SEARCHING FOR MATHEMATICAL STRUCTURES

Georgie Payne

1 Discussion: Commutative Finite Division Rings

The focus of this project is on commutative finite division rings, which are represented as a finite set of elements satisfying specific additive and multiplicative properties under some algebra. To be a *commutative ring*, the commutative property must hold for addition and multiplication on every element in the set ($ab = ba$). A commutative division ring has the additional property that every element in the set has a multiplicative inverse. When the set is also made up of a finite number of elements, this is called a commutative finite division ring. These properties for the commutative finite division ring are specified as follows over a finite set:

1.1 Addition

- *Commutative property*: For every two elements in the set a, b,

$$a + b = b + a$$

- *Associative property*: For every three elements in the set a, b, c,

$$a + (b + c) = (a + b) + c$$

- *Additive identity*: For every element a, there must be an additive identity 0_A (or 0) such that

$$a + 0 = a$$

- *Additive inverse*: For every element a, there must be an additive inverse $-a$ so that

$$a + (-a) = 0$$

1.2 Multiplication

- *Commutative property*: For every two elements in the set a, b,

$$a \times b = b \times a$$

- *Associative property*: For every three elements in the set a, b, c,

$$a \times (b \times c) = (a \times b) \times c$$

- *Multiplicative identity*: For every element a, there must be a multiplicative identity 1_A (or 1) such that

$$a \times 1 = a$$

- *Multiplicative inverse*: For every element a, there must be a multiplicative inverse b so that

$$a \times b = 1$$

1.3 Distribution

In order for addition and multiplication tables to have cohesion, they must satisfy the distributive property, which says that given any elements a, b, c,

$$a \times (b + c) = (a \times b) + (a \times c)$$

1.4 Example: $\mathbb{Z}_3$ vs $\mathbb{Z}_4$

The set of integers mod 3, represented as $\mathbb{Z}_3$, represents the set $\{0, 1, 3\}$, since these are all the possible remainders when dividing any integer by 3. This set with the mod n rules produces a commutative finite division ring, which can be verified by the axioms above. The set $\mathbb{Z}_4$, on the other hand, fails the multiplicative inverse property.

- Commutative: For both $\mathbb{Z}_3$ and $\mathbb{Z}_4$, the commutative property holds because the integer and modulo operations are identical. If $a \equiv a \pmod{n}$, meaning any number is congruent to itself mod n, then $a + b \equiv (a + b) \pmod{n}$ because $a + b = b + a$ for all integers. Similarly, $a \times b \equiv b \times a \pmod{n}$.

- Associative: The associate property also holds due to the integer properties. Since integers are associative, $a + (b + c) \equiv (a + b) + c \pmod{n}$ and $a + (b + c) \equiv (a \times b) \times c \pmod{n}$.

- Identity: The additive and multiplicative identities for all integers $\mathbb{Z}_n$ are 0 and 1 which also reflects integer properties. For addition, $a+0 \equiv a \pmod{n}$, and for multiplication, $a \times 1 \equiv a \pmod{n}$.

- Additive inverse: In $\mathbb{Z}_n$, just like integers, each element has an additive inverse and it is itself negated: $a + (-a) \equiv 0 \pmod{n}$

- Multiplicative inverse: The multiplicative inverse, which sets a commutative ring apart from a commutative division ring, is what holds for $\mathbb{Z}_3$ but fails for $\mathbb{Z}_4$. Every nonzero element in $\mathbb{Z}_3$ has an inverse: $1 \times 1 \equiv 1 \pmod{3}$ and $2 \times 2 \equiv 1 \pmod{3}$. However, in $\mathbb{Z}_4$, 2 does not have an inverse: $2 \times 0 \equiv 0 \pmod{4}$, $2 \times 1 \equiv 2 \pmod{4}$, $2 \times 2 = 0 \pmod{4}$, and $2 \times 3 \equiv 2 \pmod{4}$.

1.5 Project goal

This project aims to use the definition of commutative finite division rings to search for all possible commutative finite division rings of sizes 2, 3, 4, and 5. This search is implemented by generating all combinations of n elements in an $n \times n$ matrix for both addition and multiplication and then checking which ones the axioms hold for. In order to speed up the process, some entries are pre-populated with entries that satisfy the identity axioms. I also used the commutative property to populate the second half of the table so that I could explore fewer combinations.

2 Python Code

```
import itertools
import sys

def generate_tables(n, opp):
    # index's s1, s2, ...
    symbols = [f's{i}' for i in range(n)]
    # create empty table
```

```
    table = [[None for j in range(n)] for i in range(n)]

    # a + 0 = a for addition, a * 1 = a for multipliation
    identity = 0 if opp == "add" else 1
    # fill in identities
    for i in range(n):
        table[identity][i] = symbols[i]
        table[i][identity] = symbols[i]

    # get indices that need to be filled in
    # j from i to n leaves out half of the columns, which can be populated according the the com
    none_indices = [(i, j) for i in range(n) for j in range(i, n) if table[i][j] is None]

    # fill in other entries and add to table list
    def generate_combos():
        # Generate all possible completions of the None cells
        for combo in itertools.product(symbols, repeat=len(none_indices)):
            filled_table = [row[:] for row in table]  # copy table into new one
            for index, value in zip(none_indices, combo): # fill in empty entries
                i, j = index
                # fill in entry and commutative entry
                filled_table[i][j] = value
                filled_table[j][i] = value

            yield filled_table

    return generate_combos()


def is_commutative(table):
    # check commutativity for each pair
    for row in range(len(table)):
        for col in range(len(table)):
            if table[row][col] != table[col][row]: # check if a ~ b = b ~ a
                return False

    return True

def is_associative(table):
    n = len(table)
    element_dict = {f's{i}': i for i in range(n)}
    for a in range(n):  # Iterate over all possible 'a'
        for b in range(n):  # Iterate over all possible 'b'
            for c in range(n):  # Iterate over all possible 'c'
                # Check if a ~ (b ~ c) == (a ~ b) ~ c
                ab = element_dict[table[a][b]]
                bc = element_dict[table[b][c]]
                if table[a][bc] != table[ab][c]:
                    return False
    return True
```

```
def has_identity(table, opp):
    # a + 0 = a for addition, a * 1 = a for multipliation
    identity = 0 if opp == "add" else 1

    for a in range(len(table)):
        expected = f's{a}'
        # Check 0 + a = a and a + 0 = a for each a (or a * 1 = a and 1 * a = a)
        if table[identity][a] != expected or table[a][identity] != expected:
            return False
    return True

def has_inverse(table, opp):
    # check if each number has an inverse
    for a in range(len(table)):
        # for multiplicative, only non zero elements
        if (opp == "mult" and a == 0):
            break

        has_inverse = False
        # a + b = 0 for additive, a * b = 1 for multiplicative
        identity = 's0' if opp == "add" else 's1'
        # check if each element has an inverse
        for b in range(len(table)):
            if table[a][b] == identity and table[b][a] == identity:
                has_inverse = True
        # return false if one was not found
        if not has_inverse:
            return False

    return True

def is_distributive(add_table, mult_table):
    n = len(add_table)
    element_dict = {f's{i}': i for i in range(n)} # for converting to index

    # for each a, b, and c, check if distributive property holds
    for a in range(n):
        for b in range(n):
            for c in range(n):
                bc_sum = element_dict[add_table[b][c]]
                ab_prod = element_dict[mult_table[a][b]]
                ac_prod = element_dict[mult_table[a][c]]
                if mult_table[a][bc_sum] != add_table[ab_prod][ac_prod]:
                    return False
    return True

def find_rings(n):
    # get all possible sum and multiplication tables
    sum_tables = generate_tables(n, "add")
    mult_tables = generate_tables(n, "mult")
```

```
    # check validity of each table
    valid_sum_tables = []
    valid_mult_tables = []
    for table in sum_tables:
        if is_commutative(table) and is_associative(table) and has_identity(table, "add")
        and has_inverse(table, "add"):
            valid_sum_tables += [table]

    for table in mult_tables:
        if is_commutative(table) and is_associative(table) and has_identity(table, "mult")
        and has_inverse(table, "mult"):
            valid_mult_tables += [table]

    # check distributive property for each add/multiplaication table pair
    valid_combos = []
    for add_table in valid_sum_tables:
        for mult_table in valid_mult_tables:
            if is_distributive(add_table, mult_table):
                valid_combos += [(add_table, mult_table)]

    return valid_combos

def display_rings(n):
    valid_rings = find_rings(n)  # get the rings

    # for each add/multiplaication table pair
    for idx, (add_table, mult_table) in enumerate(valid_rings):
        print(f"Ring {idx + 1}:\n")

        # Display the addition table
        print("Addition Table:")
        for row in add_table:
            print(row)
        print()

        # display multiplication table
        print("Multiplaication Table:")
        for row in mult_table:
            print(row)
        print()


if __name__ == '__main__':
    n = int(sys.argv[1])
    display_rings(n)
```

3 Results and visualization

My Python code for generating all tables for each size revealed that there is one commutative finite division ring for sets of two and three. For elements of size four and five there were 6 rings. The

tables for each size are shown below with $\mathbb{Z}_n$ division rings colored in green:

2 Element Rings

Addition

$+$	s_0	s_1
s_0	s_0	s_1
s_1	s_1	s_0

Multiplication

$*$	s_0	s_1
s_0	s_0	s_0
s_1	s_0	s_1

3 Element Rings

Addition

$+$	s_0	s_1	s_2
s_0	s_0	s_1	s_2
s_1	s_1	s_2	s_0
s_2	s_2	s_0	s_1

Multiplication

$*$	s_0	s_1	s_2
s_0	s_0	s_0	s_0
s_1	s_0	s_1	s_2
s_2	s_0	s_2	s_1

4 Element Rings

Ring 1: Addition

$+$	s_0	s_1	s_2	s_3
s_0	s_0	s_1	s_2	s_3
s_1	s_1	s_0	s_3	s_2
s_2	s_2	s_3	s_0	s_1
s_3	s_3	s_2	s_1	s_0

Ring 1: Multiplication

$*$	s_0	s_1	s_2	s_3
s_0	s_0	s_0	s_0	s_0
s_1	s_0	s_1	s_2	s_3
s_2	s_0	s_2	s_0	s_2
s_3	s_0	s_3	s_2	s_1

Ring 2: Addition

$+$	s_0	s_1	s_2	s_3
s_0	s_0	s_1	s_2	s_3
s_1	s_1	s_0	s_3	s_2
s_2	s_2	s_3	s_0	s_1
s_3	s_3	s_2	s_1	s_0

Ring 2: Multiplication

$*$	s_0	s_1	s_2	s_3
s_0	s_0	s_0	s_0	s_0
s_1	s_0	s_1	s_2	s_3
s_2	s_0	s_2	s_1	s_3
s_2	s_0	s_3	s_3	s_0

Ring3: Addition

$+$	s_0	s_1	s_2	s_3
s_0	s_0	s_1	s_2	s_3
s_1	s_1	s_0	s_3	s_2
s_2	s_2	s_3	s_0	s_1
s_3	s_3	s_2	s_1	s_0

Ring 3: Multiplication

$*$	s_0	s_1	s_2	s_3
s_0	s_0	s_0	s_0	s_0
s_1	s_0	s_1	s_2	s_3
s_2	s_0	s_2	s_2	s_0
s_2	s_0	s_3	s_0	s_3

Ring 4: Addition

$+$	s_0	s_1	s_2	s_3
s_0	s_0	s_1	s_2	s_3
s_1	s_1	s_0	s_3	s_2
s_2	s_2	s_3	s_0	s_1
s_3	s_3	s_2	s_1	s_0

Ring 4: Multiplication

$*$	s_0	s_1	s_2	s_3
s_0	s_0	s_0	s_0	s_0
s_1	s_0	s_1	s_2	s_3
s_2	s_0	s_2	s_3	s_1
s_2	s_0	s_3	s_1	s_2

Ring 5: Addition

$+$	s_0	s_1	s_2	s_3
s_0	s_0	s_1	s_2	s_3
s_1	s_1	s_2	s_3	s_0
s_2	s_2	s_3	s_0	s_1
s_3	s_3	s_0	s_1	s_2

Ring 5: Multiplication

$*$	s_0	s_1	s_2	s_3
s_0	s_0	s_0	s_0	s_0
s_1	s_0	s_1	s_2	s_3
s_2	s_0	s_2	s_0	s_2
s_3	s_0	s_3	s_2	s_1

Ring 6: Addition

$+$	s_0	s_1	s_2	s_3
s_0	s_0	s_1	s_2	s_3
s_1	s_1	s_3	s_0	s_2
s_2	s_2	s_0	s_3	s_1
s_3	s_3	s_2	s_1	s_0

Ring 6: Multiplication

$*$	s_0	s_1	s_2	s_3
s_0	s_0	s_0	s_0	s_0
s_1	s_0	s_1	s_2	s_3
s_2	s_0	s_2	s_1	s_3
s_2	s_0	s_3	s_3	s_0

5 Element Rings

Ring 1: Addition

$+$	s_0	s_1	s_2	s_3	s_4
s_0	s_0	s_1	s_2	s_3	s_4
s_1	s_1	s_2	s_3	s_4	s_0
s_2	s_2	s_3	s_4	s_0	s_1
s_3	s_3	s_4	s_0	s_1	s_2
s_4	s_4	s_0	s_1	s_2	s_3

Ring 1: Multiplication

$*$	s_0	s_1	s_2	s_3	s_4
s_0	s_0	s_0	s_0	s_0	s_0
s_1	s_0	s_1	s_2	s_3	s_4
s_2	s_0	s_2	s_4	s_1	s_3
s_3	s_0	s_3	s_1	s_4	s_2
s_4	s_0	s_4	s_3	s_2	s_1

Ring 2: Addition

$+$	s_0	s_1	s_2	s_3	s_4
s_0	s_0	s_1	s_2	s_3	s_4
s_1	s_1	s_2	s_4	s_0	s_3
s_2	s_2	s_4	s_3	s_1	s_0
s_3	s_3	s_0	s_1	s_4	s_2
s_4	s_4	s_3	s_0	s_2	s_1

Ring 2: Multiplication

$*$	s_0	s_1	s_2	s_3	s_4
s_0	s_0	s_0	s_0	s_0	s_0
s_1	s_0	s_1	s_2	s_3	s_4
s_2	s_0	s_2	s_3	s_4	s_1
s_3	s_0	s_3	s_4	s_1	s_2
s_4	s_0	s_4	s_1	s_2	s_3

Ring 3: Addition

$+$	s_0	s_1	s_2	s_3	s_4
s_0	s_0	s_1	s_2	s_3	s_4
s_1	s_1	s_3	s_0	s_4	s_2
s_2	s_2	s_0	s_4	s_1	s_3
s_3	s_3	s_4	s_1	s_2	s_0
s_4	s_4	s_2	s_3	s_0	s_1

Ring 3: Multiplication

$*$	s_0	s_1	s_2	s_3	s_4
s_0	s_0	s_0	s_0	s_0	s_0
s_1	s_0	s_1	s_2	s_3	s_4
s_2	s_0	s_2	s_1	s_4	s_3
s_3	s_0	s_3	s_4	s_2	s_1
s_4	s_0	s_4	s_3	s_1	s_2

Ring 4: Addition

$+$	s_0	s_1	s_2	s_3	s_4
s_0	s_0	s_1	s_2	s_3	s_4
s_1	s_1	s_3	s_4	s_2	s_0
s_2	s_2	s_4	s_1	s_0	s_3
s_3	s_3	s_2	s_0	s_4	s_1
s_4	s_4	s_0	s_3	s_1	s_2

Ring 4: Multiplication

$*$	s_0	s_1	s_2	s_3	s_4
s_0	s_0	s_0	s_0	s_0	s_0
s_1	s_0	s_1	s_2	s_3	s_4
s_2	s_0	s_2	s_4	s_1	s_3
s_3	s_0	s_3	s_1	s_4	s_2
s_4	s_0	s_4	s_3	s_2	s_1

Ring 5: Addition

$+$	s_0	s_1	s_2	s_3	s_4
s_0	s_0	s_1	s_2	s_3	s_4
s_1	s_1	s_4	s_0	s_2	s_3
s_2	s_2	s_0	s_3	s_4	s_1
s_3	s_3	s_2	s_4	s_1	s_0
s_4	s_4	s_3	s_1	s_0	s_2

Ring 5: Multiplication

$*$	s_0	s_1	s_2	s_3	s_4
s_0	s_0	s_0	s_0	s_0	s_0
s_1	s_0	s_1	s_2	s_3	s_4
s_2	s_0	s_2	s_1	s_4	s_3
s_3	s_0	s_3	s_4	s_2	s_1
s_4	s_0	s_4	s_3	s_1	s_2

Ring 6: Addition

$+$	s_0	s_1	s_2	s_3	s_4
s_0	s_0	s_1	s_2	s_3	s_4
s_1	s_1	s_4	s_3	s_0	s_2
s_2	s_2	s_3	s_1	s_4	s_0
s_3	s_3	s_0	s_4	s_2	s_1
s_4	s_4	s_2	s_0	s_1	s_3

Ring 6: Multiplication

$*$	s_0	s_1	s_2	s_3	s_4
s_0	s_0	s_0	s_0	s_0	s_0
s_1	s_0	s_1	s_2	s_3	s_4
s_2	s_0	s_2	s_3	s_4	s_1
s_3	s_0	s_3	s_4	s_1	s_2
s_4	s_0	s_4	s_1	s_2	s_3