A Free Group of Rotations of Rank 2

Jagadish Bapanapally[*] & Ruben Gamboa
University of Wyoming
Laramie, Wyoming
{jbapanap,ruben}@uwyo.edu

One of the key steps in the proof of the Banach-Tarski Theorem is the introduction of a free group of rotations. First, a free group of reduced words is generated where each element of the set is represented as an ACL2 list. Then we demonstrate that there is a one-to-one relation between the set of reduced words and a set of 3D rotations. In this paper we present a way to generate this set of reduced words and we prove group properties for this set. Then, we show a way to generate a set of 3D matrices using the set of reduced words. Finally we show a formalization of 3D rotations and prove that every element of the 3D matrices set is a rotation.

1 Introduction

The Banach-Tarski theorem [7] states that we can break the unit ball into a finite number of sets, then rotate the sets to form two identical copies of the unit ball. This seems impossible because it contradicts our intuition that when we partition the ball into finite sets, the total volume of the pieces must be the same as the volume of the original ball. This would be the case if all the pieces had a well-defined volume. The Banach-Tarski theorem is possible because the construction breaks the ball into non-measurable sets [4], which means they don't have a well-defined volume. These non-measurable sets are formed by the introduction of a free group of rotations. Using properties of this free group and with the help of the Axiom of Choice, the surface of the sphere is broken down into two equivalent sets. This construction is then extended to the whole unit ball.

The free group of rotations is formed by introducing a free group of reduced words and then showing one-to-one relation between the set of reduced words and the set of rotations. In section 2, we generate the set of reduced words using ACL2 lists. In section 3, we show a way to generate a set of 3D matrices using the set of reduced words. Then we show there is a one-to-one relation between the set of reduced words and the set of 3D matrices. In section 4, we formalize 3D rotations in ACL2(r) and we show every element of the 3D matrices set is a rotation thus generating a free group of rotations. Many properties of matrix algebra [3] and modular arithmetic [2] that are needed for the proof have already been formalized in ACL2(r). The matrix algebra that is formalized using the ACL2 two dimensional arrays contains a lot of properties that we need for the proof. For example matrix multiplication, matrix equivalence, matrix transpose, and properties like associativity of the matrix multiplication and $(m_1 \times m_2)^T = m_1^T \times m_2^T$ have been formalized. Also, there are properties about dimensions of the matrices. These formalized properties about matrices made us believe we can use these books and we have been proven correct as we are able to achieve the goal of generating a free group of rotations of rank 2.

[*]The research presented in this paper was partially supported by a grant from IOG Singapore Pte. Ltd.

2 A Free Group of Reduced Words

In this section, we introduce the free group over the letters a and b. This group contains all words that can be formed from a, b, a^{-1}, and b^{-1} such that no letter and its inverse appear together. For example, $abba$ is a member of this free group but $abb^{-1}a$ is not.

We use lists in ACL2(r) to represent words. A weak word is an empty list or a list that has characters a or a^{-1} or b or b^{-1}. For example, '(a b b^{-1} a^{-1}) is a weak word. In the ACL2(r) source files, we have defined the functions `wa`, `wa-inv`, `wb` and `wb-inv` which return the ACL2(r) characters #\\$a$, #\\$b$, #\\$c$, and #\\$d$ respectively. e.g., (`wa`)=#\\$a$. We use the ACL2(r) characters #\\$a$, #\\$b$, #\\$c$, and #\\$d$ to represent a, a^{-1}, b, and b^{-1} respectively, but in this paper we will simply refer to a, a^{-1}, b, and b^{-1} to avoid confusion. The predicate *weak-wordp* recognizes elements of the set of weak words, as shown below. Since ACL2(r) does not have support for infinite sets, such as a set of weak words, we represent these sets implicitly using recognizers for their elements.

```
(defun weak-wordp (w)
  (cond ((atom w) (equal w nil))
        (t (and (or (equal (first w) (wa))
                    (equal (first w) (wa-inv))
                    (equal (first w) (wb))
                    (equal (first w) (wb-inv)))
                (weak-wordp (rest w))))))
```

A reduced word is a weak word such that character a^{-1} does not appear beside the character a and character b^{-1} does not appear beside the character b in the list. For instance, '(a b a^{-1}) is a reduced word and '(a a^{-1} b) is not a reduced word. The predicates *a-wordp*, *a-inv-wordp*, *b-wordp*, and *b-inv-wordp* represent the set of reduced words that start with characters a, a^{-1}, b, and b^{-1} respectively. The predicate *reducedwordp*, as shown below, represents the set of all reduced words. *reducedwordp* returns true if the argument belongs to the set *a-wordp* or *a-inv-wordp* or *b-wordp* or *b-inv-wordp* or if it is an empty list.

```
(defun reducedwordp (x)
  (or (a-wordp x)
      (a-inv-wordp x)
      (b-wordp x)
      (b-inv-wordp x)
      (equal x '())))
```

The function *word-inverse* finds the inverse of a reduced word. If the argument is a weak word, *word-inverse* flips each character in the list to its inverse and then reverses the list, e.g., *word-inverse*('(a a^{-1} b^{-1})) = '(b a a^{-1}). Below are the definitions of the flip function and the inverse function.

```
;; Definition of the flip function
(defun word-flip (x)
  (cond ((atom x) nil)
        ((equal (car x) (wa)) (cons (wa-inv) (word-flip (cdr x))))
        ((equal (car x) (wa-inv)) (cons (wa) (word-flip (cdr x))))
        ((equal (car x) (wb)) (cons (wb-inv) (word-flip (cdr x))))
        ((equal (car x) (wb-inv)) (cons (wb) (word-flip (cdr x))))))

;; Definition of the Inverse operation
(defun word-inverse (x)
  (rev (word-flip x)))
```

The group operation *compose* takes two arguments. If the arguments are weak words, then the *compose* function first appends the two lists and then “fixes” the result by deleting any letter and its inverse that appear beside each other. Thus, the final result of *compose* is always a reduced word. E.g., *compose*(’(a b b), ’(b^{-1})) = ’(a b). Below are the definitions of the fixing function and the group operation *compose*.

```
;; Definition of the fixing function
(defun word-fix (w)
  (if (atom w)
      nil
    (let ((fixword (word-fix (cdr w))))
      (let ((w (cons (car w) fixword)))
        (cond ((equal fixword nil)
               (list (car w)))
              ((equal (car (cdr w)) (wa))
               (if (equal (car w) (wa-inv))
                   (cdr (cdr w))
                   w))
              ((equal (car (cdr w)) (wa-inv))
               (if (equal (car w) (wa))
                   (cdr (cdr w))
                   w))
              ((equal (car (cdr w)) (wb))
               (if (equal (car w) (wb-inv))
                   (cdr (cdr w))
                   w))
              ((equal (car (cdr w)) (wb-inv))
               (if (equal (car w) (wb))
                   (cdr (cdr w))
                   w)))))))

(defun compose (x y)
  (word-fix (append x y)))
```

If we denote the set of reduced words by $W(a,b)$, the set of reduced words starting with character a by $W(a)$, and similarly for $W(a^{-1})$, $W(b)$, and $W(b^{-1})$, then

$$W(a,b) = \text{'}() \cup W(a) \cup W(a^{-1}) \cup W(b) \cup W(b^{-1})$$

Considering the empty list as the identity element, we show below the group properties of the set of reduced words.

2.1 Closure Property

If x and y are reduced words, then (append x y) is a weak word as shown below by the lemma *closure-lemma*. If x is a weak word, then *word-fix*(x) returns a reduced word as shown below by the *weak-wordp-equivalent* lemma. So, *compose*(x,y) = *word-fix*(append x y) is a reduced word. This establishes that *compose* is closed over the set of reduced words as shown below by the lemma *closure-prop*.

```
(defthmd closure-lemma
  (implies (and (reducedwordp x)
                (reducedwordp y))
           (weak-wordp (append x y))))
```

```
(defthmd weak-wordp-equivalent
  (implies (weak-wordp x)
           (reducedwordp (word-fix x))))

(defthmd closure-prop
  (implies (and (reducedwordp x)
                (reducedwordp y))
           (reducedwordp (compose x y))))
```

2.2 Associative Property

By the definition, *word-fix* ”fixes” a weak word recursively starting from the tail of the list; i.e if *x*, *y*, *z* are weak words, then (*word-fix* (append *x* (*word-fix* (append *y z*)))) is equal to (*word-fix* (append *x y z*)) as shown below by the lemma *compose-assoc-lemma1*. Another key lemma required to prove that the set $W(a,b)$ satisfies the associative property is that if *x* is a reduced word, then *word-fix*(rev(*x*)) = (rev(*word-fix*(*x*))), which we proved by induction on *x*.

```
(defthm compose-assoc-lemma1
  (implies (and (weak-wordp x)
                (weak-wordp y)
                (weak-wordp z))
           (equal (word-fix (append x (word-fix (append y z))))
                  (word-fix (append x y z))))
  :hints ...)
```

The other two lemmas required to prove the associative property which are already proved in ACL2, are: if *x* and *y* are lists, then rev(rev *x*) = *x* and rev(append *x y*) = (append (rev *y*) (rev *x*)). Using these lemmas, below is the derivation of the associative property of the *compose* function with respect to the set of reduced words. If *x*, *y*, *z* are reduced words, then

$$\begin{aligned}
(\textit{compose}\ (\textit{compose}\ x\ y)\ z) &= (\textit{word-fix}\ (\text{append}\ (\textit{word-fix}\ (\text{append}\ x\ y))\ z))\\
&= (\text{rev}\ (\text{rev}\ (\textit{word-fix}\ (\text{append}\ (\textit{word-fix}\ (\text{append}\ x\ y))\ z))))\\
&= (\text{rev}\ (\textit{word-fix}\ (\text{rev}\ (\text{append}\ (\textit{word-fix}\ (\text{append}\ x\ y))\ z))))\\
&= (\text{rev}\ (\textit{word-fix}\ (\text{append}\ (\text{rev}\ z)\ (\text{rev}\ (\textit{word-fix}\ (\text{append}\ x\ y))))))\\
&= (\text{rev}\ (\textit{word-fix}\ (\text{append}\ (\text{rev}\ z)\ (\textit{word-fix}\ (\text{rev}\ (\text{append}\ x\ y))))))\\
&= (\text{rev}\ (\textit{word-fix}\ (\text{append}\ (\text{rev}\ z)\ (\textit{word-fix}\ (\text{append}\ (\text{rev}\ y)\ (\text{rev}\ x))))))\\
&= (\text{rev}\ (\textit{word-fix}\ (\text{append}\ (\text{rev}\ z)\ (\text{rev}\ y)\ (\text{rev}\ x))))\\
&= (\textit{word-fix}\ (\text{rev}\ (\text{append}\ (\text{rev}\ z)\ (\text{rev}\ y)\ (\text{rev}\ x))))\\
&= (\textit{word-fix}\ (\text{append}\ x\ y\ z))\\
&= (\textit{word-fix}\ (\text{append}\ x\ (\textit{word-fix}\ (\text{append}\ y\ z))))\\
&= (\textit{compose}\ x\ (\textit{compose}\ y\ z))
\end{aligned}$$

2.3 Inverse Property

By induction on *x*, first we show if *x* is a reduced word, then (rev *x*) and (*word-flip x*) are reduced words and thus (*word-inverse x*) is a reduced word. Now since (*word-inverse x*) is a reduced word, using the associative property and by induction on x, (*compose* x (*word-inverse* x)) results in an empty list as shown below by the *reduced-inverse* lemma below. This proves that right inverse of the reduced word x is (*word-inverse* x). To prove the left inverse of x is also equal to (*word-inverse* x), we can use the *reduced-inverse* lemma. In the *reduced-inverse* lemma in place of x if we have (*word-inverse* x) and if (*word-inverse* (*word-inverse* x)) is equal to x, then (*word-inverse* x) becomes the left inverse of x. We have proved (*word-inverse* (*word-inverse* x)) is equal to x by functionally instantiating the *equal-by-nths* [1] lemma. We have functionally instantiated the *equal-by-nths* lemma with the hypothesis being x a weak word, left hand side of the equivalence being (*word-inverse* (*word-inverse* x)) and the right hand side of the equivalence being just x. To finish the proof, we needed proofs that both the lists (*word-inverse* (*word-inverse* x)) and x have same characters at any specified index and they both have the same length. Thus we have proved that for every element x in the reduced word set there exists an inverse of x which is equal to (*word-inverse* x).

```
(defthmd reduced-inverse
  (implies (reducedwordp x)
           (equal (compose x (word-inverse x)) '()))
  :hints ...)
```

3 A Free Group of 3D Matrices

Matrices in ACL2 are represented with the data structure *array2p*. We define a predicate *r3-matrixp* that recognizes the set of 3D matrices: *r3-matrixp* returns true if the argument is of type *array2p*, if its dimensions are 3×3, and if each element of the matrix is a real number.

We now define the four matrices A^+, A^-, B^+, and B^- as

$$A^{\pm} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{3} & \mp\frac{2\sqrt{2}}{3} \\ 0 & \pm\frac{2\sqrt{2}}{3} & \frac{1}{3} \end{bmatrix} \quad B^{\pm} = \begin{bmatrix} \frac{1}{3} & \mp\frac{2\sqrt{2}}{3} & 0 \\ \pm\frac{2\sqrt{2}}{3} & \frac{1}{3} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

and we associate these matrices with the letters a, a^{-1}, b, and b^{-1} from the free group respectively. Moreover, we associate a list $(x_1, x_2, \ldots, x_n) \in W(a,b)$ with the matrix $X_1 \times X_2 \times \cdots \times X_n$, where $\times$ denotes matrix multiplication, and X_i is the matrix associated with letter x_i. The recursive function *rotation* performs this mapping from words in the free group to 3D matrices. If we denote the resulting set as $R(a,b)$, then $R(a,b) = \{rotation(w) \mid w \in W(a,b)\}$. By induction, it is easy to verify that every element of the set $R(a,b)$ belongs to *r3-matrixp*.

To show the set $R(a,b)$ is a free group homomorphic to $W(a,b)$, we show that if $w \in W(a,b)$ and w is not the empty list, then $rotation(w)$ is not equal to I, the identity matrix. Equivalently, we show that $(rotation(w))(0,1,0) \neq (0,1,0)$, unless w is the empty list.

To do this, suppose that $w \in R(a,b)$, and consider the rotation $R(w)$. In particular, suppose that $R(w)$ transposes the point $(0,1,0)$ to (x', y', z'). Define (x,y,z) as

$$(x,y,z) = 3^n \left(\frac{x'}{\sqrt{2}}, y', \frac{z'}{\sqrt{2}} \right)$$

where $n = |w|$. Using induction, we show x, y, and z are integers.

So now suppose that $(rotation(w))(0,1,0) = (0,1,0)$ for some non-empty word w. It follows that $(x,y,z) = (0, 3^n, 0)$, where $n = |w| > 0$, thus $x \equiv y \equiv z \equiv 0 \pmod 3$. But this cannot be the case. If

$|w| = 1$, then $rotation(w)$ is one of $A^{\pm}$ or $B^{\pm}$, and considering each of the four cases by brute force, it is clear that $(x,y,z) \not\equiv (0,0,0) \pmod 3$. Using induction, there are 16 cases to consider, but in all of these cases we can again conclude that $(x,y,z) \not\equiv (0,0,0) \pmod 3$. This shows that if $|w| > 0$, then $rotation(w)$ is not the identity matrix.

Here we want to mention two key lemmas needed to prove the one-to-one relation between the set of 3D matrices and the set of reduced words. First, if $w_1, w_2 \in W(a,b)$, then by the definition of *rotation* and *compose*, $rotation(w_1) \times rotation(w_2) = rotation(compose(w_1,w_2))$. Second, if $r \in R(a,b)$, then $\exists w \in W(a,b)$ such that $r = rotation(w)$, and by the previous lemma, $r^{-1} = rotation(w^{-1})$. Moreover, since $w^{-1} \in W(a,b)$, $r^{-1} \in R(a,b)$. Now, if $w_1, w_2 \in W(a,b)$ and $w_1 \neq w_2$ and $r_1 = rotation(w_1)$ and $r_2 = rotation(w_2)$, then using the proof that, if $|w| > 0$, then $rotation(w) \neq I$, $r_1 \times r_2^{-1} \neq I$, which implies $r_1 \neq r_2$. This proves there is a one-to-one relation between the set $R(a,b)$ and the set $W(a,b)$. So defining $R(a) = \{rotation(w) \mid w \in W(a)\}$, $R(a^{-1}) = \{rotation(w) \mid w \in W(a^{-1})\}$, $R(b) = \{rotation(w) \mid w \in W(b)\}$, and $R(b^{-1}) = \{rotation(w) \mid w \in W(b^{-1})\}$, then the set of rotations $R(a,b)$ can be partitioned as

$$R(a,b) = I \cup R(a) \cup R(a^{-1}) \cup R(b) \cup R(b^{-1}).$$

4 A Free Group of Rotations of Rank 2

In this section we formalize 3D rotations and prove every element of the 3D matrices set is a rotation. As discussed previously, the matrix transpose operation (*m-trans*) was formalized in prior work [3], and as part of that, it was shown that $(A \times B)^T = B^T \times A^T$.

We extended that formalization by introducing the function *r3-m-determinant* that computes the determinant of a matrix, the function *r3-m-inverse* that computes the inverse of a 3D matrix (when possible). Using these functions, we defined the predicate *r3-rotationp* that recognizes rotations in $\mathbb{R}^3$. A matrix M is a rotation matrix if it satisfies these conditions [6]:

- M is a 3D matrix,
- $M^{-1} = M^T$, and
- $\det(M) = 1$.

Another important detail is that every element of $R(a,b)$ must be a rotation of $\mathbb{R}^3$. Given the correspondence between $R(a,b)$ and $W(a,b)$ established in section 3, what we need to show is that for any $w \in W(a,b)$, $rotation(w)$ satisfies the axioms of a rotation. This was done using induction on the list w. It is easy to verify that the base cases are rotations; i.e., I, A^+, A^-, B and B^- are all rotation matrices. For the induction to go through, the lemma we need to prove $rotation(xw)$ is a rotation in $\mathbb{R}^3$ given that $rotation(w)$ is a rotation, is that the product of two rotation matrices M_1 and M_2 is also a rotation matrix. Below is the the proof of this lemma, and some other lemmas from matrix algebra that we proved in ACL2(r).

- $\textit{r3-matrixp}(m_1) \wedge \textit{r3-matrixp}(m_2) \implies \textit{r3-matrixp}(m_1 \times m_2)$
- $\textit{r3-matrixp}(m_1) \wedge \textit{r3-matrixp}(m_2) \implies \det(m_1 \times m_2) = \det(m_1) \cdot det(m_2)$
- $\textit{r3-matrixp}(m) \implies m \times I = I \times m = m$
- $\textit{r3-matrixp}(m) \wedge \det(m) \neq 0 \implies m \times m^{-1} = m^{-1} \times m = I$
- $\textit{r3-matrixp}(m_1) \wedge \det(m_1) \neq 0 \wedge \textit{r3-matrixp}(m_2) \wedge \det(m_2) \neq 0$
  $\implies (m_1 \times m_2)^{-1} = m_2^{-1} \times m_1^{-1}$

- $\textit{r3-rotationp}(m_1) \wedge \textit{r3-rotationp}(m_2) \implies \textit{r3-rotationp}(m_1 \times m_2)$
- $\textit{r3-rotationp}(m) \implies \textit{r3-rotationp}(m^{-1})$
- Rotations preserve distances [5]. Let $p_1 = (x_1, y_1, z_1)$ and R be a rotation matrix, and consider $p_2 = Rp_1 = (x_2, y_2, z_2)$. Using the previous lemmas,

$$\begin{aligned}
x_1^2 + y_1^2 + z_1^2 &= p_1^T \times p_1 \\
&= p_1^T \times (I \times p_1) \\
&= p_1^T \times ((R^{-1} \times R) \times p_1) \\
&= p_1^T \times ((R^T \times R) \times p_1) \\
&= (p_1^T \times R^T) \times (R \times p_1) \\
&= (R \times p_1)^T \times (R \times p_1) \\
&= p_2^T \times p_2 \\
&= x_2^2 + y_2^2 + z_2^2.
\end{aligned}$$

5 Conclusion

In this paper we presented a way to generate the free group of reduced words using ACL2 lists. Using this set we have generated a free group of 3D matrices. Then we have shown a formalization of 3D rotations in ACL2(r) and we proved that every element of the 3D matrices set is a 3D rotation. When we apply these rotations on S^2, then with the help of the Axiom of Choice we can form two copies of S^2 minus the set of the poles of the rotations. This is called the Hausdorff's Paradox which is the next step in the proof of the Banach-Tarski theorem. We are currently working to formalize the Hausdorff's paradox, and then we will prove the Banach-Tarski theorem.

References

[1] *Equal-by-nths*. `https://www.cs.utexas.edu/users/moore/acl2/manuals/current/manual/index-seo.php/ACL2____EQUAL-BY-NTHS`. Accessed: 2022-02-26.

[2] Piergiorgio Bertoli & Paolo Traverso (2000): *Design Verification of a Safety-Critical Embedded Verifier*, pp. 233–245. Springer US, Boston, MA, doi:`10.1007/978-1-4757-3188-0_14`.

[3] Ruben Gamboa, John Cowles & JV Baalen (2003): *Using ACL2 Arrays to Formalize Matrix Algebra*. In: *Fourth International Workshop on the ACL2 Theorem Prover and Its Applications (ACL2'03)*, 1.

[4] Thomas J Jech (2008): *The Axiom of Choice*. Courier Corporation.

[5] Rotation matrix (2021): *Rotation matrix — Wikipedia, The Free Encyclopedia*. `https://en.wikipedia.org/wiki/Rotation_matrix`. Online; Accessed: 2022-02-04.

[6] Madeline Tremblay (2017): *The Banach-Tarski Paradox*. Unpublished.

[7] Tom Weston (2016): *The Banach-Tarski Paradox*. *Citado* 2, p. 15.