The word problem in one-relator groups

Chris Hughes

Imperial College London

5th January 2021

The problem

The one-relator tactic can prove an equality in a group given one equality of group expressions.

It is an implementation of an algorithm by Wilhem Magnus (1932).

For example,

Let G be a group and let a, b, c be elements of G.

$$\text{If } abab^2 = 1 \text{ then prove } ab = ba$$
$$\text{If } ab = b^2a \text{ then prove } a^2b = b^4a^2$$

A Story

I wrote this tactic. Once I wrote it, my supervisor Kevin Buzzard tweeted about it.

A bunch of people replied saying that there are better ways of solving this problem.

There are practical *semidecision* procedures that solve the problem for more than one relation.

Two Semidecision Procedures

Knuth Bendix is one well know algorithm that can be used as a semidecision procedure for multiple relations.

Kyle Miller (Berkeley) commented on twitter that he had another semidecision procedure for multiple relations. (It is on his website).

Semi Decision Procedures vs Decision Procedures

Semidecision procedures terminates in a proof when the equality is true, but may not terminate when the goal is not provable.

Decision procedures always terminate on whether the goal is provable or not.

Given a group G, and $a, b \in G$, if $ab = ba^2$ and $ba = ab^2$ then $a = 1$.

There exists a group G, and $a, b \in G$, such that $ab = ba^2$ and $ba = ab^2$, but $a \neq 1$.

Knuth Bendix

Given a set of relations the Knuth Bendix algorithm finds a confluent rewriting system for those relations.
E.g, given the equalities

$$a^4 = 1, b^2 = 1, bab = a^3$$

Knuth Bendix generates the rewriting procedure

$$\begin{aligned} aa^{-1} &= 1 \\ a^{-1}a &= 1 \\ b^2 &= 1 \\ a^{-2} &= a^2 \\ b^{-1} &= b \\ ba &= a^{-1}b \\ ba^{-1} &= ab \end{aligned}$$

Usually there is no such *finite* rewriting procedure

Path Search Method

As an example consider the relation $abab^2 = 1$ and suppose we are trying to prove $ab = ba$.

First rearrange $ab = ba$ to the form $aba^{-1}b^{-1} = 1$

The relation $abab^2$ can be rearranged to a bunch of different equations, each with one letter on the lhs.

$$\begin{aligned}
a = b^{-2}a^{-1}b^{-1}, &\quad a^{-1} = bab^2 \\
b = a^{-1}b^{-2}a^{-1}, &\quad b^{-1} = ab^2a \\
a = b^{-1}a^{-1}b^{-2}, &\quad a^{-1} = b^2ab \\
b = a^{-1}b^{-1}a^{-1}b^{-1}, &\quad b^{-1} = baba \\
b = b^{-1}a^{-1}b^{-1}a^{-1}, &\quad b^{-1} = abab
\end{aligned}$$

We generate the set of all words that can be reached from my starting node by applying one rewrite rule once, reduce the generated words, and add each generated word to a set of nodes S.

For example, applying the rewrite $a = b^{-2}a^{-1}b^{-1}$, we obtain the equality $aba^{-1}b^{-1} = b^{-2}a^{-1}b^{-1}ba^{-1}b^{-1} = b^{-2}a^{-2}b^{-1}$. The word $b^{-2}a^{-2}b^{-1}$ is then added to S.

We then take the shortest word in S and repeat the process, maintaining a list of seen words to avoid repeating any words.

Word Length

Cyclically reduce words at each stage

$$aba^{-1}b^{-1}a^{-1}$$

Word Length

Cyclically reduce words at each stage

$$aba^{-1}b^{-1}a^{-1}$$

Word Length

Cyclically reduce words at each stage

$$a^{-1}$$

Tactic Outline

1. Convert all hypotheses to the form $r_i = 1$, and the goal to the form $w = 1$
2. Reify hypotheses r_i and goal w in the local context into a list of free group expressions.
3. Search for a path using an efficient representation of the free group, whilst keeping track of enough information to retrace the path.
4. Construct a Lean proof from the traced path.

Keeping track of proofs

Given an old word w, such that w, a proof step is the following

$$\text{Words } w_1 \text{ and } w_2\text{, such that } w_1 w_2 = w$$
$$\text{A relation } r_i \text{ such that } r_i \approx 1$$
$$\text{Two words } a \text{ and } b$$

If $w \approx 1$, then $a^{-1} w_1 b^{-1} r b w_2 a \approx 1$.

Proof

```
inductive proof_eq_one
  {G : Type*} [group G] (atoms : list G) :
  free_group → Prop
| one : proof_eq_one 1
| step :
  Π (word₁ rel word₂ conj old_word new_word rel_conj : free_group),
  proof_eq_one new_word →
  word₁ ++ word₂ = old_word
  → eval atoms rel = 1
  → conj⁻¹ * word₁ * rel_conj⁻¹ * rel * rel_conj * word₂ * conj = new_word
  → proof_eq_one old_word
```

Comparison of Three Methods

Magnus' Method

- Decision procedure
- One relation
- Implemented in Lean
- Very complex (months of work to implement)

Path Search Method

- Semidecision Procedure
- Multiple relations
- Implemented in Lean
- Quite simple ($\sim$1 week to implement)

Knuth Bendix

- Semidecision Procedure
- Multiple relations
- Not implemented in Lean
- GAP, MAGMA have nonverified implementations

Demo

What I've Learnt

- Brute forcey methods work when there are a small number of hypotheses, and the proof, if it exists, is not very long.
- Often we don't really want a decision procedure, we only need a semidecision procedure.

Brute forcey methods work when there are a small number of hypotheses, and the proof, if it exists, is not very long.

***Proposition 2.1**. Let $\mathscr{C}$ be a category or more generally an $(\infty,1)$-category or derivator. Consider a commuting diagram in $\mathscr{C}$ of the following shape:*

$$\begin{array}{ccccc} x & \longrightarrow & y & \longrightarrow & z \\ \downarrow & & \downarrow & & \downarrow \\ u & \longrightarrow & v & \longrightarrow & w \end{array}$$

Then:

1. *if the right square is a pullback, then the total rectangle is a pullback precisely if the left square is a pullback.*
2. *if the left square is a pushout, then the total rectangle is a pushout precisely if the right square is a pushout.*

https://ncatlab.org/nlab/show/pasting+law+for+pullbacks

Questions