Semigroup where BBBC=BBCB

Tim devised the following problem.

Consider the semigroup with three generators: A, B, and C. We know:

- AB = BA
- BC = CAB
- ABB = ABBC

We want to prove:

- BBBC = BBCB

The solution

Since this is a semigroup, the only things I could do were:

- use one of the three rewrite rules,
- forward-reason by multiplying both sides by an element, or
- backward-reason by cancelling an element from both sides.

I noticed that in trying to prove `BBBC = BBCB`, I could hop the C all the way to the front by repeatedly applying `BC=CAB`. So, I realized proving the goal is equivalent to proving `CABABAB = CABABB`. Then, I thought it might be nicer to rearrange, using the `AB=BA` rule, to something like `CAAABBB = CAABBB`. So that became my new goal — to "create an A" in a specific situation.

Now, I had to prove I could "create an A" in a specific situation. After a few attempts at the problem, the "intuition" behind the rules became engrained in me:

```
AB = BA     (the way to move around As and Bs)
BC = CAB    (the way to introduce/destroy As)
ABB = ABBC (the way to introduce/destroy Cs)
```

So how to prove that `CAAABBB = CAABBB`? At this point, I realized the only rule I hadn't used yet was `ABB=ABBC`. So I started from that statement. And I knew I wanted to hop the C (to create an A) by applying the rule `BC=CAB`.

```
ABB = ABBC
    = ABCAB     (hop up the C)
    = ACABCAB   (hop up the C)
```

To make my state look like the target, I needed to get the C all the way to the front. But I realized there is no way to move the C up without a B before it. So I started afresh.

```
ABB  = ABBC
BABB = BABBC      (set up a B at the front)
     = BABCAB     (hop up the C)
     = BACABAB    (hop up the C)
     = ABCABAB   (put the B before the C...)
     = ACABBAB   (...so I can hop up the C)
```

Ah but still, there was no way for that C to get ahead of that B. I realized moving a C up wasn't doable as long as there is an A in front. Another issue was that I had hopped the C twice, so now I had too many As.

Then I realized — I don't have to get the C all the way to the front. I could remove it entirely. That is, I could change my goal to proving the stronger statement: `AABBB = AAABBB`.

So then I started off the same track, but tweaked a little:

```
ABB  = ABBC
BABB = BABBC      (set up a B at the front)
     = BABCAB     (hop up the C)
     = ABBCAB     (set up for destruction)
     = ABBAB      (destroy the C)
ABBB = AABBB      (tidy up)
AABBB= AAABBB     (make it look like the goal)
```

And so, I can create "A"s!

So, using these intermediary lemmas, I could prove the statement:

```
AABBB = AAABBB       (by the lemma)
CAABBB = CAAABBB     (by multiplying by C)
BBBC = BBCB          (by the lemma)
```