A framework to drive a proof or refutation of the Collatz Conjecture

The problem

Given the function defined for the natural numbers:

f(n) = (3n+1)/2 of n is odd n/2 if n is even (

[/ f(n) =]

And its kth application f^k(n). By definition f^0(n) = n

Notice that for odd branches we are using the shortcut version provided that 3n+1, being n odd is guaranteed to be even and the next step is always dividing by two.

Let's define that a number n is *reductible* if it exist a finite k so that f^k(n)=1

By brute force we know that all the first checked natural numbers are reductible. As for today this has been proved until 2^68 (David Bařina https://github.com/hellpig/collatz).

The Collatz conjecture states that: all natural numbers are reductible.

Toolbox

Powers of three

We can relate a power of 3 with lesser powers of 3

Given that for any base b and natural power n:

```
b^n - 1 = (b-1) * sum[0<=i<n](b^i)
b^n = 1 + (b-1) * sum[0<=i<n](b^i)
```

For b = 3:

```
3^n = 1 + 2*sum[0<=i<n](3^i)
3^n = 1 + 2*(3^n-1 + 3^n-2 ... + 3 + 1)
```

This is also equivalent to those formulas:

```
(3^n - 1)/2 = (3^n-1 + 3^n-2 ... + 3 + 1)
(3^n - 1)/2 = sum[0<=i<n](3^i)

sum(3^k)[0<=k<n] = (3^n - 1) / 2

(3^n + 1)/2 = 1 + (3^n-1 + 3^n-2 ... + 3 + 1)
(3^n + 1)/2 = 1 + sum[0<=i<n](3^i)
```

We can also relate powers of 3 in terms of powers of 2 by using the binomial theorem

```
3^n = (2 + 1)^n
3^n = sum[0<=i<=n]( 2^i * n! / (n-i)! / i! )
3^n = sum[0<=i<=n]( 2^i * bincoef(i,n) )
```

Boolean with integer arithmetics

Enable us to integrate boolean conditions into natural numbers expresions. Useful to eliminate formula branching.

Let's define a boolean integer as B€[0,1]

Being a,b,c… boolean integers, we can represent boolean operations with integer algebra like this:

- not: not a = 1-a
- and: a and b = (a*b)
    - Properties:
        - a*1 = a
        - a*0 = 0
        - a*a = a — because both 1 and 0 multiplied by themselves return themselves
        - a*(1-a) = a - a*a = a - a = 0
- or: a or b = (a + b - a*b)
    - Properties:
        - a or 1 = a + 1 - 1*a = 1
        - a or 0 = a + 0 - 0*a = a
        - a or a = a + a - a*a = a
        - a or not a = a + (1 -a) - a * (1-a) = a +1 -a -a +a*a = 1

Other derived operators

- xor: a xor b = a*(1-b) + b*(1-a) = a-ab+b-ab = a+b-2ab = a*a +b*b -2ab = (a-b)^2
    - a xor 0 = (a-0)^2 = a
    - a xor 1 = (a-1)^2 = a*a +1 - 2*a = 1-a = not a
    - a xor a = (a-a)^2 = 0
    - a xor not a = (a - (1-a))^2 = (2*a-1)^2 = 4*a + 1 - 4*a = 1
- eq: a eq b = a*b + (1-a)*(1-b) = 2*a*b -a -b +1 = 1 - (a-b)^2 = not (a xor b)

Be careful that a+b-ab is an OR while a+b-2ab is an XOR.

Those operations ensure a closure among booleans integers. Meaning that while the operands are 0 or 1, the result will be also 0 or 1.

So, how to use those boolean expresions inside an algebraic fórmula? We can make multiplication factors and addition terms optional.

Being x and y natural numbers and 'a' a boolean condition represented as integer,

- conditional addition of a term x: a*x + y
- conditional multiplication by a factor x: x^a * y

Oddity of algebraic expressions

Oddity is a function that returns a boolean integer, 1 if the number is odd. Also useful to eliminate branching.

Being a and b integer expressions:

```
odd(1) = 1
odd(2*a) = 0
odd(a+b) = odd(a) xor odd(b) = odd(a) + odd(b) - 2*odd(a)*odd(b) = (odd(a)-odd(b))^2
odd(a*b) = odd(a) and odd(b) = odd(a)*odd(b)
```

Pair terms can be ignored for oddity:

```
odd(2*a + b) = odd(2*a) xor odd(b) = 0 xor odd(b) = odd(b)
odd((1+2*a)**b) = 1
odd((2*a)**b) = (b!=0)   --- TODO: which opp gives this?
```

Unbranched formula

Unbranched formula for the generator function:

```
f(n) = 1/2 * (n*3^odd(n) + odd(n) )
```

Thus we can define the series recursively by:

```
f0(n) = n
fk+1(n) = 1/2 * ( fk(n) * 3^(odd(fk(n))) + odd(fk(n)) )
```

Another formulation is:

```
fk+1(n) = 1/2 * ( fk(n) * (1 + 2*odd(fk(n))) + odd(fk(n)) )
```

Which can be convenient to extract factors of two.

Lets define Ok as odd(fk(n)). Then we can express both formulations as

```
fk+1(n) = (3^Ok * fk(n) + Ok) / 2
```

Or also:

```
fk+1(n) = ((1+2*Ok) * fk(n) + Ok) / 2
```

For compactness, we will omit (n), so fk = fk(n)

```
fk+1 = (3^Ok * fk + Ok) / 2     # Ok exponential form
fk+1 = (fk + 2*Ok*fk  + Ok) / 2 # Ok factor form
```

Additive/Substractive view

```
fk+1 - fk =
= (fk + 2*Ok*fk  + Ok) / 2 - fk   # Using Ok factor form
= (-fk + 2*Ok*fk  + Ok) / 2       # fk inside 1/2
    Odd: (-fk + 2*fk  + 1) / 2 = fk/2 + 1/2
```

```
    Even: -fk/2

fk odd : +fk/2 + 1/2
fk even: -fk/2
```

Conclusión: Depending on the oddity of the previous result, we are adding or substracting half of the sequence value, rounding up for odds.

Strategies

Hypothesis: Exists a first natural number A that it is not reductible.

Strategy 1: If exists a first non reductible natural A implies that any `n<A` is reductible. Because A is not reductible, `f_k(A)>=A` for all k. Having `f_k(A)<A` will contradict the hypothesis. If the hypothesis is true, it could lead to a search algorithm for A.

Suposition: The oddness of the f_k(n) just depends on the lower k+1 bits of n.

Strategy 2: Being A a finite number, at a given k all remaining bits are zero. Because the oddity of f_k(n) is inverted depending on the kth bit, maybe we could find a pattern for which appending 0 bits gets higher and higher or cyclic.

Strategy 3: Constructive. Requires demonstrating that kth bit of A controls oddity of f_k(x). Instead of starting with every number and apply the function to reduce it. Start on 1 and invert f so that we can get to that number by the odd and even formula.

Solution structure

Theorem: All f^k(n) can be expressed as (an+b)/c being a and c extrictly positive integers, and b positive integer.

Proof:

```
f^0(n) = n; (a0=1, b0=0, c0=1)

Given that f^k(n) can be expressed as (ak*n +bk)/ck,
can f^k+1(n) be expressed as (a'*n+b')/c' ?

if f^k(n) is odd: f^k+1(n) = (3*a*n + 3*b + c)/2*c
(a'=3*a, b'=3*b+c, c'=2*c)

if f^k(n) is even: f^k+1(n) = (a*n + b) /2*c
(a'=a, b'=b, c'=2*c)
```

qvd

In single branch

```
f^k+1(n) = ( a + 2*Ok*a, b + 2*Ok*b + Ok, 2*c )

a'= (2*Ok +1) * a
b'= b + b*2*Ok + c * Ok
c'= 2*c

ak+1 = prod[i=0..k] 3^Oi = 3 ^ sum[i=0..k] Oi
bk+1 = bk * 3^Ok + ck + Ok = bk * 3^Ok + 2^k-1 + Ok
ck+1 = 2^k
```

Empirical observations

Being nk the kth bit of n binary base representation. And Nk = N » k, this is the integer division by 2^k.

All developed solutions have the form:

```
fk(n) = 3^Bk*Nk + Ck
```

Where Bk and Ck depend only on bits nk-1 to n_0. 0<=Bk<=k and 0<=Ck<3^Bk Indeed max(Ck)=3^Bk-1 and happens when the Bk *higher processed* bits are 1 (Caution this has been observed just for the first 5 levels)

Lets try to demonstrate those observations.

All solutions as fk(n) = 3^Bk*Nk + Ck

Hypothesis: solutions can be represented as:

```
fk(n) = Nk*3^Bk + Ck
```

So that:

```
0 <= Bk <= k
0 <= Ck < 3^Bk

for k=0, O0 = n0
f0(n) = N0;  thus B0 = 1, C0 = 0

for k=1
f1(n) = (3^n0 * N0 + n0)/2
f1(n) = (3^n0 * (2*N1+n0) + n0)/2    --- Expand N0=2*N1 + n_0
f1(n) = (3^n0*2*N1 + 3^n0 * n0 + n0)/2    -- distribute 3^n0
f1(n) = (3^n0*2*N1 + n0 * 3^n0 + n0)/2   -- reorder factors
f1(n) = (3^n0*2*N1 + n0 * 3^n0 + n0)/2   -- 3^n0 = 1 + 2*n0 for n0€(0,1)
f1(n) = (3^n0*2*N1 + n0 * (1+2*n0) + n0)/2   -- 3^n0 = 1 + 2*n0 for n0€(0,1)
f1(n) = (3^n0*2*N1 + 2*n0 * 2*n0*n0 )/2   -- 3^n0 = 1 + 2*n0 for n0€(0,1)
f1(n) = (3^n0*2*N1 + 4*n0 )/2   -- n0^2 = n0 for n0€(0,1)
f1(n) = 3^n0*N1 + 2*n0 --  divide by 2
```

```
f1(n) = N1 + 2*n0*N1 + 2*n0 --  3^n0 = 2*n0 + 1

O1 = odd(f1(n)) = odd(N1 + 2*n0*N1 + 2*n0) = odd(N1) = n1
```

So for k=1, B1=n0 and C1 = 2*n0

```
0 <= B1 = n0 <= k = 1
0 <= C1 = 2*n0 < 3^n0 = 1+2*n0
```

Now, suposing that:

```
fk = Nk*3^Bk + Ck
0 <= Bk <= k
0 <= Ck < 3^Bk
```

Let's demonstrate that:

```
fk+1(n) = (Nk+1)*3^Bk+1 + Ck+1
0 <= Bk <= Bk+1 <= k+1
0 <= Ck+1 < 3^Bk+1

Ok = odd(fk)
= odd(Nk*3^Bk + Ck)
= odd(Ck) + odd(Nk*3^Bk) -2*odd(Ck)*odd(Nk*3^Bk)   -- exclusive or
= odd(Ck) + odd(Nk) - 2*odd(Ck)*odd(Nk)   --- odd(Nk*3^Bk) = odd(Nk)
= odd(Ck) + nk - 2*nk*odd(Ck)   --- odd(Nk) = odd(2*Nk+1 + nk) = nk

fk+1 = (3^Ok*fk + Ok)/2    --- Single branch formula
fk+1 = (3^Ok*(Nk*3^Bk + Ck) + Ok)/2    -- fk = Nk*3^Bk + Ck
fk+1 = (3^Ok*Nk*3^Bk + 3^Ok*Ck + Ok)/2    -- distribute
fk+1 = (3^(Bk+Ok)*Nk + Ck*3^Ok + Ok)/2    -- adding exponents
fk+1 = (3^(Bk+Ok)*(2*Nk+1 + nk) + Ck*3^Ok + Ok)/2    -- Nk = 2*Nk+1 + nk
fk+1 = (3^(Bk+Ok)*2*Nk+1 + 3^(Ok+Bk)*nk + Ck*3^Ok + Ok)/2    -- distribute
fk+1 = 3^(Bk+Ok)*Nk+1 + (3^(Ok+Bk)*nk + Ck*3^Ok + Ok)/2    -- divide Nk+1 term

Bk+1 = Bk + Ok
Bk+1 = Bk + nk + Odd(Ck) - 2*nk*Odd(Ck)

2*Ck+1 =
= 3^(Bk+Ok)*nk + Ck*3^Ok + Ok    -- from fk+1 expression
= 3^Ok*3^Bk*nk + Ck*3^Ok + Ok    -- split powers
= (2*Ok + 1)*3^Bk*nk + Ck*(2*Ok + 1) + Ok    -- 3^Ok = 2*Ok + 1
= 2*Ok*3^Bk*nk + 3^Bk*nk + Ck*2*Ok + Ck + Ok    -- distribute
= 2*(odd(Ck) + nk - 2*nk*odd(Ck))*3^Bk*nk + 3^Bk*nk + Ck*2*(odd(Ck) + nk - 2*nk*odd(Ck)) + (
=    ---   just reorder
    + 2*nk*3^Bk*(odd(Ck) + nk - 2*nk*odd(Ck))
    + 3^Bk*nk
    + Ck*2*(odd(Ck) + nk - 2*nk*odd(Ck))
    + Ck
```

```
    + (odd(Ck) + nk - 2*nk*odd(Ck))
=    ---   distribute
    + 2*nk*3^Bk*(odd(Ck))
    + 2*nk*3^Bk*(nk)
    + 2*nk*3^Bk*(- 2*nk*odd(Ck))
    + 3^Bk*nk
    + Ck*2*(odd(Ck))
    + Ck*2*(nk)
    + Ck*2*(- 2*nk*odd(Ck))
    + Ck
    + odd(Ck)
    + nk
    - 2*nk*odd(Ck)
=    ---   distribute
    + 2*nk*3^Bk*odd(Ck)
    + 2*nk*3^Bk*nk
    - 4*nk*3^Bk*nk*odd(Ck)
    + 3^Bk*nk
    + 2*Ck*odd(Ck)
    + 2*Ck*nk
    - 4*Ck*nk*odd(Ck)
    + Ck
    + odd(Ck)
    + nk
    - 2*nk*odd(Ck)
=    ---   nk*nk = nk
    + 2*nk*3^Bk*odd(Ck)
    - 4*nk*3^Bk*odd(Ck)
    + 2*nk*3^Bk
    + 3^Bk*nk
    + 2*Ck*odd(Ck)
    + 2*Ck*nk
    - 4*Ck*nk*odd(Ck)
    + Ck
    + odd(Ck)
    + nk
    - 2*nk*odd(Ck)
=    ---   grouping factors
    - 2*nk*3^Bk*odd(Ck)
    + 3*nk*3^Bk
    + nk
    + 2*Ck*nk
    - 4*Ck*nk*odd(Ck)
    + 2*Ck*odd(Ck)
    + Ck
    + odd(Ck)
```

```
    - 2*nk*odd(Ck)
=    ---   grouping factors
    + nk*3^Bk * (3 - 2*odd(Ck))
    + nk
    + 2*Ck*(nk + odd(ck) - 2*nk*odd(Ck)
    + Ck
    + odd(Ck)
    - 2*nk*odd(Ck)
```

In order to endup with a natural number 2*Ck+1 should be even:

```
odd(2*Ok*3^Bk*nk + 3^Bk*nk + Ck*2*Ok + Ck + Ok) =
= odd(3^Bk*nk + Ck + Ok)            --- Removed even terms
= odd(3^Bk*nk + Ck + odd(Ck) + nk - 2*nk*odd(Ck))  --- Ok = odd(Ck) + nk - 2*nk*odd(Ck)
= odd(Ck + odd(Ck) - 2*nk*odd(Ck))   ---  odd(3^Bk*nk + nk) = 0
= odd(Ck + odd(Ck))   ---  pair term ignored
= odd(0)   ---  odd(Ck + odd(Ck)) = 0
= 0  (qvd)
```

Ck+1 = = (— grouping factors + nk*3^Bk* (3 - 2*odd(Ck)) + 2*Ck*(nk + odd(ck)
- 2*nk*odd(Ck)*
*+ Ck + nk + odd(Ck) - 2*nk*odd(Ck)) / 2

By cases nk, odd(Ck).

```
nk=0; odd(Ck)=0; Bk+1=Bk*(1+2*Ok) = Bk
    2*Ck+1 =
    =
        + nk*3^Bk * (3 - 2*odd(Ck))
        + 2*Ck*(nk + odd(ck) - 2*nk*odd(Ck)
        + Ck
        + nk
        + odd(Ck)
        - 2*nk*odd(Ck)
    = Ck

    Ck+1 = Ck/2 < Ck < 3^Bk = 3^Bk+1

nk=0; odd(Ck)=1; Bk+1 = Bk + 1
    2*Ck+1 =
    =
        + nk*3^Bk * (3 - 2*odd(Ck))
        + 2*Ck*(nk + odd(ck) - 2*nk*odd(Ck)
        + Ck
        + nk
        + odd(Ck)
        - 2*nk*odd(Ck)
    = 3*Ck + 1
```

```
    2*Ck+1 = 3*Ck + 1 <? 2*3*3^Bk
    3*Ck <? 2*3*3^Bk -1
    Ck <? 2*3^Bk - 1/3
    Ck < 3^Bk <! 2*3^Bk - 1/3

nk=1; odd(Ck)=0
    2*Ck+1 =
    =
        + nk*3^Bk * (3 - 2*odd(Ck))
        + 2*Ck*(nk + odd(ck) - 2*nk*odd(Ck)
        + Ck
        + nk
        + odd(Ck)
        - 2*nk*odd(Ck)
    =
        + 3^Bk * 3
        + 3*Ck
        + 1

    2*Ck+1 = 3*3^Bk + 3*Ck + 1 <? 2*3*3^Bk
    3*Ck + 1 <? 3*3^Bk
    Ck <? 3^Bk - 1/3

nk=1; odd(Ck)=1
    2*Ck+1 =
    =
        + nk*3^Bk * (3 - 2*odd(Ck))
        + 2*Ck*(nk + odd(ck) - 2*nk*odd(Ck)
        + Ck
        + nk
        + odd(Ck)
        - 2*nk*odd(Ck)
    =
        + 3^Bk
        + Ck
    2*Ck+1 = 3^Bk + Ck <? 2*3^Bk
    3^Bk + Ck <? 2*3^Bk
    Ck <! 3^Bk
```

Thus, it's demonstrated that for every k:

```
fk(n) = Nk*3^Bk + Ck
```

Where:

```
0 <= Bk <= k
0 <= Ck < 3^Bk
```

Ck Oddity

From the previous demonstration we got an expression of what feeds Cks from nks

```
Ck+1 =
    =   (
        + nk*3^Bk * (3 - 2*odd(Ck))
        + 2*Ck*(nk + odd(ck) - 2*nk*odd(Ck)
        + Ck
        + nk
        + odd(Ck)
        - 2*nk*odd(Ck)
    ) / 2
```

It would be nice to have a generalization of Ck oddity

```
odd(Ck+1) =
    odd((
        + nk*3^Bk * (3 - 2*odd(Ck))
        + 2*Ck*(nk + odd(ck) - 2*nk*odd(Ck))
        + Ck
        + nk
        + odd(Ck)
        - 2*nk*odd(Ck)
    ) /2)
```

Again by cases:

```
odd(Ck) = 0; nk = 0
    odd(Ck+1) =
    =
        odd((
            + nk*3^Bk * (3 - 2*odd(Ck))
            + 2*Ck*(nk + odd(ck) - 2*nk*odd(Ck))
            + Ck
            + nk
            + odd(Ck)
            - 2*nk*odd(Ck)
        ) /2)
    =                   --- nk = 0
        odd((
            + 2*Ck*odd(ck)
            + Ck
            + odd(Ck)
        ) /2)
    =                   --- odd(Ck) = 0
        odd((
```

```
            + Ck
        ) /2)
    =                  --- simplify
        odd(Ck/2)

odd(Ck) = 0; nk = 1
    odd(Ck+1) =
    =
        odd((
            + nk*3^Bk * (3 - 2*odd(Ck))
            + 2*Ck*(nk + odd(ck) - 2*nk*odd(Ck))
            + Ck
            + nk
            + odd(Ck)
            - 2*nk*odd(Ck)
        ) /2)
    =                  --- odd(Ck) = 0
        odd((
            + nk*3^Bk * (3)
            + 2*Ck*(nk)
            + Ck
            + nk
        ) /2)
    =                  --- nk = 1
        odd((
            + 3^Bk * (3)
            + 2*Ck
            + Ck
            + 1
        ) /2)

    =                  --- even outside half
        odd(
            Ck + Ck/2 +
            (
            + 3^Bk * (3)
            + 1
            ) /2
        )

    =                  --- Ck being even does not affect overall oddity
        odd(
            Ck/2 +
            (
            + 3^Bk * (3)
            + 1
```

```
            ) /2
        )
    =                   --- factor of
        odd(
            Ck/2 +
            (
            + 3^(Bk + 1)
            + 1
            ) /2
        )

odd(Ck) = 1; nk = 0
    odd(Ck+1) =
    =
        odd((
            + nk*3^Bk * (3 - 2*odd(Ck))
            + 2*Ck*(nk + odd(ck) - 2*nk*odd(Ck))
            + Ck
            + nk
            + odd(Ck)
            - 2*nk*odd(Ck)
        ) /2)

odd(Ck) = 1; nk = 1
    odd(Ck+1) =
    =
        odd((
            + nk*3^Bk * (3 - 2*odd(Ck))
            + 2*Ck*(nk + odd(ck) - 2*nk*odd(Ck))
            + Ck
            + nk
            + odd(Ck)
            - 2*nk*odd(Ck)
        ) /2)
```

The last significative bit

Let's define the upper Nk and lower bits Lk so that:

```
N0 = 2**k * Nk + Lk
Lk = sum(ni * 2**i for i in range(k))
Nk = sum(ni+k * 2**i for i in range(n-k))
```

For any finite number there exists a position of the most significative bit k so that nk=1 and ni=0 for any `i>k`.

Also Nk=1, Ni = 0 for `i>k`.

```
fk(N0) = Nk * 3^Bk + Ck
fk(N0) = 3^Bk + Ck

fk(Lk) = Ck
```

Let’s be N0 the first unreductible natural number. Because Lk = N0 - 2^k < N0, both Lk and Ck are reductible

```
fk+1 = (Nk+1)*3^Bk + Ck+1 = Ck+1

2*fk+1 = 2*Ck+1 =
=
    + nk*3^Bk * (3 - 2*odd(Ck))
    + nk
    + 2*Ck*(nk + odd(ck) - 2*nk*odd(Ck)
    + Ck
    + odd(Ck)
    - 2*nk*odd(Ck)
=                        ---- nk = 1
    + 3^Bk * (3 - 2*odd(Ck))
    + 1
    + 2*Ck*(1 + odd(ck) - 2*odd(Ck)
    + Ck
    + odd(Ck)
    - 2*odd(Ck)
=                        ---- nk = 1
    + 3^Bk * (3 - 2*odd(Ck))
    + 1
    + 2*Ck*(1 - odd(Ck))
    + Ck
    - odd(Ck)
=                        ---- split terms
    + 3^Bk * (3 - 2*odd(Ck))
    + 1
    + 3*Ck
    - 2*Ck*odd(Ck))
    - odd(Ck)

For odd(Ck) = 1
    2*fk+1 = 3^Bk + Ck
    fk+1 = (3^Bk + Ck)/2
    fk+1 = sum[0<=i<Bk](3^i) + (Ck + 1)/2

For odd(Ck) = 0
    2*fk+1 = 3*3^Bk + 3*Ck + 1
    fk+1 = (3*3^Bk + 3*Ck +1)/2
```

Prunning outcomes

An outcome gets prunned whenever exists a k so that:

```
2^k * Nk + Lk > 3^Bk * Nk + Ck

Nk (2^k - 3^Bk) > Ck - Lk
```

Beyond the most significative bit:

```
Ck < Lk = NO   ## Nothing new apparently
```

For the most significative bit

```
Nk=1
(2^k - 3^Bk) > ( Ck - Lk)
```

For the less significative: Nk>1

```
Nk > ( Ck - Lk) / (2^k - 3^Bk)
```

Because Nk Can be discarded whenever the right side is negative: - Lk <= Ck and 2^k >= 3^Bk - Lk >= Ck and 2^k <= 3^Bk

```
3^h/2^n < 2^h
ln(3^h/2^n) < ln(2^h)
ln(3^h)-ln(2^n) < ln(2^h)
h*ln(3)-n*ln(2) < h*ln(2)
h*ln(3) < h*ln(2)-n*ln(2)
h*ln(3) < (h-n)*ln(2)
```

B = A » 1

A Even, A = 2*B A:B0 f^1(A) = B/2 = B

A Odd, A = 2*B + 1 A:B1 f^1(A) = (3*A+1)/2 = (6*B+3+1)/2 = 3*B + 2 So: f^1(A) = 3*B + 2 same oddity as B

```
C = B >> 1

B Even, B = 2*C
A:C01
f^2(A) = f^1(3*B + 2) = (6*C + 2)/2 = 3*C + 1
So: f^2(A) = 3*C + 1  (oposite oddity than C)

    D = C >> 1

    C Even, C = 2*D
    A:D001
    f^3(A) = f^1(3*C + 1) = f^1(6*D + 1) =  (18*D + 4)/2 = 9*D+2
    So: f^3(A)  = 9*D + 2  (same odity)
```

```
    C Odd, C = 2*D +1
    A:D101
    f^3(A) = f^1(3*C + 1) = f^1(6*D + 4) =  3*D + 2
    So: f^3(A)  = 3*D + 2  (same odity)

B Odd, B = 2*C +1
A:C11
f^2(A) = f^1(3*B + 2) = (9*B+6+1)/2 = (9*(2*C +1)+6+1)/2 = 9*C + 8
So: f^2(A) = 9*C + 8  (same oddity as C)

    D = C >> 1

    C Even, C = 2*D
    A:D011
    f^3(A) = F^1(9*C + 8) = F^1(18*D+8) = (18D+8)/2 = 9*D +4
    f^3(A) = 9*D + 4 (same oddity as C)

    C Odd, C = 2*D + 1
    A:D111
    f^3(A) = F^1(9*C + 8) = F^1(18*D+17) = (3*18D+3*17 +1)/2 = 27*D +26
    f^3(A) = 27*D +26 (same oddity as C)
```

- Let be a_k the kth bit of the binary representation of A.
- Let be r_k = A»(k) (the integer division of A by the n power of two)

If r_0 is even, a_0 is 0, then f^1(r_0) = r_1 = A/2 < A, thus imposible. Thus r_0 is odd. A = 2 r_1 + 1, a_0=1 A odd means that bit 0 is 1. And then f^1(A) = 3A+1/2 = (6B+3+1)/2 = 3B + 2

a_1=0 B even? If B was even, then f^2(A)=(3A+1)/4 >= A (for A>1) and A would be reductible. So B is odd, lets B=2C+1; A = 2(2C+1)+1 = 4C+3 B odd means that bit 1 is 1. f^2(A) = 3((3A+1)/2)+1 = (9A+3)/2+1 = (9A+5)/2 = 18C + 16 so even f^3(A) = (9A+5)/4 = 9C+8

aX+b

a even, b even -> even a even, b odd -> odd a odd, b even -> whatever X is a odd, b odd -> whatever X is not

Visualization: We will get a binary search tree, where each levels decide a bit of the number. We can prune the tree every time the function evaluates less than A. We evolve an expression for the f^n(A) in terms of A to get this comparision. We evolve an equivalent expresion for f^n(A) in terms of the undecided bits. If A is a finite number, beyond a given n, the undecided bits are always zero. Which pattern makes that even with zeros as undecided bits it keeps always>=A?

Note: If we can demostrate that in order not getting under A, we have to add bits for ever then we are demonstrating that the number is infinite.

k even: f^3(n0) = (3n0+1)/4 < n0

k odd: k=2k'+1 f^3(n0) = 3((3n0+1)/2)+1 = (9n0+3)/2 +1 = (9n0+5) f^3(n0) = 3(3k+2)+1 = 9k + 7 = 18k'

Because n0 it is odd, n0 reduces to 3n0+1 = 6k+3+1 = 6k + 4 (even) Being even reduces to 3k+2

Constructive aproach

fodd^-1 = (2n-1)/3 feven^-1 = 2n

```
1
    1/3 X
    2 10
        1 loop
        4 100
            7/3 X
            8 1000
                5 101
                    3 11
                        5/3 X
                        6 110
                            11/3 X
                            12 1100 ...
                    10 1010
                        19/3 X
                        20 10100
                            13 1101 ...
                            40 101000 ...
                16 10000
                    31/3 X
                    32 100000
                        21 10101 ...
                            41/3 X
                            42 101010
                                85/3 C
                                84 1010100
                        64 1000000 ...
```

Limiting factor

N0 will be reductible if for any k

N0 > $3^{\mathrm{Bk}Nk\ +\ Ck\ Nk2}$k + sum(ni * 2^i)(0=i<k) > 3^Bk*Nk + Ck

Expansion depending on the least significative bits

```
0 N1 + 0
    00 N2 + 0
        000 N3 + 0
            0000 N4 + 0
                00000 N5 + 0
                    000000 N6 + 0
                    100000 3*N6 + 2
                10000 3*N5 + 2
                    010000 3*N6 + 1
                    110000 9*N6 + 8
            1000 3*N4 + 2
                01000 3*N5 + 1
                    001000 9*N6 + 2
                    101000 3*N6 + 2
                11000 9*N5 + 8
                    011000 9*N6 + 4
                    111000 27*N6 + 26
        100 3*N3 + 2
            0100 3*N4 + 1
                00100 9*N5 + 2
                    000100 9*N6 + 1
                    100100 27*N6 + 17
                10100 3*N5 + 2
                    010100 3*N6 + 1
                    110100 9*N6 + 8
            1100 9*N4 + 8
                01100 9*N5 + 4
                    001100 9*N6 + 2
                    101100 27*N6 + 20
                11100 27*N5 + 26
                    011100 27*N6 + 13
                    111100 81*N6 + 80
    10 3*N2 + 2
        010 3*N3 + 1
            0010 9*N4 + 2
                00010 9*N5 + 1
                    000010 27*N6 + 2
                    100010 9*N6 + 5
                10010 27*N5 + 17
                    010010 81*N6 + 26
                    110010 27*N6 + 22
            1010 3*N4 + 2
                01010 3*N5 + 1
                    001010 9*N6 + 2
```

```
                101010 3*N6 + 2
            11010 9*N5 + 8
                011010 9*N6 + 4
                111010 27*N6 + 26
        110 9*N3 + 8
            0110 9*N4 + 4
                00110 9*N5 + 2
                    000110 9*N6 + 1
                    100110 27*N6 + 17
                10110 27*N5 + 20
                    010110 27*N6 + 10
                    110110 81*N6 + 71
            1110 27*N4 + 26
                01110 27*N5 + 13
                    001110 81*N6 + 20
                    101110 27*N6 + 20
                11110 81*N5 + 80
                    011110 81*N6 + 40
                    111110 243*N6 + 242
1 3*N1 + 2
    01 3*N2 + 1
        001 9*N3 + 2
            0001 9*N4 + 1
                00001 27*N5 + 2
                    000001 27*N6 + 1
                    100001 81*N6 + 44
                10001 9*N5 + 5
                    010001 27*N6 + 8
                    110001 9*N6 + 7
            1001 27*N4 + 17
                01001 81*N5 + 26
                    001001 81*N6 + 13
                    101001 243*N6 + 161
                11001 27*N5 + 22
                    011001 27*N6 + 11
                    111001 81*N6 + 74
        101 3*N3 + 2
            0101 3*N4 + 1
                00101 9*N5 + 2
                    000101 9*N6 + 1
                    100101 27*N6 + 17
                10101 3*N5 + 2
                    010101 3*N6 + 1
                    110101 9*N6 + 8
            1101 9*N4 + 8
                01101 9*N5 + 4
```

```
                001101 9*N6 + 2
                101101 27*N6 + 20
            11101 27*N5 + 26
                011101 27*N6 + 13
                111101 81*N6 + 80
    11 9*N2 + 8
        011 9*N3 + 4
            0011 9*N4 + 2
                00011 9*N5 + 1
                    000011 27*N6 + 2
                    100011 9*N6 + 5
                10011 27*N5 + 17
                    010011 81*N6 + 26
                    110011 27*N6 + 22
            1011 27*N4 + 20
                01011 27*N5 + 10
                    001011 27*N6 + 5
                    101011 81*N6 + 56
                11011 81*N5 + 71
                    011011 243*N6 + 107
                    111011 81*N6 + 76
        111 27*N3 + 26
            0111 27*N4 + 13
                00111 81*N5 + 20
                    000111 81*N6 + 10
                    100111 243*N6 + 152
                10111 27*N5 + 20
                    010111 27*N6 + 10
                    110111 81*N6 + 71
            1111 81*N4 + 80
                01111 81*N5 + 40
                    001111 81*N6 + 20
                    101111 243*N6 + 182
                11111 243*N5 + 242
                    011111 243*N6 + 121
                    111111 729*N6 + 728
```

Demonstrated facts

```
fk = Nk*3^Bk + Ck
where
    Nk = N0 >> k
    0 <= Ck < 3^Bk
    0 <= Bk <= k

Bk = sum[0<=i<k](odd(fi))
```

```
Ck also depends only on the kth lower bits of N0 (N0 - 2^k * Nk)
Numbers sharing lower k bits share also the oddity of the kth first terms

Bk+1 = Bk + odd(nk + Ck)

Ck+1 = (
    + nk*3^Bk * (3 - 2*odd(Ck))
    + Ck* (1 + 2*odd(nk + Ck))
    + odd(nk + Ck)
) / 2

2*Ck+1
= (2*Ok + 1)*3^Bk*nk + Ck*(2*Ok + 1) + Ok    -- 3^Ok = 2*Ok + 1
= 2*Ok*3^Bk*nk + 3^Bk*nk + Ck*2*Ok + Ck + Ok    -- distribute
Ck+1 = Ok*(nk*3^Bk + Ck) + (nk*3^Bk + Ck + Ok)/2     --- divide by 2
Ck+1 = Ok*(nk*3^Bk + Ck) + (nk*3^Bk + Ck + Ok)/2     --- Ok = nk + odd(Ck) - 2*nk*odd(Ck)
Ck+1 = (nk + odd(Ck) -2*nk*odd(Ck)*(nk*3^Bk + Ck) + (nk*3^Bk + Ck + nk + odd(Ck) -2*nk*odd((
Ck+1 =                                    --- split in lines
    + (nk + odd(Ck) -2*nk*odd(Ck)*(nk*3^Bk + Ck)
    + (nk*3^Bk + Ck + nk + odd(Ck) -2*nk*odd(Ck))/2
Ck+1 =                                    ---  distribute first term
    + (nk + odd(Ck) -2*nk*odd(Ck))*nk*3^Bk
    + (nk + odd(Ck) -2*nk*odd(Ck))*Ck
    + (nk*3^Bk + Ck + nk + odd(Ck) -2*nk*odd(Ck))/2
Ck+1 =                                    ---  nk*nk = nk
    + (1 + odd(Ck) -2*odd(Ck))*nk*3^Bk
    + (nk + odd(Ck) -2*nk*odd(Ck))*Ck
    + (nk*3^Bk + Ck + nk + odd(Ck) -2*nk*odd(Ck))/2
Ck+1 =                                    ---  nk*nk = nk
    + (1 - odd(Ck))*nk*3^Bk
    + (nk + odd(Ck) -2*nk*odd(Ck))*Ck
    + (nk*3^Bk + Ck + nk + odd(Ck) -2*nk*odd(Ck))/2
Ck+1 =                                    ---  group 3^Bk + 1
    + (1 - odd(Ck))*nk*3^Bk
    + (nk + odd(Ck) -2*nk*odd(Ck))*Ck
    + (nk*(3^Bk + 1) + odd(Ck) + Ck -2*nk*odd(Ck))/2
Ck+1 =                                    ---  split (3^Bk + 1)/2 term
    + (1 - odd(Ck))*nk*3^Bk
    + nk*(3^Bk + 1)/2
    + (nk + odd(Ck) -2*nk*odd(Ck))*Ck
    + (odd(Ck) + Ck -2*nk*odd(Ck))/2
Ck+1 =                                    ---  (3^Bk + 1)/2 = 1 + sum[0<=i<k](3^Bi)
    + (1 - odd(Ck))*nk*3^Bk
    + nk*sum([0<=i<k](3^Bi)
    + (nk + odd(Ck) -2*nk*odd(Ck))*Ck
    + (odd(Ck) + Ck -2*nk*odd(Ck))/2
```

```
Ck+1 =                                    ---  extract pair from division
    + (1 - odd(Ck))*nk*3^Bk
    + nk*sum([0<=i<k](3^Bi)
    + (nk + odd(Ck) -2*nk*odd(Ck))*Ck
    - nk*odd(Ck)
    + (odd(Ck) + Ck)/2
Ck+1 =                                    ---  extract pair from division
    + nk*sum([0<=i<k+1-odd(Ck)](3^Bi)
    + (nk + odd(Ck) -2*nk*odd(Ck))*Ck
    + (odd(Ck) + Ck)/2
    - nk*odd(Ck)
Ck+1 =                                    ---  sum(3^k)[0<=k<n] = (3^n - 1) / 2
    + (nk*3^(k+1-odd(Ck)) -nk)/2
    + (nk + odd(Ck) -2*nk*odd(Ck))*Ck
    + (odd(Ck) + Ck)/2
    - nk*odd(Ck)
```

`g_k(n) = 2^k*f_k(n)`

Let's define `g_k(n)` as the following succession:

```
g_k(n) = 2^k * f_k(n)
```

That turns the reduction condition `f_k(n)=1` into:

```
g_k(n) = 2^k f_k(n) = 2^k
```

Also by construction, being `f_k` a positive natural number:

```
g_k(n) >= 2^k
```

Recall that Ok the oddity of f_k(n).

```
Ok = Odd(fk(n)) = bin_k(gk(n))
```

Where bin_k is the kth binary bit (the one with weight 2^k, so starting at k=0)

Which is the 0th bit of f_k(n) (considering 0 the first one). Then Ok is also the kth bit of g_k(n)

This leads to the following formula

```
g0(n) = n * 2^0 = n
gk+1(n) = 2^k+1 fk+1(n)

    Ok==true
    = (3*fk(n) +1)*2^k
    = 2^k * 3*fk(n) + 2^k
    = 3*gk(n) + 2^k

    Ok==false
```

```
    = fk(n)*2^k
    = gk(n)
```

In summary:

```
gk+1(n) =
    gk(n); when no Ok
    3*gk(n) + 2^k; when Ok
```

Unified

```
gk+1(n) = gk(n) + 2 * Ok(n) * gk(n) + 2^k * Ok(n)
```

Theorem: All powers of 2 (2^p) converge in p steps: gk(2^k) = 2^k

```
By definition: n converges if exists k such that gk(n) = 2^k, so the theorem can be expresse

g0(2^k) = 2^k

gk(2^p) = 2^p => gk+1(2^p) = 2^p, for k<p ?

Ok(2^p) = bit_k(gk(2^p)) = bit_k(2^p) = 0 [k<p]

gk+1(2^p)
    = gk(2^p) + 2 * Ok(2^p) * gk(2^p) + 2^k * Ok(2^p)  [Unified for k=k, n=2^p]
    = gk(2^p)   [Ok(2^p)=bit_k(gk(2^p)) if p>k]
    = 2^p
para k=p-1 [se cumple k<p]
gk+1(2^p) = gp(2^p) = 2^p  qvd
```

Curiosity: what happens with the following iterations

```
gp(2^p) = 2^p =>  Op(2^p) = bit_p(2^p) = 1
gp+1(2^p) =
    = gp(2^p) + 2 * Op(2^p) * gp(2^p) + 2^p * Ok(2^p)
    = gp(2^p) + 2 * gp(2^p) + 2^p      [ Op(2^p) = 1 ]
    = 2^p + 2 * 2^p + 2^p      [ gp(2^p) = 2^p ]
    = 2^p + 2^p+1 + 2^p
    = 4*2^p = 2^p+2

Op+1(2^p) = bit_p(gp+1(2^p)) = bit_p(2^p+2) = 0

gp+2(2^p) =
    = gp+1(2^p) + 2 * Op+1(2^p) * gp+1(2^p) + 2^p+1 * Ok+1(2^p)
    = gp+1(2^p) [ Op+1(2^p)=0 ]
    = 2^p+2  <- converges again
```

Theorem: If the kth iteration of a number is a power of 2, sequence converges
gk(n) = 2^p => exists a r | gr(n) = 2^r

```
gk(n) = 2^p
```

```
gk+1(n) =
    = gk(n) + 2 * Ok(n) * gk(n) + 2^k * Ok(n)
    = 2^p + 2 * Ok(n) * gk(n) + 2^k * Ok(n)




    gp+1(2^p+1) =
        = gp(2^p+1) + 2 * Op(2^p+1) * gp(2^p+1) + 2^p * Op(2^p+1) [k=p, n=2^p+1]
        = gp(2^p+1) [Op(2^p+1)=0]
        = gp(2^p+1)

p=0
    g0(2^0) = 1 = 2^0

p=1 (2^1=2, k0=0)
    g1(2^1) =
        = g0(2) + 2 * O0(2) * g0(2) + 2^1 * O0(2) [k=0, n=2]
        = g0(2) [O0(2)=0]
        = 2 [g0(2) = 2
        = 2^1]

p>0
suposing that gp(2^p) = 2^p demonstrate that gp+1(2^p+1) = 2^p+1
    gp+1(2^p+1) =
        = gp(2^p+1) + 2 * Op(2^p+1) * gp(2^p+1) + 2^p * Op(2^p+1) [k=p, n=2^p+1]
        = gp(2^p+1) [Op(2^p+1)=0]
        = gp(2^p+1)



    gp+1(2^p) =
        = 3*gp(2^p) + 2^p  [Op=1]
        = 3*2^p + 2^p
        = 4*2^p
        = 2^p+2
        Op+1 = 0
    gp+2(2^p) =
        = 2^p+2 -> converges again
    gp+1(2^p+1) =
        = gp(2^p+1) +  2*Op(2^p+1) * gp(2^p+1) + 2^p * Op(2^p+1)
```