2 is always a primitive root (2^power is only congruent to 1 mod n when power is a multiple of phi(n)) mod a power of 3.

See the first post and the linked lifting lemma <https://math.stackexchange.com/questions/594782/2-is-a-primitive-root-mod-3h-for-any-positive-integer-h>

The implication of this is that 4^n/3 is congruent to 1 mod n when n is a power of 3. Normally this power would be 2/3n since that’s phi(n) but 4 is the square of the primitive root 2.

For a given power of 3 modulus this means that there will be 3 possible candidates for the power of 4 that maps to index 0 under 5n+1. These will be 4^(exponent that maps to 0 in the modulus/3), 4^that exponent + the modulus/3, and 4^that exponent + 2\*modulus/3) since these all share the congruence to 0 under 5n+1 in modulus/3. This property lets us represent this exponent as a base 3 number.

For example, the sequence 2,0,0,1 tells us that the exponent of the power of 4 congruent to 0 after 5n+1 mod 3^5 is 2\*3^0 + 0\* 3^1 + 0\*3^2 + 1\*3^3 = 29. Every power of 4 that becomes divisible by 243 by doing 5n+1 must have an exponent congruent to 29 mod 243.

A similar property applies for powers of 3 and their divisibility by powers of 2 after 5n+1.

2 is not like the other primes, its powers don’t have primitive roots (except for 4, which has just 3, but should have 2 roots). But the lifting lemma still applies (despite it saying it doesn’t work).

We know that 3^2 is congruent to 1 mod 8 but not 16.

If 3^n is congruent to 1 mod 2^k, and not congruent to 1 mod 2^(k+1), we can write 3^n as 2^k \* a +1, where a is not divisible by 2 (otherwise condition 2 is not true). If we square this we get a number of the form 2^(2k) + **2\*2^k \*a +1**. Looking at this number mod 2^(k+1) we get remainder 1, and looking at it mod 2^(k+2) we do not get remainder 1 since a is not divisible by 2.

This serves to inductively prove that 3^n/4 is congruent to 1 mod n where n is a power of 2.

This means a similar string to the one above can specify the power of 3 that maps to 0 under 5n+1 mod each power of 2. But since 2 has that slightly strange divisibility property the string starts with the powers of 8. For example, the string 1 means that 3^1 maps to 0 mod 8, the string 101 means that 3^5 maps to 0 mod 32, etc…

If you can show that every number in a collatz-like problem descends below itself, that inductively means that every number converges. This, along with these strings lets us make an algorithm for confirming the 5n+1 holdout problem. If the exponent represented by the strings is greater then the number of digits (might be a +1 in here, +2 in the case of 2) (also a ratio of logs adjustment between the bases of 4 and 3) that means that the number in question, and any other number mapping to the same power of the other base descends below itself in the next step. Numbers that don’t descend below themselves are very rare, in my first example with the base 3 string, we can see that it starts with 200, so 4^2 ends up being divisible by 3^4, so it didn’t shrink because there was a sufficiently long string of zeros. But long enough strings of zeros will become super rare the further you go, since the exponent on 4 will grow by orders of magnitude, and the exponent on the next power of 3 will grow linearly. Should one of these cases be encountered, you would have to trace the trajectory of that number, which could be done by looking up the other string and finding the congruence of the new exponent. You would also have to worry about the equivalence classes of the skipped powers, like when you skip 3^3 for 4^2, you’d have to verify that 4^5 is larger than it, or also check 4^5.

The majority of the work for confirming the numbers will be in getting these strings, which can be done by using fast modular exponentiation taking log(log(n)) time for every element in the string you generate, but there are log(log(n)) elements, bringing the time up to log(log(n))^2. The size of the numbers in each modulus also grows logarithmically with the size of the exponent, bringing the runtime to log(log(n))^3.