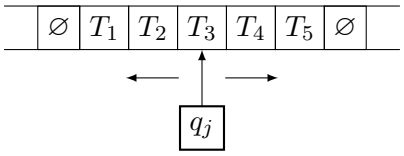
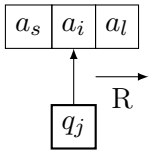
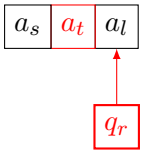


Fig. 2. Example of the Turing machine

Initially, the head is in the initial state. At consecutive moments, the head can either move one cell left or right (L, R) or not move (N). Before the movement, the head rewrites the letter it was pointing to and after the movement, the head changes its state. To determine how the head performs these operations we fix the *program of the Turing machine* (or *the transition function*) - function $p : \Gamma \times Q \to \Gamma \times Q \times \{L, R, N\}$. Below you can see a visualization of this process:



(a) $p(a_i, q_j) = (a_t, q_r, R)$



(b) Head moved right.

Fig. 3. In this example according to our program $p(a_i, q_j) = (a_t, q_r, R)$. Therefore the head moved right, rewrote the letter in the cell it was pointing to a_t, and changed its state to q_r.

Definition 2.1 (Turing Machine)**.** *Using the concepts introduced above the Turing machine is a tuple* (Γ, Q, q_0, q_T, p).

Lemma 2.1. *Equivalently one can define a Turing machine with the following limitation on its program: TM rewrites the tape letters only with command* N.

Proof. Add additional states $\widetilde{q}_r$ to Q. Next for any command $p(a_i, q_j) = (a_t, q_r, R/L)$, remove it from p and add two new commands: $p(a_i, q_j) = (a_t, \widetilde{q}_r, N)$, $p(a_t, \widetilde{q}_r, R/L)$. □

Lemma 2.2 (Countability of TM)**.** *The set of Turing machines $\mathcal{A}$ and tapes $\mathcal{T}$ is countable (up to relabeling states and tape letters).*

Proof. Let us relabel the tape/state alphabet: $r_a : a_i \mapsto i\varnothing \mapsto 0; r_q : q_j \mapsto j$. Then each Turing machine/tape is described by some finite subset of $\mathbb{N}$, therefore the cardinality of Turing machines/tapes is not larger than $\aleph_0$. Therefore $\mathcal{A} = \{\mathcal{A}_i : i \in \mathbb{N}\}, \mathcal{T} = \{\mathcal{T}_j : j \in \mathbb{N}\}$. □

Notice that despite the simplicity of the definition, the Turing machine can simulate any real-life computer. The opposite is not true, since the memory of computers is bounded, while the number of non-empty cells in the Turing Machine can grow unboundedly. This observation makes the Turing machine a fundamental way to formalize the concept of an algorithm - on the one hand, we know that any algorithm written in a modern language such as Python or Matlab can be also 'performed' by some Turing machine. On the other hand, if we know that there is no Turing machine capable of 'solving' some problem then we definitely cannot create a program that performs this task on some modern general-purpose computer.

Example 2.1. *Let tape be of type* $\ldots\varnothing\underbrace{1\ldots1}_{n}\varnothing\underbrace{1\ldots1}_{m}\varnothing\ldots$. *Write a Turing machine program such that starting working on the leftmost 1 it will terminate with tape equal* $\ldots\varnothing\underbrace{1\ldots1}_{m+n}\varnothing\ldots$ *and the head pointing to the leftmost* 1*.*

Solution. We start and keep moving right until the first $\varnothing$: $p(1,0) = (1,0,R)$. As soon we see it, we rewrite it on one, shift right, and change the state $p(\varnothing,0) = (1,1,R)$. Now again keep moving right, until the first $\varnothing$: $p(1,1) = (1,1,R)$. Turn left $p(\varnothing,1) = (\varnothing,2,L)$. Finally rewrite the rightmost one: $p(1,2) = (\varnothing,3,L)$, and move left until the first $\varnothing$: $p(\varnothing,3) = (\square,\varnothing,R)$. Below you can see a visualization of this process:

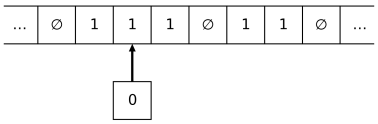


Fig. 4. Turing machine performs unary addition $11 + 111 = 11111$.

□

Example 2.2. *Can you come up with a TM program that performs (1) unary multiplication (2) binary addition (3) binary multiplication or (4) binary-unary conversion?*

Hint. Try to use our implementation of the Turing machine which includes animated visualization to create such a program. □

2.2 The Halting Problem

After we got familiar with Turing machines, let us move to the main theorem which serves as the basis for several non-results in our paper. In Lemma 2.2 we showed that we can enumerate tapes $\mathcal{T}_j$ and Turing Machines $\mathcal{A}_i$. Using it let us prove the unsolvability of the *Halting Problem for Turing machines*:

Theorem 2.1 ([T+36]). *There does not exist a TM, A, such that on the initial tape* $t_i = \underbrace{1 \dots 1}_{i}$ *A terminates on* 1 *if* $\mathcal{A}_i$ *terminates on* t_i *and A terminates on* 0 *otherwise.*

Proof. Assume such an A exists and p_A is its program with termination state q_m. Then define a program that starts an infinite cycle as soon as A wants to terminate with letter 1:

$$p_{\widetilde{A}}(a_i, q_j) = \begin{cases} (1, q_{m+1}, N), & p_A(a_i, q_j) = (1, q_m, \cdot) \text{ or } q_j = q_{m+1} \\ p_A(a_i, q_j), & else \end{cases}$$

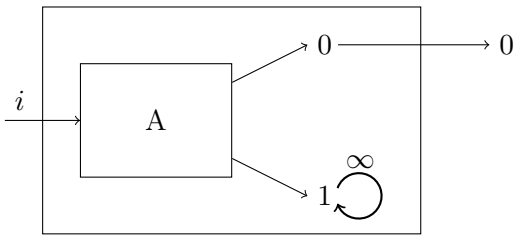


Fig. 5. Modification of Turing Machine A

Now $\exists j : \widetilde{A} = \mathcal{A}_j$, therefore (1) if $\mathcal{A}_j$ terminates on j at the same time $\widetilde{A}$ does not terminate on j and (2) if $\mathcal{A}_j$ does not terminate on j then $\widetilde{A}$ do terminate on j. Therefore we get a contradiction since $\widetilde{A} = \mathcal{A}_j$. □

3 The word problem in semigroups

3.1 The appearance of words in semigroups

From a set-theoretic perspective

Definition 3.1 (Semigroup and Monoid)**.** *Semigroup S is a set X with an associative binary operation $\cdot : X \times X \to X$ such that $\forall\ a, b, c \in X : (a \cdot b) \cdot c = a \cdot (b \cdot c)$. An element $1_S \in S$ is called neutral if $\forall a \in S : 1_S a = a 1_S = a$. Semigroups with a neutral element are called monoids.*

It is easy to show that a neutral element is unique: if 1 is another neutral element in S, then $1 = 1_S 1 = 1_S$. Another way to define a semigroup is to define it as a set of words with some 'equivalence' relation. Let us go through the details of this construction. Let Σ be some set - the *set of generators*. Let $\Sigma^+ = \{a_1 \dots a_n : a_i \in \Sigma, n \in \mathbb{N}\}$ be a set of finite words on alphabet Σ, and $\Sigma^* = \Sigma^+ \cup \{\varnothing\}$, where $\varnothing$ denote the empty word. On the set of words Σ^* we can define concatenation operation $\cdot : \Sigma^* \times \Sigma^* \to \Sigma, (a, b) \mapsto ab = a_1 \dots a_n b_1 \dots b_n$. Then $\Sigma^+(\Sigma^*)$ with concatenation operation is called the *free semigroup (monoid) on* Σ. For convenience, we will denote both free semigroup (monoid) and its underlying set by Σ^+ (Σ^*). Notice that concatenation is associative. Therefore a free semigroup is a semigroup according to definition 3.1. Furthermore since concatenating with $\varnothing$ doesn't change the word $\varnothing$ is the neutral element and Σ^* is a monoid. To define a general semigroup (monoid) in these settings we also need to specify what we mean by 'equivalence'. Let $\sim$ be an equivalence relation on $\Sigma^+(\Sigma^*)$, then $\sim$ is called *congruence relation* if $\sim$ respects concatenation: for any words a, b, c, d : $a \sim c \wedge b \sim d \Rightarrow ab \sim cd$. Now we are ready to define

Definition 3.2 (Semigroup-presentation)**.** *Let $\sim$ be a congruence relation on Σ^+. Then $(\Sigma, \sim)$ is called a presentation of a factor-semigroup $\Sigma^+/\sim := \{[a] = \{u \sim a, u \in \Sigma^+\}, a \in \Sigma^+\}$ with a binary operation $[a][b] = [ab]$.*

In the same way, we can define monoid-presentations $(\Sigma, \sim, \varnothing)$ and factor-monoids $\Sigma^*/\sim$. Let us show that these two definitions of a semigroup are equivalent using the following

Lemma 3.1 (Semigroups $\leftrightarrow$ Semigroup-presentations)**.** *1. $\Sigma^+/\sim$ is a semigroup. 2. Each monoid M is isomorphic to $\Sigma^*/\sim$ for some Σ and $\sim$. 3. Each semigroup S without neutral element is isomorphic to $\Sigma^+/\sim$ for some Σ and $\sim$.*

Proof. 1. Firstly, if we choose different representatives $a' \in [a]$ and $b' \in [b]$, then $a'b' \sim ab$, therefore $[a'][b'] = [a][b]$, hence multiplication is well-defined. Secondly $([a][b])[c] = [abc] = [a]([b][c])$, therefore multiplication is associative.

2. Let $\Sigma = \{a_i\}$ be a set bijective to the $M_0 = \{m_i \neq 1_S, m_i \in S\}$. Let $\sim$ be the minimal congruence on Σ^* such that $a_i a_j \sim a_k$ if $m_i m_j = m_k$ ($a_k = \varnothing$ if $m_i m_j = 1_S$). Notice that $a_i \sim a_j \Leftrightarrow m_i = m_j \Leftrightarrow a_i = a_j$ i.e. the only one-letter word in $[a_i]$ is a_i. On the other hand, $\forall a_i, a_j \exists! a_k : a_i a_j \sim a_k \Rightarrow \forall u \in \Sigma^* \exists! a_l \in \Sigma \cup \{\varnothing\} : [u] = [a_l]$. In summary $\Sigma^*/\sim = \{[a_i]\}$ and $[a_i] \neq [a_j]$ for $i \neq j$. Now let $\phi : \Sigma^* \to M$ such that $\phi(\varnothing) = 1_m \wedge \phi(wa_i) = \phi(w)m_i$. Then we

aim to show that $\widetilde{\phi} : \Sigma^*/\sim \to M, [u] \mapsto \phi(u)$ is a well-defined isomorphism. Firstly we need to show that ϕ respects $\sim$. This follows from an inductive argument: (base) $\phi(a_i a_j) = m_i m_j = m_k = \phi(a_k)$, (step) $u_1 \sim u_2, v_1 \sim v_2 \Rightarrow \phi(u_1 u_2) = \phi(u_1)\phi(u_2) = \phi(v_1)\phi(v_2) = \phi(v_1 v_2)$. Therefore ϕ is a homomorphism and therefore $\widetilde{\phi}$ is a well-defined homomorphism. Next 1) $\widetilde{\phi}([a_i]) = m_i$, therefore $\widetilde{\phi}$ is surjective and 2) $\widetilde{\phi}([a_i]) = \widetilde{\phi}([a_j]) \to [a_i] = [a_j]$, therefore $\widetilde{\phi}$ is injective. Therefore $\widetilde{\phi}$ is an isomorphism.

3. Consider $M := S \cup \{1_S\}$ and construct Σ, $\widetilde{\phi}|_S$ as in previous part. Since $\widetilde{\phi}(1_s) = \varnothing$ we get that $\widetilde{\phi}|_S$ is isomorphism between S and $\Sigma^+/\sim$. □

3.2 Thue's problem

Let $\Sigma = \{a_1, \ldots, a_n\}$ and $\sim$ is the minimal congruence such that $u_1 \sim v_1, \ldots, u_m \sim v_m$. Then we call $u_i \sim v_i$ *defining relations*. Notice that by the congruence definition, $uu_iv \sim uv_iv$. We call such a pair of words *directly equivalent* pair. For any pair of words $g \sim h$, there exists a chain $g = g_0 \sim g_1 \sim \cdots g_k = h$, such that $g_i \sim g_{i+1}$ are directly equivalent. Now we have the natural question:

Question 3.1 ([Thu14]). *Let $a_1, \ldots, a_n$ be a semigroup-presentation with defining relations $u_1 \sim v_1, \ldots, u_m \sim v_m$. Let $u, v \in \Sigma^+$. Is it true that $u \sim v$?*

Notice that in a free semigroup, this question is trivial since we do not have any equivalences. In an abelian semigroup, we can answer this question in linear time - we just need to count the occurrences of each generator for both words. It may seem that the problem should be easy for a general semigroup since both words are finite and a finite number of pairs encodes congruence. However, the following result proves that there is no general algorithm that can work for any finitely generated semigroup:

Theorem 3.1 ([Mar48]; [Pos47]). *There is no algorithm A such that on the input of a finite semigroup-presentation and a pair of words A answer Thue's question for the input.*

Proof. Our strategy is to show the equivalency between this problem and the Halting Problem (Theorem 2.1). So, let us show how to build a semigroup from a Turing machine. Let a Turing machine A have a tape alphabet $\Gamma = \{a_0, \ldots, a_n\}$ and a state alphabet $Q = \{q_0, q_T, q_1, \ldots q_m\}$. We aim to construct a semigroup $S = \langle a_0, \ldots, a_n, q_0, q_T, q_1, \ldots, q_m, w\rangle/\sim$, where $\sim$ is a congruence constructed from the Turing machine A and a special symbol w.

For simplicity, we will use the equivalent definition of TM that was justified in Lemma 2.1. Then the program of A has three types of commands and we add three equivalences according to them:

$$\text{I}\begin{cases} a_i \mapsto a_t \\ q_j \mapsto q_r \\ N \end{cases} \qquad \text{II}\begin{cases} a_i \mapsto a_i \\ q_j \mapsto q_r \\ L \end{cases} \qquad \text{III}\begin{cases} a_i \mapsto a_i \\ q_j \mapsto q_r \\ R \end{cases} \tag{1}$$

$$q_j a_i \sim q_r a_t \qquad \forall a_t \in \Gamma : a_t q_j a_i \sim q_r a_t a_i \qquad \forall a_t \in \Gamma : q_j a_i a_t \sim a_i q_j a_t \tag{2}$$

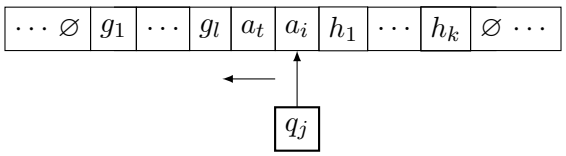


Fig. 6. The configuration before the command is $c_1 = ga_tq_ja_ih$, and after the command is $c_2 = gq_ja_ta_ih$. c_1 and c_2 are directly equivalent using equivalence of type II.

We define the *Post word* and *configuration* of TM and tape in the following way: if the tape is $g_1 \dots g_l a_i h_1 \dots h_k$ and the head has state q_i pointing on a_i, then the configuration is $c = gq_ia_ih$ and the Post word is wcw. Then we see from the definition of our commands (Equation 1) and corresponding equivalences (Equation 2), that Post words before and after the command are directly equivalent (see an example in figure 6). We also will need another type of equivalence - $\forall t$:

$$\text{(IV)} \quad wq_Tw \sim q_T, \quad q_Ta_t \sim q_T, \quad a_tq_T \sim q_T$$

Finally, we can start proving the main idea of the theorem: *if A started working with configuration c then A will terminate* $\Leftrightarrow wcw \sim q_T$.

($\Rightarrow$) Let c_f be the configuration of A when it terminated. Then $wc_fw = wg_1..g_lq_Th_1..h_kw$, where $g_i, h_j \in \Gamma$. But then using equivalences of type (IV) we get $wc_fw \sim q_T$. Now if $c, c_1, \dots c_f$ is the chain of configurations, then as we showed before $c_i \sim c_{i+1}$, therefore $q_T \sim c_f \sim c$.

($\Leftarrow$) If $wcw \sim q_T$, then there exists a chain of directly equivalent words:

$$wcw = x_1 \sim \dots \sim x_k = gq_Th \sim x_{k+1} \sim \dots \sim q_T.$$

In each such chain, there is minimal k such that a_T is in x_k. Let us pick a chain such that this k will be minimal among all possible chains. Then notice, since any $x_j, j < k$ does not contain q_T the direct equivalence between $x_{j-1} \sim x_j, j < k$ can not have type IV. Therefore these equivalences correspond to some command of A. However, the commands of A are directed while the equivalences are not. Let us write $x_j \overset{\rightarrow}{\sim} x_j$ and $x_j \overset{\leftarrow}{\sim} x_j$ to highlight the direction of command of A. Since $q_T \in x_k$ then we cannot have $x_{k-1} \overset{\leftarrow}{\sim} x_k$ because A stop working after

reaching state q_T. Therefore $x_{k-1} \overset{\rightarrow}{\sim} x_k$.

$$wcw = x_1 \sim x_2 \sim \cdots x_{k-1} \overset{\rightarrow}{\sim} x_k = gq_Th.$$

Now since chain ends with equivalence $\overset{\rightarrow}{\sim}$, then if we have at least one $\overset{\leftarrow}{\sim}$, then there is a subchain of type

$$x_j \overset{\leftarrow}{\sim} x_{j+1} \overset{\rightarrow}{\sim} x_{j+2}.$$

But in this case, since A has only one command for each configuration including x_{j+1} we get that $x_j = x_{j+2}$. Therefore we can drop x_j, x_{j+2} and get a valid chain of direct equivalences with a smaller length. This is the contradiction with our choice of k, therefore all equivalences $x_i \sim x_{j+1}$ correspond to the forward commands of A, and when algorithm A starts work on c, A terminates.

Let us summarize. We showed that answering the question about the termination of a Turing machine A is equivalent to answering the question about the equivalency of two words in the semigroup constructed from A. Therefore from Theorem 2.1 we conclude that there is no general algorithm capable of solving word problems for all semigroups simultaneously. □

This theorem shows, that there is now a general algorithm, such that given a semigroup and a pair of words we can answer whether the words are equal in this semigroup. However, as we have shown previously there are such algorithms for abelian and free semigroups, and these algorithms are different. So the natural question is

Question 3.2. *Is there a semigroup S (semigroup-presentation) such that there is no algorithm A that can in finite time check the equality of any words in S.*

Unfortunately, the answer to this question is positive. One can prove the existence of such a semigroup using the unsolvability of the Halting Problem for Turing Machines. For instance, in Chapter 12 of [Rot12] you can find proof that relies on the corollary of the halting problem - the existence of a set $X \subset \mathbb{N}$ such that we can in finite time answer the question "$x \in X$?" if $x \in X$, but cannot in finite time answer the question "$x \notin X$?" if $x \notin X$.

Let's give an elegant example of a semigroup with unsolvable word problem

Example 3.1 ([Tse56], [Sco56]). *Let $\Sigma = \{a, b, c, d, e\}$ and*

$$ac \sim ca, ad \sim da, bc \sim cb, ce \sim eca, de \sim edb, cca \sim ccae.$$

Then the word problem in semigroup with these generators and defining relations is unsolvable.

We conclude this section with two open questions [NB24].

Open Question 3.1. *Is the word problem always solvable in monoids with one or two defining relations?*

Open Question 3.2. *Is the word problem always solvable in monoids with just one defining relation of type $u \sim a$, where a is a letter?*

4 The word problem in groups

4.1 Solvability for free groups

In the previous section, we discussed word problems for semigroups. However, a more familiar and common structure for us is a group. To study the word problem for groups, we first introduce free groups.

Definition 4.1 (Free group)**.** *Free group with generators $x_1, \dots, x_n, \dots$ is a monoid with generators x_i, x_i^{-1} and defining relations $x_i x_i^{-1} \sim x_i^{-1} x_i \sim \varnothing$.*

Recall that in free monoid/semigroup, the word problem is trivial: we do not have any equivalency relations $\Leftrightarrow$ each equivalency class contains a single word. In free groups, we do not have this nice (or boring) property, for instance $x_1 x_1^{-1} x_2 \sim x_2 x_1^{-1} x_1$. In this example, both words are equivalent to x_2 and x_2 is the shortest among all words equivalent to x_2. We call words with minimal length among all equivalency class *irreducible words*. Since the set of lengths of all words in an equivalency class is a subset of $\mathbb{N}$, we see that there is at least one irreducible word in each equivalency class. In the following lemma, we prove that it is unique.

Lemma 4.1. *Each equivalency class in a free group contains a single irreducible word.*

Proof. Similarly to Theorem 3.1 to prove this lemma we need to take a closer look at chains of direct equivalent words. Words a and b are directly equivalent in a free group if you can get a from b 1) either by inserting $x_i x_i^{-1}$ (or $x_i^{-1} x_i$) somewhere inside b 2) either by deleting $x_i x_i^{-1}$ (or $x_i^{-1} x_i$) somewhere inside b. Let's denote these direct equivalences by $a \nearrow b$ and $a \searrow b$ correspondingly. Then for any equivalent words $a \sim b$ there exists some chain of direct equivalences represented (figure 7a) and our goal is to show that there is a chain of direct equivalences that goes down and then up (figure 7b).

Let us take a chain between a and b of the minimal length. If this chain does not go down and then up, it has some local maximum $c \nearrow d \searrow e$. In this case, we can find the word $\widetilde{d}$ directly equivalent to c and e such that $c \searrow \widetilde{d} \nearrow e$. $c = c_1 \dots c_n \nearrow c_1 \dots c_k f c_{k+1} \dots c_n = d$, where $f = x_i x_i^{-1}$ (or $f = x_i^{-1} x_i$). Then we get e from d by deleting subword $g = x_j x_j^{-1}$ (or $g = x_j^{-1} x_j$). Since the chain has the minimal length if f intersects with g (as a subword of d),

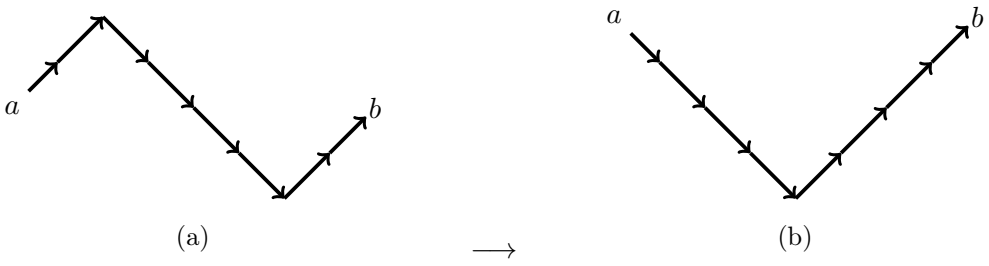


Fig. 7. Any chain of direct equivalences can be transformed into a chain with a single minimum.

then $a = c$, and this is a contradiction. Therefore f and g do not intersect and we can firstly delete g and then add f.

If a is an irreducible word then there is no word c, such that $a \nearrow c$ or $c \searrow a$, therefore the down-up chain between two irreducible words cannot exist. □

An immediate corollary of this lemma is the following:

Theorem 4.1. *The word problem in a free group can be solved in linear time.*

Proof. Let $a \sim b$ and b be irreducible. Then, by the previous lemma, there is a chain from the word a to b which goes only down. Each $\searrow$ in this chain is deleting of $x_i x_i^{-1}$ or $x_i^{-1} x_i$. Therefore after deleting all $x_i x_i^{-1}$ and $x_i^{-1} x_i$ from the word a we get irreducible form of a.

To check equivalency $c \sim d$ we 1) get their irreducible form $\widetilde{c}, \widetilde{d}$ and 2) check if $\widetilde{c} = \widetilde{d}$. □

4.2 Unsolvability for groups in general

Motivated by this positive result we might ask ourselves the following:

Question 4.1. *Is the word problem always solvable in groups?*

The answer is found in the following theorem:

Theorem 4.2 ([Nov58], [Boo58])**.** *There exist a finitely presented group G with unsolvable word problem.*

Proof strategy. The classical strategy to prove this theorem is to show its equivalency to unsolvability of word problem in some semigroup. Recall that in Theorem 3.1 we showed how to construct turing machine $A = A(S)$ by semigroup $S \cong \{x_i\}^+ / \sim$. Now for this semigroup S we build group $G \cong \{x_i, x_i^{-1}\}^* / (\sim \cup \{x_i x_i^{-1}, x_i x_i^{-1}\})$. And then show that word problem for some special pairs of words in S is equivalent to the word for some corresponding words in G. The complete proof using this strategy can be found in Chapter 12 of [Rot95]. □

5 The word problem for Coxeter groups

As we have shown earlier in this paper, the word problem is undecidable for groups in general, as well as for finitely presented semigroups. However, there are examples of specific types of groups for which the word problem is not only solvable, but for which algorithms have been found to solve it. One such example of a specific type of group for which the word problem is solvable are Coxeter groups. In 1969, the Belgian-French mathematician Jacques Tits developed an algorithm to solve the word problem for Coxeter groups. In the following paragraphs, we will formally define Coxeter groups and then present the algorithm that Tits invented.

A Coxeter group is a group defined by giving a group presentation with specific constraints on its relations. These relations are encoded in a Coxeter matrix or a Coxeter diagram, which we will define shortly. The reader might wonder what the significance is of these groups. Why should we care about them? One answer is that it's convenient to have a family of groups defined in terms of their relations, especially if one is interested in the word problem. It also turns out that many important classes of groups are instances of Coxeter groups, so Coxeter groups are a way of generalizing these instances to elucidate what they have in common and what sets them apart. For example, Dihedral groups and Symmetric groups have in common that they are both examples of Coxeter groups.

Definition 5.1 (Coxeter matrix)**.** *A Coxeter matrix M over S, is a function $M : S \times S \to N \cup \infty$ such that, for $s_i, s_j \in S, M(s_i, s_i) = 1, M(s_i, s_j) = M(s_j, s_i) \geq 2$ if $s_i \neq s_j$. The value $M(s_i, s_j)$ of M at (s_i, s_j) will also be denoted as m_{ij}.*

Definition 5.2 (Coxeter diagram)**.** *A Coxeter diagram is a graphical presentation of the relations of the generators in a Coxeter group. The diagram is an undirected edge-labeled graph whose vertices correspond to the generators, with edges between two vertices labeled with m_{ij}. If $m_{ij} = 2$, we omit the edge between vertices s_i and s_j. If $m_{ij} = 3$, we omit the label.*

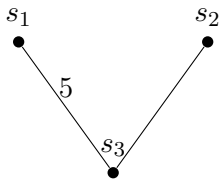
Definition 5.3 (Coxeter system)**.** *A Coxeter system (W, S) is a Coxeter group W along with a set of generators $S \subset W$, with relations $(s_i s_j)^{m(s_i, s_j)} = 1$, where $m(s_i, s_j) = m(s_j, s_i) \geq 2$ for distinct $s_i, s_j \in S$.*

Example 5.1 (Coxeter matrix and diagram)**.** *A Coxeter matrix, the Coxeter diagram, and the list of relation are given in Figure 8. The corresponding Coxeter group is generated by the set $S = \{s_1, s_2, s_3, s_4\}$ and the given relations.*

Observe the lack of edge between vertices s_1 and s_2 corresponds to $(s_1 s_2)^2 = 1$, which corresponds to a commutative relationship between s_1 and s_2. The unlabeled edge between vertices s_2 and s_3 corresponds to the relation $(s_1 s_2)^3 = 1$. The edge between vertices s_1 and s_3 labeled "5" corresponds to the relation $(s_1 s_2)^5 = 1$.

$$M = \begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \end{array} \begin{array}{c} \begin{array}{ccc} s_1 & s_2 & s_3 \end{array} \\ \begin{pmatrix} 1 & 2 & 5 \\ 2 & 1 & 3 \\ 5 & 3 & 1 \end{pmatrix} \end{array}$$

(a) Coxeter matrix



(b) Coxeter diagram

$$\begin{cases} s_1^2 = s_2^2 = s_3^2 = 1 \\ (s_1 s_2)^2 = 1 \\ (s_1 s_3)^5 = 1 \\ (s_2 s_3)^3 = 1 \end{cases}$$

(c) Group relations

Fig. 8

Now we present Tits's algorithm for solving the word problem in Coxeter groups. Tits's algorithm consists of two "rewrite" rules for a given word. They are for a word "w" with letters "i" and "j".

1. $ii \Rightarrow 1$
2. $iji \ldots$of length $m_{ij} \Rightarrow jij \ldots$of length m_{ij}.

Rewrite rules of the second kind are applied to the word "w" until one of the letters appears twice in a row. If this never happens, the word is reduced and the answer is yes if the result is the empty word and otherwise no. If it does happen, remove the double letter pair from the word, which is the rewrite rule of the first kind, and repeat the procedure. Termination will happen because the word's length has decreased. The mathematical basis for this algorithm is a theorem due to Tits first published in 1969 [Dav08]. Before we present the theorem, we need some further definitions.

First, we need some definitions and a preliminary proposition.

Definition 5.4 (M-Reduction)**.** *An elementary M-operation or M-reduction on a word in S is one of the following two types of operations:*

MI: Delete a subword of the form (s, s) from s.

MII: Replace an alternating subword of the form $(s, t, \cdots)$ of length $m_{st} = m(s, t)$ by the alternating word $(t, s, \cdots)$ of the same length.

Definition 5.5 (M-reduced)**.** *A word s is M-reduced if it cannot be shortened by M-operations.*

Definition 5.6 (Exchange Condition)**.** *Let $s_1 \cdots s_r$ and $t_1 \cdots t_r$ be two reduced words over S (where (W, S) is a Coxeter system) representing the same element $w \in W$. If $s_1 \neq t_1$, then there is an index $i \in \{2, \cdots, r\}$ such that $w = W s_1 t_1 \cdots \hat{t_i} \cdots t_r$.*

Lemma 5.1. *A Coxeter system (W, S) satisfies the Exchange Condition.*

Proposition 5.1. *Suppose (W, S) satisfies the Exchange Condition. Then two reduced expressions s and t represent the same element of W if and only if one can be transformed to the other by M-operations of type II.*

Proof. Let $s = (s_1, \cdots, s_k)$ and $t = (t_1, \cdots, t_k)$ be reduced expressions for w in W. We will show by induction on k that s can be transformed into t.

When $k = 1$, $w = s_1 = t_1$ so $s = t$. For the inductive step there will be two cases:

Case 1: $s_1 = t_1$. Then $s' = (s_2, \cdots, s_k)$ and $t' = (t_2, \cdots, t_k)$ are reduced words for $w' = s_1 w = t_1 w$. By induction s' can be changed to t' by MII operations and since $s_1 = t_1$ these operations change s to t.

Case 2: $s_1 \neq t_1$. Let $s = s_1$ and $t = t_1$. Set $m = m(s, t)$. Claim: m is finite and there exists another reduced expression u for w which begins with an alternating word $(s, t, \cdots)$ of length m.

The claim implies case 2. Since s and u both begin with s, by case 1 we have that s can be changed to u. Then we apply MII to obtain an expression u' from u, with u' beginning with a word $(t, s, \cdots)$ of length m. Since u' and t both begin with t, applying case 1 again we can change u' to t using MII operations.

Proof of Claim: Since $l(tw) < l(w)$, by the Exchange Condition there is an index i such that $w = w(t, s, \cdots, \hat{s_i}, \cdots, s_k)$. It's clear s is not exchanged because $s \neq t$ (removing s would show $s = t$). So we obtain a reduced expression for w beginning with (t, s).

Using the expression $(t, s, \cdots, \hat{s_i}, \cdots, s_k)$ for w, since $l(sw) < l(w)$ by the Exchange Condition $w = w(s, t, s, \cdots, \hat{s_{i_1}}, \cdots, \hat{s_{i_2}}, \cdots, s_k)$ where t or s cannot be removed. This gives a reduced expression for w beginning with (s, t, s).

We continue this process. Let $\mathbf{s_q}$ be the alternating word in s and t of length q ending in s. Suppose s' begins with $\mathbf{s_{q-1}}$. Let s' be the element of s, t that s' does not start with. Since $l(s'w) < l(w)$, by the Exchange condition we obtain a reduced expression for s' with s' in front. The removed letter cannot be from $\mathbf{s_{q-1}}$. This is because in the dihedral group of order $2m$ generated by s and t, a reduced expression for an element of length $\neq$ m is unique. If the removed letter were from $\mathbf{s_{q-1}}$, by cancellation we could obtain two different expressions in s and t of length not equal to m for the same element of the group.

Therefore w has a reduced expression beginning with $\mathbf{s_q}$. This works for all $q \leq m$ and $q \leq l(w)$, so m is finite.

Replacing $s_m = (t, s, \cdots)$ with $(s, t, \cdots)$ we obtain u. This proves the claim.

The converse of this proposition is clear. □

Theorem 5.1 (Tits)**.** *Suppose (W, S) satisfies the Exchange Condition. Then a word $s = (s_1, \cdots, s_k)$ is reduced if and only if it is M-reduced.*

Proof. If s is reduced then it is M-reduced. Conversely, suppose s is M-reduced. We will show that s is reduced by induction on k. Note that when $k = 1, w = s_1$ is reduced.

Let $k > 1$. By induction $s' = (s_2, \cdots, s_k)$ is reduced. Suppose, by contradiction, s is not reduced. Then $l(w) \leq k-1$. We have $w = s_1 \cdots s_k$ and $w' = s_2 \cdots s_k$. Because $l(s_1 w') = l(w) \leq k - 1$ by the Exchange Condition $w' = s_1 s_2 \cdots \hat{s_i} \cdots s_k$ i.e. w' has another reduced expression s'' that begins with s_1. By the proposition, s' can be changed to s'' by MII operations. Then s can be changed by MII operations to a word starting with (s_1, s_1). This contradicts s being M-reduced. Hence s must be reduced. □

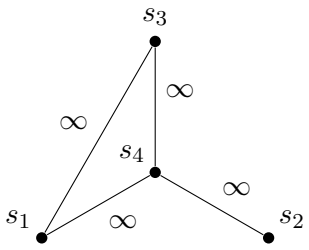
6 Right-angled Coxeter groups

Definition 6.1 (Right-angled Coxeter group)**.** *A Coxeter group is a right-angled Coxeter group (RACG) if the only relations between distinct generators are commuting relations, i.e. $(s_i s_j)^2 = 1$ for some i, j and $i \neq j$.*

Example 6.1 (A right-angled Coxeter group)**.** *The following Coxeter group is given by the set of generators $S = \{s_1, s_2, s_3, s_4\}$. The relations are given by the matrix M or the Coxeter diagram. We observe that it is an example of a right-angled Coxeter group, given that relations between distinct generators are commutative.*

$$M = \begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \\ s_4 \end{array} \begin{array}{c} \begin{array}{cccc} s_1 & s_2 & s_3 & s_4 \end{array} \\ \begin{pmatrix} 1 & 2 & \infty & \infty \\ 2 & 1 & 2 & \infty \\ \infty & 2 & 1 & \infty \\ \infty & \infty & \infty & 1 \end{pmatrix} \end{array}$$

(a) Coxeter matrix



(b) Coxeter diagram

$$\begin{cases} s_1^2 = s_2^2 = s_3^2 = s_4^2 = 1 \\ (s_1 s_2)^2 = 1 \\ (s_2 s_3)^2 = 1 \end{cases}$$

(c) Group relations

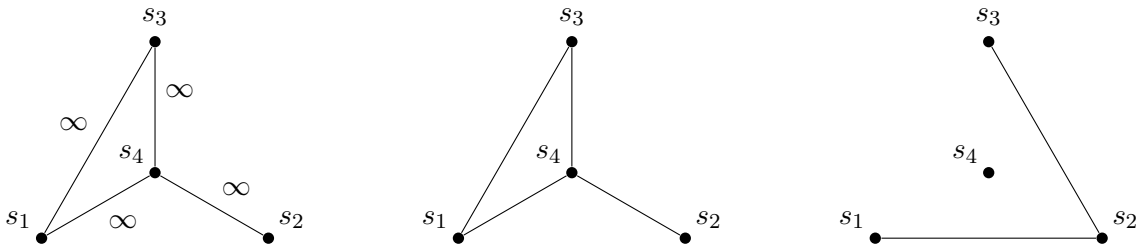
Definition 6.2 (Simple graph)**.** *A simple graph is an unlabeled, undirected graph with no loops or multiedges.*

Proposition 6.1. *There is a bijection between simple graphs on n vertices and right-angled Coxeter groups with n generators.*

Proof. Given a right-angled Coxeter group with n, we can form a unique Coxeter diagram. All edges between distinct vertices s_i and s_j on this graph have the label "∞", which corresponds

to no relation of the form $(s_i s_j)^m = 1$. Since the edge labels are all the same, we can remove these labels without consequence. Further, we can remove vertex labels as they correspond to arbitrarily-named generators, and are now left with a simple graph. Conversely, any simple graph on n vertices corresponds to a right-angled Coxeter group with n generators where commuting relations are given by two vertices that are not directly connected by an edge. □

Moving forward, we will be using a vertex-labeled graph, 10, that we will refer to as the *relations graph* as a visual aid to the algorithm for solving the word problem. Since we wish to quickly check which pair of vertices commute, we will to use the complement of the graph obtained by removing the labels from the edges of the Coxeter diagram. That way, if a pair of distinct generators commute, we can reference the graph and simply check for an edge.



(a) Coxeter diagram (b) Vertex-labeled graph (c) Relations graph

Fig. 10. Coxeter diagram and corresponding relations graph

6.1 The word problem for right-angled Coxeter groups

There are multiple linear-time approaches to solving the word problem for right-angled Coxeter groups. One approach is to apply the work of [LWZ90] for a linear-time solution to the word problem for free partially commutative monoids. Another approach we can apply (and further adapt) comes from [CGW08], where a linear-time algorithm is presented for right-angled Artin groups (RAAG). This approach is particularly favored by the authors as it utilizes a geometric tool known as a piling, which allows for visual representation of prefixes and words in the group, and allows one to easily take the perspective of an individual generator in a given word.

Artin groups are closely related to Coxeter groups; looking purely from a generators-relations perspective, the difference between the two lies in the involutive relations of the generators in Coxeter groups. Removing these relations from a Coxeter group yields an Artin group, and quotienting any Artin group by these relations results in a Coxeter group. Further, a right-angled Artin group is a direct analog to a right-angled Coxeter group: the only relations between distinct generators is commutativity.

First, we present the algorithm as given for RAAGs, then discuss the adaptation of this algorithm to apply to RACGs. We consider words within a fixed RAAG given by the generating set of n elements, $\{s_1, \ldots, s_n\}$, and their relations. For each word that we wish to check in our group, we build a *piling*. This is a collection of n different words over the alphabet $\{+, -, 0\}$, one word for each generator. A word from this piling that corresponds to generator s_i will be referred to as the s_i-*stack*. To help with clarity, we will refer to the letters of an s_i-stack as *beads* labeled from $\{+, -, 0\}$.

Algorithm for right-angled Artin groups

To start building our piling, we start with empty stacks. We read each word from our group from left-to-right, checking each letter individually, adding or subtracting beads as necessary from each stack before moving onto the next. Each letter we read, s_i^ϵ, we check the corresponding s_i-stack and either add or subtract beads as follows:

- Subtracting beads: if the last bead on the s_i-stack has the same sign as ϵ, we remove this bead as well as the last bead of any stacks that correspond to any generators that do not commute with s_i (if done correctly, when this occurs, the beads that are removed from these other stacks will always be 0). This subtraction event corresponds to the cancelling of s_i and s_i^{-1} due to the ability to commute any letters that may be between them.
- Adding beads: if the last bead on the s_i-stack does not have the same sign as $-\epsilon$ (i.e. the bead has the same sign as ϵ, is 0, or the stack is empty), we add an ϵ-signed bead to the s_i-stack and a 0 bead to each stack corresponding to a generator that does not commute with s_i.

We repeat this process of adding and subtracting beads until we run out of letters and the process terminates.

In order to understand more intuitively how this works, we can take the perspective of one of these s_i-stacks. An empty bead between two + or − beads in the stack corresponds with these letters in the word being "blocked" by the presence of a non-commuting element between them in the word. This is why we remove 0 beads from stacks at the end of a subtraction event, we have cleared a corresponding blocking element.

This process is well-defined; any equivalent words in the group will admit the same piling (i.e., each s_i-stack contains the same sequence of beads). These pilings correspond to a unique reduced word, and an empty piling corresponds to the identity. The length of each reduced

word corresponds to the sum of the number of + and − beads present in all of the s_i-stacks.

Let m be the number of letters in the word and n be the number of generators in the group. In each step of our algorithm, we make at most one change to each of the m stacks. Therefore, with proper implementation, constructing the piling requires $O(nm)$ operations. Comparing two words involves comparing the constructed pilings, which also takes $O(nm)$ operations. Thus, the complexity of comparing two words is linear with respect to their lengths.

The unique reduced word is given by a process of "extraction" from the piling. We first need to assign some total order to use on the set of stacks, we'll stick with $s_1 > s_2 > \cdots > s_n$. From here, prioritizing our total ordering on generators, we take the first stack that contains a + or − bead, extract our bead and $s_i^{\pm 1}$ becomes our first letter. We also extract the 0 beads from stack that correspond to a generator that does not commute with s_i. We continue this process, concatenating each letter onto the growing word until we run out of beads.

Example 6.2. *Given the right-angled Artin group with generating set* $S = \{s_1, s_2, s_3, s_4, s_5, s_6\}$ *and relations given by the Coxeter diagram in 12 (but noting that we wish to consider this as the Artin group, so we omit the involution relation and just consider commuting relations from the diagram), let us construct piling for the word*

$$w = s_4^{-1}s_1s_4^2s_1^{-1}s_2^{-1}s_2s_3s_1^4s_4^{-1}s_6^{-1}s_3s_6^2s_3^{-1}s_4s_3^{-1}s_3^2s_4^{-1}s_2s_6^{-1}s_5s_2^{-1}s_5^{-1}.$$

The algorithm is displayed in the animation of Fig. 11. The red lines in the graph that appear throughout the animation correspond to the elements that do not commute with the current generator.

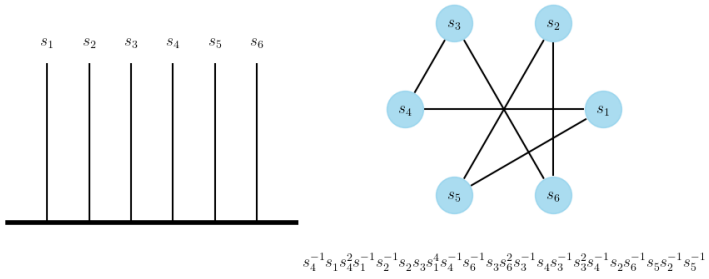


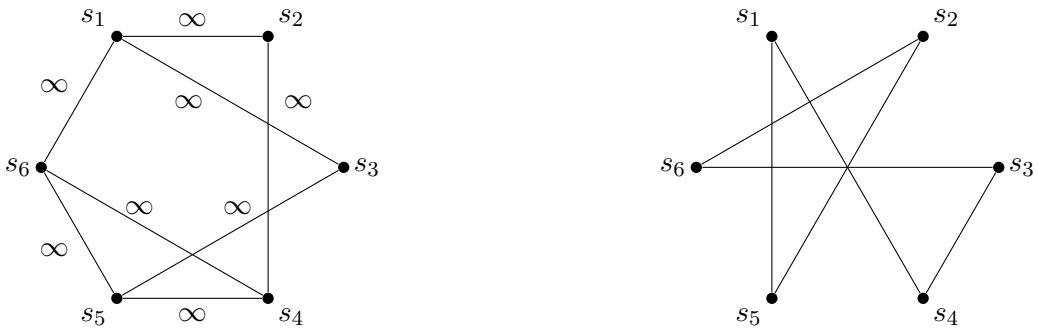
Fig. 11. Piling for w in the right-angled Artin group.

Try it out! Create a similar animation for any Artin or Coxeter group using our code.

Adapting the algorithm to right-angled Coxeter groups

In order to adapt this process to a RACG group, we can reduce our alphabet of beads to $\{+, 0\}$. This is due to the fact that all generators are involutive, so $s_i = s_i^{-1}$, so the sign of the bead is

no longer important. Now we subtract beads when the last bead on the s_i-stack is $+$, and add a bead when the last bead on the s_i-stack is 0.



(a) Coxeter diagram (b) Relations graph

Fig. 12

Example 6.3. *If we take the same generating set and commutative relations as in as in 6.2 given by 12, but this time we consider it as the Coxeter group where the generators are all involutive. We construct the piling for the word*

$$w = s_4^{-1}s_1s_4^2s_1^{-1}s_2^{-1}s_2s_3s_1^4s_4^{-1}s_6^{-1}s_3s_6^2s_3^{-1}s_4s_3^{-1}s_3^2s_4^{-1}s_2s_6^{-1}s_5s_2^{-1}s_5^{-1}.$$

The algorithm is displayed in the animation of Fig. 13.

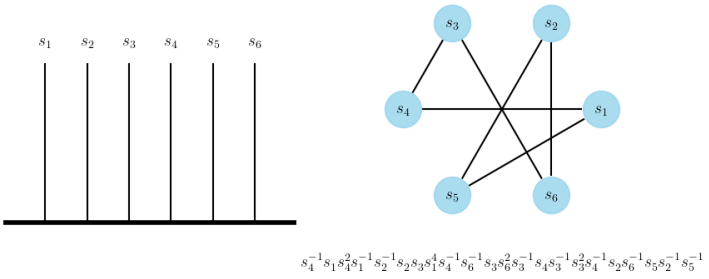


Fig. 13. Piling for w in the right-angled Coxeter group.

7 Conclusion

This paper has examined the word problem for general groups and in particular, for Coxeter groups and right-angled Coxeter groups. However, there are many other interesting questions we can jump to from here, such as how many distinct words of a given length exist? Expanding

further, Coxeter groups in general invite a number of questions we have not touched upon here. Another rich topic to explore is the mathematics of Weyl groups which are a specific kind of finite Coxeter group. More generally, automatic groups are also mathematically compelling and it can be shown that all finitely generated Coxeter groups are actually automatic. This opens up a new set of questions. There are also several open conjectures about Coxeter groups, including the "Combinatorial Invariance Conjecture" which concerns Kazhdan-Lusztig polynomials and others about unimodality.

References

[Boo58] William W Boone. The word problem. *Proceedings of the National Academy of Sciences*, 44(10):1061–1065, 1958.

[CGW08] John Crisp, Eddy Godelle, and Bert Wiest. The conjugacy problem in right-angled artin groups and their subgroups, 2008.

As the title suggests, this paper explores the conjugagy problem in right-angled Artin groups. This problem is intricately linked to the word problem. While we explore the problem for Coxeter groups, the close ties between Artin groups and Coxeter groups is why this paper contains the necessary background and algorithm. This paper explores a brief history on linear-time solutions to the conjugacy problem, as well as the importance of right-angled Artin groups before digging into the mathematics. Our paper utilizes information from sections 1 and 2 of this source.

[Dav08] Michael W. Davis. *The Geometry and Topology of Coxeter Groups*. Princeton University Press, 2008.

This book provides a nearly comprehensive treatment of Coxeter groups and their combinatorial theory, geometry and algebraic topology. The book begins by giving some basic notions in geometric group theory and then defines Coxeter groups and provides several examples of them, such as Dihedral groups. The book has a very complete treatment of the word problem for Coxeter groups and even provides the word problem algorithm as originally developed by Belgian-French mathematician Jacques Tits. Further topics are the Euler Characteristic, Growth Series, Buildings, and Hecke-Von Neumann Algebras. There are several detailed appendices on polytopes, cell complexes and algebraic topology in general.

[LWZ90] Hai-Ning Liu, C. Wrathall, and Kenneth Zeger. Efficient solution of some problems in free partially commutative monoids. *Information and computation*, 89(2):180–198, 1990.

[Mar48] Andrei Markov. Impossibility of certain algorithms in the theory of associative systems. 1948.

This is a surprisingly short note that we had the pleasure of reading in the original language. The note was originally submitted in 1946, so we see how

different was the style of mathematical writing back in the day. Without touching on the interesting features that will be understandable only to native Russian, it can be noted that many algebraic concepts such as semigroup had not yet come into use. Therefore, in the article semigroups are called 'associative systems'. The article begins with the definition of the concept of an associative system: it explains where words come from and how we can construct associative systems using words and their relations. Then the author formulates the problem of word equality and the problem of divisibility, then provides short and not very detailed proofs of the unsolvability of these problems for certain groups. The author bases the proofs of the word problem on the reduction of Church's results on lambda calculus to normal systems developed by Post. He also mentions Turing's concept of an algorithm, but doesn't directly connect it to our work on the halting problem (as we did in Section 2).

[NB24] Carl-Fredrik Nyberg-Brodda. GS Tseytin's seven-relation semigroup with undecidable word problem. *arXiv preprint arXiv:2401.11757*, 2024.

[Nov58] PS Novikoff. On the algorithmic unsolvability of the word problem in group theory. *Akademiya Nauk SSSR Matematicheskii Institut Trudy*, (44), 1958.

[Pos47] Emil L Post. Recursive unsolvability of a problem of Thue. *The Journal of Symbolic Logic*, 12(1):1–11, 1947.

[Rot95] Joseph J Rotman. The word problem. *An Introduction to the Theory of Groups*, pages 418–470, 1995.

[Rot12] Joseph J Rotman. *An introduction to the theory of groups*, volume 148. Springer Science & Business Media, 2012.

[Sco56] Dana Scott. A short recursively unsolvable problem. *J. Symbolic Logic*, 21(1):111–112, 1956.

[T+36] Alan Mathison Turing et al. On computable numbers, with an application to the entscheidungsproblem. *J. of Math*, 58(345-363):5, 1936.

[Thu14] Axel Thue. *Probleme über Veränderungen von Zeichenreihen nach gegebenen Regeln.* J. Dybwad, 1914.

[Tse56] Grigorii S Tseitin. Associative calculus with an insoluble equivalence problem. *Doklady Akademii Nauk SSSR*, 107(3):370–371, 1956.