The Word Problem for Groups and Applications to Knot Theory

Abrahim Ladha, Sarang Joshi

1 Introduction

In 1928, a mathematician by the name of Hilbert proposed a problem known as the "Decision Problem". The problem asks if there exists an algorithm such that given a set of axioms and a proposition of first order logic, it can return yes or no depending on if the statement is provable from the axioms. This really is an existential question. Are mathematicians even useful? Could we not invent an algorithm that could prove any theorem for us? For some time it seems like such an algorithm was realizable. Independently, two mathematicians by the name of Church and Turing published results showing that no such algorithm could exist. This fundamentally changed the direction and nature of mathematics since.

The word problem for groups is the problem of deciding whether two words in the group are equivalent. This problem has applications in knot theory. For instance, we show that the braid group has a decidable word problem, which implies there is an algorithm to determine when two braids are equivalent. We can also use this to show the undecidability of the unknotting problem in higher dimensions.

2 Groups

A group G is a set and an operator $\cdot : G \times G \to G$ such that the following hold:

- There exists a unique element $e \in G$ such that $a \cdot e = e \cdot a = a$ for every $a \in G$.
- For every $a, b \in G$ we have that $a \cdot b \in G$
- For every $a \in G$, there exists an element $a^{-1} \in G$ such that $a \cdot a^{-1} = a^{-1} \cdot a = e$
- For $a, b, c \in G$, it holds that $(a \cdot b) \cdot c = a \cdot (b \cdot c)$

Less formally, a group is an algebraic structure that contains a unique identity element, all elements have inverses in the group, and the group has closure, which means elements operated upon are put back into the group, and associativity. An example of a group is $\mathbb{Z}$ under the operation of addition $(+)$. The identity here is 0, the inverse of a is $-a$ since $-a + a = 0$, and the sum of any two integers will always be an integer. An example of something that is not a group is $\mathbb{N}$ under the operation of $\div$, because it does not satisfy closure, i.e. dividing two integers does not always give an integer.

To describe a group, one does not need every single element. A set of **generators** is a subset S of the elements of G such that repeated application of the group operation to a combination these generators and their inverses can produce every element in G. For example, take the group $(\mathbb{Z}/9\mathbb{Z})^\times = \{1, 2, 4, 5, 7, 8\}$, the group of all integers relatively prime to 9 less than 9 under multiplication mod 9. 7 is not a generator of this set, since $7^1, ..., 7^6 = 7, 4, 1, 7, 4, 1$. However, 2 is a generator since $2^1, ..., 2^6 = 2, 4, 8, 7, 5, 1$. If a group is generated by a single element, we say it is a cyclic group.

One method of describing a group is in the form of a **presentation**. We say that G has a presentation $\langle S|R\rangle$ if S is the set of generators for G, and R is a set of **relations** between the elements of S. For example: consider a cyclic group G of order n with a generator g. We know that $g^n = e$, so We can write $G = \langle S|R\rangle = \langle g, g^n = e\rangle = \langle g|g^n\rangle$. If a relation is simply equal to the identity, we don't write it and it is simply understood. Another example is the free group. This, by definition has no relations, so we can write it as $G = \langle S|\emptyset\rangle$, where S here, trivially, can be any set.

We say that a presentation of a group G is a **finite presentation** if both S and R are finite. A group may be finite or infinite, and that has no effect on whether or not it has a finite presentation.

3 Formal Languages and Computation

We use the notation described by Sipser in his textbook [6]. Given a set of symbols called an **alphabet**, denoted as Σ, we say a **word** is a concatenation of symbols in our alphabet. For example if $\Sigma = \{0, 1\}$, then $0, 1, 11011, 0101$ are all words. We also say that ε is the word of length zero, or the empty word. A collection of words is called a **formal language**. The collection of all possible words, we denote this as $\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, ...\}$. You can think of formal languages as subsets of Σ^*. Here are some examples of formal languages:

- If our alphabet is $\Sigma = \{0, 1\}$, then let $L \subset \Sigma^*$ be $\{0, 01, 1\}$
- $L \subset \Sigma^*$ with each $w \in L$ beginning with a 0
- $L \subset \Sigma^*$ with each $w \in L$ such that w is palindromic
- $L \subset \Sigma^*$ with each $w \in L$ having length greater than 17
- $L \subset \Sigma^*$ with each $w \in L$, if represented as an integer in base 2, would be a prime number.
- If Σ would be all symbols found on a keyboard, then let $L \in \Sigma^*$ with $w \in L$ all words found in this paper.
- For any alphabet, Σ^* is itself a language

One can describe languages like how one would describe sets – either explicitly, or with properties that each word in that language holds. Languages can be infinite or finite.

4 Decidability

Turing machines can be defined as a finite bit of logic with a pointer attached to an infinite strip of cells on a tape. Alan Turing imagined a human sitting down and physically doing arithmetic on a piece of paper. There are many formal, and equivalent definitions. The machine can take a turn to choose to replace the symbol at the end of the pointer, then move to another cell, and repeat. All computation can be done by Turing machines. We say a machine accepts a word if when we place the word on its tape, it will accept the word or reject. One can think of accepting or rejecting as ringing a small bell, or that it replaces the entire word on its tape with a 1 or a 0.

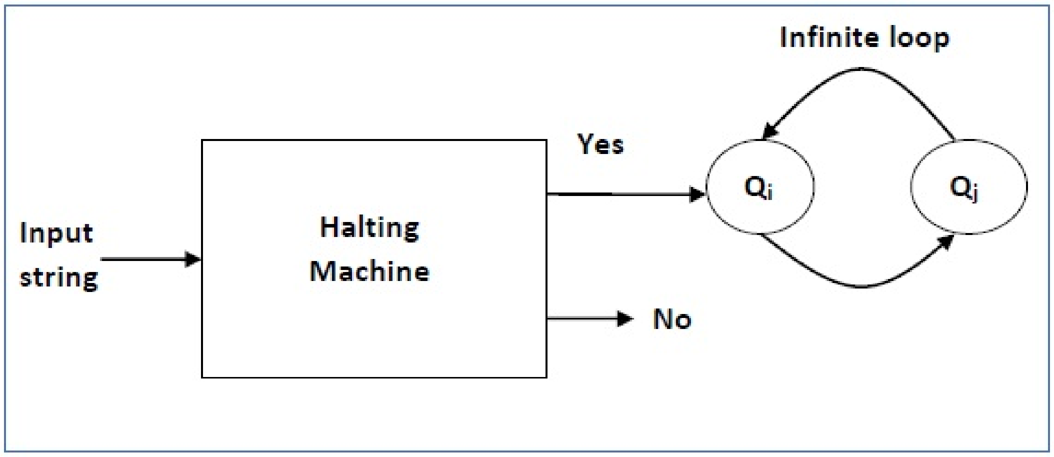
We say a turing machine **recognizes** a language L if for all $x \in L$, you put x on the tape of the TM, and the TM always accepts. If $x \notin L$, and you put it on the tape and run the TM, it will either reject or not halt at all. It could run forever. We say a turing machine **decides** a language L if it always halts and rejects for $x \notin L$.

If a machine M decides a language L, then its complement $L^c = \Sigma^* - L$ is also decidable. Simply construct a machine M', such that given a word on its tape, it passes the word to the machine M and M' simulates M. If M accepts, then M' rejects. If M rejects, then M' accepts. Unions and intersections of decidable languages are also decidable.

Evaluating if a logical statement is true or false is an equivalent problem to determining what words are in what languages. If L is a language that is encodings of prime numbers, and if w is an encoding of an integer, then to determine if $w \in L$ is the same as determining if w is prime. In general, if L is a language of words with certain properties, then $w \in L$ if w has those properties.

Clearly decidable languages are a subset of recognizable languages, but are they equal? Can we find a language who is recognizable but not decidable? To answer this question, it is important we understand first what it means for a machine to halt. Like computer code that can get stuck in an infinite loop, so can a Turing machine. Can we determine if a machine will halt or not? Well, machines are simply finite data, so let us create some encoding scheme such that a machine has an associated word. Then lets define a language H such that all words in H are encodings of machines that halt. If we can determine if H is decidable, then we can determine if the class of recognizable languages are always decidable or not.

Assume to the contrary that H is a decidable language. Then there exists a machine HM that can decide it. HM will always halt because it decides our language. Construct a new machine HM' such that on input, HM' will simply pass its input to HM. If HM outputs true, then HM' will loop forever. If HM outputs false, then HM' will halt.



HM' takes on input encodings of machines. We will encode HM' and run it on an encoding of itself! Case 1 is that HM' halts. But then if HM says HM' halts, HM' will not halt, which is a contradiction. Case 2 is that HM' does not halt, but then HM will say that HM' does not halt, which will cause HM' to halt, also a contradiction. Therefore, there cannot exist a machine HM that decides H. That is to say, in general, determining if a machine will halt is **undecidable**[7]. We are unable to determine the truth of such a statement.

This proof is really notable and deserves a few more details on how important it is. The existence of an undecidable problem was proven earlier by Church using "calculatable functions"[1]. The machine proof we have provided is the same one that Turing used a year later, and it is far more accessible than the work by Church [7]. Turing was also able to show that calculatable functions and turing machines are equivalent. The proof here quietly relies on the same kind of self-referential paradoxes that Gödel used to prove his incompleteness theorem. The machine we constructed will halt if it does not halt. Gödels unprovable statement was "This sentence is false". This kind of proof is called a proof by reduction. One assumes some problem is decidable and constructs a machine that decides it. Then one can construct a contradictory machine that uses the decidable machine as a subroutine proving that it cannot exist.

5 The Word Problem

Given a group G, and a presentation $\langle S \mid R \rangle$, we can construct a set Σ, which we call the alphabet, such that for each generator $g \in S$, we add the elements g, g^{-1} to Σ. Also, we add the identity element of G, which is e, to Σ. Now, any element of G can be written as a word whose letters are made of elements in Σ. For instance, consider the cyclic group of order 4, which has a presentation $\langle a \mid a^4 \rangle$. This group has four elements e, a, a^2, a^3, and a is a generator of G. For $A = a^{-1}$, our alphabet will be $\Sigma = \{a, A, e\}$. Further, we can write any element of G using letters from Σ. For example, the element a^3 can be written as aaa, which is the same as $aaaAa$ and also A.

Definition 5.1. The word problem for groups is the problem of determining whether two words consisting of letters from Σ are equivalent under the group relations R.

For the cyclic group, this is easy to decide. Given any two words, we can replace $aaaa$, $AAAA$ or aA by e. Doing this repeatedly, a word of any length can be brought down to a reduced word with fewer than 4 letters. By comparing the two reduced words, we can determine whether the two words are equivalent. This means that the word problem for the cyclic group is decidable. For some classes of groups, including braid groups, the word problem is decidable. In general however, this is not true. In [2], Collins shows an example of a group with a finite presentation having 10 generators and 27 relations, whose word problem is undecidable.

6 Word Problem for the Braid Group

Garside [3] showed that the braid group has a decidable word problem. The main idea of their proof is to define a "standard form" for a braid word. By showing that the standard form is unique, and by defining a procedure to convert any braid word to its standard form, we get a deterministic way of determining whether two words are equal in a finite number of steps.

Definition 6.1. A word consisting of an ordered sequence of generators σ_i in which no inverse of any generator occurs is called a **positive word**.

Definition 6.2. Two words are said to be **positively equal** if (i) they are identically equal, or (ii) they can be converted to each other through a sequence of positive words, using the braid relations. If A is positively equal to B, we write $A \doteq B$.

Definition 6.3. Let Π_k be the word $\sigma_1\sigma_2 \dots \sigma_k$ where all symbols appear in ascending order. The **fundamental word** of order $r+1$, denoted by Δ_r, is defined as the following word:

$$\Delta_r = \Pi_r \Pi_{r-1} \dots \Pi_1$$

For example, the fundamental word for the braid group on four strands, B_4, is $\Delta_3 = \sigma_1\sigma_2\sigma_3\sigma_1\sigma_2\sigma_1$. We study Δ_n because it has some nice properties that help us reach a standard form.

Let $\mathfrak{R}$ be a function from σ_i onto itself given by $\mathfrak{R}\sigma_i = \sigma_{n+1-i}$. This is the reflection function whose input is a generator σ_i and output is its reflection in B_{n+1}. For example, $\mathfrak{R}\sigma_1 = \sigma_n$, $\mathfrak{R}\sigma_2 = \sigma_{n-1}$ and so on. We can define the reflection of a word as the word obtained on taking the reflection of each symbol in order. Also, the reflection of a positive word will always be a positive word.

Lemma 1. *In B_{n+1}, we have $P\Delta_n = \Delta_n \mathfrak{R} P$ for any word P.*

Lemma 1 says that any word which contains Δ_n can be rearranged such that we "shift" Δ_n to the first position. For example, consider this braid word in B_3: $\sigma_2\sigma_2\sigma_1\sigma_2\sigma_1$, which is $\sigma_2\sigma_2\Delta_2$. Lemma 1 says that this is equivalent to $\Delta_2\mathfrak{R}(\sigma_2\sigma_2) = \Delta_2\sigma_1\sigma_1 = \sigma_1\sigma_2\sigma_1\sigma_1\sigma_1$.

Lemma 2. *In B_{n+1}, there exists a positive word X such that $\sigma_i X = \Delta_n$, for all σ_i.*

Lemma 2 means that the inverse of any generator, σ_i^{-1}, can be written as

$$\sigma_i^{-1} = X\Delta_n^{-1}$$

where X is a positive word. This gives us a procedure of systematically getting rid of all generator inverses, except for the ones in Δ_n^{-1}.

Theorem 3. *In B_{n+1}, any word can be expressed uniquely in the form $(\Delta_n)^m \bar{A}$*

The idea behind the proof is to write any braid word W in terms of positive generators and powers of Δ_n. We do this by separating out all generator inverses and positive words as follows:

$$\begin{aligned}
W &= W_1\sigma_k^{-1}W_2\sigma_l^{-1}\dots\sigma_p^{-1}W_{s+1} && \text{(where } W_i \text{ are positive words)}\\
&= W_1X_k\Delta_n^{-1}W_2X_l\Delta_n^{-1}\dots X_p\Delta_n^{-1}W_{s+1} && \text{(from Lemma 2)}\\
&= \Delta_n^{-1}\mathfrak{R}(W_1)X_kW_2\Delta_n^{-1}X_l\dots\Delta_n^{-1}X_pW_{s+1} && \text{(from Lemma 1)}\\
&\vdots\\
&= \Delta_n^{-s}A
\end{aligned}$$

Here, A is a positive word made from generators of W_i and their reflections. Notice that while A is guaranteed to be a positive word, it may still contain Δ_n. We then find the largest power t such that $A \doteq (\Delta_n)^t\bar{A}$, and then we write $W = \Delta_n^{-s}A = \Delta_n^{t-s}\bar{A}$. $\bar{A}$ is said to be **prime** to A. This step can be done by looking at the Cayley graph of A, details of which are left out because it is outside the scope of the course.

Definition 6.4. The form $(\Delta_n)^k\bar{A}$, where $\bar{A}$ is prime to Δ_n, is called as the **standard form** of W.

The uniqueness of the standard form can be shown as a proof by contradiction.

This gives us a procedure to go from our braid word to the representation $\Delta_m\bar{A}$ in finitely many steps. This would mean that words P and Q are equivalent if and only if their standard forms have the same m and $\bar{A}$. This shows that the braid group has a decidable word problem.

Example:
Let us work out an example of this procedure. Consider B_3, so that we have the fundamental word as $\Delta_2 = \sigma_1\sigma_2\sigma_1$. Consider a word $W = \sigma_2\sigma_2^{-1}\sigma_1\sigma_1\sigma_2^{-1}$ in B_3.

From Lemma 2, we can write $\sigma_1^{-1} = X\Delta_2^{-1}$ and $\sigma_2^{-1} = Y\Delta_2^{-1}$. Notice that $\sigma_1^{-1} = \sigma_2\sigma_1(\sigma_1\sigma_2\sigma_1)^{-1} = \sigma_2\sigma_1\Delta_2^{-1}$, which means $X = \sigma_2\sigma_1$. Similarly, we get $\sigma_2^{-1} = \sigma_1\sigma_2(\sigma_2\sigma_1\sigma_2)^{-1} = \sigma_1\sigma_2\Delta_2^{-1}$, so $Y = \sigma_1\sigma_2$.

Now, let us try to get W into its canonical form. First, getting rid of the inverses, we have

$$\begin{aligned}
W &= \sigma_2(\sigma_1\sigma_2\Delta_2^{-1})\sigma_1\sigma_1(\sigma_1\sigma_2\Delta_2^{-1}) \\
&= \Delta_2^{-1}\mathfrak{R}(\sigma_2\sigma_1\sigma_2)\sigma_1\sigma_1\sigma_1\sigma_2\Delta_2^{-1} \\
&= \Delta_2^{-1}\sigma_1\sigma_2\sigma_1\sigma_1\sigma_1\sigma_1\sigma_2\Delta_2^{-1} \\
&= (\Delta_2)^{-2}\mathfrak{R}(\sigma_1\sigma_2\sigma_1\sigma_1\sigma_1\sigma_1\sigma_2) \\
&= (\Delta_2)^{-2}\sigma_2\sigma_1\sigma_2\sigma_2\sigma_2\sigma_2\sigma_1
\end{aligned}$$

After this, we notice that $A = \sigma_2\sigma_1\sigma_2\sigma_2\sigma_2\sigma_2\sigma_1 = \Delta_2\sigma_2\sigma_2\sigma_2\sigma_1$, so that $\bar{A} = \sigma_2\sigma_2\sigma_2\sigma_1$, and $\Delta_2^{-1}\bar{A}$ is the standard form of W.

7 Knots in Higher Dimensions

A knot K in $\mathbb{R}^{n+2}$ is an embedding of S^n in $\mathbb{R}^{n+2}$. This is a higher dimensional analogue of a knot in $\mathbb{R}^3$. The unknotting problem is the problem of deciding whether a given knot K is isotopic to the unknot (i.e., S^n). Nabutovsky [5] showed that the unknotting problem is undecidable in $\mathbb{R}^n$ for $n \geq 4$. This means there is no algorithm that can decide whether or not K is a trivial knot for $n \geq 4$. This is somewhat surprising considering Haken [4] gives a procedure to solve the unknotting problem for $\mathbb{R}^3$.

The proof by Nabutovsky uses a result that there exists a group G with a specific structure corresponding to any knot K in $\mathbb{R}^n$ for $n \geq 4$. Further, the trivial group G corresponds to the trivial knot. Further, there is a reduction from the Halting problem to the problem of deciding whether G has a decidable word problem. This means that if we can decide whether the group has a decidable word problem, which corresponds to the unknot, we would be able to solve the Halting problem. However, we showed earlier that the Halting problem is undecidable. This means that our unknotting problem is also undecidable.

References

[1] Alonzo Church. An unsolvable problem of elementary number theory. *American journal of mathematics*, 58(2):345–363, 1936.

[2] Donald J Collins et al. A simple presentation of a group with unsolvable word problem. *Illinois Journal of Mathematics*, 30(2):230–234, 1986.

[3] Frank A Garside. The braid group and other groups. *The Quarterly Journal of Mathematics*, 20(1):235–254, 1969.

[4] Wolfgang Haken. Theorie der normalflächen. *Acta Mathematica*, 105(3-4):245–375, 1961.

[5] Alexander Nabutovsky and Shmuel Weinberger. Algorithmic unsolvability of the triviality problem for multidimensional knots. *Commentarii Mathematici Helvetici*, 71(1):426–434, 1996.

[6] Michael Sipser. *Introduction to the Theory of Computation*, volume 2. Thomson Course Technology Boston, 2006.

[7] Alan M Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London mathematical society*, 2(1):230–265, 1937.