David Rackl

Cycles in the Collatz sequence An approach using 2-adic arithmetic

BACHELOR'S THESIS

DEGREE PROGRAM
Technical Mathematics

University of Klagenfurt
Department of Mathematics

ADVISOR
Dr. Benjamin Hackl
Department of Mathematics
University of Klagenfurt

Klagenfurt, September 22, 2021

Abstract

The Collatz conjecture has earned itself the reputation of being one of the hardest problems in modern mathematics. While it is relatively easy to define and understand, a rigorous proof has eluded even some of the most talented mathematicians for almost a century. It is unclear when the Collatz problem was first introduced to the world of mathematics. Perhaps, because of this reality it has also acquired a few other names such as Syracuse problem, Hasse's Algorithm, Kakutani's Problem and Ulam's problem. According to Lagarias [1] the first publication on the problem dates back to 1963, although it appears to have circulated around the mathematical community for some time already at that point. Some speculation even dates it back as far as the 1930's.

In this thesis we discuss the problem itself, some results related to it and the possibility of cycles in the Collatz sequence. To achieve this, we will be using an accelerated Collatz sequence, which may be obtained using 2-adic arithmetic. The main result of this thesis will be a test to check whether cycles can exist in this accelerated sequence for any given length, as well as a resulting bound for minimum cycle length.

Contents

1 Introduction

Throughout this thesis, we take $\mathbb{N}$ to be the set of positive integers.

In this thesis we will investigate the Collatz conjecture and more specifically cycles in the corresponding Collatz sequence using 2-adic arithmetic. However, before we begin studying the Collatz conjecture we must define some basic concepts and formulate the conjecture itself. Firstly, we start by defining the Collatz sequence in its original form.

Definition 1 (Collatz sequence).
The Collatz sequence for a positive starting integer c_0 is defined as

$$c_{n+1} = \begin{cases} 3c_n + 1 & c_n \equiv 1 \mod 2 \\ \frac{c_n}{2} & c_n \equiv 0 \mod 2 \end{cases}.$$

Now let us take a look at some short examples to get an idea of how this sequence behaves. Set c_0 to 13 and begin iterating under the above rule. We get the sequence

$$(13, 40, 20, 10, 5, 16, 8, 4, 2, 1, 4, 1, 4, \ldots).$$

We can stop iterating after the second occurrence of 4 as the sequence has entered a cycle. This is visualized as a digraph in Figure 1.1. We can do this for some other numbers and we would find that the sequence enters the same cycle as in the above example. Now let us take a look at some larger numbers to see if this behavior continues. Since these calculations can be very long, some examples are visualized in Figures 1.2 and 1.3.

$$\begin{tikzcd}
& & & & & & & & 2 \arrow[dr] & \\
13 \arrow[r] & 40 \arrow[r] & 20 \arrow[r] & 10 \arrow[r] & 5 \arrow[r] & 16 \arrow[r] & 8 \arrow[r] & 4 \arrow[ur] & & 1 \arrow[ll, bend left]
\end{tikzcd}$$

Figure 1.1: Sequence digraph for 13

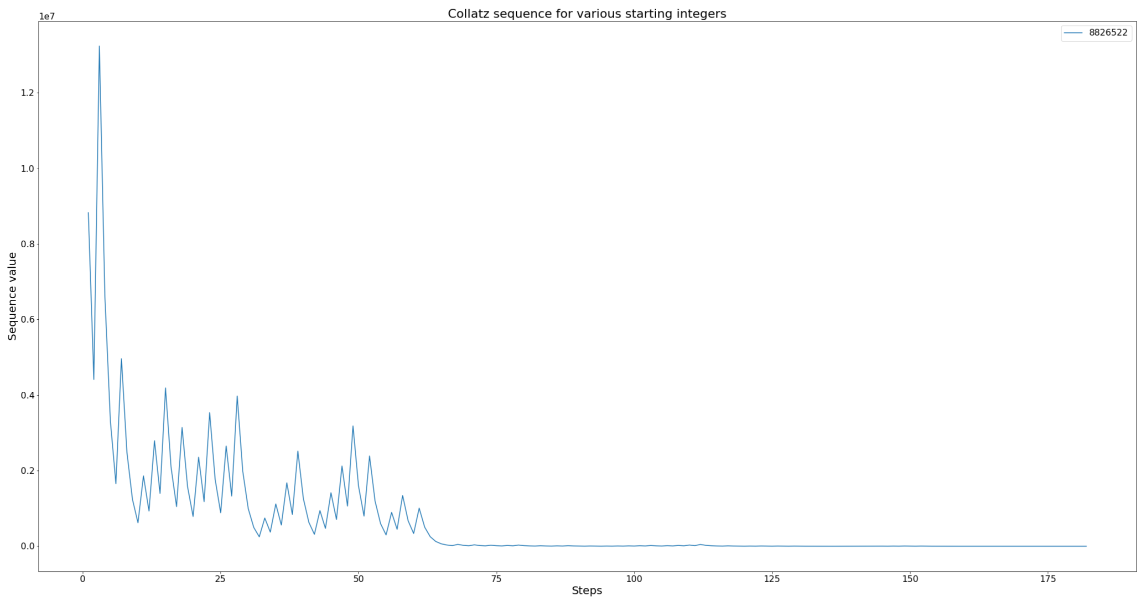

Figure 1.2: Sequence graph for 8826522

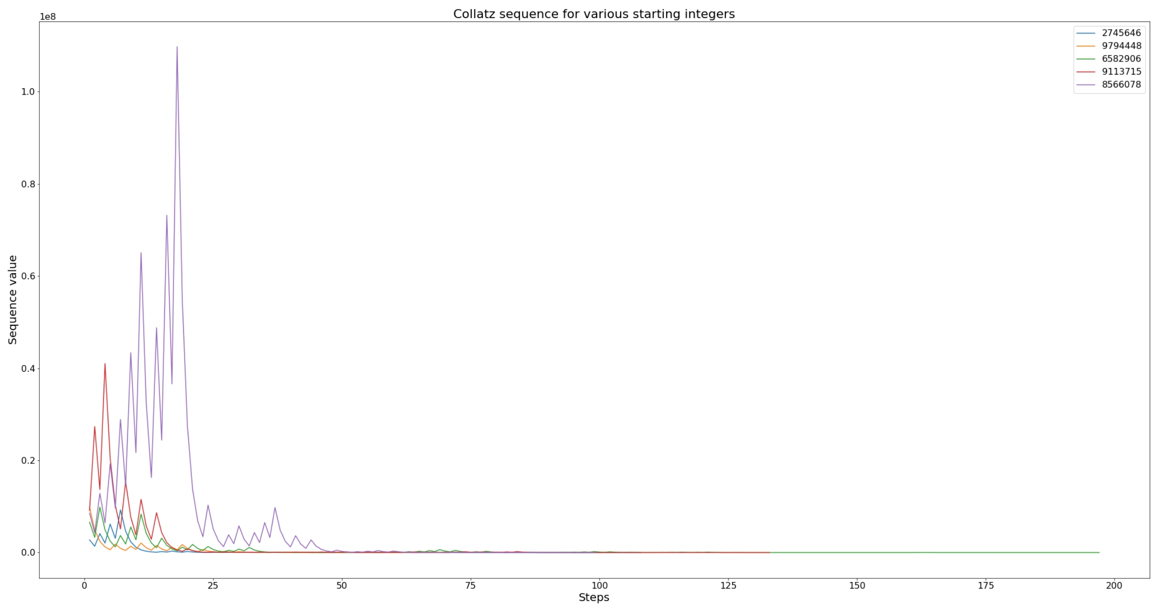

Figure 1.3: Sequence graph for various starting integers

Again we observe the same behavior—after a finite number of steps the sequence enters the exact same cycle: $(1, 4, 2)$—this is called the trivial cycle.

$$\begin{tikzcd}
 & 4 \arrow[dl] \\
2 \arrow[dr] & \\
 & 1 \arrow[uu, bend right]
\end{tikzcd}$$

Figure 1.4: The trivial cycle

This may lead us to conjecture that we get this behaviour for every possible positive starting integer, which is exactly what the Collatz conjecture formally states. If we put this idea into more formal terms, we get

Conjecture 1 (Collatz).
Let c_n denote the Collatz sequence with starting integer c_0. Then it holds that

$$\forall c_0 \in \mathbb{N} : \exists n \in \mathbb{N} : c_n = 1.$$

The above definition is equivalent to our observation as 1 *is contained in* $(1, 4, 2)$.

While this problem may not look very difficult at first glance it is indeed one of the hardest problems in modern mathematics. Countless mathematicians have studied it and discovered various different results about it [1, 2]. However, a rigorous proof of the conjecture is still missing. In the next chapter we will take a look at some of the most famous results as well as some results that will be relevant for the topic of this thesis.

2 Known results

In this chapter we will present some relevant results on the Collatz conjecture. However, as most of these results have complicated proofs, we will only present the results itself without going into technical detail. For this and further discussion of the results we refer the reader to the original publications. We will begin this chapter by presenting one of the most prominent results on the Collatz conjecture at this time, which has only been discovered recently. Although only a probabilistic result, it is likely as close as we can get without delivering a rigorous proof of the conjecture itself. We define $c_{\min}(N) := \inf_{n\in\mathbb{N}} c_n(N)$ with $c_0(N) := N$, i.e. we define define $c_{\min}(N)$ to be the minimal value of the collatz orbit of N.

Theorem 1 (Almost all Collatz orbits attain almost bounded values [3]).
Let $f : \mathbb{N} \to \mathbb{R}$ be any function with $\lim_{N\to\infty} f(N) = +\infty$. Then we have $c_{\min}(N) < f(N)$ for almost all $N \in \mathbb{N}$.

Our next result concerns itself with the computational verification of the Collatz conjecture.

Theorem 2 (Convergence verification of the Collatz conjecture [4]).
The Collatz conjecture has been verified for all starting integers up to 2^{68}.

The next two results focus specifically on cycles in the Collatz sequence.

Theorem 3 (Minimal cycle length [5]).
The length of any nontrivial cycle in Collatz sequence is at least 1,027,712,276.

For an extensive overview of the Collatz conjecture we refer the reader to [1, 2].

We will close this chapter with a major result on a close relative of the Collatz conjecture. Consider the following natural generalization of the Collatz conjecture as introduced by Conway [6].

Definition 2 (Collatz function).
Let P be a positive integer and let $a_0, a_1, ..., a_{P-1}$ and $b_0, b_1, ..., b_{P-1}$ be non-negative rational numbers s.t. $g(n)$ given by

$$g(n) = a_i n + b_i \quad \text{if} \quad n \equiv i \mod P$$

is always an integer. Then $g(n)$ is called a Collatz function.

In this notation the Collatz function that yields the original Collatz sequence is given by $P = 2$ and

$$\begin{aligned} a_0 &= \frac{1}{2} & b_0 &= 0, \\ a_1 &= 3 & b_1 &= 1. \end{aligned}$$

With the concept of Collatz functions we can formulate a natural generalization of the Collatz Problem.

Definition 3 (Natural generalization of the Collatz Problem).
Let g be a Collatz function and let n_0 be a positive starting integer. Does the sequence of iterates g^k

reach 1 *for every n_0, i.e. does the statement*

$$\forall n_0 \in \mathbb{N} : \exists k \in \mathbb{N} : g^k(n_0) = 1$$

hold?

It is important to note that we abstained from calling the above statement a conjecture, as the class of Collatz functions contains elements for which this statement can be easily proven. One such function is given by choosing $P = 1, a_0 = 0, b_0 = 1$, in which case $g^k(n_0) = 1$ for every k and every n_0. However, for most functions, this problem is very difficult. We have

Theorem 4 (Undecidability of the generalized Collatz Problem [7]).
The generalized Collatz Problem is undecidable.

As discussed in [7], the generalized Collatz conjecture is strongly related to the universal halting problem and in terms of computational complexity, it is even located in the same layer of the arithmetical hierarchy. For the interested reader we provide the following result.

Theorem 5 (Complexity of the generalized Collatz Problem [7]).
The generalized Collatz Problem is Π_2^0-complete.

3 The reduced Collatz sequence

Moving forward, it will be useful to work with an alternate version of the Collatz sequence. We introduce this version mainly for the purpose of obtaining a closed form of the i-th sequence element. In order to achieve this we will combine one $3x + 1$-step with all of its subsequent $x/2$-steps. This leads us to the following definition.

Definition 4 (Reduced/Accelerated Collatz Sequence)**.**
The reduced Collatz sequence for a positive starting integer r_0 (with $r_0 \equiv 1 \mod 2$) is defined as

$$r_{n+1} = \frac{3r_n + 1}{2^{v_2(3r_n+1)}},$$

where $v_2(n)$ is the 2-adic valuation on $\mathbb{Z}$.

This sequence is called reduced since it does not contain all elements of the original sequence—only the odd ones. The 2-adic valuation removes all even elements. We can see this reduction by plotting both the reduced and the standard sequence in Figure 3.1.

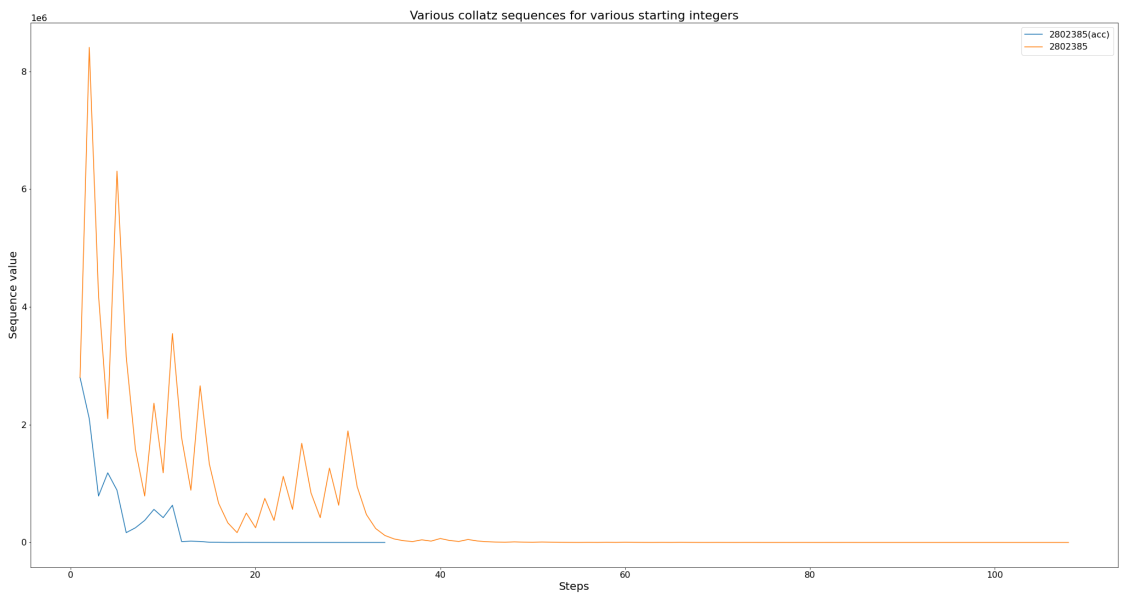

Figure 3.1: The reduced and standard sequence for 2802385

It is important to note that it does not matter much which sequence we study— the reduced sequence is merely a subsequence of the standard sequence. More precisely, it is the subsequence formed by the odd entries of the original sequence. Results from the standard sequence can be translated into results about the reduced sequence and vice versa. We can also just copy our original definition of the Collatz conjecture since 1 is an odd number. Figure 3.2 illustrates several accelerated Collatz sequences.

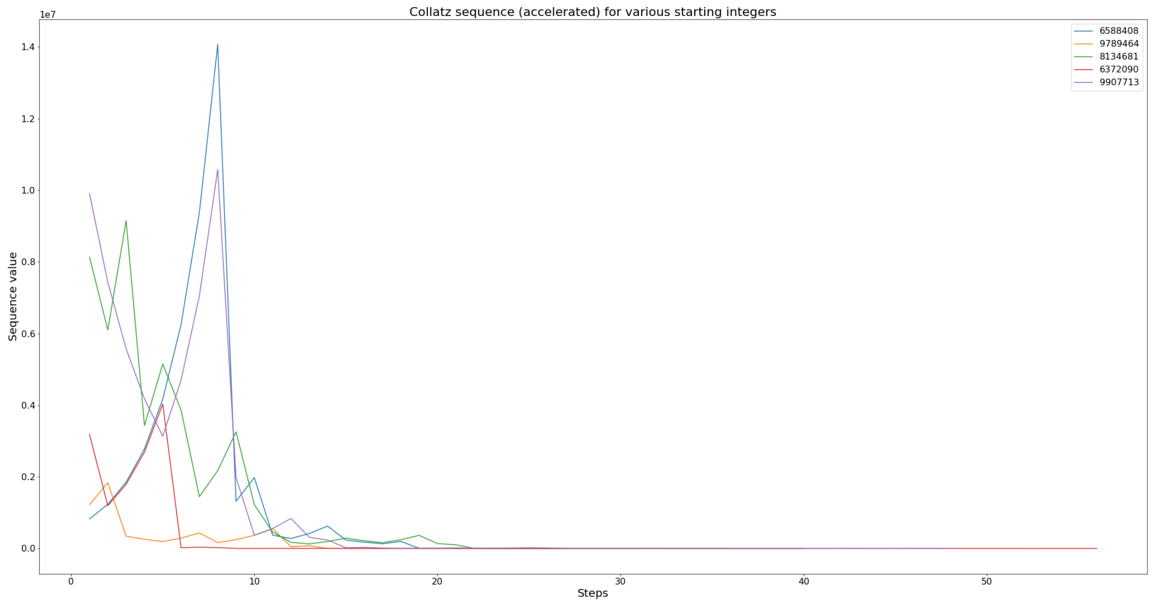

Figure 3.2: The accelerated sequence for various (odd) starting integers

We can see that the sequences show similar behavior as the standard ones. However, there is one detail we have to change — the trivial cycle $(1, 4, 2)$ is now reduced to the cycle (1). For the reduced sequence, we also have a known lower bound for cycle length.

Theorem 6 (Minimal cycle length [8]).
The length of any nontrivial cycle in the Reduced/Accelerated Collatz sequence is at least 17,985.

3.1 Basic properties of the reduced Collatz sequence

We begin this section by establishing the closed form of the reduced sequence from Definition 3.

Theorem 7 (Closed form of the reduced sequence).
For any given $n \in \mathbb{N}$ the n-th element of the reduced Collatz sequence can be written as

$$r_n = \frac{3^n r_0}{2^{s(r_0,n,1)}} + \sum_{i=1}^{n} 3^{n-i} 2^{-s(r_0,i,n)},$$

where $s(r_0, i, n)$ is defined as

$$\sum_{j=i}^{n} \nu_2(3r_{j-1} + 1).$$

Proof.
This result can be obtained simply by iterating the reduced sequence.

$$\begin{aligned} r_1 &= \frac{3r_0}{2^{\nu_2(3r_0+1)}} + \frac{1}{2^{\nu_2(3r_0+1)}} \\ r_2 &= \frac{3^2 r_0}{2^{\nu_2(3r_0+1)+\nu_2(3r_1+1)}} + \frac{3}{2^{\nu_2(3r_0+1)+\nu_2(3r_1+1)}} + \frac{1}{2^{\nu_2(3r_1+1)}} \\ &\vdots \end{aligned}$$

After n steps this will give us the desired result. □

Corollary 7.1.
If the reduced Collatz sequence is constant, then $\forall n \in \mathbb{N} : r_n = 1$.

Proof.
If the sequence is constant then it must hold that

$$\begin{aligned} r_n &= r_{n+1} \\ r_n &= \frac{3r_n + 1}{2^{\nu_2(3r_n+1)}} \\ 2^{\nu_2(3r_n+1)} &= \frac{3r_n+1}{r_n} \\ 2^{\nu_2(3r_n+1)} &= 3 + \frac{1}{r_n}. \end{aligned}$$

Since $2^{\nu_2(3r_n+1)}$ is an integer it follows that $r_n = 1$. □

This corollary establishes that the trivial cycle is the only cycle of length 1. We can take this concept further and apply it to cycles of any given length. This leads us to the following theorem.

Theorem 8.
If the reduced Collatz sequence is k-cyclic (i.e., has a cycle of length k) then r_0 can be written as

$$r_0 = \frac{1}{2^{s(r_0,1,k)} - 3^k} 2^{s(r_0,1,k)} \sum_{i=1}^{k} 3^{k-i} 2^{-s(r_0,i,k)},$$

where $s(r_0, i, k)$ is defined as in Theorem 7.

Proof.
Without loss of generality, we can assume that the sequence r is cyclic from the first element onwards. Otherwise we can define a new sequence $\bar{r}$ starting where the original sequence r entered a cycle. Then we have that

$$r_0 = r_k,$$

or using the closed form from Theorem 7,

$$r_0 = \frac{3^k r_0}{2^{s(r_0,1,k)}} + \sum_{i=1}^{k} 3^{k-i} 2^{-s(r_0,i,k)}.$$

Rearranging this a little gives

$$\begin{aligned}
r_0 &= \frac{3^k r_0}{2^{s(r_0,1,k)}} + \sum_{i=1}^{k} 3^{k-i} 2^{-s(r_0,i,k)} \\
r_0 - \frac{3^k r_0}{2^{s(r_0,1,k)}} &= \sum_{i=1}^{k} 3^{k-i} 2^{-s(r_0,i,k)} \\
2^{s(r_0,1,k)} r_0 - 3^k r_0 &= 2^{s(r_0,1,k)} \sum_{i=1}^{k} 3^{k-i} 2^{-s(r_0,i,k)} \\
(2^{s(r_0,1,k)} - 3^k) r_0 &= 2^{s(r_0,1,k)} \sum_{i=1}^{k} 3^{k-i} 2^{-s(r_0,i,k)} \\
r_0 &= \frac{1}{2^{s(r_0,1,k)} - 3^k} 2^{s(r_0,1,k)} \sum_{i=1}^{k} 3^{k-i} 2^{-s(r_0,i,k)},
\end{aligned}$$

which yields the desired result. □

It is important to note that we can generalize this result to any cycle element by simply shifting the sequence.

Corollary 8.1. *If the reduced Collatz sequence is k-cyclic (i.e., has a cycle of length k) then any cycle element r can be written as*

$$r = \frac{1}{2^{s(r,1,k)} - 3^k} 2^{s(r,1,k)} \sum_{i=1}^{k} 3^{k-i} 2^{-s(r,i,k)}.$$

Proof.
Shift the sequence such that the first element is the desired one and apply Theorem 7. □

Using this form we can now analyze the 2-adic reductions that can occur in cycles.

3.2 2-adic reductions in cycles

In this section we will take a look at what types of 2-adic reductions can occur if the sequence is cyclic. We will also take a look at the implications this has for the sequence elements themselves. We will start off using the identity we proved for cyclic elements in the last chapter. Using this identity we can prove the following theorem.

Theorem 9.
Assume that the odd positive integer r_0 starts a cycle of length k in the reduced Collatz sequence. Then we have

$$\begin{aligned}
r &\equiv 1 \mod 6 \implies v_2(3r_{k-1}+1) \quad \textit{even}, \\
r &\equiv 5 \mod 6 \implies v_2(3r_{k-1}+1) \quad \textit{odd}.
\end{aligned}$$

Proof.
Since we have a cyclic sequence the following holds by Corollary 2.

$$r = \frac{1}{2^{s(r,1,k)} - 3^k} 2^{s(r,1,k)} \sum_{i=1}^{k} 3^{k-i} 2^{-s(r,i,k)}$$

For the next part it will be useful to eliminate fractions from this equation. We can achieve this by simply multiplying the equation by $2^{s(r,1,k)} - 3^k$ and multiplying $2^{s(r,1,k)}$ into the sum. This gives us the equation

$$(2^{s(r,1,k)} - 3^k) r = \sum_{i=1}^{k} 3^{k-i} 2^{s(r,1,k)-s(r,i,k)}.$$

Since $s(r,1,k) > s(r,i,k)$ (this follows directly from the definition of s in Theorem 7 as it simply denotes a sum of 2-adic valuations) we have successfully eliminated all fractions from the equation. Since this is now an integer equation we can apply modular arithmetic to it. In this particular case it is most useful to work modulo 6. Taking this whole equation modulo 6 gives

$$(2^{s(r,1,k)} - 3^k) r \equiv 3^{k-i} + 2^{s(r,1,k)-s(r,k,k)} \mod 6.$$

To simplify this further we need the following two observations. First, note that

$$\forall n \in \mathbb{N} : 3^n \equiv 3 \mod 6.$$

This is easy to check as $3^1 \equiv 3 \mod 6$ and $3^2 = 3 \cdot 3 = 9 \equiv 3 \mod 6$ and so on. The second observation concerns powers of 2. We have:

$$\forall n \in \mathbb{N} : 2^n \equiv \begin{cases} 2 \mod 6 & \text{if } n \text{ is odd,} \\ 4 \mod 6 & \text{if } n \text{ is even.} \end{cases}$$

This is again easy to check as $2^1 \equiv 2 \mod 6$, $2^2 \equiv 4 \mod 6$, $2^3 = 2^2 \cdot 2 = 8 \equiv 2 \mod 6$, and so on. Using these observations we can simplify our previous equation to find

$$\begin{aligned} (2^{s(r,1,k)} - 3^k) r &\equiv 3^{k-i} + 2^{s(r,1,k)-s(r,k,k)} \mod 6, \\ (2^{s(r,1,k)} - 3) r &\equiv 3 + 2^{s(r,1,k)-s(r,k,k)} \mod 6. \end{aligned}$$

Since powers of 2 can take 2 distinct values $\mod 6$ we need to tackle this next step using four different cases, namely

$$\begin{aligned} 2^{s(r,1,k)} \equiv 2 \mod 6 \quad &\land \quad 2^{s(r,1,k)-s(r,k,k)} \equiv 2 \mod 6, \\ 2^{s(r,1,k)} \equiv 2 \mod 6 \quad &\land \quad 2^{s(r,1,k)-s(r,k,k)} \equiv 4 \mod 6, \\ 2^{s(r,1,k)} \equiv 4 \mod 6 \quad &\land \quad 2^{s(r,1,k)-s(r,k,k)} \equiv 2 \mod 6, \\ 2^{s(r,1,k)} \equiv 4 \mod 6 \quad &\land \quad 2^{s(r,1,k)-s(r,k,k)} \equiv 4 \mod 6. \end{aligned}$$

We can now obtain some information about r in each of these cases. For the first cases we have

$$\begin{aligned} (2^{s(r,1,k)} - 3) r &\equiv 3 + 2^{s(r,1,k)-s(r,k,k)} \mod 6 \\ (2-3) r &\equiv 3 + 2 \mod 6 \\ -r &\equiv 5 \mod 6 \\ r &\equiv 1 \mod 6. \end{aligned}$$

Repeating this procedure for all four cases gives us

$$\begin{aligned}
2^{s(r,1,k)} &\equiv 2 \mod 6 \quad \wedge \quad 2^{s(r,1,k)-s(r,k,k)} \equiv 2 \mod 6 \implies r \equiv 1 \mod 6,\\
2^{s(r,1,k)} &\equiv 2 \mod 6 \quad \wedge \quad 2^{s(r,1,k)-s(r,k,k)} \equiv 4 \mod 6 \implies r \equiv 5 \mod 6,\\
2^{s(r,1,k)} &\equiv 4 \mod 6 \quad \wedge \quad 2^{s(r,1,k)-s(r,k,k)} \equiv 2 \mod 6 \implies r \equiv 5 \mod 6,\\
2^{s(r,1,k)} &\equiv 4 \mod 6 \quad \wedge \quad 2^{s(r,1,k)-s(r,k,k)} \equiv 4 \mod 6 \implies r \equiv 1 \mod 6.
\end{aligned}$$

Let us take a look at the first case. If we have $2^{s(r,1,k)} \equiv 2 \mod 6$ and $2^{s(r,1,k)-s(r,k,k)} \equiv 2$ then $s(r,k,k)$ must be an even number by our observation about powers of 2 before. The same logic applies to all other cases as well. This gives

$$\begin{aligned}
r \equiv 1 \mod 6 &\implies s(r,k,k) \quad \text{even},\\
r \equiv 5 \mod 6 &\implies s(r,k,k) \quad \text{odd}.
\end{aligned}$$

Now we need one last observation. By definition of s we have

$$s(r,k,k) = \sum_{j=k}^{k} v_2(3r_{j-1}+1) = v_2(3r_{k-1}+1).$$

So $s(r,k,k)$ is exactly the last 2-adic valuation to occur in the cycle. Combining this with our results from above completes the proof. □

This theorem gives us some insight into how cyclic sequences behave under iteration. We can also extract a test to check if a number may be contained in a cycle from this theorem. For this we take any odd number q and iterate one step in the reduced sequence to obtain its successor q_{next}. Now we check what the 2-adic valuation of $3q+1$ is. If it is odd then q_{next} must be congruent to 5 modulo 6. If q_{next} is congruent to 1 modulo 6 then we know that q cannot be contained in a cycle by Theorem 8. We can also do a similar check if the 2-adic valuation of $3q+1$ is even. It is important to note that if a number passes this test we do not get any result — it could be contained in a cycle, or it could not.

This result concludes our investigation into 2-adic reductions in cycles. Next we wish to investigate the existence of cycles for any given length. Our goal will be to obtain a simple test to check if a cycle can exist for a given length.

3.3 Existence of cycles with given length

Before we begin to work on test to check if a cycle of any given length can exist, it is interesting to see that we can also devise such a method using some of the results of our previous two sections. To demonstrate the core idea we will begin by proving no cycles of length 2 can exist.

Theorem 10.
The reduced Collatz sequence can not contain any cycles of length 2.

Proof.
We will again use our identity for elements in cyclic sequences from Theorem 7. So for any odd number r_0 we have

$$(2^{s(r_0,1,k)} - 3^k)r_0 = \sum_{i=1}^{k} 3^{k-i} 2^{s(r_0,1,k)-s(r_0,i,k)}.$$

If the cycle has length $k = 2$ this equation takes a much simpler form. If we call the first 2-adic valuation in the cycle z_1 and the second z_2 we have

$$(2^{z_1+z_2} - 3^2)r_0 = 3 + 2^{z_1}.$$

We can now use the fact that $r_0 \geq 1$ since the reduced Collatz sequence is only defined for positive starting integers.

$$3 + 2^{z_1} = (2^{z_1+z_2} - 3^2)r_0 \geq 2^{z_1+z_2} - 3^2$$

We can now analyze the resulting inequality a little more. Using some basic transformations we obtain

$$\begin{aligned} 3 + 2^{z_1} &\geq 2^{z_1+z_2} - 3^2 \\ 3 + 3^2 &\geq 2^{z_1+z_2} - 2^{z_1} \\ 2^{z_1}(2^{z_2} - 1) &\leq 12. \end{aligned}$$

Since $2^{z_2} > 1$ we get our first condition, $2^{z_1} \leq 12$. Applying the same logic to 2^{z_1} gives $2^{z_2} \leq 13$. So we have $z_1 \in \{1, 2, 3\}$, $z_2 \in \{1, 2, 3\}$.
We can also obtain a set of constraints for r_0 by using a similar argument. Since $3 + 2^{z_1} > 0$, we have $(2^{z_1+z_2} - 3^2) > 0$ and since both are integers and $2^n \neq 9$ for all positive integers n, we also have $(2^{z_1+z_2} - 3^2) > 1$. This gives us

$$3 + 2^{z_1} = (2^{z_1+z_2} - 3^2)r_0 \geq r_0.$$

Since we already have $z_1 \in \{1, 2, 3\}$ we get

$$\begin{aligned} &3 + 2^3 \geq 3 + 2^{z_1} \geq r_0, \\ &r_0 \leq 11. \end{aligned}$$

We can now try all possible combinations for z_1, z_2 and r_0. We will find that none satisfy our condition

$$(2^{z_1+z_2} - 3^2)r_0 = 3 + 2^{z_1}.$$

□

We could have also used the same trick for length 3, length 4 and so on. This method will quickly become very inefficient as an increase in length also causes an increase in both the number of variables and the corresponding bounds for them. In other words for large cycle lengths this test turns into a combinatorial nightmare.

We will now take a different approach to obtain a simpler criterion. For this it will be useful to characterize cycles in a different way. We will now use the observation that if r_0 is an integer starting a Collatz cycle of length k, then

$$\prod_{i=0}^{k-1} \frac{r_{i+1}}{r_i} = 1.$$

It is important to note that this is equivalent to our previous characterization $r_0 = r_k$, since

$$\prod_{i=0}^{k-1} \frac{r_{i+1}}{r_i} = \frac{r_1}{r_0} \cdot \frac{r_2}{r_1} \cdot \frac{r_3}{r_2} \cdots \frac{r_k}{r_{k-1}} = \frac{r_k}{r_0},$$

which is equal to 1 if and only if $r_0 = r_k$.

Next it will be useful to find an explicit form for $\frac{r_{i+1}}{r_i}$. We can get one by substituting the definition of the reduced sequence,

$$\frac{r_{i+1}}{r_i} = \frac{\frac{3r_i+1}{2^{v_2(3r_i+1)}}}{\frac{r_i}{1}} = \frac{3r_i+1}{2^{v_2(3r_i+1)}r_i}.$$

This characterization allows us to get the following criterion for existence of cycles.

Theorem 11.
The integers r_0, r_1, ..., r_{k-1} form a cycle of length k in the reduced Collatz sequence if and only if

$$\sum_{i=0}^{k-1} \log_2\left(3 + \frac{1}{r_i}\right) = \sum_{i=0}^{k-1} v_2\left(3r_i + 1\right).$$

Proof.
By our considerations above we know a cycle exists if and only if

$$\prod_{i=0}^{k-1} \frac{r_{i+1}}{r_i} = 1.$$

It is useful to apply the base 2 logarithm to both sides to transform this condition. Doing so yields

$$\log_2\left(\prod_{i=0}^{k-1} \frac{r_{i+1}}{r_i}\right) = \log_2(1) \quad \Longleftrightarrow \quad \sum_{i=0}^{k-1} \log_2\left(\frac{r_{i+1}}{r_i}\right) = 0.$$

Next we will substitute our previous identity for $\frac{r_{i+1}}{r_i}$ and use some logarithm rules to get

$$\sum_{i=0}^{k-1} \log_2\left(\frac{3r_i+1}{2^{v_2(3r_i+1)}r_i}\right) = 0$$

$$\sum_{i=0}^{k-1} \log_2\left(\frac{3r_i+1}{r_i}\right) - \log_2\left(2^{v_2(3r_i+1)}\right) = 0$$

$$\sum_{i=0}^{k-1} \log_2\left(3 + \frac{1}{r_i}\right) - \sum_{i=0}^{k-1} v_2\left(3r_i + 1\right) = 0$$

$$\sum_{i=0}^{k-1} \log_2\left(3 + \frac{1}{r_i}\right) = \sum_{i=0}^{k-1} v_2\left(3r_i + 1\right),$$

which leaves us with the expression we were looking for. □

For the next part we will need to use that the Collatz conjecture has been verified for all starting integers up to 2^{68} [4]. Using this we want to obtain some bounds for the expression

$$\sum_{i=0}^{k-1} \log_2\left(3 + \frac{1}{r_i}\right),$$

which we encountered in our previous theorem. This gives us the following result.

Lemma 1.
Let r_0 be the starting value of a Collatz cycle of length k, then

$$k\log_2(3) = \sum_{i=0}^{k-1}\log_2(3) < \sum_{i=0}^{k-1}\log_2\left(3+\frac{1}{r_i}\right) < \sum_{i=0}^{k-1}\log_2\left(3+\frac{1}{2^{68}}\right) = k\log_2\left(3+\frac{1}{2^{68}}\right)$$

Proof.
We will start with the lower bound. Since $\frac{1}{r_i}$ is strictly positive and the logarithm is strictly increasing on $\mathbb{R}$ we have,

$$\sum_{i=0}^{k-1}\log_2\left(3+\frac{1}{r_i}\right) > \sum_{i=0}^{k-1}\log_2(3).$$

For the upper bound we will use the fact that $\frac{1}{r_i} < \frac{1}{2^{68}}$, which follows directly from the verification of the Collatz conjecture. This gives us

$$\sum_{i=0}^{k-1}\log_2\left(3+\frac{1}{r_i}\right) < \sum_{i=0}^{k-1}\log_2\left(3+\frac{1}{2^{68}}\right).$$

Combining the upper and lower bounds gives us the inequality chain we wanted. □

This lemma opens up the possibility to verify whether a cycle of length k is possible for any given $k \in \mathbb{N}$. It is important to note that we always have

$$\sum_{i=0}^{k-1} v_2(3r_i+1) \in \mathbb{N}$$

for any given cycle since 2-adic valuations may only take positive integer values. Before we continue, we rewrite the inequality chain from our previous lemma using intervals. This leaves us with

$$\sum_{i=0}^{k-1}\log_2\left(3+\frac{1}{r_i}\right) \in \left(\sum_{i=0}^{k-1}\log_2(3);\sum_{i=0}^{k-1}\log_2\left(3+\frac{1}{2^{68}}\right)\right).$$

This leads us to our next theorem, which gives us a practically computable condition for the existence of cycles in the reduced sequence.

Theorem 12.
A cycle in the reduced Collatz sequence of length k can exist if and only if

$$\left(k\log_2(3);k\log_2\left(3+\frac{1}{2^{68}}\right)\right) \cap \mathbb{N} \neq \emptyset.$$

Proof.
We will begin by using our existence condition

$$\sum_{i=0}^{k-1}\log_2\left(3+\frac{1}{r_i}\right) = \sum_{i=0}^{k-1} v_2(3r_i+1),$$

which we established earlier. Additionally we will use the fact that

$$\sum_{i=0}^{k-1} v_2(3r_i+1) \in \mathbb{N},$$

which we also established earlier. If we have that

$$\left(\sum_{i=0}^{k-1} \log_2(3) ; \sum_{i=0}^{k-1} \log_2\left(3+\frac{1}{2^{68}}\right)\right) \cap \mathbb{Z} = \emptyset,$$

then

$$\sum_{i=0}^{k-1} \log_2\left(3+\frac{1}{r_i}\right)$$

cannot take any value in $\mathbb{N}$ by our previous lemma. So if we want the identitiy

$$\sum_{i=0}^{k-1} \log_2\left(3+\frac{1}{r_i}\right) = \sum_{i=0}^{k-1} v_2(3r_i+1)$$

to hold then we must have that

$$\left(k\log_2(3) ; k\log_2\left(3+\frac{1}{2^{68}}\right)\right) \cap \mathbb{N} \neq \emptyset,$$

which is exactly the condition we wanted to establish. □

This theorem will give us a much easier condition to check than the one we devised at the beginning of this chapter, if we want to know if a cycle of length k exists for any given $k \in \mathbb{N}$. Notice that for a fixed $k \in \mathbb{N}$ we have

$$\sum_{i=0}^{k-1} \log_2(3) = k\log_2(3)$$

and

$$\sum_{i=0}^{k-1} \log_2\left(3+\frac{1}{2^{68}}\right) = \sum_{i=0}^{k-1} \log_2(3) + \left(\log_2\left(3+\frac{1}{2^{68}}\right) - \log_2(3)\right).$$

If we define

$$\epsilon := \log_2\left(3+\frac{1}{2^{68}}\right) - \log_2(3),$$

we then have that

$$\sum_{i=0}^{k-1} \log_2\left(3+\frac{1}{2^{68}}\right) = \sum_{i=0}^{k-1} \log_2(3) + \epsilon = k\left(\log_2(3) + \epsilon\right).$$

This gives us this us exactly what we were looking for—an easy test to check if a cycle exists for any given length.

Corollary 12.1.
A cycle in the reduced Collatz sequence of length k for given $k \in \mathbb{N}$ can exist if and only if

$$\lceil k\log_2(3)\rceil - k\log_2(3) < k\epsilon$$

Proof.
We start with the observation that if we want the open interval

$$\left(k\log_2(3) ; k\log_2\left(3+\frac{1}{2^{68}}\right)\right)$$

from Theorem 12 to contain an integer we must have that

$$k\left(\log_2(3)+\epsilon\right) > \lceil k\log_2(3)\rceil,$$

otherwise we would have that

$$\left(k\log_2(3)\,;k\log_2\left(3+\frac{1}{2^{68}}\right)\right) \subset \left[\lfloor k\log_2(3)\rfloor;\lceil k\log_2(3)\rceil\right],$$

which means the interval cannot contain any integers. This means a cycle can only exist if we have

$$\lceil k\log_2(3)\rceil - k\log_2(3) < k\epsilon,$$

otherwise the interval will not contain any integers which means that

$$\left(k\log_2(3)\,;k\log_2\left(3+\frac{1}{2^{68}}\right)\right) \cap \mathbb{N} = \emptyset.$$

Using Theorem 12 we get that no cycle of this length can exist. □

We now have a condition that we can easily check computationally. However, this test also comes with it's own set of limitations. Most importantly the bound $k\epsilon$ increases with the cycle length k. This means for large enough k ($k \geq 1/\epsilon$) this test will not provide any useful results. We can counteract this by verifying more numbers, which would decrease ϵ and therefore our bound $k\epsilon$. However, this will never produce a test that will work for arbitrarily large numbers.

Finally, we will put our newfound test in action. In the next chapter we will discuss an implementation using interval arithmetic and also provide a lower bound for the length cycles in the reduced sequence can have.

3.4 A bound for cycle length

We will start this section by proposing the following algorithm to numerically check our existence condition from the previous section.

Algorithm 1.
Remark: The cycle length k should be a positive integer, precisions should be specified in powers of 2.

```
1 : **input** cycle length $k$, starting precision $p$, maximum precision $p_{\max}$
2 : **output**
3 : **while** $p \leq p_{\max}$ **do**
4 :   **with** precision $p$ :
5 :     $\epsilon := \log_2(3+\frac{1}{2^{68}})-\log_2(3)$
6 :     **if** $\lceil k\log_2(3)\rceil - k\log_2(3) > k\epsilon$
7 :       **return**
8 :     **else**
9 :       $p \longleftarrow 2p$
```

This algorithm does not terminate unless the condition $\lceil k\log_2(3)\rceil - k\log_2(3) > k\epsilon$ is met or the maximum precision $p_{\max}$ is reached. The second condition is put in place since in practice it is only useful to check up to a certain precision, as eventually missing precision is unlikely to be the cause of failure of the condition. As mentioned previously, eventually the bound $k\epsilon$ will simply be too large to obtain proper results. In our case we will check up to a precision of 512 bits. To ensure the numerical validity, this algorithm was implemented with interval arithmetic using C++ with the MPFR and MPFI libraries. Using this approach all numbers up to 2^{35} were checked with no failures found. This gives us the final theorem of this thesis.

Theorem 13.
The reduced Collatz sequence cannot contain any cycles with length $k < 2^{35}$.

Proof.
This proof was done computationally using the implementation discussed above. The source for this implementation and additional details may be found in the appendix. □

4 Appendix

Below is the source code for the C++ implementation used to verify the cycle existence condition from Corollary 2 up to a given integer n. Required libraries are GMP, MPFR and MPFI.

```
#include <gmp.h>
#include <mpfr.h>
#include <mpfi.h>
#include <math.h>
#include <chrono>
#include <iostream>
#include <vector>

using namespace std::chrono;

int check_bound(const unsigned long &n, const mpfi_t &eps,
  const mpfi_t &log, mpfi_t &x, mpfi_t &x_eps, mpfi_t &diff,
  mpfi_t &x_ceil, mpfr_t &max_left, mpfr_t &max_right)
{

    mpfi_mul_ui(x, log, n);
    mpfi_mul_ui(x_eps, eps, n);

    mpfr_rint_ceil(max_left,&(x->left),MPFR_RNDD);
    mpfr_rint_ceil(max_right,&(x->right),MPFR_RNDU);

    mpfr_set(&(x_ceil->left),max_left,MPFR_RNDD);
    mpfr_set(&(x_ceil->right),max_right,MPFR_RNDU);

    mpfi_sub(diff,x_ceil,x);

    return (diff < x_eps);

}

int main()
{
    /* Start execution timing */

    auto start = high_resolution_clock::now();

    /* Set maximum iterations */

    unsigned long MAX = exp2(35);

    /* Set result variable */

    int result = 0;

    /* Set fail counter */

    unsigned int fails = 0;

    /* Set fail table */

    std::vector<unsigned long> f;

    /* Declare helper variables */

    mpfi_t help_1, help_2;

    /* 64-bit variable set */

    mpfi_t eps_64, log_64, x_64, x_eps_64, x_ceil_64, diff_64;
    mpfr_t max_left_64, max_right_64;

    mpfi_init2(eps_64, 64);
    mpfi_init2(log_64, 64);
    mpfi_init2(x_64, 64);
    mpfi_init2(x_eps_64, 64);
    mpfi_init2(x_ceil_64, 64);
    mpfi_init2(diff_64, 64);

    mpfr_init2(max_left_64, 64);
    mpfr_init2(max_right_64, 64);

    /* Compute eps and log2(3) in 64-bit */

    mpfi_init2(help_1,64);
    mpfi_init2(help_2,64);

    mpfi_set_d(help_1,1.);
    mpfi_set_d(help_2,68.);

    mpfi_exp2(help_2,help_2);
    mpfi_div(help_1,help_1,help_2);

    mpfi_set_d(help_2, 3.);

    mpfi_add(help_1, help_2, help_1);

    mpfi_log2(help_1, help_1);
    mpfi_log2(help_2, help_2);

    mpfi_sub(eps_64,help_1,help_2);

    mpfi_set(log_64, help_2);

    /* 128-bit variable set */

    mpfi_t eps_128, log_128, x_128, x_eps_128, x_ceil_128, diff_128;
    mpfr_t max_left_128, max_right_128;

    mpfi_init2(eps_128, 128);
    mpfi_init2(log_128, 128);
    mpfi_init2(x_128, 128);
    mpfi_init2(x_eps_128, 128);
    mpfi_init2(x_ceil_128, 128);
    mpfi_init2(diff_128, 128);

    mpfr_init2(max_left_128, 128);
    mpfr_init2(max_right_128, 128);

    /* Compute eps and log2(3) in 128-bit */

    mpfi_init2(help_1,128);
    mpfi_init2(help_2,128);

    mpfi_set_d(help_1,1.);
    mpfi_set_d(help_2,68.);

    mpfi_exp2(help_2,help_2);
    mpfi_div(help_1,help_1,help_2);

    mpfi_set_d(help_2, 3.);

    mpfi_add(help_1, help_2, help_1);

    mpfi_log2(help_1, help_1);
    mpfi_log2(help_2, help_2);

    mpfi_sub(eps_128,help_1,help_2);

    mpfi_set(log_128, help_2);

    /* 256-bit variable set */

    mpfi_t eps_256, log_256, x_256, x_eps_256, x_ceil_256, diff_256;
    mpfr_t max_left_256, max_right_256;

    mpfi_init2(eps_256, 256);
    mpfi_init2(log_256, 256);
    mpfi_init2(x_256, 256);
    mpfi_init2(x_eps_256, 256);
    mpfi_init2(x_ceil_256, 256);
    mpfi_init2(diff_256, 256);

    mpfr_init2(max_left_256, 256);
    mpfr_init2(max_right_256, 256);

    /* Compute eps and log2(3) in 256-bit */

    mpfi_init2(help_1,256);
    mpfi_init2(help_2,256);

    mpfi_set_d(help_1,1.);
    mpfi_set_d(help_2,68.);

    mpfi_exp2(help_2,help_2);
    mpfi_div(help_1,help_1,help_2);

    mpfi_set_d(help_2, 3.);

    mpfi_add(help_1, help_2, help_1);

    mpfi_log2(help_1, help_1);
    mpfi_log2(help_2, help_2);

    mpfi_sub(eps_256,help_1,help_2);

    mpfi_set(log_256, help_2);

    /* 512-bit variable set */

    mpfi_t eps_512, log_512, x_512, x_eps_512, x_ceil_512, diff_512;
    mpfr_t max_left_512, max_right_512;

    mpfi_init2(eps_512, 512);
    mpfi_init2(log_512, 512);
    mpfi_init2(x_512, 512);
    mpfi_init2(x_eps_512, 512);
    mpfi_init2(x_ceil_512, 512);
    mpfi_init2(diff_512, 512);

    mpfr_init2(max_left_512, 512);
    mpfr_init2(max_right_512, 512);

    /* Compute eps and log2(3) in 512-bit */

    mpfi_init2(help_1,512);
    mpfi_init2(help_2,512);

    mpfi_set_d(help_1,1.);
    mpfi_set_d(help_2,68.);

    mpfi_exp2(help_2,help_2);
    mpfi_div(help_1,help_1,help_2);

    mpfi_set_d(help_2, 3.);

    mpfi_add(help_1, help_2, help_1);

    mpfi_log2(help_1, help_1);
    mpfi_log2(help_2, help_2);

    mpfi_sub(eps_512,help_1,help_2);

    mpfi_set(log_512, help_2);

    /* Testing the numbers up to MAX */

    for(unsigned long i = 1; i < MAX; i++)
    {

        /* Check with 64-bit precision */

        result = check_bound(i, eps_64, log_64, x_64, x_eps_64,
                  diff_64, x_ceil_64, max_left_64,
                  max_right_64);

        /* If it fails check again
        with 128-bit precision */

        if(result == 1)
        {

            result = check_bound(i, eps_128, log_128, x_128,
                      x_eps_128, diff_128, x_ceil_128,
                      max_left_128, max_right_128);

            /* If it fails check again
            with 256-bit precision */

            if(result == 1)
            {

                result = check_bound(i, eps_256, log_256, x_256,
                          x_eps_256, diff_256, x_ceil_256,
                          max_left_256, max_right_256);

                /* If it fails check again
                with 512-bit precision */

                if(result == 1)
                {

                    result = check_bound(i, eps_512, log_512, x_512,
                              x_eps_512, diff_512, x_ceil_512,
                              max_left_512, max_right_512);

                    /* If it fails again print the index
                     - the result is equal up to 512-bit
                      precision */

                    if(result == 1)
                    {

                        fails++;

                        f.push_back(i);

                    }

                }

            }

        }

    }

    auto stop = high_resolution_clock::now();

    auto duration = duration_cast<seconds>(stop - start);

    std::cout << "All integers tested up to: " << MAX << '\n';
    std::cout << fails << " have failed with 512-bit precision" << '\n';
    std::cout << "Execution time: " << duration.count() << 's' << '\n';

    std::cout << "Fails occurred for numbers:" << '\n';

    for(unsigned long n : f)
    {

        std::cout << n << '\n';

    }

    return 0;

}
```

Below is the source for a proposed C++ implementation using multiprocessing. Required libraries are GMP, MPFR and MPFI.

```cpp
#include <gmp.h>
#include <mpfr.h>
#include <mpfi.h>
#include <math.h>
#include <stdio.h>
#include <chrono>
#include <iostream>
#include <vector>

using namespace std::chrono;

int check_bound(unsigned long n, const mpfi_t &eps,
     const mpfi_t &log, unsigned int p)
{

    mpfi_t x, x_eps, x_ceil, diff;
    mpfr_t max_left, max_right;


    mpfi_init2(x, p);
    mpfi_init2(x_eps, p);
    mpfi_init2(x_ceil, p);
    mpfi_init2(diff, p);

    mpfr_init2(max_left, p);
    mpfr_init2(max_right, p);

    mpfi_mul_ui(x, log, n);
    mpfi_mul_ui(x_eps, eps, n);

    mpfr_rint_ceil(max_left,&(x->left),MPFR_RNDD);
    mpfr_rint_ceil(max_right,&(x->right),MPFR_RNDU);

    mpfr_set(&(x_ceil->left),max_left,MPFR_RNDD);
    mpfr_set(&(x_ceil->right),max_right,MPFR_RNDU);

    mpfi_sub(diff,x_ceil,x);

    int result = (diff < x_eps);

    mpfi_clear(x);
    mpfi_clear(x_eps);
    mpfi_clear(x_ceil);
    mpfi_clear(diff);

    mpfr_clear(max_left);
    mpfr_clear(max_right);

    return result;

}

int main()
{
    /* Start execution timing */

    auto start = high_resolution_clock::now();

    /* Set maximum iterations */

    unsigned long MAX = exp2(35);

    /* Set result variable */

    int result = 0;

    /* Set fail counter */

    unsigned int fails = 0;

    /* Set fail table */

    std::vector<unsigned long> f;

    /* Declare helper variables */

    mpfi_t help_1, help_2;

    /* 64-bit variable set */

    mpfi_t eps_64, log_64;

    mpfi_init2(eps_64, 64);
    mpfi_init2(log_64, 64);

    /* Compute eps and log2(3) in 64-bit */

    mpfi_init2(help_1,64);
    mpfi_init2(help_2,64);

    mpfi_set_d(help_1,1.);
    mpfi_set_d(help_2,68.);

    mpfi_exp2(help_2,help_2);
    mpfi_div(help_1,help_1,help_2);

    mpfi_set_d(help_2, 3.);

    mpfi_add(help_1, help_2, help_1);

    mpfi_log2(help_1, help_1);
    mpfi_log2(help_2, help_2);

    mpfi_sub(eps_64,help_1,help_2);

    mpfi_set(log_64, help_2);

    /* 128-bit variable set */

    mpfi_t eps_128, log_128;

    mpfi_init2(eps_128, 128);
    mpfi_init2(log_128, 128);

    /* Compute eps and log2(3) in 128-bit */

    mpfi_init2(help_1,128);
    mpfi_init2(help_2,128);

    mpfi_set_d(help_1,1.);
    mpfi_set_d(help_2,68.);

    mpfi_exp2(help_2,help_2);
    mpfi_div(help_1,help_1,help_2);

    mpfi_set_d(help_2, 3.);

    mpfi_add(help_1, help_2, help_1);

    mpfi_log2(help_1, help_1);
    mpfi_log2(help_2, help_2);

    mpfi_sub(eps_128,help_1,help_2);

    mpfi_set(log_128, help_2);

    /* 256-bit variable set */

    mpfi_t eps_256, log_256, x_256;

    mpfi_init2(eps_256, 256);
    mpfi_init2(log_256, 256);

    /* Compute eps and log2(3) in 256-bit */

    mpfi_init2(help_1,256);
    mpfi_init2(help_2,256);

    mpfi_set_d(help_1,1.);
    mpfi_set_d(help_2,68.);

    mpfi_exp2(help_2,help_2);
    mpfi_div(help_1,help_1,help_2);

    mpfi_set_d(help_2, 3.);

    mpfi_add(help_1, help_2, help_1);

    mpfi_log2(help_1, help_1);
    mpfi_log2(help_2, help_2);

    mpfi_sub(eps_256,help_1,help_2);

    mpfi_set(log_256, help_2);

    /* 512-bit variable set */

    mpfi_t eps_512, log_512;

    mpfi_init2(eps_512, 512);
    mpfi_init2(log_512, 512);

    /* Compute eps and log2(3) in 512-bit */

    mpfi_init2(help_1,512);
    mpfi_init2(help_2,512);

    mpfi_set_d(help_1,1.);
    mpfi_set_d(help_2,68.);

    mpfi_exp2(help_2,help_2);
    mpfi_div(help_1,help_1,help_2);

    mpfi_set_d(help_2, 3.);

    mpfi_add(help_1, help_2, help_1);

    mpfi_log2(help_1, help_1);
    mpfi_log2(help_2, help_2);

    mpfi_sub(eps_512,help_1,help_2);

    mpfi_set(log_512, help_2);

    /* Testing the numbers up to MAX */

    #pragma omp parallel for simd
    for(unsigned long i = 1; i < MAX; i++)
    {

        /* Check with 64-bit precision */

        result = check_bound(i, eps_64, log_64, 64);

        /* If it fails check again
         with 128-bit precision */

        if(result == 1)
        {

            result = check_bound(i, eps_128, log_128, 128);

            /* If it fails check again
            with 256-bit precision */

            if(result == 1)
            {

                result = check_bound(i, eps_256, log_256,
                          256);

                /* If it fails check again
                 with 512-bit precision */

                if(result == 1)
                {

                    result = check_bound(i, eps_512, log_512,
                              512);

                    /* If it fails again print the index - the
                     result is equal up to 512-bit precision */

                    if(result == 1)
                    {

                        fails++;

                        f.push_back(i);

                    }

                }

            }

        }

    }

    auto stop = high_resolution_clock::now();

    auto duration = duration_cast<seconds>(stop - start);

    std::cout << "All integers tested up to: " << MAX << '\n';
    std::cout << fails << " have failed with 512-bit precision" << '\n';
    std::cout << "Execution time: " << duration.count() << 's' << '\n';

    std::cout << "Fails occurred for numbers:" << '\n';

    for(unsigned long n : f)
    {

        std::cout << n << '\n';

    }

    return 0;

}
```

Bibliography

[1] Lagarias, Jeffrey C., *The 3x+1 Problem: An Annotated Bibliography (1963–1999)*. arXiv, 2012. [Online]. Available: https://arxiv.org/abs/math/0608208v3

[2] ——, *The Ultimate Challenge: The 3x+1 Problem*. AMS, 2010. [Online]. Available: N/A

[3] Tao, Terrence, *Almost all orbits of the Collatz Map attain almost bounded values*. arXiv, 2020. [Online]. Available: https://arxiv.org/abs/1909.03562

[4] Barina, David, *Convergence verification of the Collatz problem*. The Journal of Supercomputing, 2021. [Online]. Available: https://doi.org/10.1007/s11227-020-03368-x

[5] Sinisalo, Matti K., *On the minimal cycle lengths of the Collatz sequences*. University of Oulu, 2003. [Online]. Available: N/A

[6] Conway, John H., *Unpredictable Iterations*. University of Colorado Boulder, 1972. [Online]. Available: N/A

[7] Kurtz, Stuart A.; Simon, Janos, *The Undecidability of the Generalized Collatz Problem*. Springer, 2007. [Online]. Available: https://doi.org/10.1007/978-3-540-72504-6_49

[8] Crandall, Richard E., *On the 3x+1 problem*. AMS, 1978. [Online]. Available: N/A