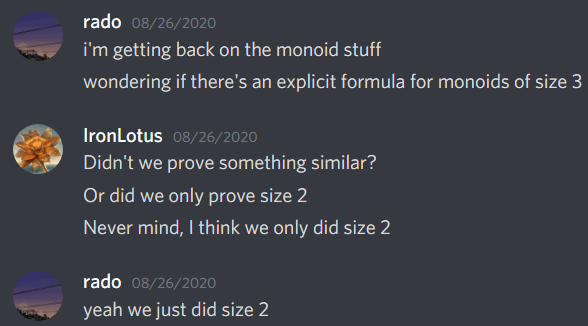
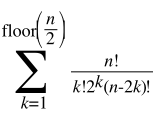
Finding Explicit Formulas for families of submonoids of size m=3, of the composition monoid of integer valued functions [0,n)∩ℤ[0,n)∩ℤ

Background

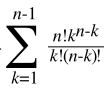
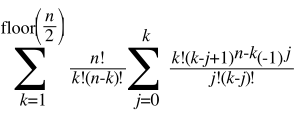


For reference, size m=2 was this:

Insert formula for # of functions consisting of only 2-loops given n elements



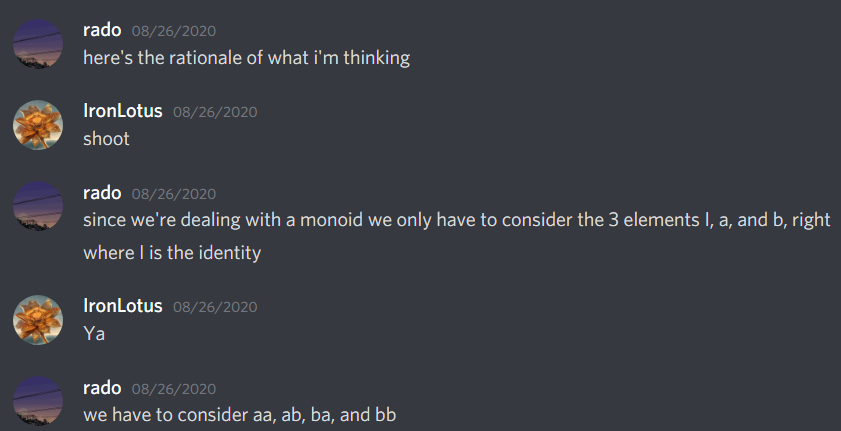
OEIS Formula: <https://oeis.org/A001189>

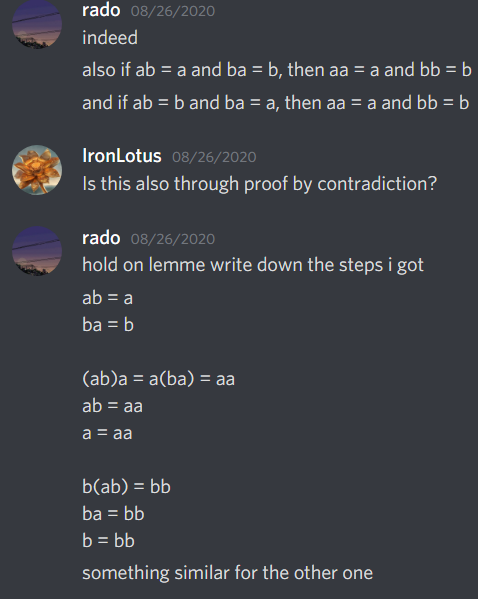
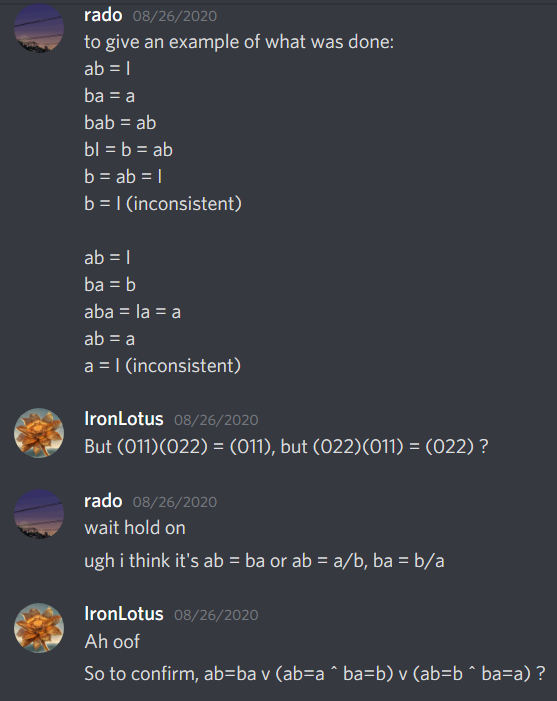
Insert formula for # of functions consisting of only 1-paths given n elements  
 Alternate: 

OEIS Formula: <https://oeis.org/A235596> and <https://oeis.org/A000248>

Now onwards to m=3.

Composition table approach:







Here are the 5 families discovered.

family 1 (size formula known):

aa = b

ab = I

ba = I

bb = a

family 2 (size formula known):

aa = a

ab = a

ba = b

bb = b

family 3 (size formula known):

aa = a

ab = b

ba = a

bb = b

family 4 (size formula known):

aa = b

ab = a

ba = a

bb = b

family 5:

aa = a

ab = a

ba = a

bb = I, a, or b

Example monoids of each family, where n=3:

1: (012), (120), (201)

2: (012), (011), (022); (012), (111), (222)

3: (012), (010), (011)

4: (012), (121), (212)

5:

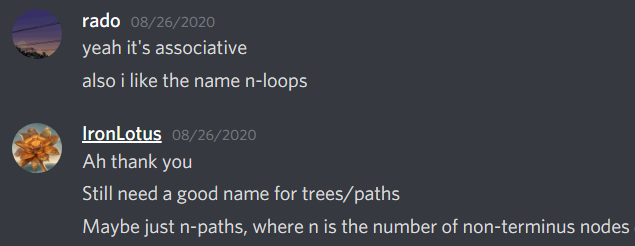
.1: (012), (000), (021)

.2: (012), (000), (001)

.3: (012), (000), (002)

Function graph approach:

After defining families through algebraic properties / composition tables, and finding example monoids of each family, we turned to a graph based approach to possible functions, and connected them with the aforementioned families.



Here is an example of this terminology applied on the function level (a function’s own higher order compositions)

000: self-loop

001 - one-path to self-loop (000)

002 - self-loop

010 - self-loop

011 - self-loop

012 - self-loop

020 - 1-path to self-loop (000)

021 - two-loop (021 - 012 - ...)

022 - self-loop

100 - two-loop (100 - 011 - ...)

101 - two-loop (101 - 010 - ...)

102 - two-loop (102 - 012 - ...)

110 - one-path to self-loop (111)

111 - self-loop

112 - self-loop

120 - three-loop (120 - 201 - 012 - ...)

121 - two-loop (121 - 212 - ...)

122 - one-path to self-loop (222)

200 - two-loop (200 - 022 - ...)

201 - three-loop (201 - 120 - 012 - ...)

202 - one-path to self-loop (222)

210 - two-loop (210 - 012 - ...)

211 - one-path to self-loop (111)

212 - self-loop

220 - two-loop (220 - 002 - ...)

221 - two-loop (221 - 112 - ...)

222 - self-loop

Claim: all monoids of size 3 must have functions that do not contain any higher than 3-loops, or 2-path

A 4-loop would need to be composed with itself 4 times before reaching the identity, and all three intermittent functions are unique, but we only have room for 2 non identity elements

A 3-path would need to be composed with itself 3 times before reaching a non-identity function with only self loops or 1-paths, and non-identity because the existence of paths prevents the identity function from being a higher-order composition with itself

Here are all possible combinations of micro structures in each function graph.

1. All self-loops (Identity function) (any following type can also contain self-loops)

2. All 2 loops

3. All 3 loops

4. 2 loops and 3 loops

5. 1 paths

6. 2 paths

7. 1 paths and 2 paths

8. 2 loops and 1 paths

9. 3 loops and 1 paths

10. 2 loops and 2 paths

11. 3 loops and 2 paths

12. No 2 loop

13. No 3 loop

14. No 1 path

15. No 2 path

16. All

There are a few categories that can be eliminated as invalid right off the bat. Type 4 introduces 6 distinct higher order compositions, so there is no chance of that fitting in size 3. That then also eliminates 14, 15, and 16.

3 loops also are not compatible with any path, as paths make the higher order composition of the function not the identity, therefore the 3 higher order compositions would turn the nodes of a loop into a self loop, but there would still be 1 paths elsewhere. This will not fit in size 3. Therefore, 9, 11, and 12 are also out.

2 loops also are not compatible with 2 paths, as the 2 paths collapse into 1 paths in 2 different configurations: the 2 loop still being a loop, or being self loops. Won't fit in size 3, so 10. and 13. are out.

The remaining valid types are:

1. All self-loops (Identity function) (any following type can also contain self-loops)

2. All 2 loops

3. All 3 loops

5. 1 paths

6. 2 paths

7. 1 paths and 2 paths

8. 2 loops and 1 paths

Now, we will consider what combination of these are valid, which may require subdividing these types further.

(1) Of course is required exactly once, so will be omitted for brevity at the moment

(22) is seemingly impossible, as the composition of 2 loops either create larger loops, other distinct 2 loops, or distinct combinations of 2 loops.

(33) is the only possible way for 3 to exist, as any 3 must come with its inverse. This is EV-1

(85) works, as 5 becomes itself, and 8 can become 5 and back to 8. In fact, this is the only combination 8 is valid in. All 1-paths must either be the same if pure paths, or point to the other node in the 2-loop if connected to the loop. This is EV-4

(55) will only work if the 1-paths are not disjoint between the two, otherwise the composition will contain even more 1-paths. There are 4 situations I can think of:

1. All 1-paths are flipped (terminus becomes head) This is EV-2

2. All 1-paths share a head. This is EV-3

3. All 1-paths going to one terminus instead goes to another terminus, where the two termini are flipped 1-paths in the other function. This is EV-2. In fact, situation 1 is a subset of this, where fan-in of all nodes is no more than 2. Indeed, all 1-paths except the 2-termini share a head here as well. So the distinction between EV-2 and EV-3 is having termini that is a head in the other or not.

4. All heads and termini of 1-paths go to termini in the other function that was not a head in this one, possibly adding heads. This is EV-5.3

(56) and (57) are similar, in that 6/7 becomes 5. Indeed, this is the only requirement (and only possible combination for 6 and 7). EV-5.2

(52) is the only remaining combination to test, which ultimately is EV-5.1. The loop must not interfere with the path (no termini in a loop), and nodes in a loop must be heads of a 1-path.

After this, the same example monoids in each function was rewritten in formats that show this loop and path structure more clearly. Cycle notation is used in parenthesis. Self loops are omitted for brevity.

1: I, (012), (021)

2: I, (2->1), (1->2); I, ((0,2)->1), ((0,1)->2)

3: I, (2->0), (2->1)

4/5: I, (12)(0->1), (0->2)

6/7:

.1: I, ((1,2)->0), (12)

.2: I, ((1,2)->0), (2->1->0)

.3: I, ((1,2)->0), (1->0)

Simplified:

1: I, (012), (021)

2: I, [21], [12]; I, [(0,2)1], [(0,1)2]

3: I, [20], [21]

4/5: I, (12)[01], [02]

6/7:

.1: I, [(1,2)0], (12)

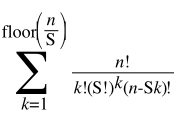
.2: I, [(1,2)0], [210]

.3: I, [(1,2)0], [10]

Explicit Formulas:

Family 1:

As noted earlier, this family consists of only 3-loops and its associated inverted counterpart.



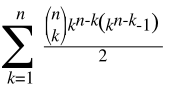
S is the order of the loop, which in this case is S = 3. This appeared earlier in the background describing when m = 2, in which case S = 2.

Family 2:

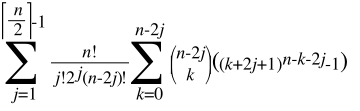
so the formula for family 2, given #elements n, is the sum over all distinct partitions (x1, x2, ..., xk) of the set {1, 2, ..., n}, of (|x1| |x2|...|xk|)(|x1| |x2|...|xk| - 1)/2.

It would be cool if we can express the sum over distinct partitions as a simpler sum.

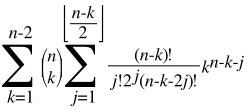
Family 3:



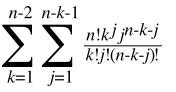
Family 4:



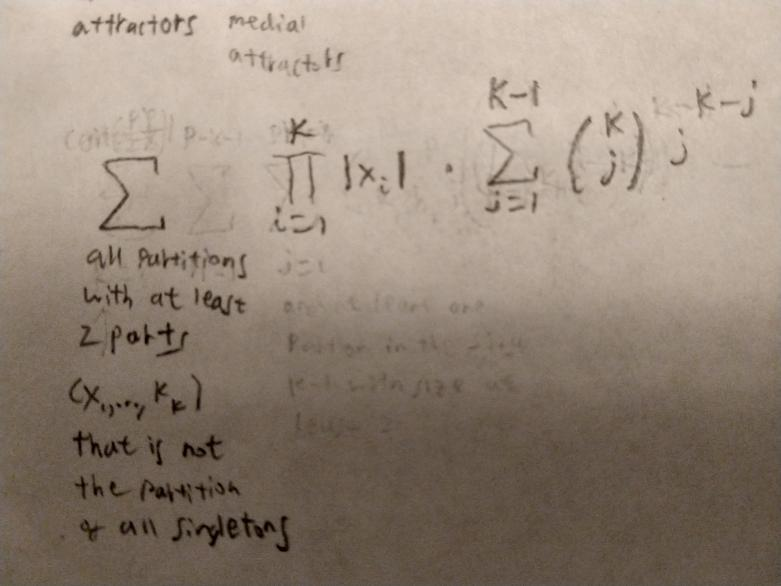
Family 5.a:



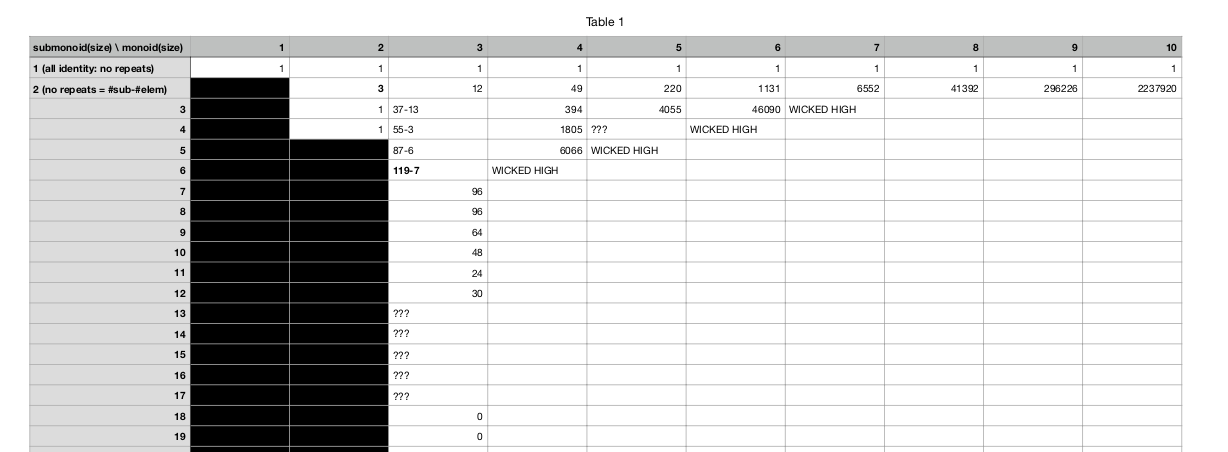
Family 5.b:



Family 5.c:



Now let’s demonstrate that these formulas match up:



**Goal for n = 3; 37** total.

Family 1 : 1

Family 2 : 6

Family 3 : 3

Family 4 : 6

Family 5.a : 3

Family 5.b : 6

Family 5.c : 12

Family 2 Calculations: **6** total

1,2:

(3 choose 2) = 3 instances

3 \* (1 \* 2 choose 2) = 3

3:

(3 choose 3) = 1 instance

1 \* (3 choose 2) = 3

Family 5.c Calculations: **12** total

1,2: 3 instances

3 \* (1 \* 2 \* 2) = 12

**Goal for n = 4; 394** total.

Family 1 : 4

Family 2 : 42

Family 3 : 48

Family 4 : 60

Family 5.a : 24

Family 5.b : 60

Family 5.c : 156

Family 2 Calculations: **42** total

1,1,2:

(4 choose 2) \* (2 choose 1) / 2! = 6 instances

6 \* (1 \* 1 \* 2 choose 2) = 6

1,3:

(4 choose 3) = 4 instances

4 \* (1 \* 3 choose 2) = 12

2,2:

(4 choose 2) / 2! = 3 instances

3 \* (2 \* 2 choose 2) = 18

4:

(4 choose 4) = 1 instance

1 \* (4 choose 2) = 6

Family 5.c Calculations: **156** total

1,1,2: 6 instances

6 \* 1 \* 1 \* 2 \* 9 = 108

1,3: 4 instances

4 \* 1 \* 3 \* 2 = 24

2,2: 3 instances

3 \* 2 \* 2 \* 2 = 24

**Goal for n = 5; 4055** total.

(total 4045; 10 under)

Family 1 : 10

Family 2 : 310

Family 3 : 670

Family 4 : 530

Family 5.a : 195

Family 5.b : 560

Family 5.c : 1770

Family 2 Calculations: **310** total

1,1,1,2:

(5 choose 2) \* (3 choose 1) \* (2 choose 1) / 3! = 10 instances

10 \* (2 choose 2) = 10

1,1,3:

(5 choose 3) \* (2 choose 1) / 2! = 10 instances

10 \* (3 choose 2) = 30

1,2,2:

(5 choose 2) \* (3 choose 2) / 2! = 15 instances

15 \* (4 choose 2) = 90

1,4:

(5 choose 4) = 5 instances

5 \* (4 choose 2) = 20

2,3:

(5 choose 3) = 10 instances

10 \* (6 choose 2) = 150

5:

(5 choose 5) = 1 instance

1 \* (5 choose 2) = 10

Family 5.c Calculations: **1770** total

1,1,1,2: 10 instances

10 \* 2 \* 40 = 800

1,1,3: 10 instances

10 \* 3 \* 9 = 270

1,2,2: 15 instances

15 \* 4 \* 9 = 540

1,4: 5 instances

5 \* 4 \* 2 = 40

2,3: 10 instances

10 \* 6 \* 2 = 120

**Goal for n = 6; 46090** total.

Family 1 :

Family 2 :

Family 3 :

Family 4 :

Family 5.a :

Family 5.b :

Family 5.c :

Family 2 Calculations:

1,1,1,1,2:

(6 choose 2) = 15 instances

1,1,1,3:

(3 choose 2) =

1,1,2,2:

1,1,4:

1,2,3:

1,5:

2,4:

3,3:

6: