The Thesis Committee for Matthew Alexander Denend
Certifies that this is the approved version of the following Thesis:

Challenging Variants of the Collatz Conjecture

APPROVED BY

SUPERVISING COMMITTEE:

Scott Aaronson, Supervisor

Marienus Heule, Co-Supervisor

Challenging Variants of the Collatz Conjecture

by

Matthew Alexander Denend

THESIS

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT AUSTIN

December 2018

Dedicated to my late mother, who passed away earlier this year, yet never let her three battles with cancer stop her from endlessly loving her sons and her husband.

Acknowledgments

I want to give a huge thank you to both of my advisors for this project: Professor Scott Aaronson and Professor Marijn Heule.

Professor Scott Aaronson came to UT Austin from MIT in fall 2016. I knew that I wanted to write a master's thesis from pretty much the time I started my master's degree, and I heard he might be a great person to work with. We come from very different backgrounds. He is well-renowned for his work in theoretical quantum computing. I came into UT's CS program with virtually no math/CS theory background, as my undergrad degree is in electrical engineering. He had a project idea related to SAT solving that interested me, and eventually suggested that I talk to Dr. Marijn Heule about SAT solving. As a result, this project was born. I got a chance to witness firsthand the enthusiasm that Scott has for CS and math theory. It's no wonder that Scott has done so much work in theoretical computer science... he just keeps going and going! This entire project would not have been possible without him and he is as much, if not more, of an author of this thesis as I am. I'm also deeply thankful to him for his support as I had to go through the challenge of losing my mom.

Professor Marijn Heule became an assistant research professor at UT Austin in fall 2017. He was a research scientist with UT when I first met him. Marijn is a master at SAT solving, and is famous for his proof of the Boolean Pythagorean

Triples Problem in 2016 that was the largest proof ever at the time at 100 TB. Much like Scott, Marijn has a great deal of enthusiasm for mathematical problems. This entire project is originally Marijn's idea, so I am thankful that he had this idea, and I am very happy to have written a thesis on such a cool topic. Without his advice and direction and his many read-overs and edits, this thesis would have not been possible to finish, so he is as much, if not more, of an author of this thesis as I am. I am also deeply thankful of the support he gave me as I had to go through the challenge of losing my mom.

When Scott and Marijn first met each other, they raced each other to solve several proofs related to this project: Scott by hand, Marijn using computation. I happened to be in the middle of it all, unable to keep up with the blazing speed of these two brilliant minds. It took me some time but I can now confidently read through most of these thoughts that Scott and Marijn had and understand them much better now. I am indebted to these two men and can only hope to someday achieve the greatness and brilliance of them.

I am also thankful for the many friends that I have made along the way towards finishing this thesis, especially those who gave me support as I had to deal with the loss of my mom. There are far too many to list, but you know who you are. From my time here in Austin, I have made many friends that I want to keep for many years to come.

Finaly, I want to thank my immediate family. First, my older brother, for being one of the most patient, hard working people I know. He has been a great role model for me, and continues to do so, and is a great friend of mine as well. Even

though we haven't lived together in a while, I've enjoyed spending lots of time with him over the phone.

My father, for keeping me in line during my childhood and instilling discipline into me that I managed to carry with me through both undergrad and graduate school. He is a great person to talk to in difficult situations and always puts others before himself, and is also a great friend of mine as well.

Most of all, I am so thankful that I had such a loving, caring mother. She told me, a couple of months before she passed, her dream job was to be a house wife and take care of her kids. Her passion for this really showed... without all of the love and caring she gave to me, I can't imagine that I'd be sitting here, writing the acknowledgements section of my master's thesis. She was one of my best friends, always willing to listen to me whether I had time to talk for 10 minutes or 2 hours, always caring and loving me. But most of all, she taught me how important it is to never quit. Cancer may wear you down and make you tired, weak, and treatment may compromise your immune system, but it can never take away your ability to be human, your ability to lovem your soul. She fought three valiant battles with cancer, one before I was born, and never gave up in any of them. I had difficult time dealing with my mom slowly dying 2000 miles away for almost the entirety of my time in grad school, but she didn't want me to leave school to come back for her. She wanted me to keep fighting, alongside her. And here's the product of that determination. Thank you so much for everything, mom.

Abstract

Challenging Variants of the Collatz Conjecture

Matthew Alexander Denend, M.S. Comp.Sci.
The University of Texas at Austin, 2018

Supervisors: Scott Aaronson
Marienus Heule

The Collatz Conjecture (also known as the $3N + 1$ problem) is simple to explain, yet proving that all positive integers following the Collatz Mapping must converge to 1 has eluded mathematicians for over half a century. Aaronson and Heule are exploring solving the Collatz Conjecture using an approach involving string rewrite systems: Aaronson transformed the Conjecture into a string rewrite system and Heule has been applying parallel SAT solvers on instances of this system. Similar approaches have been applied successfully to other mathematical problems.

We started looking into simpler variants of the conjecture. This thesis defines some of these variants and investigates easily provable as well as very hard variants. We study the hardness of unsolved variants by computing the number of rewrite steps needed up to 1 billion. Our hardness prediction method suggests that proving termination of the challenging variants should be considerably easier compared to solving the original conjecture.

Table of Contents

List of Figures

Chapter 1

Introduction

Computers have been successfully applied to many complex problems, such as those in finance, healthcare, and the Internet. However, computers still struggle with several problems. Consider whether a given program on any input will always terminate. One can easily show that certain programs will always halt. For example, we can easily show a program that takes an integer input x, computes $y = x + 1$, prints y, then halts, will always terminate. Nevertheless, there exist simple programs that are much harder to determine if they always terminate. Algorithm 1 is such an example. Will this program always return 1 for any positive integer N, causing it to halt? This problem is a reformulation of the Collatz Conjecture, also known as the $3N + 1$ problem.

Algorithm 1 The Collatz Conjecture Sequence, Col(N)

1: **if** $N \leq 1$ **then return** N
2: **if** $N \equiv 0 \pmod 2$ **then return** Col($N/2$)
3: **return** Col($3N + 1$)

We know from Turing that no program exists to determine if an arbitrary program with arbitrary input can halt [1], but that does not exclude the possibility of a program that determines if specifically Algorithm 1 halts. However, no program has been found to show that all positive integer inputs for Algorithm 1 will halt, even

though this problem can be explained to an elementary grade student. The problem has been extensively analyzed, according to surveys by Lagrias [2] [3], yet no proof has been found. The search is further motivated by extensive empirical evidence suggesting that the Collatz Conjecture is true. According to a website maintained by Roosendahl [4], all numbers up to $87 \cdot 2^{60}$, or about 10^{20}, have been tried as N in Algorithm 1, and have converged to 1.

Since the Collatz Conjecture has been challenging to prove, we propose another supposedly simpler variant of it. Can we prove that the code in Algorithm 2, where $A = \{1\}$, and $b = 8$, always terminates for any positive integer N? Even though this program seems to be easier, we do not have a program showing this variant always terminates either!

Algorithm 2 A Collatz Conjecture Variant $\text{Col}_{\text{mod}}(N, A, b)$

1: **if** $(N \leq 1) \vee (N \equiv a_1 \pmod{b}) \vee \ldots \vee (N \equiv a_s \pmod{b})$ **then return** N
2: **if** $N \equiv 0 \pmod{2}$ **then return** $\text{Col}_{\text{mod}}(N/2, A, b)$
3: **return** $\text{Col}_{\text{mod}}(3N + 1, A, b)$

One of the goals of this thesis is to try and determine how hard certain variants of the Collatz Conjecture are to solve. A contribution of this thesis is that it uses empirical data to try and find trends of hardness for difficult variants, and compare these trends to the hardness of solving the whole Collatz Conjecture. This thesis also follows an approach that Heule and Aaronson have devised attempting to craft a program to determine if Algorithm 1 always halts [5]. At a high level, it involves taking a completely reworked formulation of Algorithm 1 and using known techniques that, if certain conditions are met, the reworked formulation can be shown

to terminate for any positive integer input. The formulation requires SAT solvers, string rewrite systems, and a technique called matrix interpretation, all topics which will be covered briefly in this paper as background. This thesis also investigates a rewrite system that Aaronson crafted, and we believe that, if Aaronson's system is found to terminate for all input, the Collatz Conjecture holds. We analyze properties of this rewrite system, and the previously variants are investigated with this rewrite system as well.

The rest of this thesis is outlined as follows: Chapter 2 introduces definitions that will be used throughout the paper. Chapter 3 defines several Collatz Conjecture Variants, both solved and unsolved, including $\mathrm{Col}_{\mathrm{mod}}(N, \{1\}, 8))$. Chapter 4 analyzes the difficulty of these variants using algebra. Chapter 5 discusses the results so far of Heule using Aaronson's rewrite system and parallel SAT solving to prove the Collatz Conjecture and hard Collatz Variants, as well as necessary background to understand the approach. Finally, Chapter 6 investigates hardness of solving the same variants covered in Chapter 4, but derived from Aaronson's rewrite system instead.

Chapter 2

Definitions

We will use the following terms throughout this thesis:

- **$3N + 1$ sequence**: Define this as follows:

$$N_0 = N = \text{ initial input number}$$
$$N_{i+1} = \begin{cases} N_i/2 & \text{if } N_i \text{ is even} \\ 3N_i + 1 & \text{if } N_i \text{ is odd} \end{cases}$$

  This sequence can continue for arbitrarily large values of i, but we are only interested in following the sequence until N_i, for some i, is 1, as any value after 1 follows the cycle of $1 \to 4 \to 2 \to 1$ infinitely.

  Note that if we run Col(N), we would, after i recursive calls, end up passing, into the $i + 1^{\text{th}}$ recursive call of Algorithm 1, N_{i+1}. Algorithm 1 can be modified to store the sequence of numbers $N, N_1, \ldots, N_i$, meaning it can effectively compute $3N + 1$ sequences as well. However, for simplicity, throughout the remainder of this thesis, we just refer to $3N + 1$ sequences, with N as the initial input, and N_i as the i number in the $3N + 1$ sequence that started with N.

- **$3N + 1$ mapping**: One recursive call of Col(N) for input number N, giving us the number N_1, as defined in the $3N + 1$ sequence.

- **Avoidance set A:** The second parameter of Algorithm 2, $\mathrm{Col}_{\mathrm{mod}}(N, A, b)$, which was defined in the introduction. $A = \{a_1 \dots a_s\}$ is all of the numbers modulo the positive integer b that cause termination of the algorithm. This termination condition is in addition to the sole termination condition of Algorithm 1: when the input into $\mathrm{Col}(N)$, N, is 1. Note that for all $a \in A$, $0 \leq a < b$, and $A = \varnothing$ turns Algorithm 2 into Algorithm 1.

- **Collatz Variant:** When we say "Collatz Variant" we are referring to a specific instance of Algorithm 2.

- **$\mathrm{Col}_{\mathrm{mod}}(N, A, b)$:** A specific instance of a Collatz Variant for a positive integer N, avoidance set A, and positive integer b.

- **Collatz Variant A:** The vast majority of our analysis on Collatz Variants is on instances when $b = 8$, so when we say Collatz Variant A, it is shorthand for $\mathrm{Col}_{\mathrm{mod}}(N, A, 8)$. We often list several base 8 instances of Collatz Variants together, so for instance, if we say Collatz Variants 1, 5, 7, and $\{1, 5\}$; we mean the instances $\mathrm{Col}_{\mathrm{mod}}(N, \{1\}, 8)$, $\mathrm{Col}_{\mathrm{mod}}(N, \{5\}, 8)$, $\mathrm{Col}_{\mathrm{mod}}(N, \{7\}, 8)$, and $\mathrm{Col}_{\mathrm{mod}}(N, \{1, 5\}, 8)$; respectively. Note that when listing variants in this manner, we omit the braces normally around singleton sets.

- **Number of bits:** For any number N, we say that, written in binary, it has m bits. Note that $m = \log_2 N$.

Chapter 3

Alternative Termination Conditions

This chapter explores some of the possible Collatz Variants. Before we investigate them, we define a graph paradigm that transforms the $3N + 1$ problem into directed graphs that depict the flow for all input numbers modulo base b, where b is a power of 2. Using this paradigm, we can show that some variants are easily provable. However, others are not.

3.1 Base *b* Collatz Graph Definition

Define $G_b = (V, E)$ to be a "Base b Collatz Graph", where $b = 2^k$, and k is nonnegative. Choosing a power of 2 for b allows us to easily reason with binary numbers, which is useful for both several proofs in this chapter and the Collatz SRS that we will discuss starting in Chapter 5. V has b vertices in it, where each vertex is labeled a unique integer in the interval $[0, b-1]$. We say that a number N is visiting vertex $v \in V$ if and only if $N \equiv v \pmod{b}$ for vertex v labeled with integer v. Out of convenience, note that we use v as a vertex and integer interchangeably in this paper. Let N be an input number, and N_1 the result of applying the $3N + 1$ mapping to N. $E = V \times V$ is a set of directed edges, where, for nodes $u, v \in V$, $(u, v) \in E$ if and only if $N_i \equiv u \pmod{b}$ and $N_{i+1} \equiv v \pmod{b}$ for some N and N_1.

3.2 Base b Collatz Graph Lemmas

This section contains some lemmas that are used throughout this chapter. First, we start with a lemma about the number of node transitions in any Collatz Base b Graph G_b.

Lemma 3.2.1. *Given a Collatz Base b Graph, for all $v \in V$, if node v is even, then it has two outgoing edges. Otherwise, if node v is odd, then it has only one outgoing edge.*

Proof. Take some number N that is visiting node v, and consider N in binary. First, let us consider the case where node v is even. When we divide by 2, we just remove the lowest 0 bit from N to get N_1. The bit at index $k-1$ of N_1 can be either a 0 or a 1, allowing for two options for even nodes.

Now, let N visit an odd node v. Multiplying N by 3 and adding 1 will give us a N_1 where the binary string for it grows at least one bit longer than N, since $3 \cdot N = 2 \cdot N + N$, and $2N$ in binary is shifting the bits of N one index to the left, then adding a 0 bit to the least significant bit. This gives us only one option for the least significant k bits, meaning N_1 can only visit one node, and only one such outgoing edge from v exists. □

Now, we have a couple of important properties about cycles that exist in any Collatz Base b Graph, and the fact that these cycles cannot continue indefinitely. First, we introduce the "0 cycle" lemma.

Lemma 3.2.2. *For any Collatz Base b Graph (where b is a positive power of 2), a self-loop occurs on a 0 node. This cycle cannot continue indefinitely.*

Proof. Assume we have an input number N such that $N \equiv 0 \pmod{b}$. This means that the k least significant bits are all 0. Apply the $3N + 1$ mapping once to this string. We remove a 0 from the end, since we divide the input number by 2. We now look at the new number N_1 and the k least significant bits. The $k - 1$ least significant bits are all 0. But what is the value of the bit in the most significant of the k lowest bits of N_1? If it is a 0, then $N_1 \equiv 0 \pmod{b}$, and we have a self-loop.

To show that the self loop cannot continue indefinitely: every time we follow the self-loop, remove a 0. Since we also know that $N > 1$ in order for the algorithm to continue running, we also know that at least one binary digit is a 1. So as we continue dividing by 2 and removing bits that are 0, we eventually reach, after i visits to node 0, a point where the least k significant bits are $1000\ldots0$, meaning N_i is visiting node $b/2$ instead, ending the cycle. □

Now, we introduce another important lemma about cycles: the fact that we will always have a cycle between nodes $b - 2$ and $b - 1$, and it cannot continue indefinitely.

Lemma 3.2.3. *For any Collatz Base b Graph, a cycle between exists between nodes $b - 2$ and $b - 1$. This cycle cannot continue indefinitely.*

Proof. In this proof, we will show first that a $b-1 \to b-2 \to b-1$ cycle exists, and second, that this cycle cannot continue indefinitely; more precisely, that some number

		x_n	x_{n-1}	$\dots$	x_{j+1}	x_j	$\dots$	x_{k+1}	x_k	1	1	$\dots$	1	1	1
$\times$														1	1
$^{c_{n+2}}$	$^{c_{n+1}}$	^{c_n}	$^{c_{n-1}}$		$^{c_{j+1}}$	^{c_j}		$^{c_{k+1}}$	^{1}	^{1}	^{1}		^{1}	^{1}	
		x_n	x_{n-1}	$\dots$	x_{j+1}	x_j	$\dots$	x_{k+1}	x_k	1	1	$\dots$	1	1	1
$+$	x_n	x_{n-1}	x_{n-2}	$\dots$	x_j	x_{j-1}	$\dots$	x_k	1	1	1	$\dots$	1	1	1_{+1}
y_{n+2}	y_{n+1}	y_n	y_{n-1}	$\dots$	y_{j+1}	y_j	$\dots$	y_{k+1}	y_k	1	1	$\dots$	1	1	0

Figure 3.1: The long multiplication corresponding to $3x + 1$. Note how, in the addition step, the 1_{+1} is in place of a zero, and the addition is just $x + 2x + 1$.

congruent modulo to $b - 2 \pmod b$, after division by 2, actually becomes congruent modulo to $b/2 - 1 \pmod b$ instead. We will prove this using long multiplication in binary.

Let x be a arbitrary number[1] congruent modulo to $b - 1 \pmod b$. We want to express x in binary, so let x have m bits. We know that since x is congruent modulo to $b - 1 \pmod b$, all bits from indices 0 to $k - 1$ are 1. Let x_j denote the j^{th} bit of x, $k \leq j \leq m$. These bits are unknown. Let 1_{+1} correspond to the adding of 1 after multiplying by 3, which is added to the least significant bit of x. Let c_j denote the unknown carry for the j^{th} addition of bits. Let y be the result after $3x + 1$ is computed, and let index j of y be the same index as x. The multiplication is referenced in Figure 3.1.

As mentioned in lemma 3.2.1, multiplication of a binary number x by 3 is just $x + 2x$, and $2x$ is just placing all bits of x one index to the left, adding a 0 bit at the newly vacant spot. In the multiplication we show in Figure 3.1, we write this

[1]Normally, we use the notation N to denote an arbitrary number in the $3N + 1$ sequence, but we mean N_i to mean the i^{th} number in the $3N + 1$ sequence, whereas in this proof, we have x_j to denote the j^{th} bit of x. Hence, we use variables x and y instead of N and N_1.

but replace the new least significant 0 bit of $2x$ with the special 1_{+1} bit, allowing us to perform $3x + 1$ with just one addition instead of two.

Notice that all bits in the resulting binary number y from indices 0 to $k-1$ are 1, except for index 0, which is 0. This is because all bits of y, save the least significant bit, are computed by adding $1+1$ and carrying over the 1 from the previous addition. As a result, y is congruent modulo to $b-2 \pmod b$, meaning an edge from node $b-1 \pmod b$ to node $b-2 \pmod b$ exists in our graph.

When we divide y by 2, we just remove the least significant 0 bit from y, decreasing the indices of all bits in y by 1. Hence, bit y_k is now in position $k-1$. If y_k is 1, our number is now congruent modulo to $b-1 \pmod b$. This means an edge also exists from node $b-2 \pmod b$ to node $b-1 \pmod b$, proving the existence of the $b-2 \to b-1 \to b-2$ cycle.

Now we show that this cycle will always eventually terminate. This happens when, after dividing a number congruent modulo to $b-2 \pmod b$ by 2, bit y_{k-1} is 0. So we need to show this eventually occurs for any positive integer x. There are two cases for this:

1. Some bit in x is 0. Let x_j be the least significant bit of x that is 0 (all bits which have indices lower than j are 1). Look back at Figure 3.1, and replace $x_j = 0$, and have all bits at indices less than j be 1. After taking the $3x + 1$ step, all bits of y between 1 and $j - 1$ will be 1, since for each bit, we add $1 + 1$ and carry over a 1. However, when we get to index j, we add $1 + 0$ plus a carry of 1, making bit $y_j = 0$. Since after we divide by 2 we move all bits one

index to the right, bit y_j now moves to index $j-1$. We repeat this process a total of $j-k+1$ times. After this, the 0 bit will be in position y_{k-1}, making y congruent modulo to $b/2-1 \pmod{b}$ instead, breaking the cycle.

2. No bit in x is 0. In this case, again looking at Figure 3.1, all bits in y from indices 1 to m will be 1. However, bit $y_{m+1}=0$, because $y_{m+1}=c_{m+1}+x_m+x_{m+1}$, and $c_{m+1}=1$ and $x_m=1$, but $x_{m+1}=0$. Since bit $y_{m+1}=0$ we move it over one index to index m after dividing by 2. We then follow case 1, where $j=m$, and hence, the cycle also breaks in this case.

Hence, no input number can follow the $b-2 \to b-1 \to b-2$ cycle indefinitely. $\square$

We introduce one more lemma that shows that cycles where the magnitude of divisions by 2 outweigh the magnitude of multiplications by 3. We call this the "even node dominance lemma".

Lemma 3.2.4. *Given a cycle in any Collatz Base b Graph G, let V_e be the set of even nodes in the cycle, and V_o be the set of odd nodes. If $2^{|V_e|} > 3^{|V_o|}$, then the cycle cannot continue indefinitely.*

Proof. Let $2^{|V_e|} > 3^{|V_o|}$ and $j = |V_e| + |V_o|$. Let N be visiting a node v in the cycle, and assume N will run through the cycle at least once without terminating, implying no number between N and N_j inclusive is 1, and that N_j visits the same node v as N did. We visited $|V_e|$ vertices in the cycle, so we divided N by $2^{|V_e|}$ after one trip

around the cycle. We also visited $|V_o|$ vertices in the cycle, each time multiplying by 3, and overall, multiplied N by about $3^{|V_o|}$.[2]

Hence, after we visited the cycle once, we computed $N_j \approx \frac{3^{|V_o|}}{2^{|V_e|}}N$. Since $2^{|V_e|} > 3^{|V_o|}$, $N_j < N$. Therefore, one of two things must happen:

1. N eventually becomes 1, which means the cycle no longer can be repeated.

2. The cycle is eventually broken.

In both cases, the cycle cannot continue indefinitely. □

3.3 Base 4 Collatz Graph and Variants

In this section, we build a simple example of a Base b Collatz Graph: G_4. We chose G_4 because we can prove that $\mathrm{Col}_{\mathrm{mod}}(N, A, 4)$ terminates for any nonempty $A \subseteq \{0, 1, 2, 3\}$. In G_4, there are 4 different nodes: one for each integer between 0 and 3. When determining which node v input number N visits, we look at the lowest 2 bits of N, since $4 = 2^2$. For example, $N \equiv 0 \pmod 4$ has 00 as the 2 least significant bits, whereas $N \equiv 2 \pmod 4$ has 10 as its 2 least significant bits.

3.3.1 Base 4 Collatz Graph Construction

Figure 3.2 shows G_4. We describe the construction in this subsection. We start by describing the transitions for even nodes, then the transitions for odd nodes.

[2]We also added one each time we visit an odd node, but this is asymptotically insignificant compared to multiplying by 3 or dividing by 2, so we ignore it in this proof.

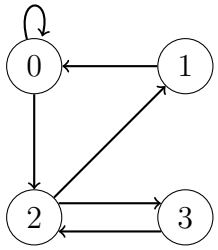

Figure 3.2: The Base 4 Collatz Graph, G_4. There are 4 nodes, each one corresponding to a value mod 4.

Assume, in all cases, that a number N is written in binary. First, the transitions for even nodes:

- N visiting node 0 means N ends with binary string "00". Removing the last 0 bit of N leaves either "00", meaning N_1 visits node 0 again, or "10", meaning N_1 visits node 2 instead.
- N visiting node 2 means N ends with binary string "10". Removing the last 0 bit of N leaves either "01", meaning N_1 visits node 1, or "11", meaning N_1 visits node 3.

Now, the transitions for odd nodes. Let x_2 and x_3 be unknown bits:

- N visiting node 1 means N ends with binary string "01". Multiplying this by 3 and adding 1 results in N_1 ending with "$x_2$00", so N_1 visits node 0.
- N visiting node 3 means N ends with binary string "11". Multiplying this by 3 and adding 1 results in N_1 ending with "$x_3x_2$10", so N_1 visits node 2.

3.3.2 Base 4 Collatz Variants

We introduced the Base 4 case for nodes because we can prove that we need to visit all of the nodes in this graph, which is equivalent to saying that each of the Collatz Variants, $\mathrm{Col}_{\mathrm{mod}}(N, A, 4)$ terminates for nonempty $A \subseteq \{0, 1, 2, 3\}$, and for any input number N.

Theorem 3.3.1. $\mathrm{Col}_{mod}(N, A, 4)$ *terminates for nonempty* $A \subseteq \{0, 1, 2, 3\}$.

Proof. Assume that no $3N + 1$ sequence described in this proof reaches 1, otherwise $\mathrm{Col}_{\mathrm{mod}}(N, A, 4)$ terminates trivially for any A. We will start with proving that $\mathrm{Col}_{\mathrm{mod}}(N, \{2\}, 4)$ will terminate for any input number N, because the 2 node is central to the Base 4 graph.

Lemma 3.3.2. $\mathrm{Col}_{mod}(N, \{2\}, 4)$ *terminates for any* N.

Proof. We use the graph to help in this proof. An equivalent question is this: Can we show that node 2 must be visited for all input numbers? To show that this is the case, we have to show that all other nodes must visit node 2.

- 2: If the input number N is visiting node 2, we are already done.
- 3: If the input number N is visiting node 3, then after the $3N + 1$ mapping is applied to N, N_1 is now visiting node 2.
- 1 and 0: Let N be visiting node 1. N_1 visits node 0 after applying the $3N + 1$ mapping once. To show N_{1+j} must leave node 0, we use lemma 3.2.2 (the "0

cycle" lemma) for $b = 4$, and N_{1+j} will visit node 2 after j more applications of the $3N + 1$ mapping, both causing $\text{Col}_{\text{mod}}(N, \{2\}, 4)$ to terminate.

Since all other nodes must visit node 2, it means that for all N, $\text{Col}_{\text{mod}}(N, \{2\}, 4)$ terminates. □

Lemma 3.3.3. $\text{Col}_{mod}(N, \{1\}, 4)$ *terminates for any* N.

Proof. We use lemma 3.3.2 to show that node 2 must be visited, meaning that for any input number N, $N_i \equiv 2 \pmod 4$ after i steps. Then we use lemma 3.2.3 to show that the cycle between nodes 2 and 3 cannot continue indefinitely, so after j more steps, N_{i+j} visits node 1, proving termination of this variant. □

Lemma 3.3.4. $\text{Col}_{mod}(N, \{0\}, 4)$ *terminates for any* N.

Proof. Given lemma 3.3.3, we know after i steps N_i must visit node 1, and given lemma 3.2.1, an odd node can only have one outgoing edge, so N_{i+1} visits node 0. □

Lemma 3.3.5. $\text{Col}_{mod}(N, \{3\}, 4)$ *terminates for any* N.

Proof. Given the "even node dominance lemma", 3.2.4, the $2 \to 1 \to 0 \to \ldots \to 2$ cycle cannot continue forever because, if we assume that the 0 node never self-cycles, $2^2 > 3^1$. The 0 self-cycle makes even nodes dominate even more. So an input number N in the $2 \to 1 \to 0 \to \ldots \to 2$ cycle must, after i steps, visit node 3, causing $\text{Col}_{\text{mod}}(N, \{3\}, 4)$ to terminate. □

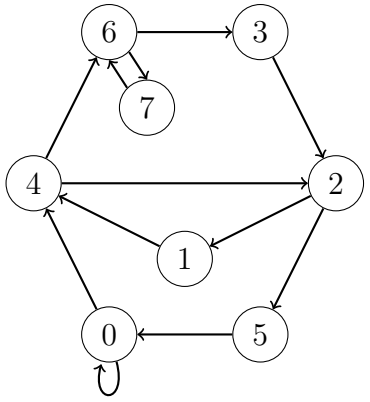

Figure 3.3: The Base 8 Collatz Graph, G_8. There are 8 nodes, each one corresponding to a value mod 8.

Since all of these lemmas hold, it follows that any singleton set from $A \subseteq \{0, 1, 2, 3\}$ causes $\text{Col}_{\text{mod}}(N, A, 4)$ to terminate. Also, by definition of Algorithm 2, any larger size sets for A also terminate as larger set sizes add more termination conditions. So any nonempty set A will cause $\text{Col}_{\text{mod}}(N, A, 4)$ to terminate for any input N. □

3.4 Base 8 Collatz Graph and Variants

After proving $\text{Col}_{\text{mod}}(N, A, 4)$ terminates for any nonempty base set A and input number N, we decided to see what would happen if we expanded to $k = 3$ bits. We have not been able to prove all variants of $\text{Col}_{\text{mod}}(N, A, 8)$ for all nonempty sets A. This will motivate further computation undertaken in Chapter 4, as we try to determine how hard figuring out these unproven variants are. Figure 3.3 shows the base 8 graph. There are 8 different nodes, since $8 = 2^3$.

3.4.1 Base 8 Collatz Graph Construction

Like G_4 before, we show how to construct G_8. We examine the even nodes first. In this case, there are four different nodes: 0, 2, 4, and 6. Using lemma 3.2.1, each vertex has two different transitions, depending on what the next bit to the left of the 3 bits after removing the least significant 0.

- N visiting node 0 means N ends with binary string "000". Removing the last 0 bit of N leaves either "000", meaning N_1 loops to node 0; or "100", meaning N_1 visits node 4 instead.
- N visiting node 2 means N ends with binary string "010". Removing the last 0 bit leaves either "001", meaning N_1 visits node 1; or "101", meaning N_1 visits node 5.
- N visiting node 4 means N ends with binary string "100". Removing the last 0 bit leaves either "010", meaning N_1 visits node 2; or "110", meaning N_1 visits node 6.
- N visiting node 6 means N ends with binary string "110". Removing the last 0 bit leaves either "011", meaning N_1 visits node 3, or "111", meaning N_1 visits node 7.

Now, the odd nodes. Let x_3 and x_4 be unknown bits.

- N visiting node 1 means N ends with binary string "001". Multiplying this by 3 and adding 1 results in N_1 ending with "100", so N_1 visits node 4.

- N visiting node 3 means N ends with binary string "011". Multiplying this by 3 and adding 1 results in N_1 ending with "$x_3$010", so N_1 visits node 2.
- N visiting node 5 means N ends with binary string "101". Multiplying this by 3 and adding 1 results in N_1 ending with "$x_4x_3$000", so N_1 visits node 0.
- N visiting node 7 means N ends with binary string "111". Multiplying this by 3 and adding 1 results in N_1 ending with "$x_4x_3$110", so N_1 visits node 6.

3.4.2 Base 8 Collatz Graph Cycle Analysis

Since we do not have proofs for all Collatz Variants $\mathrm{Col}_{\mathrm{mod}}(N, A, 8)$, we analyzed whether certain cycles can last indefinitely. We have found that all simple cycles of G_8 can be proven to not last indefinitely. However, showing that some combinations of these simple cycles cannot continue forever is much more difficult to do.

We tie in interesting Collatz Variants to these analyses. We start with smaller cycles and work our way into longer cycles, as well as combinations of them.

- The 0 self-cycle, $(0 \to \ldots)$ cannot continue forever, as per lemma 3.2.2.
- The $6 \to 7 \to 6$ cycle cannot continue forever as per lemma 3.2.3.
- The $4 \to 2 \to 1 \to 4$ cycle cannot continue forever, as even nodes dominate ($2^2 > 3$), so according to lemma 3.2.4, this cycle cannot last forever.
- $4 \to 2 \to 5 \to 0 \to \ldots \to 4$ cycle: Even nodes dominate, even without any 0 self-cycles ($2^3 > 3$), so according to lemma 3.2.4, this cycle cannot continue

forever. We can also combine this cycle with the $4 \to 2 \to 1 \to 4$ cycle, and since both cycles cause input numbers to decrease, the combination of these two cycles must visit a new node to prevent the number from converging to 1. The only other choice is node 6, so this argument is used to prove Collatz Variant 6 must terminate.

- $4 \to 6 \to 3 \to 2 \to 1 \to 4$ cycle: There are three different variations of this cycle. We have a proof for only one of them:
  - No transition allowed from $4 \to 2$: The following theorem explains this case.

    Theorem 3.4.1. *Aaronson '17: The* $4 \to 6 \to 3 \to 2 \to 1 \to 4$ *cycle cannot continue indefinitely.*

    Proof. If we start some number N such that $N \equiv 4 \pmod 8$, and follow the $4 \to 6 \to 3 \to 2 \to 1 \to 4$ cycle once, we turn N into $(9N + 20)/8 = \frac{9}{8}(N + 20) - 20$. If we were to repeat the cycle k times, we would turn N into $\frac{9}{8}^k(N + 20) - 20$. This quantity must be an integer for all k if the cycle is to continue forever. However, $N + 20$ will only have a finite number of factors of 8, so the cycle must terminate. □

  - Transition allowed between $4 \to 2$: This creates a conflict between two different cycles: one that causes growth by approximately a factor of $9/8$ ($4 \to 6 \to 3 \to 2 \to 1 \to 4$), and another that causes the cycle to decay by a factor of $4/3$ ($4 \to 2 \to 1 \to 4$). Even though we can prove that both of

these cycles terminate independently, it has been a challenge to show that the combination of them cannot continue indefinitely, as we cannot prove that one cycle must stop transitioning to the other. Hence, no proof, by hand or machine, is known that we must break out of this combination of cycles, by visiting either node 5 or 7. A proof that this combination of cycles must be broken would prove termination of Collatz Variant $\{5, 7\}$, which we explore in Chapter 4.

  - Building on the prior point, we can also consider visits to the node 7 as well in this cycle. This adds the $6 \to 7 \to 6$ cycle to the already challenging two cycle case. If we can't prove that the smaller combination of cycles $4 \to 6 \to 3 \to 2 \to 1 \to 4$ and $4 \to 2 \to 1 \to 4$ cannot terminate, it would be far more difficult to add a third cycle, even though all three cycles must terminate individually. A proof of this cycle would solve Collatz Variant 5, also explored in Chapter 4.

- $4 \to 6 \to 3 \to 2 \to 5 \to 0 \to \ldots \to 4$ cycle: There are three different variations to showing this cycle cannot continue forever: keeping both nodes 1 and 7 omitted, or omitting one node or the other. Adding in both nodes yields the entire base 8 graph. The variant where both 1 and 7 are omitted is proven, the other two are not.

  - Strictly following this cycle, no changes: assuming no zero-cycles, there are 4 distinct even nodes in this cycle, and 2 distinct odd nodes. $2^4 > 3^2$, so according to the "even node dominance lemma", 3.2.4, this cycle cannot

continue forever. The nodes that must be visited to break this cycle are either 1 or 7, so this is a proof that Collatz Variant $\{1, 7\}$ terminates.

– Allowing 7 but avoiding 1: Like variant $\{5, 7\}$, we have two conflicting cycles: the $6 \to 7 \to 6$ cycle which cause the number to grow by approximately $\frac{3}{2}$ every time it takes this cycle, and the base cycle $4 \to 6 \to 3 \to 2 \to 5 \to 0 \to \ldots \to 4$ that reduces it by approximately $\frac{16}{9}$, depending on number of zero self cycles. These two cycles are interesting in that they clash the fastest growing part of the graph: the $6 \to 7 \to 6$ cycle, and the fastest decaying part of the graph: the 0 self-cycle, followed by two more even numbers. We are also not aware of a proof for this case either. Such a proof that these two cycles cannot continue combined forever would be equivalent to proving termination of Collatz Variant 1. We present analysis of hardness of this cycle in Chapter 4.

– Allowing 1 but avoiding 7: Adding back in node 1 but disallowing node 7 actually allows for three different cycles: $4 \to 6 \to 3 \to 2 \to 5 \to 0 \to \ldots \to 4$, $4 \to 6 \to 3 \to 2 \to 1 \to 4$, and $4 \to 2 \to 1 \to 4$. Finding a proof for this case is expected to be harder than the unsolved two cycle case of $4 \to 6 \to 3 \to 2 \to 1 \to 4$, and $4 \to 2 \to 1 \to 4$. Solving that the combination of these three cycles must terminate is equivalent to proving termination of Collatz Variant 7.

3.4.3 Base 8 Collatz Variants

As for which nodes we are forced to visit during the computation of a $3N+1$ sequence, we can prove Collatz Variants 2, 3, 4 and 6 terminate, meaning nodes 2, 3, 4, and 6 must be visited in the base 8 graph eventually. Variant 0 can be shown to terminate if another unproven variant terminates. It will still be mentioned with the already proven variants. The following will explain how these five variants terminate. We present them in an order to build arguments off of one another. Like the proofs for Base 4 Collatz Variants, assume that no $3N+1$ sequence described in this proof reaches 1, otherwise $\mathrm{Col}_{\mathrm{mod}}(N, A, 8)$ terminates trivially for any A.

- **$\mathbf{Col_{mod}}(N, \{6\}, 8)$:** Like in the Collatz Base 4 Graph, we have to show that all other nodes must visit node 6 eventually. There are several different cases we enumerate here:

  1. N visits node 6. We are already done.
  2. N visits node 7. Then after one application of the $3N+1$ mapping, N_1 visits node 6.
  3. N is visiting nodes 0, 1, 2, 4, or 5. The $3N+1$ sequence for N in this case, to avoid node 6, would have to traverse one of two cycles: $4 \to 2 \to 1 \to 4$ or $4 \to 2 \to 5 \to 0 \to \ldots \to 4$. We talked about how this combination of cycles cannot continue forever in Subsection 3.4.2. So an N input number in either one of these cycles, after i steps, must visit node 6.
  4. N visits node 3. Then after one application of the $3N+1$ mapping,

N_1 visits node 2, and we apply the previous argument to show it must transition to node 6 eventually.

Hence, $\text{Col}_{\text{mod}}(N, \{6\}, 8)$ must terminate for any input N.

- **$\text{Col}_{\text{mod}}(N, \{3\}, 8)$:** We know that $\text{Col}_{\text{mod}}(N, \{6\}, 8)$ terminates, so as a result, an input number N must transition to node 6 after i steps. Given lemma 3.2.3, the $6 \to 7 \to 6$ cycle cannot continue forever. Hence, after another j steps, N_{i+j} visits node 3, meaning $\text{Col}_{\text{mod}}(N, \{3\}, 8)$ must terminate.

- **$\text{Col}_{\text{mod}}(N, \{2\}, 8)$:** Since we know that N must visit node 3 after i steps, we apply the $3N+1$ mapping once, and N_{i+1} visits node 2, proving $\text{Col}_{\text{mod}}(N, \{2\}, 8)$ must terminate.

- **$\text{Col}_{\text{mod}}(N, \{4\}, 8)$:** Given that we know we need to visit node 2, we know that N_i visits node 2. We look at the graph and see two different paths, both which lead to node 4: Either visit node 1 then 4, or visit node 5 then 0. From lemma 3.2.2, the 0 cycle cannot continue forever, so either way, the path taken must traverse to node 4. Hence after j steps for $j \geq 2$, N_{i+j} visits node 4, and $\text{Col}_{\text{mod}}(N, \{4\}, 8)$ terminates.

- **$\text{Col}_{\text{mod}}(N, \{0\}, 8)$:** In the graph, we can see that to visit node 0, we must come from node 5. So we cannot prove this yet unless we prove termination for $\text{Col}_{\text{mod}}(N, \{5\}, 8)$.

We do not have proofs for $\text{Col}_{\text{mod}}(N, \{1\}, 8)$, $\text{Col}_{\text{mod}}(N, \{5\}, 8)$, $\text{Col}_{\text{mod}}(N, \{7\}, 8)$, but we can prove a couple of combined variants of them:

- **$\mathbf{Col_{mod}(N, \{1, 5\}, 8)}$:** We already know that Collatz Variant 2, so N_i visits node 2. Looking at the base 8 graph, 2 must traverse to either node 1 or 5. Hence, after one application of the $3N + 1$ mapping, N_{i+1} visits either 1 or 5, meaning $\text{Col}_{\text{mod}}(N, \{1, 5\}, 8)$ must terminate.

- **$\mathbf{Col_{mod}(N, \{1, 7\}, 8)}$:** This was discussed in Subsection 3.4.2, but repeated here: Since the $4 \to 6 \to 3 \to 2 \to 5 \to 0 \to \ldots \to 4$ cannot continue forever, either node 1 or 7 must be visited, since there are not other choices for nodes. Hence, $\text{Col}_{\text{mod}}(N, \{1, 7\}, 8)$ must terminate.

However, the combination $\text{Col}_{\text{mod}}(N, \{5, 7\}, 8)$ has not been proven. Hence, we've run some computational experiments to try and better understand the difficulty of coming up with a proof for termination of variant $\{5, 7\}$, as well as Collatz Variants 1, 5, and 7.

Chapter 4

Collatz Variant Hardness Prediction

In this section, we attempt to determine how difficult variants from Algorithm 2 are for $\text{Col}_{\text{mod}}(N, A, 8)$, where $A = \{1\}$, $\{5\}$, $\{7\}$, or $\{5, 7\}$. We start by defining some measures, talk about the process of running experiments, and talk about the results. We analyze termination of the singleton set Collatz Variants 1, 5, and 7; and the combined 2-element variant of $\{5, 7\}$; in separate sections.

4.1 Defining Measures

We define hardness off of the notion that odd numbers make the Collatz Conjecture harder, whereas even numbers make it easier. To more precisely define the measures, define the following numbers, given some input number N:

- $f(N)$: The total number of steps in the sequence for N before it converges to 1.
- $f_{\text{odd}}(N)$: Number of odd numbers visited in the sequence from N to 1. Note that $f_{\text{even}}(N) = f(N) - f_{\text{odd}}(N)$.
- A: The base avoidance set. Same as defined in Algorithm 2 and Chapter 2. For the Collatz Variants we are exploring in this chapter, $A \subseteq \{1, 3, 5, 7\}$ and

$A \neq \varnothing$.

- Collatz Variant A Sequence: The sequence of numbers that, for input number N, runs i numbers in length, such that $\forall a \in A, j \in [0, i], (N_j \not\equiv a \pmod 8 \wedge N_j > 1)$.

- $g(N, A)$: The highest number of steps that an input number N, while computing Algorithm 1, also avoids termination of Collatz Variant A. More precisely, the longest number of steps in a Collatz Variant A Sequence for all numbers in the $3N + 1$ sequence for N until it reaches 1.

- $g_{\text{odd}}(N, A)$: The number of odd numbers within the given $g(N, A)$.

- Slice: a batch of numbers from some low number to some high number for a fixed A.

- N_{min}: the lowest number of any slice.

- N_{max}: the highest number of any slice.

- Record: any number r in the range that has $g(r, A)$ higher than all numbers measured so far in the slice. More formally, any new record r_{new} must have the properties compared to the current record r_{current}: $r_{\text{new}} > r_{\text{current}}$, and $g(r_{\text{new}}, A) > g(r_{\text{current}}, A)$ for a specific A. Note that we measure records off of total steps, *not* total number of odd numbers.

Using these numbers, three different measures are defined, and the intuition behind why they were chosen is given as well:

Hardness: Defined to be $H(N, A) = \frac{g_{\text{odd}}(N,A)}{\log_2 N}$. This assesses whether or not increasing the number of bits needed to represent the number N changes the difficulty of determining a proof for Collatz Variants 1, 5, 7, or $\{5, 7\}$.

Classical hardness: Defined to be $H_C(N) = \frac{f_{\text{odd}}(N)}{\log_2 N}$. This is a comparison to our hardness measure, but we compute H_C with respect to the whole sequence, instead of trying to avoid specific numbers. Records for classical hardness occur when $r_{\text{new}} > r_{\text{current}}$ and $f(r_{\text{new}}) > f(r_{\text{current}})$.

Note that classical hardness is much like the gamma value mentioned by Lagrias [2] [3] and Roosendaal [4], except that $\gamma(N) = \frac{f_{\text{even}}(N)}{\log N}$. We chose to define our measure based on odd numbers for purposes of this thesis, because the Collatz String Rewrite System we define in Chapter 5 is made much harder by the presence of odd numbers.

Percentage of Sequence: Defined to be $P(N, A) = \frac{g_{\text{odd}}(N,A)}{f_{\text{odd}}(N)}$. This assesses what percentage of all odd numbers in the Collatz Sequence lie within Record Collatz Variant Sequences for Collatz Variants 1, 5, 7, or $\{5, 7\}$.

4.2 Generating Measures

We wrote a program that computes Collatz Sequences using Java, and ran it on all odd numbers from 1 to 1 billion. The program has various modes which evolved over the lifetime of this project. In these modes, let $\mathcal{A}$ be a family of avoidance sets A, and let $r(A)$ be the record for set $A \in \mathcal{A}$.

- `baseavoid` is the default option. This allows us to check all $A \in \mathcal{A}$ by running through all odd numbers from $N_{\min}$ (we usually use 1) to $N_{\max}$ (we usually use 1 billion), and determines the record $r(A)$ for all of these numbers. When it finishes, it prints out, for each A, separate csv files that have $r(A)$, $g(r(A), A)$, and the Record Collatz Variant A Sequence. Algorithm 3 outlines how we run this for the family of sets $\mathcal{A}$ for a given N. Before printing results, we repeat this algorithm for each odd $N \in [N_{\min}, N_{\max}]$.

Algorithm 3 Base Avoid Mode for input N

Input: The initial number N; family of avoidance sets $\mathcal{A}$; dictionary that stores, for all sets $A \in \mathcal{A}$, $r(A)$ and $g(r, A)$
Let M be a dictionary
for $A \in \mathcal{A}$ **do**
    $M[A] \leftarrow 0$
while $N > 1$ **do**
    if $N \equiv 0 \pmod 2$ **then**
        $N \leftarrow N/2$
    else
        $N \leftarrow 3N + 1$
    $y \leftarrow N \pmod b$
    for $A \in \mathcal{A}$ **do**
        if $y \in A$ **then**
            if $M[A] > g(r, A)$ **then**
                $g(r, A) \leftarrow M[A]$
                $r(A) \leftarrow N$
            $M[A] \leftarrow 0$
        else
            $M[A] \leftarrow M[A] + 1$

- `entirechain` just runs Algorithm 1 for all odd N in the range $N_{\min} \leq N \leq N_{\max}$, and prints out the smallest number that has the longest length $3N + 1$

sequence. More precisely, it prints out the smallest odd number r such that $(\forall x \in [N_{\min}, N_{\max}] | x \equiv 1 \pmod 2)(f(r) \geq f(x))$.

- `untildecay` means that, for each odd number in between $N_{\min}$ and $N_{\max}$, we continue to run until, after i steps, we have a number N_i such that $N_i < N$. We return only the longest sequence of numbers that occurs until the resulting number is smaller than the initial number, as well as the steps i needed.

- `updown` is a quite different mode. For each odd number N such that $N_{\min} \leq N \leq N_{\max}$, determine two things. First, the number of steps it takes for N to become some number N_i such that $N_i < N$, like in the `untildecay` mode. Second, the number of steps it takes for another number N_g to grow to N if such an N_g exists (no multiple of 3 can grow from a smaller number, for instance). The output prints out, for all odd numbers in the range, N_i, the number of steps it takes for N to turn into N_i, and, if N_g exists, the number N_g and the number of steps it takes for N_g to grow into N. The process of computing N_g and the number of steps it takes for N_g to grow into N is included in a README for the git repository mentioned at the end of this section.

- `avoidingmodgrowth` computes, for $N_{\min} \leq N \leq N_{\max}$ and for all $A \in \mathcal{A}$, $r(A)$ and $g(r(A), A)$ in the same manner as the `baseavoid` mode, except we do not overwrite old records with new ones. Instead, we store all records in a table. A csv file is made for each $A \in \mathcal{A}$, and given such an A, we print progressively growing $r(A)$ and $g(r(A), A)$. This is the mode we used to generate the hardness results in this thesis.

As mentioned, we used the `avoidingmodgrowth` mode to generate the records defined in Subsection 4.2. We run for sequences of odd numbers in multiple slices, usually 8, in order to take advantage of parallel computing via a distributed computing program called Condor that was made by The University of Wisconsin-Madison [6]. We then combine the records of these 8 tables by hand using the defined record criterion.

Within slices we are running, we added an option to avoid recomputing odd numbers already part of a prior Collatz Sequence, as these will never generate new records. However, this option can be disabled if we wish to compute extremely large numbers and slices and are limited in our memory storage.

The program could have been rewritten to build off of previously used results, which should run faster, but this would have been difficult without many GBs of memory available and good memory management in our program. So the space efficient approach was chosen for this project.

The code can be accessed via a public GitHub repository located at `https://github.com/mdenend/CollatzRewriteSystem`. A README file is included that explains how to run the code and available options.

4.3 Single Base Collatz Variant Analysis

Our analysis for analyzing the termination of Collatz Variants 1, 5, and 7 is broken into three subsections: two exploring our defined computations, H and P, and a third one analyzing interesting properties of sequence similarities that may provide insight to eventual proofs showing that these three Collatz Variants must

terminate. For exploring H and P, we took all of the records for Collatz Variants 1, 5, and 7, and plotted, for records r, the number of bits ($\log_2 r$) versus $H(r, A)$ and $P(r, A)$, respectively. We also added in Collatz Variant 3 as a control case.

4.3.1 Hardness Function Results and Analysis

Figure 4.1 shows the results, for records r, of $H(r, A)$ versus the number of bits ($\log_2 r$). Comparing the three unproven Collatz Variants 1, 5, and 7 to the proven variant 3, the known variant is easier. The known variant actually slight decreases in hardness as the number of bits increases, meaning that there are fewer odd numbers per bit.

Comparing the unknown variants to themselves, there is no consistent leader among the three as the number of bits increases. However, they all seem to be within a hardness range of 1-3, with only a couple of exceptions. Variant 7 seems to remain in the same range with no definite increase or decrease, whereas variants 1 and 5 grow slightly from about 12 bits onward. The growth for both variants 1 and 5 may be because as numbers get larger, there are more opportunities to visit the $6 \to 7 \to 6$ cycle, which adds odd numbers more quickly to the sequence than any other base 8 graph traversal. More experiments for higher numbers should be consider in order to determine whether any of these three unknown variants continue to trend the same way.

Classical hardness actually tends to grow linearly against the log scale, meaning that as the input number increases in number of bits, H_C increases logarithmically. This contrasts to H for all the plotted Collatz Variants, which tend to stay

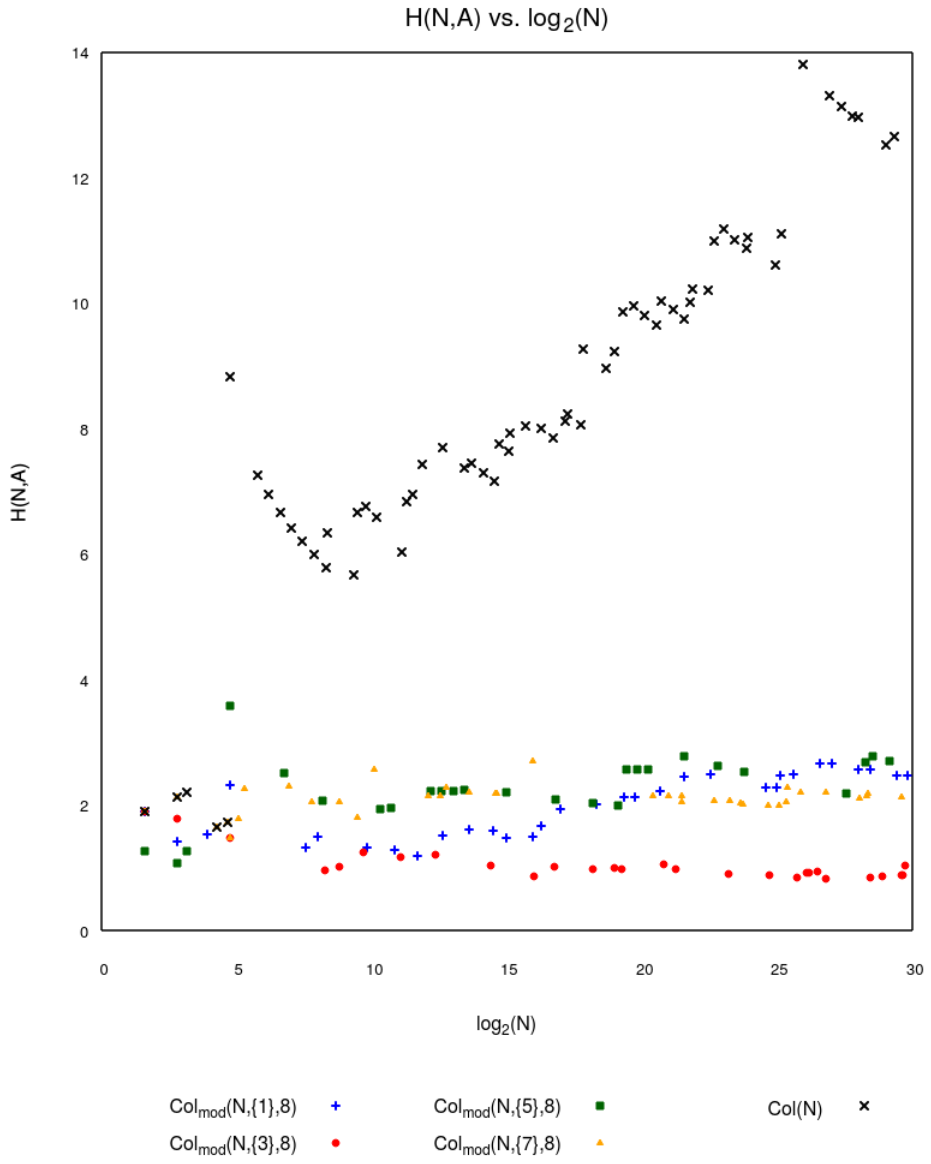

Figure 4.1: This graph visualizes how, for records r, the H values for Collatz Variants 1, 3, 5, and 7 compare to each other, and how they compare to H_C. The number of bits for the records ($\log_2 N$) is the x-axis, and the hardness measure H as defined in Subsection 4.1 is the y-axis.

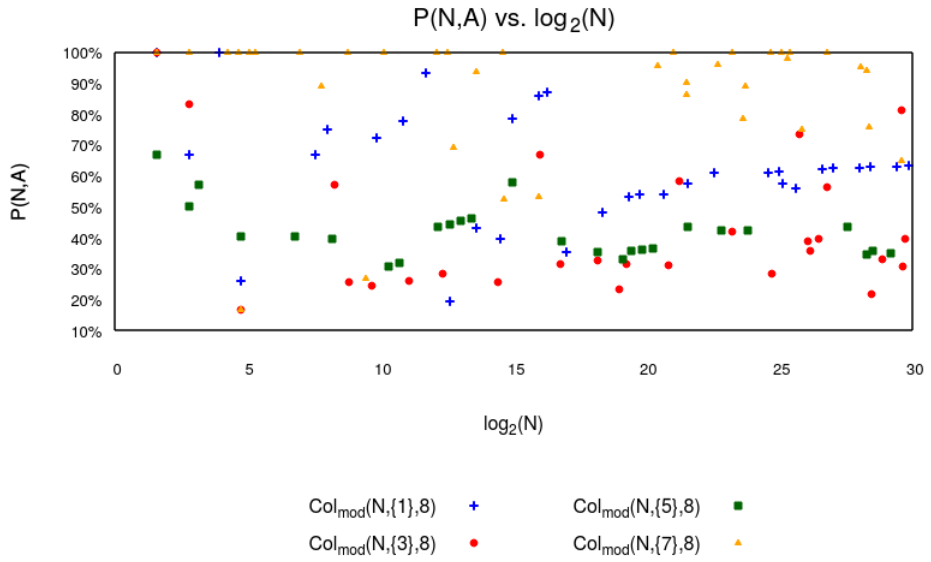

Figure 4.2: This graph visualizes how, for records r, P values for Collatz Variants 1, 3, 5, and 7 compare to each other. The number of bits for the records ($\log_2 N$) is the x-axis, and the percentage measure P as defined in Subsection 4.1 is the y-axis.

below an H of 3, meaning that figuring out proofs for the variants' termination is expected to be easier than proving the Collatz Conjecture.

4.3.2 Percentage of Sequence Function Results and Analysis

Figure 4.2 shows, for records r, the results of $P(r, A)$ versus the number of bitsin r. $P(N, A)$, as discussed earlier, is just calculating what percentage of odd numbers are part of record sequences for Collatz Variants 1, 3, 5, and 7.

Collatz Variant 7 comprises the highest percentage overall, with a couple of exceptions. Following Collatz Variant 7 causes the sequence to decline rapidly, since the $6 \to 7 \to 6$ cycle causes an input number to grow faster than any other cycle in

the base 8 graph. Almost all of the records for variant 7 terminate at 1 instead of actually reaching a number that is 7 (mod 8).

Variant 5 tends to have a low percentage, and appears to be the least erratic of all four variants. Variant 5 avoids the 0 self-cycle, which causes many divisions by 2. Numbers having record sequences that avoid 5 (mod 8) tend to turn into very large numbers when variant 5 terminates, meaning that many more steps in the $3N + 1$ mapping must often be taken before these numbers converge to 1.

Variant 1 is interesting, because as the input numbers grow larger, the line changes from erratic behavior to a more steady percentage at around 20 bits. This is likely a consequence of the sequence similarity that is seen in larger records for variant 1, which will be analyzed in Subsection 4.3.3. Further, as mentioned in the cycle analysis in Subsection 3.4.2, the $4 \to 6 \to 3 \to 2 \to 5 \to 0 \to \ldots \to 4$ cycle combined with the $6 \to 7 \to 6$ cycle causes a clash between the decay of the 0 self-cycle and the growth of the $6 \to 7 \to 6$ cycle. This may explain some of the erratic percentage for variant 1, aside from the small part with chain similarity.

Variant 3 record sequences tend to have the lowest percentage of all odd numbers compared to other variants, even lower than variant 5, but also has erratic percentage. This could be explained by the fact that avoiding termination of variant 3 causes the sequence to follow some combination of the $6 \to 7 \to 6$ cycle, the $4 \to 2 \to 1 \to 4$ cycle, or the $4 \to 2 \to 5 \to 0 \to \ldots \to 4$ cycle with some number of 0 self-cycles. The first cycle causes growth, whereas both other cycles cause decay. If the growth cycle is followed, the number gets larger, likely reducing the percentage of odd numbers making up long chains avoiding termination of variant 3, whereas

the decay cycles cause the number to shrink, tending the percentages to be higher. This may explain why variant 3 causes widely different percentages.

4.3.3 Sequence Similarity Analysis

We analyzed the sequences of the records for Collatz Variants 1, 5, and 7 as well to see if we could find any similarities:

- Variant 1: There are two groups of records that were particularly interesting: Those from 325,791 to 32,505,681 (call this group S), and those from 35,651,835 to 949,643,331 (call this group T). Group S numbers all terminated at number 161, and group T numbers all terminated at number 35,369. These sequences all matched *number-by-number* at least one other sequence starting at most 8 steps from the beginning. This is a striking similarity meaning that records for variant 1 might be predictably related to groups S or T, or perhaps to other groups.

- Variant 7: All record sequences, except for input number 27, terminated at 1. While there was some similarity between sequences (all numbers ≥ 62079 had the same last 41 numbers), there were many different paths taken from the input, so not as many patterns as variant 1.

- Variant 5: This had few matches and was the most changing of the records, so chain similarity appears not to have affected the low variance that P has for variant 5.

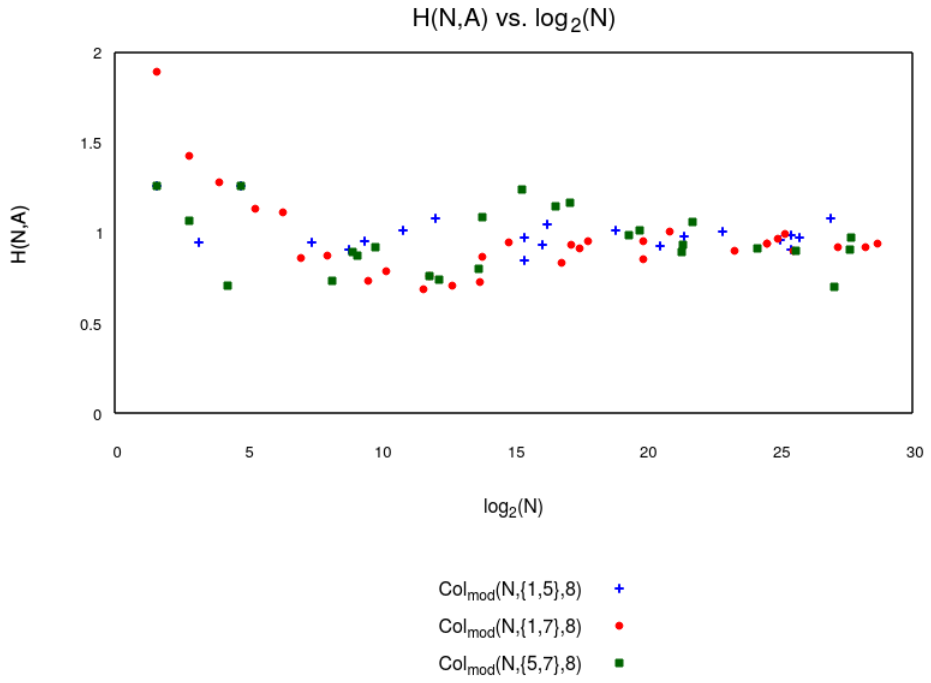

Figure 4.3: This graph visualizes how, for records r, H measure for the three Collatz Variants $\{1,5\}$, $\{1,7\}$, and $\{5,7\}$. The number of bits for the records ($\log_2 N$) is the x-axis, and the hardness measure H as defined in Subsection 4.1 is the y-axis. Classical hardness was omitted from this graph to eliminate distortion.

4.4 Paired Base Avoidance Analysis

Since termination of Collatz Variants 1, 5, and 7 appear to be difficult to prove, we also analyzed two element combinations of them. The termination of two such variants were already proved in Subsection 3.4.3: $\{1,5\}$ and $\{1,7\}$. However, termination of variant $\{5,7\}$ has yet to be proven. This section will analyze what happens to $H(N,A)$ and $P(N,A)$ where $A = \{1,5\}$, $\{1,7\}$, and $\{5,7\}$.

4.4.1 Hardness Function Results and Analysis

Figure 4.3 shows, for records r, $H(r, A)$ versus bits in r. These results were quite surprising. At first thought, it would have appeared that the unproven variant $\{5, 7\}$ should be the hardest to determine, compared to the two variants we have proofs for. But both variants $\{1, 5\}$ and $\{1, 7\}$ had alike predictive hardness to $\{5, 7\}$! These numbers suggest that a proof for determining why variant $\{5, 7\}$ must terminate is either easier than we anticipated, or our hardness measures are not very good. However, given the fact that variant 3 for the single base cases is clearly easier than variants 1, 5, and 7, we have reason to believe this measure should be good. Further investigation needs to be considered.

4.4.2 Percentage of Sequence Function Results and Analysis

Figure 4.4 shows, for records r, $P(r, A)$ versus bits in r. Record sequences for variant $\{1, 7\}$ make up the highest percentage of their overall Collatz Sequences, because avoiding both the $6 \to 7 \to 6$ and the $4 \to 6 \to 3 \to 2 \to 1 \to 4$ cycles allows for the sequence to only go through the $4 \to 2 \to 5 \to 0 \to \ldots \to 4$ cycle, causing fast decay, like variant 7.

Both variants $\{1, 5\}$ and $\{5, 7\}$ are much closer to each other in the percentage that their Record Collatz Variant Sequences comprise of their $3N + 1$ sequences, although as the numbers grow past 17 bits in size, variant $\{5, 7\}$ comprises of the higher percentage. A possible explanation is the fact that the $6 \to 7 \to 6$ cycle allowed in variant $\{1, 5\}$, but not variant $\{5, 7\}$, causes a number to grow larger than the $4 \to 6 \to 3 \to 2 \to 1$ cycle that variant $\{5, 7\}$ allows. Also, since variant $\{1, 5\}$

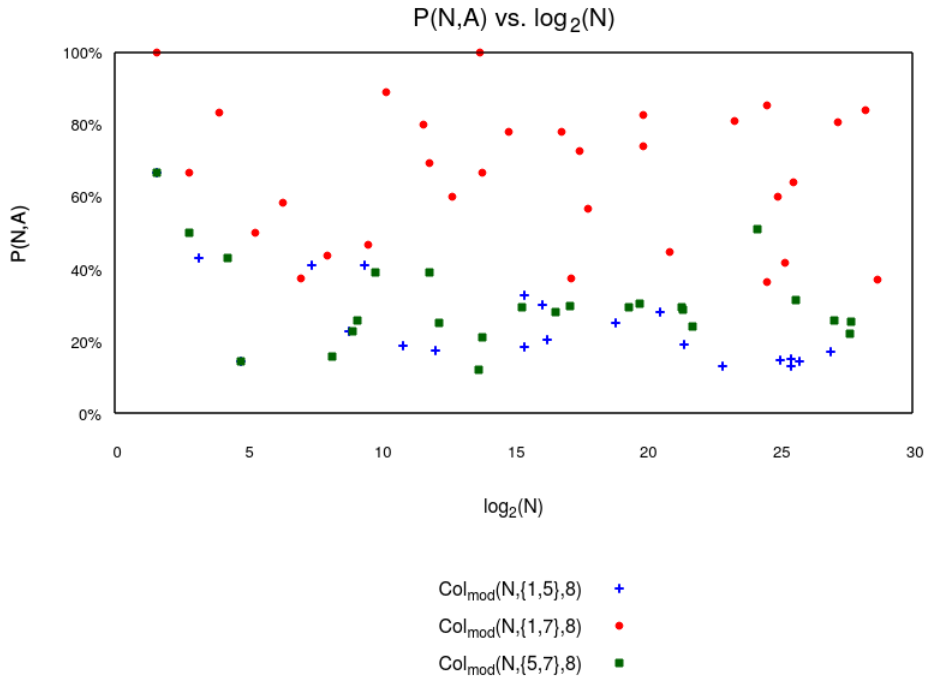

Figure 4.4: This graph visualizes how, for records r, the P measure for variants $\{1,5\}$, $\{1,7\}$, and $\{5,7\}$. The number of bits for the records ($\log_2 N$) is the x-axis, and the hardness measure P as defined in Subsection 4.1 is the y-axis.

allows for larger numbers, and larger numbers generally (but not always) take more steps to decline, allowing for more growth should mean that Record Collatz Variant $\{1, 5\}$ Sequences comprise a lower percentage of all odd numbers in record numbers' overall $3N + 1$ sequences.

Chapter 5

Heule's and Aaronson's Attempts to Prove the Collatz Conjecture

In this chapter, we touch upon the approach that Heule and Aaronson came up with to try and prove the Collatz Conjecture. Before doing so, we mention background needed to understand the approach: SAT solvers, string rewrite systems, and matrix interpretation. We then introduce the string rewrite system that Aaronson built, which, if it terminates, we believe it is equivalent to showing that the Collatz Conjecture holds. We then present results of attempts where Heule applied parallel SAT solving instances on Aaronson's rewrite system to see if the Collatz Conjecture holds.

5.1 SAT Solver Background

SAT solvers are powerful programs that can solve some incredibly complex problems, such as those found in hardware verification, software verification, and combinatorics. SAT solvers leverage the fact that k-SAT, where k is the maximum number of literals per clause, is a decision problem that is NP-Complete when $k > 2$[1], meaning that if $P \neq NP$, the worst case runtime is exponential. However, with

[1]See the Cook-Levin Theorem for a proof showing that k-SAT is NP-complete.

clever heuristics, we can actually solve many interesting propositional logic formulas in linear time. Also, since k-SAT is NP-Complete, other NP-Complete problems may be reduced to it, meaning some instances of NP-Complete problems can be solved with SAT solvers. Knowing SAT solving background is not important for this thesis, but there is a great deal of literature talking about SAT solving, so one can check, for instance, [7].

5.2 String Rewrite System and Matrix Interpretation Background

A string rewriting system (SRS), at a high level, takes a input string of an certain alphabet (set of valid symbols) and applies string rewriting rules (SRRs) in an arbitrary order on the input string to see if the string can be transformed further. The SRS continues to apply SRRs on the input string until the input string no longer has any substrings as input for an SRR. This causes the SRS to terminate.

An example SRS to explain further is given, which is from [5].

SRS A: The alphabet is $\Sigma = \{a, b, c\}$ and the SRRs are as follows:

1. $aa \rightarrow bc$
2. $bb \rightarrow ac$
3. $cc \rightarrow ab$

A problem, called Zantema's Other Problem [8], using SRS A, asks the following question:

Zantema's Other Problem: Does the system laid out in SRS A terminate for any input string $(a|b|c)*$?

If one thinks about this problem a little, it would seem that a proof should be easy to show. Surprisingly, both humans and computers struggled to come up with a proof for this problem when initially presented. However, Hofbauer and Waldmann came up with a proof [8], and from this found that, if the alphabet of an SRS can be converted to functions with certain properties, we can prove that any input string to this SRS can terminate [9].

We explain in more detail using Zantema's other problem. The matrices needed, which are from [8], are the following:

$$a(\vec{x}) = \begin{pmatrix} 1 & 0 & 0 & 3 \\ 0 & 0 & 2 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \vec{x} + \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

$$b(\vec{x}) = \begin{pmatrix} 1 & 2 & 0 & 0 \\ 0 & 2 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \vec{x} + \begin{pmatrix} 1 \\ 2 \\ 0 \\ 0 \end{pmatrix}$$

$$c(\vec{x}) = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 2 & 0 & 0 \end{pmatrix} \vec{x} + \begin{pmatrix} 1 \\ 0 \\ 3 \\ 0 \end{pmatrix}$$

We can build a vector representation of an input string by composing the functions for symbols together. We follow the example from [5] here. For instance, if we build the vector representation of the input string "*bbaa*," we would compute$b \circ b \circ b \circ a(\vec{0})$,

which gives us the following vector:

$$b(b(a(a(\vec{0})))) = \begin{pmatrix} 18 \\ 14 \\ 6 \\ 0 \end{pmatrix}$$

The matrices chosen earlier are not constructed arbitrarily. They are chosen in such a manner that, any time an input string has a rewrite rule applied to it, the magnitude of the vector representation of the input string decreases. The vector must continue to decrease in magnitude until the input string can no longer accept new SRRs, causing the SRS to terminate. The search for correct matrices is challenging, but SAT solvers can help in this process by checking if a $d \times d$ matrix is large enough to meet the aforementioned requirements. The process of building these matrices is technical so further background will not be covered here, but if one is interested, they can check the paper by Endrullis, Waldmann and Zantema [10]. We will use our example string "*bbaa*" to show that the vector representation of it is always decreasing, but first, define the $\succ$ operator to show that, for vectors $(x_1 \dots x_d)$ and $(y_1 \dots y_d)$, $(x_1 \dots x_d) \succ (y_1 \dots y_d)$ if $x_1 > y_1$ and $x_i \geq y_i$ for $i \in \{2, \dots, d\}$. In other words, the first element of a vector must be always decreasing after we apply a rule, while the other $d - 1$ elements must either be the same number or decrease. This operator ensures the magnitude of the vector always decreases.

The following is a possible set of rules that can be applied to "*bbaa*" as well as the vector representations of the string:

$$
\begin{array}{ccccccccccccccccc}
bb\underline{aa} & \to & b\underline{bb}c & \to & ba\underline{cc} & \to & b\underline{aa}b & \to & \underline{bb}cb & \to & a\underline{cc}b & \to & aa\underline{bb} & \to & a\underline{aa}c & \to & ab\underline{cc} & \to & abab \\
\begin{pmatrix}18\\14\\6\\0\end{pmatrix} & \succ & \begin{pmatrix}17\\14\\6\\0\end{pmatrix} & \succ & \begin{pmatrix}15\\14\\6\\0\end{pmatrix} & \succ & \begin{pmatrix}14\\14\\6\\0\end{pmatrix} & \succ & \begin{pmatrix}13\\14\\6\\0\end{pmatrix} & \succ & \begin{pmatrix}7\\14\\5\\0\end{pmatrix} & \succ & \begin{pmatrix}6\\14\\5\\0\end{pmatrix} & \succ & \begin{pmatrix}4\\14\\3\\0\end{pmatrix} & \succ & \begin{pmatrix}3\\0\\3\\0\end{pmatrix} & \succ & \begin{pmatrix}2\\0\\3\\0\end{pmatrix}
\end{array}
$$

The vector representation of the strings are always decreasing as defined by the $\succ$ operator. We could apply any string with symbols a, b, and c and apply rules until termination and the vectors representing the strings would always decrease.

5.3 The Collatz Conjecture as a String Rewriting System

Aaronson built an SRS representing the $3N + 1$ mapping [5]. We'll call this the Collatz SRS throughout the remainder of this thesis.

Let the alphabet of the Collatz SRS consist of the symbols a, b, c, d, e, f, g. The symbols can be written as these linear functions:

$$
\begin{aligned}
a(x) &= 2x & e(x) &= 3x \\
b(x) &= 2x + 1 & f(x) &= 3x + 1 \\
c(x) &= 1 & g(x) &= 3x + 2 \\
d(x) &= x
\end{aligned}
$$

The symbols a and b are binary symbols. They represent a binary system: a is 0 and b is 1. The symbols e, f, and g are ternary symbols. They represent a ternary system: e is 0, f is 1, and g is 2. c and d are placeholder symbols to represent

the leading 1 and the end of the string, respectively. They help this SRS know where the beginning and end of the string are.

Note that in order to correctly compute the values that the strings represent using the above linear functions, one needs to read the functions from left to right. That is to say, the string *cabad*, which is equal to 10, is not equal to $c \circ a \circ b \circ a \circ d$, where $\circ$ is the composition of functions. Instead, $cabad = d \circ a \circ b \circ a \circ c$. Aaronson chose to write the strings like this since they follow the way we would write numbers.

Using the provided alphabet, Aaronson created the following series of SRRs:

$$\begin{array}{llll} D_1 : ad \to d & A_1 : ae \to ea & B_1 : be \to fb & C_1 : ce \to cb \\ D_2 : bd \to gd & A_2 : af \to eb & B_2 : bf \to ga & C_2 : cf \to caa \\ & A_3 : ag \to fa & B_3 : bg \to gb & C_3 : cg \to cab \end{array}$$

The SRRs provided here allow for Aaronson's SRS to be equivalent to the $3N + 1$ mapping, but a formal proof showing this is the case is difficult, because we have yet to find a proof showing that the SRRs can be applied in arbitrary order. However, we can explain how the rules work, and show that any valid input string correctly follows the $3N + 1$ mapping that the number the string represents follows, and after this, we will show the ordering we follow for the remainder of this thesis, and why this ordering is correct.

Each of the rules denotes how to handle the symbols $a - g$ and various combinations of them that occur. The D rules represent handling the $3N + 1$ mapping for a binary system. D_1 is how we handle an even number. It computes division of N by 2 by removing the a symbol that represents a binary 0. D_2 is actually a

combination of several steps. If we were to represent $3N + 1$, we could just write $bd \to bfd$, meaning take all previous symbols and multiply the result by 3 and add 1. The problem with this rule is that it increases the size of the resulting string, making the system more difficult to prove. However, $bfd \to gad$ is a valid rule, as $d \circ f \circ b = 3(2x + 1) + 1 = 6x + 4$ and $d \circ a \circ g = 2(3x + 2) = 6x + 4$, and from here, we can apply the rule $ad \to d$ to allow us to turn gad into gd. Since $3N + 1$ always results in an even number, we can just make rule D_2 compute $(3N + 1)/2$ without growing the string size, ultimately making rule D_2 into $bd \to gd$. D_2 is the rule that makes termination of our system hard to prove. Without it, we would not need the A, B, or C rules.

The A, B and C rules all deal with the handling of the ternary symbols and the eventual conversion of these ternary symbols into binary symbols. The A and B rules deal with the case when a ternary symbol is to the right of the binary symbol, and how to switch the ternary symbol and the binary symbol without changing the number the string represents. We will show that all 6 of these rules preserve the same number by showing that the string represents the same value after each rule has been applied:

- $\boldsymbol{ae \to ea}$: $ae = e \circ a = e(a(x)) = 2(3x) = 6x$, and $ea = a \circ e = a(e(x)) = 3(2x) = 6x$.
- $\boldsymbol{af \to eb}$: $af = f \circ a = f(a(x)) = 3(2x) + 1 = 6x + 1$, and $eb = b \circ e = b(e(x)) = 2(3x) + 1 = 6x + 1$.

- $\boldsymbol{ag \rightarrow fa}$: $ag = g \circ a = g(a(x)) = 3(2x) + 2 = 6x + 2$, and $fa = a \circ f = a(f(x)) = 2(3x + 1) = 6x + 2$.

- $\boldsymbol{be \rightarrow fb}$: $be = e \circ b = e(b(x)) = 3(2x+1) = 6x+3$, and $fb = b \circ f = b(f(x)) = 2(3x + 1) + 1 = 6x + 3$.

- $\boldsymbol{bf \rightarrow ga}$: $bf = f \circ b = f(b(x)) = 3(2x + 1) + 1 = 6x + 4$, and $ga = a \circ g = a(g(x)) = 2(3x + 2) = 6x + 4$.

- $\boldsymbol{bg \rightarrow gb}$: $bg = g \circ b = g(b(x)) = 3(2x + 1) + 2 = 6x + 5$, and $gb = b \circ g = b(g(x)) = 2(3x + 2) + 1 = 6x + 5$.

Hence, these rules are all correct.

The C rules take advantage of the fact that the c symbol is a binary 1, and, in a strictly binary string, it is the most significant bit of the number the string corresponds to. When the ternary symbol is adjacent to the c symbol, we apply one of the three c rules to convert the ternary symbol into binary symbol(s). These rules also preserve the number the string represents, shown here:

- $\boldsymbol{ce \rightarrow cb}$: $ce = e \circ c = e(c(x)) = 3(1) = 3$, and $cb = b \circ c = b(c(x)) = 2(1)+1 = 3$.

- $\boldsymbol{cf \rightarrow caa}$: $cf = f \circ c = f(c) = 3(1)+1 = 4$, and $caa = a \circ a \circ c = a(a(c(x))) = 2(2(1)) = 4$.

- $\boldsymbol{cg \rightarrow cab}$: $cg = g \circ c = g(c) = 3(1) + 2 = 5$, and $cab = b \circ a \circ c = b(a(c(x))) = 2(2(1)) + 1 = 5$.

Hence, we have shown that the A, B, and C rules all preserve value, and the D rules correctly apply the $3N + 1$ mapping.

Here is how one can run the SRS and preserve an ordering we know to be valid:

1. Take the initial input number, and convert it to binary. Make the leading 1 a c symbol, and all 0's and other 1's a's and b's, respectively.
2. Append a d to the end of the string.
3. Until we have the string cd:
    - Apply the appropriate D rule.
    - If a ternary character is generated, apply A and B rules until the ternary symbol and the c are adjacent, then apply the appropriate C rule.

This order of applying the SRRs is correct, because one takes a string that is strictly in binary symbols and applies the correct D rules until a ternary character is generated. In this SRS, the only D rule that generates a ternary character is rule D_2. When a ternary symbol is generated, we immediately apply A, B, and C rules until the ternary symbol is converted into binary symbol(s). The number is not changed during application of the A, B, and C rules, making the ordering correct. We use this to investigate properties of the Collatz SRS, which we will discuss in Chapter 6.

If we take the Collatz SRS and find matrix functions for all symbols that cause all vector representations of input strings to decrease when SRRs are applied,

then we believe we can prove the Collatz Conjecture. We don't have a formal proof, because we don't have a proof that applying the Collatz SRRs in arbitrary order results in the same output string. However, we strongly believe that termination of the Collatz SRS implies termination of Algorithm 1, since the rewrite system operates on input strings the same way as Algorithm 1 would on positive integers.

5.4 Proving Collatz Conjecture Results

This section discusses some of the results of attempting to solve the Collatz Conjecture using matrix interpretation [5]. Heule initially tried to run the Collatz SRS on state-of-the-art matrix interpretation solvers like AProVE [11]. However, 4 of the 11 rules needed to be removed before the system could be solved. As a result, a custom matrix interpretation solver was built by Heule specifically for this problem. Heule, as of now, has been unable to prove termination of the entire 11 rule system with matrix interpretation. However, any combination of 10 of the 11 rules has been proven.

Heule has also run the matrix interpretation on several Collatz Variants: the ones that remain unproven in this thesis; namely Collatz Variants 1, 5, 7, and $\{5, 7\}$; also have not been proved by the matrix interpretation system yet. The ones which were proven; variants 2, 3, 4, and 6; were proven by the system. Hence, the motivation for investigating hardness of the unproven Collatz Variants came about, as did the motivation for writing this thesis. Further work on this and related problems will be done under NSF grant CCF-1813993 in the coming years [12].

Chapter 6

Collatz SRS Analysis

Now that we have explained Heule's and Aaronson's attempts to solve the Collatz Conjecture, we focus on analyzing the Collatz SRS further. We do so by simulating the Collatz SRS and various modifications of it in order to analyze the number of steps these SRSes take. First, we discuss a program we created that replays the Collatz SRS for any input number, which is used throughout this chapter. We then attempt to establish a reasonable bound for number of rewrite steps by comparing classical hardness records from Chapter 4. We use this bound to compute reasonable hardness measures for modifications of the Collatz SRS that align with unproven Base 8 Collatz Variants. Finally, we explore a modification to Aaronson's rewrite system which reduces rewrite steps.

6.1 Simulating the Collatz SRS

We wrote a program that simulates the Collatz SRS in Java. It takes two different required inputs: some positive integers (either one number, or a batch of numbers, one per line), and a string file which has one SRR per line in the format "input output", which represents the rule $input \to output$. The # character can be used to comment out a line. This is a convenience to easily remove a rule to create

SRRs that correspond to Collatz Variants.

The output is a file that shows how the initial number is transformed into a valid input string for the Collatz SRS, and the terms that this initial string get turned into as we run the Collatz SRS. The number which each string represents is also written, as well as a header cell that counts the total number of steps. If a batch of numbers is run, each number gets a separate file. Also, an aggregate file can be printed for batches which has rows that list, for each input number, what the number is, the final number it eventually turns into, and the number of rewrite steps.

The program converts an input number into a binary rewrite string with symbols a, b, c, and d. The rewrite term is stored in a character array that has a "sliding" string in it. This is an efficient way to take advantage of Aaronson's SRS, because a number only grows from the c term, and shrinks from the d term. Hence, the spare space of the array is past the c term, and any time a D rule decreases size, we move in the end of the d symbol, and all other symbols past the ending d symbol are thrown out if more space is needed. For instance, when we apply rule D_2, the a term gets replaced with a d term, and a pointer denoting the end of the string gets moved to the new d symbol. If we run out of space in the array, we double the size of it, and discard any unnecessary symbols past the first d symbol.

As discussed in Section 5.3, we don't apply SRRs in arbitrary order. Given a rewrite string with only symbols a, b, c, and d, we check to see if any D rule can be applied. If not, the program terminates. If we do find a D rule, then apply it, and check if a ternary symbol is generated by it. If so, we apply the A and B rules

to swap the ternary symbol with binary symbols until we can apply a C rule, which removes the ternary symbol.

The code can be accessed via a public GitHub repository located at `https://github.com/mdenend/CollatzRewriteSystem`. A README file is included that explains how to run the code and available options.

6.2 Determining Extra Steps the Collatz SRS Adds

Looking at how the Collatz SRS operates, one can establish a reasonable bound on how many more steps the Collatz SRS adds compared to the algebraic method of computing Collatz Sequences. Starting with a number that is purely in binary symbols a and b, plus the placeholder symbols c and d, there are two different events that can occur. When a is the symbol to the immediate left of the d, we divide by 2. It only takes one step to complete this division. On the other hand, when a b symbol is to the immediate left of the d, we turn the b symbol into a g symbol to effectively compute $(3N + 1)/2$, and, following the established order, apply A, B, and C rules to convert the ternary symbol to binary. This adds $\Theta(m)$ rewrite steps per odd number.

We can compare the number of steps that the rewrite system takes to classical hardness records from Chapter 4. Let m in this case be the bits for classical hardness record r. If we do in fact apply a factor of $\Theta(m)$ more steps per odd number in the Collatz SRS, then classical hardness should be pretty close to the rewrite steps for odd numbers divided by $\log^2 r$. We call this "rewrite hardness". We compute the rewrite hardness to classical hardness by inputting classical hardness records into

the Collatz SRS. The results of this analysis are presented in Figure 6.1. From this graph, it appears that dividing the odd number of rewrite steps by $\log^2 r$ creates a reasonably tight bound against classical hardness, so the Collatz SRS adding a factor of $\Theta(m)$ rewrite steps per odd number appears to be correct.

6.3 Collatz SRS Subproblem Analysis

Now we know that the Collatz SRS adds a factor of m more steps compared to the steps in Collatz Sequences, we look back at the Collatz Variants. In this section, we modify the Collatz SRS to create various "Collatz Subproblems". These subproblems are modifications of the SRRs for the Collatz SRS with the goal of having Collatz Subproblems that align with our previously defined Collatz Variants. We show how to create Collatz Subproblems, then run computation using our Collatz SRS simulation.

6.3.1 Modified Base 8 Rewrite System

We modified the Collatz SRS to make it tie into Collatz Variants. Recall the D rules of the Collatz SRS:

$$D_1 : ad \to d \qquad 0 \pmod 2$$

$$D_2 : bd \to gd \qquad 1 \pmod 2$$

D_1 handles numbers congruent modulo to 0 (mod 2) by dividing the number by 2, while D_2 handles numbers congruent modulo to 1 (mod 2) by computing $(3N+1)/2$.

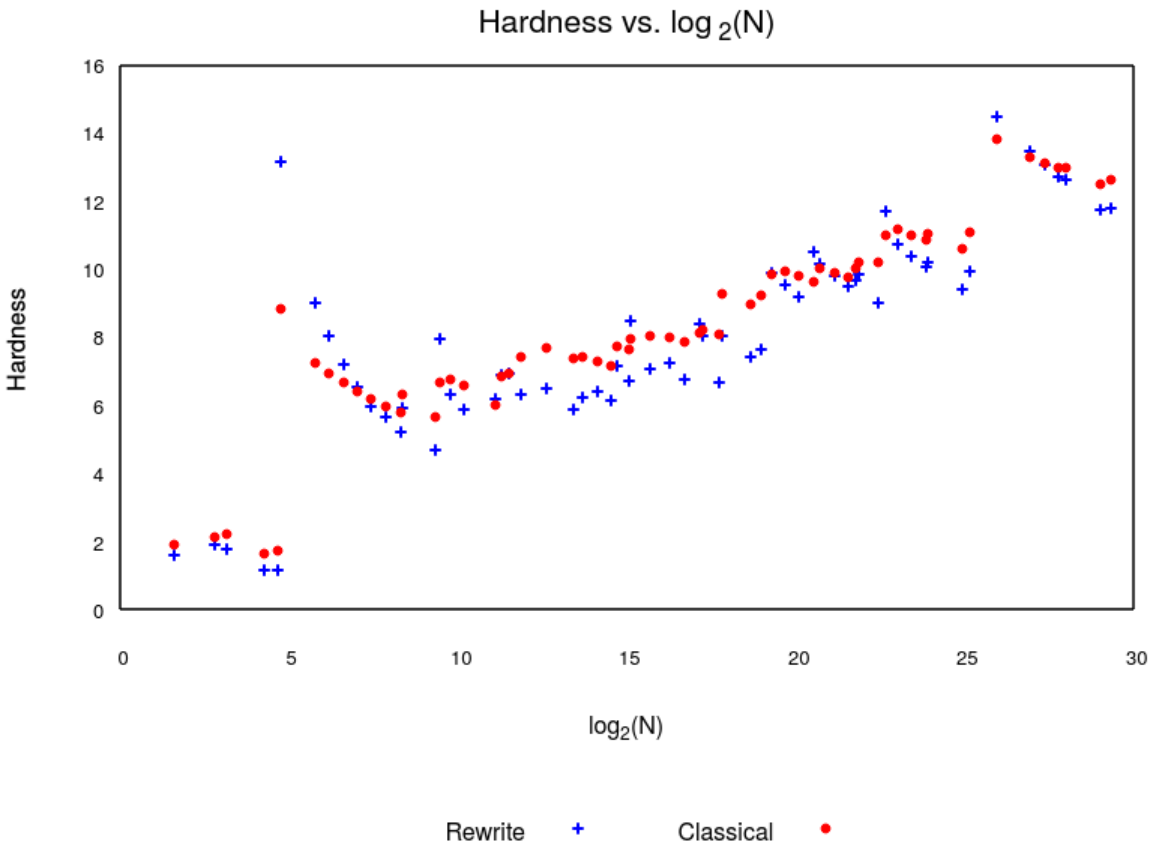

Figure 6.1: This graph compares, on the same set of input numbers, classical hardness as defined in Chapter 4, and rewrite hardness, which is the number of steps divided by the number of input bits squared. The number of bits for the records ($\log_2 r$) is the x-axis, and the hardness is the y-axis.

Also note that all input strings for these rules are just one bit, since the placeholder d is not a bit.

We can expand the input from 1 bit to 3 bits and come up with 8 D rules to create an equivalent system:

$$\begin{aligned}
aaad &\to aad && 0 \pmod 8\\
aabd &\to ebad && 1 \pmod 8\\
abad &\to abd && 2 \pmod 8\\
abbd &\to fabd && 3 \pmod 8\\
baad &\to bad && 4 \pmod 8\\
babd &\to gaad && 5 \pmod 8\\
bbad &\to bbd && 6 \pmod 8\\
bbbd &\to gbbd && 7 \pmod 8
\end{aligned}$$

Observe that the input substrings for these rules all correspond to a node in graph G_8. The rules take in substrings that represent numbers congruent modulo to 0-7 (mod 8), meaning the numbers represented by the whole input strings would be visiting nodes 0-7, respectively. The outputs are the result of dividing by 2 or multiplying by 3 and adding 1, in context of the SRS. All of the odd node rule outputs, like rule D_2 in the original system, are just a combination of several rules, which ensure that the output string is not longer, and it reduces a couple of steps by moving the ternary term toward the front. All of the even node rules are just the same exact rule D_1 in the original system, so we can remove any even rules and replace them with the

original D_1. We can also eliminate any extra a terms in the output of odd rules that we know would be eliminated by rule D_1. We ended up using the following SRRs in the Base 8 Collatz SRS:

$$D_{8_1} : ad \to d \qquad 0 \pmod 2$$

$$D_{8_2} : aabd \to ebd \qquad 1 \pmod 8$$

$$D_{8_3} : abbd \to fabd \qquad 3 \pmod 8$$

$$D_{8_4} : babd \to gd \qquad 5 \pmod 8$$

$$D_{8_5} : bbbd \to gbbd \qquad 7 \pmod 8$$

Because these rules were constructed using only SRRs in the Collatz SRS that we know to be correct, we know these new D rules, plus the A, B, and C rules, are equivalent to the original Collatz SRS. However, removing one of these rules could make the system easier to prove, which in turn constructs something much like a Collatz Variant. We present such an SRS with rule D_{8_2} removed:

$$D_{8_1} : ad \to d \qquad 0 \pmod 2$$

$$D_{8_3} : abbd \to fbbd \qquad 3 \pmod 8$$

$$D_{8_4} : babd \to gd \qquad 5 \pmod 8$$

$$D_{8_5} : bbbd \to gbbd \qquad 7 \pmod 8$$

Since we removed the SRR that corresponds to input 1 (mod 8), termination of this SRS should imply termination of Collatz Variant 1, as removing a rule causes any string with this input to terminate the system. This modified Collatz SRS is called Collatz Subproblem 1 in this paper. Note how removing an SRR is equivalent to

adding the corresponding termination condition in Algorithm 2. We analyzed this and other Collatz Subproblems that should imply termination of unproven Collatz Variants.

6.3.2 Defining Measures for Subproblem Analysis

For the Collatz Subproblem computations, instead of defining hardness by number of odd numbers, we define hardness based off of the total steps applied, because of the fact that odd numbers add significantly more steps than even numbers. Define the following numbers, given some input number N:

- $f_r(N)$: The total number of Collatz SRS rewrite steps for N before it converges to 1.
- A: The base avoidance set, same as used in Algorithm 2 and Chapter 4. In this chapter, $A \subseteq \{1, 5, 7\}$ and $A \neq \varnothing$.
- Collatz Subproblem A: Defined much in the same way as Collatz Variant A, but with modified Collatz SRSes instead. Let A be the base avoidance set. Collatz Subproblem A is a modified Base 8 Collatz SRS that, for all $a \in A$, the SRR corresponding to $a \pmod 8$ is removed. Therefore, any string that has the input to a dropped rule will caused the modified Collatz SRS to terminate in the same manner as Collatz Variants. Listing several Collatz Subproblems is done in the same way as Collatz Variants, so we can write Collatz Subproblems 1 and $\{5, 7\}$ to mean these two modifications of the Base 8 Collatz SRS.

- Record Sequence for Collatz Subproblem A: This is the same record sequence as for Collatz Variant A, which was defined in Chapter 4, but we start with an input string representing the first number in the Record Collatz Variant Sequence, and run rewrite rules until we reach the number whose input string causes Collatz Subproblem A to terminate (same number that causes Collatz Variant A to terminate). We only run SRSes for the record numbers we determined with the algebraic Collatz Program defined in subsection 4.2, as running computation for only the Collatz SRS on all odd numbers up to 1 billion would take an extremely long time.

- $R(N, A, 8)$: The number of rewrite steps that the record sequence for Collatz Variant A for base 8 and number N takes when run as Collatz Subproblem A instead.

We define a hardness measure: H_{SRS}, where $H_{SRS} = \frac{R(N,A,8)}{\log_2^2 N}$. This effectively computes the hardness of the SRRs that corresponds to determining termination of Collatz Variant A, but H_{SRS} also takes into account the extra factor of m steps that the Collatz SRS adds. The number of bits we divide by is determined by our starting number, not the first number in the record length sequence. This is the same as for our Collatz Variant computation in Chapter 4.

6.3.3 Subproblem Hardness Analysis

Figure 6.2 shows the analysis of hardness for the modified SRS with four different Collatz Subproblems: 1, 5, 7, and $\{5, 7\}$. Unlike the analysis for the Collatz

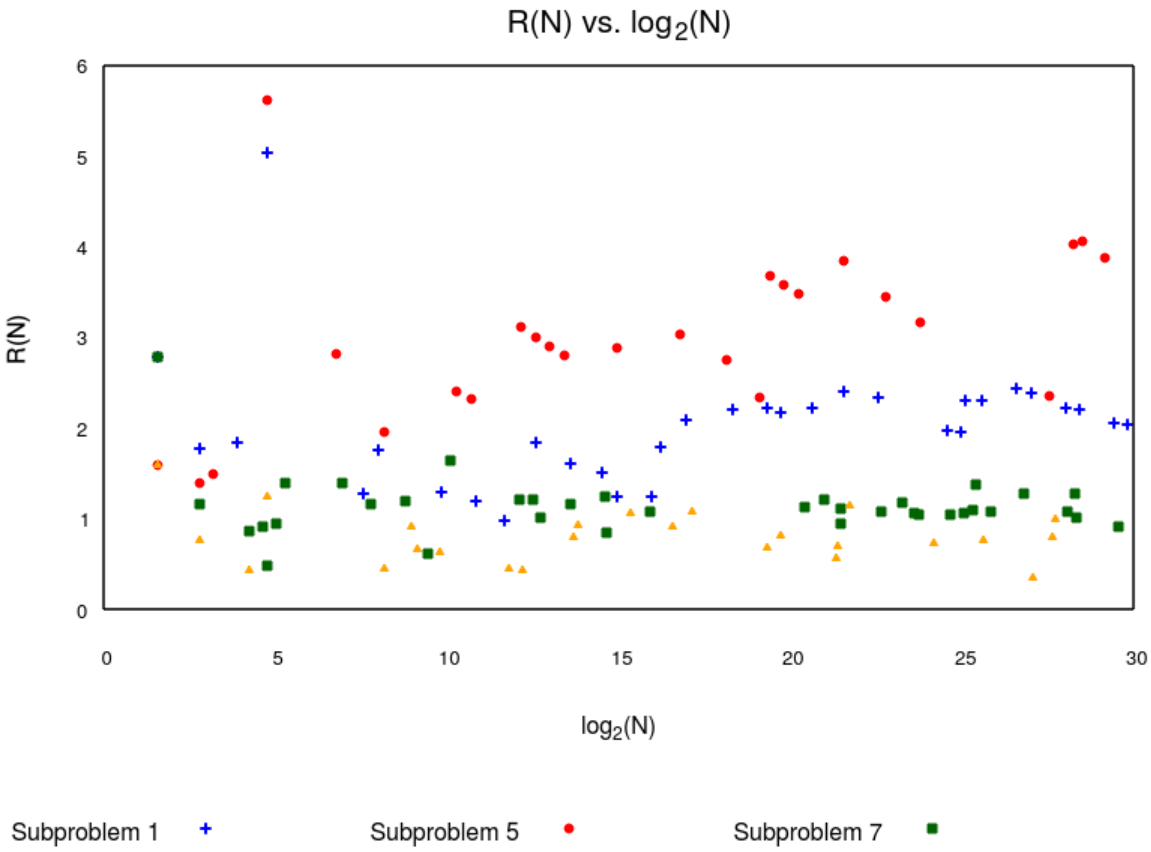

Figure 6.2: This graph visualizes how, for records r, the R values for subproblems 1, 5, and 7 compare to each other. The number of bits for the records ($\log_2 N$) is the x-axis, and the hardness measure R as defined in section 6.3.2 is the y-axis.

Variants, we have clearer separation. Subproblem 5 has the highest H_{SRS} on almost all of its records. Since Collatz Variant 5 record sequences tend to grow to very large numbers, these extra bits that the large numbers generate add even more steps needed by the rewrite system. Further, subproblem 5 appears to get harder as the number of bits increases, a trend not apparent in the Collatz Variant 5 H measure. Subproblem 1 exhibits the growth as numbers get bigger, but tapers off once again for numbers more than 20 bits, much like Collatz Variant 1. Subproblem 7 shows no increase in H_{SRS} as numbers get larger. This suggests subproblem 7 might be easier to solve than subproblems 1 or 5. Subproblem $\{5, 7\}$ has the lowest hardness, which makes sense given that it has one less rule than any of the other three subproblems.

6.4 Further SRS Rule Modifications

It is possible that the set of SRRs in the Collatz SRS can be optimized to reduce number of rewrite steps. These optimizations may possibly lead to solutions for the subproblems listed in this chapter. We present one modification of the Collatz SRS by replacing the base D rules with rules that have both input and output correspond to odd numbers. We then look at the results of doing so.

6.4.1 Odd Collatz SRS

It is possible to replace both D_1 and D_2 in the Collatz SRS with the following set of rules that calculates with strictly odd numbers:

$$D_{o1} : bbd \to gbd \qquad 3 \pmod 4$$

$$D_{o2} : aabd \to ebd \qquad 1 \pmod 8$$

$$D_{o3} : babd \to bd \qquad 5 \pmod 8$$

Notice that rule D_{o1} and D_{o2} are borrowed from the base 8 modification of the Collatz SRS. D_{o1} combines both rule D_{8_3} and D_{8_5}, removing the leading a or b, respectively. D_{o1} has the A, B, and C rules determine the next symbol(s). Rule D_{o2} is the same as rule D_{8_2}.

Rule D_{o3} is not as intuitive at first glance, but we can look at rule D_{8_4}, which has the same input as D_{o3}, and see that the output for D_{8_4} is the same as the normal D_2 rule, gd, so we can "step backwards" using rule D_2 to allow for us to get bd as our output, and hence, have a full set of correct D rules that go from odd number to odd number.

Observe that all of the D_o rules have a left handed side and a right handed side ending with bd. As a consequence, we can remove the b symbol in the second to last symbol for both the input string and the rules, resulting in the following SRRs:

$$D_{o1^*} : bd \to gd \qquad 3 \pmod 4$$

$$D_{o2^*} : aad \to ed \qquad 1 \pmod 8$$

$$D_{o3^*} : bad \to d \qquad 5 \pmod 8$$

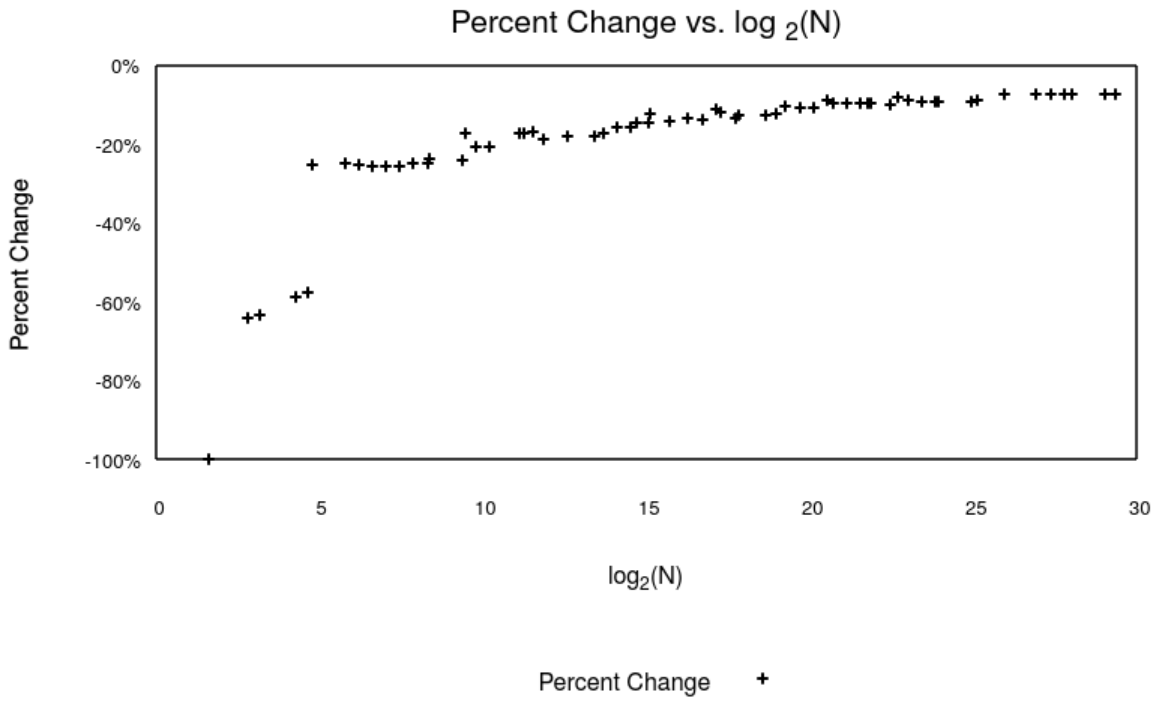

Figure 6.3: This graph shows what percentage the odd rules change the rewrite steps, as opposed to the original Collatz SRS. The number of bits for the records ($\log_2 N$) is the x-axis, and the percent decrease is the y-axis.

6.4.2 Change in Rewrite Steps

Figure 6.3 shows the percentage of rewrite steps the Odd Collatz SRS changes, compared to the original Collatz SRS, when run on classical hardness records. The Odd Collatz SRS reduces the number of steps greatly for smaller numbers. However, as the numbers get larger, the percentage reduction of steps lowers significantly. This is probably because records for larger numbers have $3N + 1$ sequences that tend to have more odd numbers, which add in more rewrite steps. That said, this

modification to the rewrite system does overall reduce the number of rewrite steps. Other SRR modifications for the Collatz SRS should be considered, as the right set of clever rules may prove termination of Collatz Subproblems, and perhaps, even prove that the Collatz Conjecture holds.

Chapter 7

Conclusion

In this thesis, we analyzed the Collatz Conjecture and simpler, yet still challenging variants, and proposed a hardness prediction for determining the answers that these variants terminate. We started by building a program that investigates Collatz Variants by running many $3N + 1$ sequences for all odd numbers up to 1 billion, and seeing how many odd numbers occur in record breaking sequences that avoid terminating the Collatz Variants.

We also investigated the Collatz SRS that Aaronson created and that Heule tried to prove with matrix interpretation. Even though this methodology has been unable to solve the Collatz Conjecture so far, we think it still has promise. We also built a simple program that ran the Collatz SRS and determined the hardness for subproblems, finding out that the hardness varies more than in the algebraic case. We also explored other modifications to try and reduce rewrite steps needed.

We believe this approach for trying to solve the Collatz Conjecture, as well as Collatz Variants, has merit and needs further investigation. Further work will be done under the recently approved grant [12].

Bibliography

[1] A. M. Turing, "On Computable Numbers, with an Application to the Entscheidungsproblem," *Proceedings of the London Mathematical Society*, vol. s2-42, no. 1, pp. 230–265, 1936.

[2] J. C. Lagarias, "The 3x+1 problem: An annotated bibliography (1963–1999) (sorted by author)." `https://arxiv.org/abs/math/03092`, Sept. 2003.

[3] J. C. Lagarias, "The 3x+1 Problem: An Annotated Bibliography, II (2000-2009)." `https://arxiv.org/abs/math/0608208`, Aug. 2006.

[4] E. Roosendaal, "On the 3x+1 Problem." `http://www.ericr.nl/wondrous/`, 2018.

[5] M. J. H. Heule and S. Aaronson. Personal communication, Sept. 2017.

[6] D. Thain, T. Tannenbaum, and M. Livny, "Distributed Computing in Practice: The Condor Experience: Research Articles," *Concurr. Comput. : Pract. Exper.*, vol. 17, pp. 323–356, Feb. 2005.

[7] A. Biere, A. Biere, M. Heule, H. van Maaren, and T. Walsh, *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications*. Amsterdam, The Netherlands, The Netherlands: IOS Press, 2009.

[8] D. Hofbauer and J. Waldmann, "Termination of $aa \to bc, bb \to ac, cc \to ab$," *Inf. Process. Lett.*, vol. 98, pp. 156–158, May 2006.

[9] D. Hofbauer and J. Waldmann, *Termination of String Rewriting with Matrix Interpretations*, pp. 328–342. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006.

[10] J. Endrullis, J. Waldmann, and H. Zantema, *Matrix Interpretations for Proving Termination of Term Rewriting*, pp. 574–588. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006.

[11] J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke, "Automated termination proofs with aprove," in *Rewriting Techniques and Applications* (V. van Oostrom, ed.), (Berlin, Heidelberg), pp. 210–220, Springer Berlin Heidelberg, 2004.

[12] M. J. H. Heule and S. Aaronson, "SHF: Small: MaPaMaP: Massively Parallel Solving of Math Problems." NSF Award CCF-1813993.

Vita

Matthew Alexander Denend was born in Spokane, Washington. He received the Bachelor of Science degree in Electrical Engineering, cum laude, from The University of Washington, Seattle, in 2012. He worked as a Packet Core Performance Engineer at T-Mobile in Bellevue, Washington for 3 years. He decided to pursue a Master of Science in Computer Science degree, and after he was accepted to The University of Texas at Austin in 2015, he left his job at T-Mobile to move to Austin, Texas and became a full-time student.

Email address: mdenend@gmail.com

This thesis was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.