An Analysis of The Collatz Map Structure
And A New Algorithm for Evaluation [*]

Gabriel M. Perry

August 7, 2024

Abstract

We present an algorithm to evaluate the Collatz path, which uses a variable modulus to preserve the path using (sometimes) smaller numbers than the naïve approach. We explore the algorithm's perspective on the 2-adic numbers and present some interesting diagrams, notation, and a measure organizing the structure of the Collatz map.

1 Introduction

The Collatz map is defined as a function over the integers such that $C(x) = 3x + 1$ if x is odd and $C(x) = x/2$ if x is even. Throughout this paper, I refer mostly to the reduced Collatz function, which is summarized and iterated as follows:

Algorithm 1 *Reduced Collatz Function*

```
procedure $C_p(x)$
    $d \leftarrow 0$
    $r \leftarrow x$
    while $d \leq p$ do
        $d \leftarrow d + 1$
        if $r$ is odd then
            $r \leftarrow 3r + 1$
        end if
        $r \leftarrow r/2$
    end while
end procedure
```

[*]The author is a graduate student in Brigham Young University's Computer Science department, under the Information and Decision Algorithms (IDeA) Labs. This work is not associated with any project therein, which primarily focuses on control and optimization. This work was mainly performed recreationally during the author's undergraduate studies. The author can be reached at gmp99@byu.edu.

where we wish to iterate the Collatz map for p steps, and the current iteration is d, and r is the result (the labeling and order of incriminating d and r are described in the explanation of Alg. 2).

The Collatz map (one iteration of Alg. 1's loop) is known to be non-injective and non-surjective. Lothar Collatz's famous 1937 conjecture that for all inputs x, there exists some p for which $C_p(x) = 1$, remains unproven at the present time. The procedure is often introduced as a type of game, and this is the spirit in which I approach this paper; what I produce here is an illustrative view meant to cause the reader to ask questions general to number theory, while at the same time providing a new perspective on this fun function.

2 A Modular Algorithm

We motivate the algorithm by exploring the expression of a number during execution in a particular manner. Expressing the numbers in this way will produce an inductive proof of the algorithm.

Given an input x, after d steps of the reduced Collatz map (Alg. 1), we can express the current state as:

$$\begin{aligned} C_d(x) &= 3^k x_d + r && (1)\\ &= 2 \cdot 3^k x_{d+1} + r + 3^k \cdot (x \mod 2) && (2)\\ &= 2 \cdot 3^k x_{d+1} + r' && (3)\end{aligned}$$

where p is the number of $n/2$ steps already performed, k is the number of $3n+1$ steps already performed, and x_d is the input x bit-shifted to the left d bits (in other words, the number of $n/2$ steps already performed). When we begin the first iteration, $k = r = d = 0$, correctly giving us $C_0(x) = x$. This is the base case. For the inductive step, we apply $3n+1$ if the current value is odd. As the first term of (3) is even, we look to r' for parity. If r' is even, we do not apply this step and label $r'' = r'$. If r' is odd, we apply $3n+1$ to the entire expression:

$$\begin{aligned} C_{d+1}(x) &= 3(2 \cdot 3^k x_{d+1} + r') + 1 && (4)\\ &= 2 \cdot 3^{k+1} x_{d+1} + 3r' + 1 && (5)\\ &= 2 \cdot 3^{k'} x_{d+1} + r'' && (6)\end{aligned}$$

where $r'' = 3r' + 1$ and $k' = k + 1$ if r' was odd, else they are not changed from (3). Finally, as we know this expression is even, we complete the $d+1$ iteration by dividing by 2:

$$C_{d+1}(x) = 3^{k'} x_{d+1} + \frac{r''}{2} \qquad (7)$$

This gives us an expression of the same form as (1), so we can apply this procedure inductively. This is summarized by the following algorithm:

Algorithm 2 *Modular Collatz Function*

```
procedure ModCollatz(x)
    $d, k, r \leftarrow 0, 0, 0$
    while $d \leq \lfloor \log_2 x \rfloor$ do
        if $x_d$ is odd then                                    ▷ see (2)
            $r \leftarrow r + 3^k$
        end if
        $d \leftarrow d + 1$
        if $r$ is odd then                                      ▷ see (6)
            $k \leftarrow k + 1$
            $r \leftarrow 3r + 1$
        end if
        $r \leftarrow r/2$                                      ▷ see (7)
    end while
end procedure
```

One can equivalently continue the *while* loop past the number of digits in x, using the trailing zeros as input, but continuing in this way is equivalent to Collatz (as the first *if* statement will never again execute). Also, since we only care about the parity of x_d in each iteration, we do not need the entire shifted x, but only the bit in the d position of the original x. We also have flexibility to increment d later in the main loop.

Table 1: Modular Evaluation of 9663 (10010110111111_2)

d'th bit of x	**Current State (d, k, r)**		
	if bit is 1 add 3^k to r	if r is odd inc. k, $3r+1$	always $r/2$, inc. d
1	(0, 0, 1)	(0, 1, 4)	(1, 1, 2)
1	(1, 1, 5)	(1, 2, 16)	(2, 2, 8)
1	(2, 2, 17)	(2, 3, 52)	(3, 3, 26)
1	(3, 3, 53)	(3, 4, 160)	(4, 4, 80)
1	(4, 4, 161)	(4, 5, 484)	(5, 5, 242)
1	(5, 5, 485)	(5, 6, 1456)	(6, 6, 728)
0	no change	no change	(7, 6, 364)
1	(7, 6, 1093)	(7, 7, 3280)	(8, 7, 1640)
1	(8, 7, 3827)	(8, 8, 11482)	(9, 8, 5741)
0	no change	(9, 9, 17224)	(10, 9, 8612)
1	(10, 9, 28295)	(10, 10, 84886)	(11, 10, 42443)
0	no change	(11, 11, 127330)	(12, 11, 63665)
0	no change	(12, 12, 190996)	(13, 12, 95498)
1	(13, 12, 626939)	(13, 13, 1880818)	(14, 13, 940409)

The initial configuration is (0, 0, 0).

Table 2: Modular Evaluation of 27335764 ($1101000001000111000101010 0_2$)

d'th bit of x	**Current State (d, k, r)**		
	if bit is 1 add 3^k to r	if r is odd inc. k, $3r+1$	always $r/2$, inc. d
0	no change	no change	(1, 0, 0)
0	no change	no change	(2, 0, 0)
1	(2, 0, 1)	(2, 1, 4)	(3, 1, 2)
0	no change	no change	(4, 1, 1)
1	(4, 1, 4)	no change	(5, 1, 2)
0	no change	no change	(6, 1, 1)
1	(6, 1, 4)	no change	(7, 1, 2)
0	no change	no change	(8, 1, 1)
0	no change	(8, 2, 4)	(9, 2, 2)
0	no change	no change	(10, 2, 1)
1	(10, 2, 10)	no change	(11, 2, 5)
1	(11, 2, 14)	no change	(12, 2, 7)
1	(12, 2, 16)	no change	(13, 2, 8)
0	no change	no change	(14, 2, 4)
0	no change	no change	(15, 2, 2)
0	no change	no change	(16, 2, 1)
1	(16, 2, 10)	no change	(17, 2, 5)
0	no change	(17, 3, 16)	(18, 3, 8)
0	no change	no change	(19, 3, 4)
0	no change	no change	(20, 3, 2)
0	no change	no change	(21, 3, 1)
1	(21, 3, 28)	no change	(22, 3, 14)
0	no change	no change	(23, 3, 7)
1	(23, 3, 34)	no change	(24, 3, 17)
1	(24, 3, 44)	no change	(25, 3, 22)

The initial configuration is again (0, 0, 0).

Another way to visualize this method is to "save" the original x in a variable separate from the current remainder or state of the Collatz path. At each iteration of the path, we only care about the current parity (least significant bits), so we move the pertinent bit from x to the working variable r, scaled according to how many times we have multiplied it by 3. We then apply Collatz as normal on r, keeping track of whether we scale x even more. Adding 1 then only affects the lowest bits or r. As we are working in base 2, the division by two at each iteration does not change the bits of x but shifts us to the next bit for the next iteration. Due to the linearity of the multiplication and division operations, we can separate x from r and only recombine them in this way when needed (and only for the bits needed).

For the purposes of justifying the use of modulus 3^k and $2 \cdot 3^k$ throughout our discussion, we guarantee that $r < 2 \cdot 3^k$ at every major point of the procedure. We know that this is true when the procedure starts, as $r = 0 < 2 = 2 \cdot 3^0$. In fact, at the beginning of each iteration, the bound is even stronger: $r < 3^k$ (which is also true on the first iteration, as $0 < 1 = 3^0$). The movement of bits in (2) adds 3^k to the "modulus" part (the coefficient on x_d) and at most 3^k to the "remainder" part (r and r'), so the relationship still holds (the remainder r' is still less than the modulus $2 \cdot 3^k$). As the start bound 3^k is sufficient for the global bound $2 \cdot 3^k$, if $r = r'$ then $r' < 2 \cdot 3^k$ still holds for the next step. Following the logic that produces (6), r' is at least one less than the modulus $2 \cdot 3^k$ (since these are all integers), so scaling both by a factor of three guarantees that the remainder is at least three less than the new modulus (which, as we incremented k, is now also scaled by 3); adding one places this bound at two. If r' were even, we did nothing and the relation still holds from the last step. Thus, $r'' < 2 \cdot 3^{k'}$. Finally, dividing both terms by 2 preserves their relation, so $r''/2 < 3^{k'}$, which is the more strict bound ready for the next iteration. As the strict bound is valid for the first iteration and is preserved during execution, we can guarantee that $r < 2 \cdot 3^k$ at the end of each major step and $r < 3^k$ at the end of each iteration.

Refer to Tables 1 and 2 for examples of a complex and a simple input for the algorithm, respectively. We will discuss what makes some inputs' execution simpler than others in Sec. 4 (in short, I picked 9663 because it moves "up" layers quickly in the diagrams therein explained, and 27335764 follows paths "along" the cycles, rarely moving upwards; refer to Sec. 4 for more details). Remember that at any point of Alg. 2, we can recover the standard Collatz value via (1) through (7).

2.1 Using More Remainders

We can extend the algorithm to include more levels of components with moduli of various powers of three (Alg. 2 has two, the input multiplied by 3^k and the current remainder in the 'ones' place). This is done using arrays to store the various values of x, r s, and multiple k that indicate the differences between the various moduli. Alg. 3 explains this extension further:

Algorithm 3 *Extended Modular Collatz Function*

```
Require: $r = [x, r_1, r_2, ...]$, the registers for each base place (base case $[x]$)
Require: $k = [k_0 - k_1, k_1 - k_2, ..., 0]$, the differences in magnitude between each
    base place (base case $[0]$)
Require: $T$, the threshold at which we create a new register
procedure MultiModCollatz($r, k, T$)
    if the arrays only contain one element then
        append a 0 to $r$ and $k$                    ▷ Ensure we have a work register
    end if
    for $i$ from 0 to $len(r) - 2$ inclusive do
        if $r[i]$ is odd then                         ▷ move value down a register
            $r[i] \leftarrow r[i] - 1$
            $r[i+1] \leftarrow r[i+1] + 3^{k[i]}$
        end if
        $r[i] \leftarrow r[i] \div 2$                                      ▷ shift bits
        if $r[i]$ is zero then                       ▷ drop an unneeded register
            drop $r[i]$
            if $i$ is not the first register then
                $k[i-1] \leftarrow k[i-1] + k[i]$        ▷ update the register distance
            end if
            drop $k[i]$
        else
            $i \leftarrow i + 1$        ▷ the for loop doesn't automatically increment i
        end if
    end for
    if $r == [1, 0]$ and $k == [0, 0]$ then            ▷ the Collatz break condition
        return $[1], [0]$
    end if
    if $r[-1] > T$ then          ▷ the threshold instructs us to add a register
        append a 0 to $r$ and $k$
        if $r[-2]$ is odd then                      ▷ apply the for loop procedure
            $r[-2] \leftarrow r[-2] - 1$
            $r[-1] \leftarrow r[-1] + 3^{k[-2]}$
        end if
        $r[-2] \leftarrow r[-2] \div 2$
    end if
    if $r[-1]$ is odd then       ▷ apply normal Collatz to the working register
        $r[-1] \leftarrow 3 \cdot r[-1] + 1$
        if $k[-2]$ exists then
            $k[-2] \leftarrow k[-2] + 1$      ▷ remember factor of 3 in higher registers
        end if
    end if
    $r[-1] \leftarrow r[-1] \div 2$                    ▷ bit-shift the working register
    return $r, k$
end procedure
```

Alg. 2 is the special case of Alg. 3 where $T = \infty$ (i.e. we only use two registers, the input x and the remainder is $r[1]$). We can imagine this algorithm as running on an array representing a number written in some numeric base (the registers are multiplied by powers of 3, though there is no rule that each register has to be generated at any particular or consistent T). We can recover the current value of the process similar to (1) through (7):

$$= (((r[0] \cdot 3^{k[0]} + r[1]) \cdot 3^{k[1]} + r[2]) \cdot 3^{k[2]} + r[3]) \cdot 3^{k[3]} + r[4]... \tag{8}$$

$$= r[0] \cdot 3^{k[0]+...+k[-1]} + r[1] \cdot 3^{k[1]+...+k[-1]} + r[2] \cdot 3^{k[2]+\cdots} + ... + r[-1]) \tag{9}$$

Recalling that $k[-1]$ will always be zero.

3 Dissecting Binary Patterns in the Modular Algorithm

3.1 The Collatz-base Expression

Consider the following expressions:

$$= 218686112 \tag{10}$$

$$= -\frac{2^5}{3^1} - \frac{2^{11}}{3^2} - \frac{2^{20}}{3^3} - \frac{2^{29}}{3^4} - \frac{2^{30}}{3^5} - \frac{2^{32}}{3^6} - \frac{2^{35}}{3^7} - \frac{2^{39}}{3^8} - \frac{2^{41}}{3^9} - \frac{2^{43}}{3^{10}} - ... \tag{11}$$

$$= ... - \frac{2^{29}}{3^4} - \frac{2^{30}}{3^5} - \frac{2^{32}}{3^6} - \frac{2^{35}}{3^7} - \frac{2^{39}}{3^8}(\frac{2^0}{3^0} + \frac{2^2}{3^1} + \frac{2^4}{3^2} + ...) \tag{12}$$

$$= ... - \frac{2^{29}}{3^4} - \frac{2^{30}}{3^5} - \frac{2^{32}}{3^6} - \frac{2^{35}}{3^7} - \frac{2^{39}}{3^8}(\frac{4^0}{3^0} + \frac{4^1}{3^1} + \frac{4^2}{3^2} + ...) \tag{13}$$

$$= ... - \frac{2^{29}}{3^4} - \frac{2^{30}}{3^5} - \frac{2^{32}}{3^6} - \frac{2^{35}}{3^7} - \frac{2^{39}}{3^8}(\frac{1}{1-\frac{4}{3}}) \tag{14}$$

$$= ... - \frac{2^{29}}{3^4} - \frac{2^{30}}{3^5} - \frac{2^{32}}{3^6} - \frac{2^{35}}{3^7} - \frac{2^{39}}{3^8}(-3) \tag{15}$$

$$= -\frac{2^5}{3^1} - \frac{2^{11}}{3^2} - \frac{2^{20}}{3^3} - \frac{2^{29}}{3^4} - \frac{2^{30}}{3^5} - \frac{2^{32}}{3^6} - \frac{2^{35}}{3^7} + \frac{2^{39}}{3^7} \tag{16}$$

We can easily derive such an expression for any integer by building such a fraction sequence, as we iterate the Collatz map. For example, as the first step for 218686112 is dividing by 2^5, the numerators must all contain 2^5:

$$= -\frac{2^5}{3^1} - \frac{2^{11}}{3^2} - \frac{2^{20}}{3^3} - \frac{2^{29}}{3^4} - \frac{2^{30}}{3^5} - \frac{2^{32}}{3^6} - \frac{2^{35}}{3^7} + \frac{2^{39}}{3^7} \tag{17}$$

$$\div 2^5 \rightarrow -\frac{1}{3^1} - \frac{2^6}{3^2} - \frac{2^{15}}{3^3} - \frac{2^{24}}{3^4} - \frac{2^{25}}{3^5} - \frac{2^{27}}{3^6} - \frac{2^{30}}{3^7} + \frac{2^{34}}{3^7} \tag{18}$$

The $3x + 1$ steps indicate when a new term is created (here, working backward or executing Collatz, we see the least significant term removed):

$$= -\frac{1}{3^1} - \frac{2^6}{3^2} - \frac{2^{15}}{3^3} - \frac{2^{24}}{3^4} - \frac{2^{25}}{3^5} - \frac{2^{27}}{3^6} - \frac{2^{30}}{3^7} + \frac{2^{34}}{3^7} \tag{19}$$

$$\times 3 \to -1 - \frac{2^6}{3^1} - \frac{2^{15}}{3^2} - \frac{2^{24}}{3^3} - \frac{2^{25}}{3^4} - \frac{2^{27}}{3^5} - \frac{2^{30}}{3^6} + \frac{2^{34}}{3^6} \tag{20}$$

$$+1 \to -\frac{2^6}{3^1} - \frac{2^{15}}{3^2} - \frac{2^{24}}{3^3} - \frac{2^{25}}{3^4} - \frac{2^{27}}{3^5} - \frac{2^{30}}{3^6} + \frac{2^{34}}{3^6} \tag{21}$$

For simplicity, I use a new notation to omit many of the symbols; as each successive fraction term is strictly increasing in the power degrees (by exactly one each term in the denominators, and by a variable amount in the numerator), I list only of the amount the numerator's power increases on each term:

$$218686112 = \{5, 6, 9, 9, 1, 2, 3, 4, 2, 2, 2, 2, ...\} \tag{22}$$

$$= \{5, 6, 9, 9, 1, 2, 3, 4; -3\} \tag{23}$$

where the sequence denotes the increase in the power of 2 in successive terms, and the second notation separates the factor by which the last term is multiplied (see (15)), which factor comes from the trailing repeated sequence. I will default to the first notation. In this notation, the evaluation of the Collatz map is simple; dividing the number by 2 decrements the least significant position, and the $3x + 1$ steps have the effect of shifting the sequence left one place (the number is odd precisely when the least significant figure is zero, so the zero is shifted leftward out of the expression). The Collatz conjecture is equivalent to the statement that all integers have such an expression ending in a trail of twos (equivalently, all integers have a -3 past the semicolon). The justification for this statement is:

$$x = \{2, 2, 2, 2, 2, 2, ...\} \tag{24}$$

$$\frac{x}{4} = \{0, 2, 2, 2, 2, 2, ...\} \tag{25}$$

$$3 \cdot \left(\frac{x}{4}\right) + 1 = \{2, 2, 2, 2, 2, ...\} \tag{26}$$

$$\frac{3}{4}x + 1 = x \tag{27}$$

$$x = 4 \tag{28}$$

which suggests that every sequence of trailing twos corresponds with the 4-2-1-4 cycle. This derivation is easily extended to any repeating sequence, and each repeating sequence has a unique value by the fact that the resulting equation is always linear (see 3.2 for elaboration).

This form allows us to view the entire Collatz path of a number; the auxiliary variables d (the number of $x/2$ steps performed) and k (the number of $3x + 1$ steps performed) in Alg. 2 are easily visible in the sequence. At any point in the sequence, d will be precisely the running total of the values in the sequence up to the point of execution (including only partially reducing the current least significant place), and k will be precisely the number of positions that we will shift by that point.

Some properties of this form include the fact that a zero can only appear in the least significant term (these are exactly all odd numbers). A zero cannot appear anywhere else in the integer case, as immediately after a $3n+1$ step, we divide by 2 at least once (so every entry after the first must be at least one) If a zero appears elsewhere, then the current x must have been odd, and $3x+1$ must have also been odd. This is the case only if the denominator of x is even, in which case the sequence will continue in a trail of zeros and the values will diverge.

The Collatz path only increases when a term in this expression is 0 or 1, as a reduction occurs when dividing by 4 or greater powers of 2 after multiplication by 3. This means that the Collatz conjecture is equivalent to the statement that, for all integers, there are only finitely many ones in their Collatz base expression (which is redundant, as we already showed that there should be only finitely many non-2 terms under the conjecture). This statement does not hold under the rationals, as some cycles can contain ones, but in general any sequence (including those with trailing zeros as mentioned above) represents a valid rational number. Even a sequence that terminates (or more precisely, has infinity as its last term, as we never perform $3x+1$ again but instead divide by 2 ad infinitum, as when $x=0$) is valid, as the sequence containing one infinite term is equal to zero, and any sequence ending in such has a power of three in its denominator.

3.2 Extension to 2-adic Numbers

This method can be applied for any repeating sequence at the tail of a Collatz-base expression. These different patterns of tails correspond to different rational numbers and fall under the same family as Jeffrey Lagarias's equation of parity sequence [1]:

$$x = \frac{3^{k-1}2^{k_0} + ... + 3^0 2^{k_{k-1}}}{2^d - 3^k} \tag{29}$$

where k_i is the position of each $3n+1$ step in the 0-indexed parity sequence of the traditional Collatz path. For example, the parity sequence (1011001) (read from left to right) has odd steps at indexes $k_0 = 0$, $k_1 = 2$, $k_2 = 3$, and $k_3 = 6$. In Collatz-base form, this number's expression is:

$$x = \{0,\ 2, 1, 3, 1,\ 2, 1, 3, 1, ...\} \tag{30}$$

We see that $k = 4$ is the length of the repeated portion of the expression (the number of times the second "if" statement executes in the modular algorithm, or the number of odd steps in the "shortcut" Collatz algorithm) and $d = 7$ is the length of the iterated cycle (the number of divisions by 2 of Collatz, or sum of terms in a single cycle in my notation). We can derive the k_i values directly as the running totals at each term of the sequence up to the point where the cycle repeats. For example, $k_0 = 0$ as the first term is zero, $k_1 = 2$ as the running total of the first two terms is 2, $k_2 = 3$ adds the following one to the running sum, and $k_3 = 6$ adds the three, etc. We can generate the other members of this 7-cycle by shifting the expression and reducing the least significant term as desired (i.e., by executing Alg. 2). We can use a method similar to (24) through (28) to derive the value of x, as well as using (29).

Note that the parity sequence mentioned above, as well as the Collatz-base expression, can be used to recover the 2-adic expression of the number generating the parity sequence. For example, say we wish to recover the following number given its Collatz-base expression:

$$\frac{1}{5} = \{0, 3, 3, 3, 3, ...\} \tag{31}$$

$$= ...\overline{0110}1 \tag{32}$$

We can use this form to generate the 2-adic representation via the following algorithm:

Algorithm 4 Collatz-base to 2-adic Expression

```
($y_d$ is the d'th bit of the result and the default of all bits is 0)
procedure RecoverBinary($x$={...})
    $d, k, r, y \leftarrow 0, 0, 0, 0$
    while $x$ contains terms do
        while $x$ has 0 in the least significant place do
            if $r$ is even then
                $y_d \leftarrow 1$
                $r \leftarrow r + 3^k$
            end if
            $r \leftarrow 3r + 1$
            $k \leftarrow k + 1$
            shift out the least significant place of $x$ (the zero)
        end while
        if $r$ is odd then
            $y_d \leftarrow 1$
            $r \leftarrow r + 3^k$
        end if
        $d \leftarrow d + 1$
        $r \leftarrow r/2$
        decrement the least significant term of $x$
    end while
end procedure
```

This algorithm is similar to Alg. 2, and the proof of this algorithm is straightforward. It essentially is just setting the bits of the output y so that Alg. 2 runs the same sequence of its first, second, and third steps in y as in the given x (the three columns of Tables 1 and 2). Even steps occur at each iteration, and the bit of y is set to ensure that r is even at the correct point (by adding a power of three if needed; a similar guard is used to ensure that r is odd when we need to perform a step of $3n + 1$. The odd steps are entirely determined by the parity of r and the current bit of the binary expression, and they determine the point the Collatz-base expression begins a new term. This algorithm is an "inverse" procedure in that it instead uses the shift points in the Collatz-base expression as well as the parity of r at the start of each iteration to determine the bit sequence of the binary representation of an input that produces the correct odd and even step sequence. This is not the same as performing the reverse Collatz mapping; Alg. 4 performs the forward Collatz algorithm on the Collatz-base expression while outputting the binary sequence required to produce the same execution pattern.

Alg. 4 can be applied to any Collatz-base expression to recover a 2-adic integer, and we can feed any 2-adic integer into Alg. 2 to produce a Collatz path. These can both be done regardless of whether the expression terminates, or even repeats. For example, using the 2-adic expression of ln(3) found in [2], we find:

$$\ln(3) = ...10100011110100 \tag{33}$$

$$= \{2,\ 3,\ 1,\ 1,\ 4,\ 1,\ 1,\ 1,\ ...\} \tag{34}$$

which does not repeat.

Relating this generalization to previous extensions of the Collatz map, 2-adic *integers* will never be fractions with a factor of 2 in the denominator (since this would shift the integer part below the one's place, producing a mantissa). Also, all rational numbers with odd numerators have a one-bit of 1, and all even-numerator numbers have a 0 one-bit, so these algorithms are consistent with existing extensions of the Collatz map over the rational numbers. The same applies to negative numbers. As seen in (33), irrational numbers can also be assigned a parity (e.g. ln(3) is "even" in some sense).

4 Layer-k Diagrams

r	x bit	$r \to$
odd	odd	$\frac{r+3^k}{2}$
even	odd	$\frac{3r+3^{k+1}+1}{2}$
odd	even	$\frac{3r+1}{2}$
even	even	$\frac{r}{2}$

Consider what happens in Alg. 2 if we never increment k. The modulus 3^k remains constant, so there are only finitely many configurations of r, and we must always have an even r when we reach the second if condition (so we do not increase k). To narrow down the possible r further, we note that continuing along any Collatz path, we will never encounter a multiple of three; this means that for any k the only possible values of r when we reach this if condition are the even numbers not a multiple of 3. After the if condition is skipped, we divide the even r by 2, then at the beginning of the next iteration add 3^k if needed to keep r even to avoid incriminating k. This process is entirely deterministic for each k, and in fact, we will find that r will traverse in reverse the subring of $\mathbb{Z}_{3^k}^{\times}$ produced by continually doubling any number r not already divisible by 3. Because of this, we can model each k-layer of the algorithm as a graph, a cycle of such numbers, and append other paths corresponding to how we enter the ring and how we exit (both upon incriminating k). In Fig. 1 and 2, I arrange these cycles to depict the evolution of r during the algorithm's execution. Fig. 1 depicts the k-layers for $k = 0, 1, 2$ and Fig. 2 depicts $k = 3$.

At layer k, there are $2 \cdot 3^{k-1}$ values. The multiples of 3 mod 3^k (including 0) are not reachable, but have a virtual up-transition into the next layer (one can easily find where they belong on the diagram by providing these numbers as input to the algorithm).

4.1 The Collatz Base Representation Revisited

Note that the Collatz base entries are precisely the number of transitions we take along each layer before moving up, plus one more when we move upwards (drop the leading zero).

4.2 An Algorithmic Measure

Define a quantity $\bar{x} = -r/3^k$. This quantity $-1 < \bar{x} \leq 0$ encodes the position of the algorithm on the layer-k diagram, with the 3^k term indicating which layer execution is on, and the r term indicating which node in that layer the algorithm is currently at. It is not difficult to show that, during the execution of Alg. 2, if the next bit seen is a 1, the value of $\bar{x}$ will approximately jump to the average between -1 and the current $\bar{x}$, whereas if the next bit is 0, the value will move near the average between 0 and $\bar{x}$. This is summarized in Table 4.2.

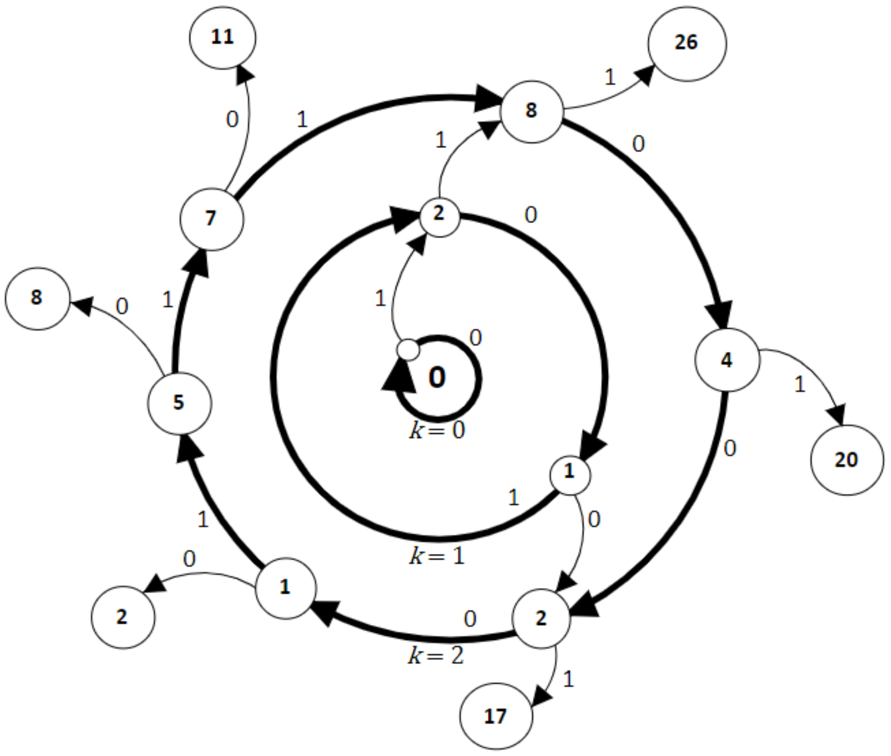

Figure 1: One depiction of the structure of the r values (nodes) during execution. The next bit of the input x_d is the label on each edge. Layer $k = 0$ is the case that keeps dividing by two until an odd bit of x is reached, so the innermost loop is very simple. Note that r cannot reach multiples of 3. Note also that layer k contains all integers $< 3^k$ which are not multiples of three. Also note that moving around the ring is simply multiplying by $2^{-1} \mod 3^k$. Equivalently, traversing the ring backward (counterclockwise) is equivalent to doubling $r \mod 3^k$.

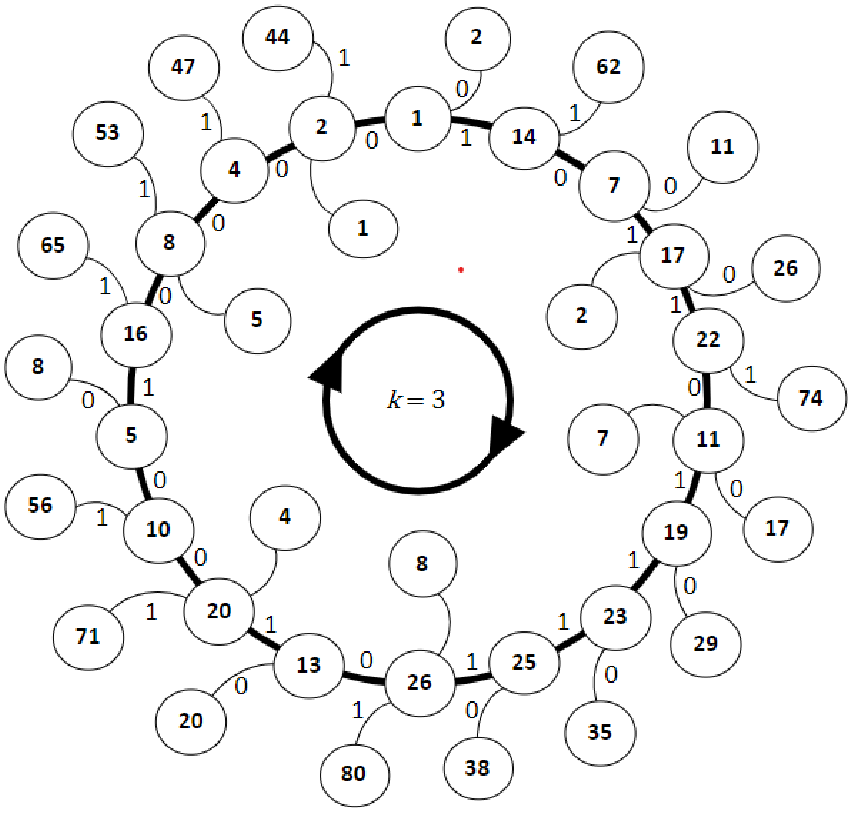

Figure 2: Another depiction of the execution structure, this time showing layer $k = 3$, also including transitions from the previous and to the next layers. Note that for even r, the branches are one each even and odd, and dividing out the factor of 2 always remains on the current layer k. Similarly, odd rs produce even children if $r \equiv 1 \mod 4$ and odd children if $r \equiv 3 \mod 4$, but both children are always greater than the parent.

All integers eventually end in a trail of zeros (on the significant end), so $\bar{x}$ for all integers will converge toward 0.

r	x bit	$\bar{x} \to$
odd	odd	$\bar{x}/2 - 1/2$
even	odd	$\bar{x}/2 - 1/2 - 1/(2 \cdot 3^{k+1})$
odd	even	$\bar{x}/2 - 1/(2 \cdot 3^{k+1})$
even	even	$\bar{x}/2$

In order for a path to reach the 1-4-2-1 cycle, the value of $\bar{x}$ must approach zero; this is a necessary condition for this end case. This condition is not sufficient, however; any end cycle, or in fact any path, taken by the integers will converge toward $\bar{x} \to 0$. This is easily seen by noting that, once we run out of bits in x, no matter whether the modulus increases or not, the value of $\bar{x}$ will approach zero.

Also note that, on the Layer-k diagrams, if we are at layer k on residue r, then the bit sequence of $\bar{x}$ is exactly that bit sequence of x which will keep us on layer k indefinitely, and any deviation from this sequence will take us to layer $k+1$.

Of greater interest is the iterative process, or abstract machine, which we might build to encode this process:

$$\bar{x}_{d+1} = \bar{x}_d/2 - \vec{e}_{x_d}^{\,T} \cdot \begin{bmatrix} 0 & \frac{1}{2 \cdot 3^{k+1}} \\ \frac{1}{2} + \frac{1}{2 \cdot 3^{k+1}} & \frac{1}{2} \end{bmatrix} \cdot \vec{e}_{r \bmod 2} \tag{35}$$

Using $\bar{x}$ as a measure of execution, the Collatz conjecture reduces to the question, "Does $\bar{x}$ eventually become a reciprocal power of three for all integer inputs?" This can also be seen as a contraction of a particular representation of the generalized Collatz map $g(x) = a_i x + b_i$ (when $x \equiv i \mod p$):

$$g(x) = \vec{e}_{x \bmod p}^{\,T} \cdot \begin{bmatrix} a_0 & b_0 \\ \ldots & \ldots \\ a_{p-1} & b_{p-1} \end{bmatrix} \cdot \begin{bmatrix} x \\ 1 \end{bmatrix} \tag{36}$$

Similar measures $\bar{x}$ can be constructed for these generalizations, including the $5x+1$ and $7x+1$ problems ($\bar{x}$ is $-r/5^k$ and $-r/7^k$, respectively).

5 Conclusion

5.1 Future Research

I hope to further investigate the fact that, using the reduced Collatz map (where we divide out the factors of 2 instantly, mapping odds to odds), no number shares any factors with its Collatz successor (and no Collatz successor has a factor of 2 or 3), i.e. $\gcd(x, C(x)) = 1$. This is easily deduced via a single lemma: that in the sum of any two numbers $a + b = c$, the only factors c shares with a or b are exactly those factors which a and b already had in common (all other factors of c are unique to c, or $\gcd(a, b) = 1 \implies \gcd(a, c) = \gcd(b, c) = 1$).

Acknowledgment

I want to dedicate this particular paper to my grandmother. During Summer 2020 I had the opportunity to spend two and a half weeks with her in the mountains of Northern California, during which time I discovered most of the material described here during breaks from gardening in the summer heat with her. Thank you Grandma for your support, love, and care.

References

[1] Lagarias, Jeffrey C. “The 3x + 1 Problem and Its Generalizations.” The American Mathematical Monthly, vol. 92, no. 1, 1985, pp. 3–23. JSTOR, https://doi.org/10.2307/2322189. Accessed 9 Sept. 2023.

[2] Rowland, Eric S. (2009), “Regularity versus Complexity in the Binary Representation of 3^n.” Complex Systems, vol. 18 issue 3. https://doi.org/10.48550/arXiv.0902.3257

[3] G. Perry, “Colaboratory,” 2024, verified: 2024-08-07. [Online]. Available: https://colab.research.google.com/drive/1xz3dsfY49Ay3ft-Yf8eDKnITY72jj8yq