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

Marijn Heule, 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

THE UNIVERSITY OF TEXAS AT AUSTIN

August 2018

Dedicated to my late mother, who never let her three battles with cancer stop her from loving her two sons and her husband.

Abstract

Challenging Variants of the Collatz Conjecture

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

Supervisors: Scott Aaronson
Marijn Heule

This hasn't been written by me yet. Going to do this last.

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 seemingly simple 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.

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 [18], 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. Many

approaches have been tried since 1963 [11] [12]. Further, empirical evidence suggests that the Collatz Conjecture is true: according to a website maintained by Eric Roosendahl [14], 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 $b = 8$ ($\text{Col}_{\text{mod}}(N, \{1\}, 8)$ is shorthand for this problem), 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 to attempt to craft a program to determine if Algorithm 1 always halts [6]. 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. An investigation on the difficulty of solving Algorithm 2 for certain values of A and b using a rewrite system that represents the Collatz Conjecture instead of algebra is another contribution by this thesis.

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, including both solvable ones and unsolved ones, including $\mathrm{Col}_{\mathrm{mod}}(N, \{1\}, 8))$. Chapter 4 analyzes the difficulty of these variants using algebra. Chapter 5 provides a brief background on SAT Solvers and String Rewrite Systems, how SAT solvers can be used to prove termination of them, describes Aaronson's rewrite system that Heule tried to use with matrix interpretation and SAT solving to prove, and the results of this 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 the paper to talk about the $3x+1$ problem:

- **$3x+1$ mapping**: A one step application of $Col(N)$ to some input number N.

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

$$x_0 = x = \text{ initial input number}$$
$$x_{i+1} = \begin{cases} x_i/2 & \text{if } x_i \text{ is even} \\ 3x_i + 1 & \text{if } x_i \text{ is odd} \end{cases}$$

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

- **Collatz Variant *A*:** When we say "Collatz Variant A," we use Algorithm 2, which was defined in the introduction. A shorthand for referring to Algorithm 2 is $\text{Col}_{\text{mod}}(N, A, b)$, where $A = \{a_1 \dots a_s\}$ is all of the numbers modulo b that cause termination of the algorithm. Note that for all $a \in A$, $0 \leq a < b$, and $A = \emptyset$ is the same as Algorithm 1.

- **Collatz Variant A:** The vast majority of our analyis is when $b = 8$, so when we say Collatz Variant A, it is shorthand for $\mathrm{Col}_{\mathrm{mod}}(N, A, 8)$ For singleton sets A, we omit the braces normally around sets. We can list several Collatz Variants together, for instance, Collatz Variants, 1, 5, 7, and $\{1, 5\}$ we mean the three variants $\mathrm{Col}_{\mathrm{mod}}(N, \{1\}, 8)$, $\mathrm{Col}_{\mathrm{mod}}(N, \{5\}, 8)$, $\mathrm{Col}_{\mathrm{mod}}(N, \{7\}, 8)$, and $\mathrm{Col}_{\mathrm{mod}}(N, \{1, 5\}, 8)$.

- **Collatz String Rewrite System(SRS):** Created by Aaronson **??**, and introduced in chapter 5, along with background of string rewrite systems, this is used in the latter portion of this paper as an alternative way of expressing the Collatz Conjecture.

- **Number of bits:** For an input number x, we say that, written in binary, it has m bits.

Chapter 3

Alternative Termination Conditions

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

3.1 Defining a Collatz base b graph

Define $G_b = (V, E)$ to be a "Collatz base b 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 String Rewrite System 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 $[0, b-1]$. Vertex $v \in V$ corresponds to $v \pmod b$. This paper uses the term "node v" as shorthand to correspond to the number v in $v \pmod b$, as opposed to "vertex" v which corresponds to the actual vertex of the graph. Let x_i be the i^{th} step of the $3x + 1$ sequence for input number x, and x_{i+1} be the step after. $E = V \times V$ is a set of directed edges, where, for nodes $u, v \in V$, $(u, v) \in E$ if and only if $x_i \equiv u \pmod b$ and $x_{i+1} \equiv v \pmod b$ for some x_i

and x_{i+1}.

3.2 Lemmas that hold for any base b Collatz Graph

This section contains some lemmas that are used throughout this Thesis. First, we start with a lemma about the number of node transitions in any Collatz base b graph $G_b = (V, E)$.

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

Proof. Take a number x_i that corresponds to node v. First, let us consider the case where node v is even. When we divide by 2, we just remove the lowest 0. The new bit at index $k-1$ can be either a 0 or a 1, allowing for two options for even nodes.

Now, take x_i that corresponds to a node v that is odd. Multiplying x_i by 3 and adding 1 will make the binary string grow at least one bit longer (since $3 \cdot x_i = 2 \cdot x_i + x_i$), giving us only one option for the least significant k bits, meaning x_{i+1} can only correspond to one node $w \in V$, 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 $3x + 1$ 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 $N/2$ 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 bits? If it is a 0, then our new number $N/2 \equiv 0 \pmod{b}$, 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 cutting off 0's, we eventually reach a point where the least k significant bits are $1000\ldots0$, meaning we transition to 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 $b - 1$. We call this the "1 Consumption" Lemma.

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

		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 $x + 2x + 1$.

number congruent modulo to $b - 2 \pmod b$, after division by 2, actually becomes $b/2 - 1 \pmod b$ instead. We will do so using long multiplication in binary, adding the 1 in as well for shorthand.

Let x be a arbitrary number 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 indices correspond to unknown bits. Let 1_{+1} correspond to the adding of 1 after multiplying by 3, which is added to the 0^{th} index of x. Let c_j denote the unknown carry for the j^{th} index of bits being added. Let y be the result after $3x + 1$ is computed, and let the j^{th} index of y be the same index as x. The multiplication is referenced in Figure 3.1.

Observe that, in binary, 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. In the multiplication we show in Figure 3.1, we replace the extra 0 vacated by $2x$ with the addition of 1, 1_+1, allowing us to perform $3x + 1$ with just one addition instead of two.

Observe 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 index 0, 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 lowest 0 from y. 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 cycle.

Now we show that this $b - 2 \to b - 1 \to b - 2$ 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 arbitrary 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 indices of y between 1 and $j - 1$ will be 1, as each time, we add $1 + 1$ and carry over a 1. However, when we get to index i, we add $1 + 0$ plus a carry of 1, making bit $y_j = 0$. Since after we divide by 2 we right shift all bits one index, bit y_j now moves to index $j - 1$. We repeat this process a total of $j-k+1$ steps. 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} will be 0, as we carry over a 1, and add only one other 1, since bit x_{m+1} is an implied 0. Since bit $y_{m+1} = 0$ we right shift it to index m after dividing by 2. We then follow case 1, where $j = m$, and the cycle will also break in this case.

Hence, no $b-2 \to b-1 \to b-2$ cycle for any legal value of $b = 2^k$ can have an indefinite cycle. □

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 number of even nodes in the cycle, and v_o be the number 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 let x be the number at some point in the cycle. Run through the cycle once, back to the same vertex that we started at with x. We visited $|V_e|$ vertices in the cycle, so we divided x by $2^{|V_e|}$ after one trip around the cycle. We also visited $|V_o|$ vertices in the cycle, so each time we multiplied x by 3, and overall, multiplied x by about $3^{|V_o|}$. (We also added one to x, but this is asymptotically insignificant compared to multiplying by 3 or dividing by 2, so ignored in this proof.)

So after we visited the cycle once, we computed a number approximately equal to $\frac{x \cdot 3^{|V_o|}}{2^{|V_e|}}$. Since $2^{|V_e|} > 3^{|V_o|}$, $\frac{x \cdot 3^{|V_o|}}{2^{|V_e|}} < x$. Therefore, one of two things must happen:

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

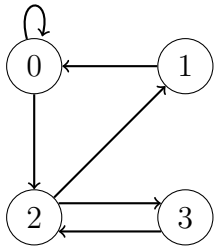

Figure 3.2: The defined graph when $b = 4$. There are 4 nodes, each one corresponding to a value mod 4.

2. The cycle is eventually broken.

In both cases, the cycle cannot continue indefinitely. □

3.3 Base 4 Graph and Variants

In this section, we build a simple example of a Base 4 graph and show that we can prove $\mathrm{Col}_{\mathrm{mod}}(N, A, 4)$ terminates for any nonempty $A \subseteq \{0, 1, 2, 3\}$. The graph is shown in Figure 3.2.

There are 4 different nodes: one for each integer between 0 and 3. Because $4 = 2^2$, we are looking at the last 2 bits of each number. For example, a number congruent modulo to 0 (mod 4) has 00 as the 2 least significant bits, while a number congruent modulo to 2 (mod 4) has 10 as its 2 least significant bits.

3.3.1 Base 4 graph construction

Figure 3.2 shows the Base 4 Collatz Graph, and this subsection describes the construction of it. First, we will start with the even nodes, then the odd nodes:

- Node 0 corresponds to the number ending with binary string "00". Removing the last 0 leaves either "00", looping to 0 (mod 4), or "10", changing it to 2 (mod 4).

- Node 2 corresponds to the number ending with binary string "10". Removing the last 0 leaves either "01", changing it to 1 (mod 4), or "11", changing it to 3 (mod 4).

Now, the odd nodes:

- Node 1 corresponds to the number ending with binary string "01". Multiplying this by 3 and adding 1 results in the binary number ending in "$y_2$00", where y_2 is an unknown bit. Even though we don't know y_2, the number is still 0 (mod 4).

- Node 3 corresponds to the number ending with binary string "11". Multiplying this by 3 and adding 1 results in the binary number ending in "$y_3y_2$10", which is still 2 (mod 4).

3.3.2 Base 4 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. This 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. 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 is this, we are already done.
- 3: This must, by definition, transition after the $3x+1$ mapping has been applied in one step.
- 1 and 0: 1 must transition to 0 after applying the $3x + 1$ mapping once. For 0, use Lemma 3.2.2 (the "0 cycle" lemma) for $b = 4$, and the number will eventually transition to the 2 node.

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

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

Proof. We use Lemma 3.3.2 to show that node 2 must be visited, and Lemma 3.2.3 to show that the cycle between nodes 2 and 3 cannot continue indefinitely, so node 2 must eventually transition to 1, proving termination of this variant. □

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

Proof. Given Lemma 3.3.3, we know we must visit node 1, and given Lemma 3.2.1, an odd node can only have one outgoing edge, so it must transition to 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 $2 \to 1 \to 0 \to 2$ 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 either $\text{Col}_{\text{mod}}(N, \{3\}, 4)$ terminates by eventually becoming 1, or the cycle is broken, and the only other node to visit outside this cycle is node 3, causing $\text{Col}_{\text{mod}}(N, \{3\}, 4)$ to always terminate. □

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 Graph and Variants

We showed that, using the Base 4 graph, $\text{Col}_{\text{mod}}(N, A, 4)$ 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 ran into a problem, and could not prove all variants of $\text{Col}_{\text{mod}}(N, A, 8)$. This will motivate further computation undertaken in chapter 4, as we try to determine how hard figuring out these unproven variants are.

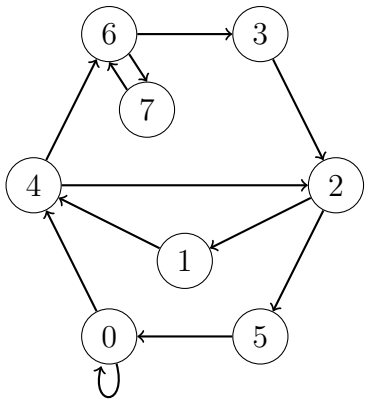

Figure 3.3: The defined graph when $b = 8$. There are 8 nodes, each one corresponding to a value mod 8.

Figure 3.3 shows the base 8 graph. There are 8 different nodes, since $8 = 2^3$, and we are looking at the last 3 bits of each number.

3.4.1 Base 8 graph construction

Like the base 4 graph, let us show how to construct the base 8 graph. Like in the base 4 case, let us 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.

- Node 0 corresponds to the binary string "000". Removing the last 0 leaves either "000", looping to 0 (mod 8), or "100", changing it to 4 (mod 8).
- Node 2 corresponds to the binary string "010". Removing the last 0 leaves either "001", changing it to 1 (mod 8), or "101", changing it to 5 (mod 8).

- Node 4 corresponds to the binary string "100". Removing the last 0 leaves either "010", changing it to 2 (mod 8), or "110", changing it to 6 (mod 8).
- Node 6 corresponds to the binary string "110". Removing the last 0 leaves either "011", changing it to 3 (mod 8), or "111", changing it to 7 (mod 8).

Now, the odd nodes. Let y_3 and y_4 be unknown bits.

- Node 1 corresponds to the binary string "001". Multiplying this by 3 and adding 1 results in the string "100", or 4 (mod 8).
- Node 3 corresponds to the binary string "011". Multiplying this by 3 and adding 1 results in the string "$y_3$010", which is 2 (mod 8).
- Node 5 corresponds to the binary string "101". Multiplying this by 3 and adding 1 results in the string "$y_4y_3$000", which is 0 (mod 8).
- Node 7 corresponds to the binary string "111". Multiplying this by 3 and adding 1 results in the string "$y_4y_3$110", which is 6 (mod 8).

3.4.2 Cycle analysis

Since we do not have proofs for all Collatz Variants of base 8 with singleton A, we analyzed whether certain cycles can last indefinitely. Some of these analyses can be used in proving some Collatz Variants, whereas others that are unknown can be tied to variants we have not proven yet.

- 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.
- $4 \to 6 \to 3 \to 2 \to 1 \to 4$ 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 x such that $x \equiv 4 \pmod 8$, and follow the $4 \to 6 \to 3 \to 2 \to 1 \to 4$ cycle once, we turn x into $(9x + 20)/8 = \frac{9}{8}(x + 20) - 20$. If we were to repeat the cycle k times, we would turn x into $\frac{9}{8}^k(x+20)-20$. This quantity must be an integer for all k if the cycle is to continue forever. However, $x + 20$ 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

$(4 \to 6 \to 3 \to 2 \to 1)$, and another that causes the cycle to decay by a factor of 4/3 $(4 \to 2 \to 1)$. This combintation of cycles is one of the more challenging cases to show that we cannot continue indefinitely and no proof is known that we must break out of this combination of cycles, by visiting either node 5 or 7. Naturally, a proof 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. Since each iteration of the $6 \to 7 \to 6$ adds one even node and one odd node to the aready challenging two cycle case. Finding a proof for this case may be harder than the two cycle case. 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 Dominiance 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 the variant $\{5, 7\}$, we have two conflicting

cycles: the $6 \to 7 \to 6$ cycle which cause the number to grow by $3/2$ 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 at least an approxmiate factor of $\frac{16}{9}$, 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. A proof that the combination of these cycles cannot continue forever would be equivalent to proving terminaton of Collatz Variant 1. We present analysis of hardness of this cycle in chapter 4.

- Allowing 1 but avoiding 7: Adding back in 1 actually allows for three different cycles: $4 \to 6 \to 3 \to 2 \to 5 \to 0 \to \ldots \to 4$, $4 \to 6 \to 3 \to 2 \to 1$, and $4 \to 2 \to 1$. Naturally, if we can't prove that the simpler combination of cycles $4 \to 6 \to 3 \to 2 \to 1$, and $4 \to 2 \to 1$ cannot terminate, it would be far more difficult to add a third cycle. Solving that the combination of these three cycles must terminate is equivalent to proving termination of Collatz Variant 7.

3.4.3 Base 8 variants

As for which nodes we are forced to visit during the computation of a $3x + 1$ sequence, we can prove these four following Collatz Variants 2, 3, 4 and 6 terminate. Variant 0 can be shown to terminate if another unproven variant terminates. It will still be mentioned in this section. The following will explain how these five variants terminate. We present them in an order to build arguments off of one another.

- $\text{Col}_{\textbf{mod}}(N, \{6\}, 8)$: In the graph, not visiting node 6, means the Collatz Sequence would have to traverse one of two cycles: $4 \to 2 \to 1$ or $4 \to 2 \to 5 \to 0 \to \ldots \to 4$, both of which are cycles that cannot continue forever. Naturally, the combination of these two cycles cannot continue forever. So $\text{Col}_{\text{mod}}(N, \{6\}, 8)$ must terminate.

- $\text{Col}_{\textbf{mod}}(N, \{3\}, 8)$: We know that $\text{Col}_{\text{mod}}(N, \{6\}, 8)$ terminates, so as a result, we know that node 6 must be visited in our base 8 graph. Given Lemma 3.2.3, the $6 \to 7 \to 6$ cycle cannot continue forever. Hence, node 6 must transition to node 3 eventually, meaning $\text{Col}_{\text{mod}}(N, \{3\}, 8)$ must terminate.

- $\text{Col}_{\textbf{mod}}(N, \{2\}, 8)$: Since we know that we must visit node 3, this transitions to node 2, proving $\text{Col}_{\text{mod}}(N, \{2\}, 8)$ must terminate.

- $\text{Col}_{\textbf{mod}}(N, \{4\}, 8)$: Given that we know we need to visit node 2, we look at the graph and see two possible 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 the path taken must traverse to node 4. Hence, $\text{Col}_{\text{mod}}(N, \{4\}, 8)$ terminates.

- $\text{Col}_{\textbf{mod}}(N, \{0\}, 8)$: 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 $\text{Col}_{\text{mod}}(N, \{1\}, 8)$, $\text{Col}_{\text{mod}}(N, \{5\}, 8)$, $\text{Col}_{\text{mod}}(N, \{7\}, 8)$, or the combination $\text{Col}_{\text{mod}}(N, \{5, 7\}, 8)$, all of which were mentioned in the cycle analy-

sis. Hence, we've run some computational experiments to try and better understand the difficulty of coming up with a proof for termination of these variants.

Chapter 4

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 $A = \{1\}$, $\{5\}$, or $\{7\}$. We start by defining some measures, talk about the process how we ran experiments, and talk about the results, both analyzing termination of Collatz Variants 1, 5, and 7, as well as exploring the Collatz Variant $\{5, 7\}$.

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 x:

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

[1] If one wanted to figure out the number of visited even numbers, then $f_{\text{even}}(x) = f(x) - f_{\text{odd}}(x)$

- $g(x, A)$: The highest number of steps, for an input number x that converges to 1, where $\forall a \in A$, $x_i \not\equiv a \pmod 8 \wedge x_i > 1$. This is counting the maximum number of steps before Collatz Variant $\text{Col}_{\text{mod}}(N, A, 8)$ would terminate for input number N.
- $g_{\text{odd}}(x, A)$: The number of odd numbers within the given $g(x, A)$
- Slice: a batch of numbers from some low number to some high number for a fixed A.
- x_{min}: the lowest number of any slice.
- x_{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, two different measures are defined, and the intuition behind why they were chosen is given as well:

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

Also define "Classical Hardness" as a comparison: $H_C(x) = \frac{f_{\text{odd}}(x)}{\log_2 x}$. This computes H 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 $f(r_{\text{new}}) > f(r_{\text{current}})$.

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

4.2 Generating Results

We wrote a program that computed 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 we wish to compute, and let $x_{\min}$ denote the lowest number in any slice, whereas $x_{\max}$ denotes the highest number in any slice.

- baseavoid is the default option. This allows us to check all $A \in \mathcal{A}$ by running through all odd numbers from a given minimum (we usually use 1) to a given maximum (we usually use 1 billion), and determines the maximum number of steps we can run Algorithm 2 for each set A, or in other words, compute $g(x, A)$ for all odd x in the range $x_{\min} \leq x \leq x_{\max}$. When it finishes, it prints out the longest sequence for each A.

- entirechain just runs Algorithm 1 for all odd x in the range $x_{\min} \leq x \leq x_{\max}$, and prints out the longest sequence.

- untildecay means that, for each odd number in between $x_{\min}$ and $x_{\max}$, we continue to run until we have a number lower than the original number. We

return only the longest sequence of numbers that occurs until the resulting number is smaller than the initial number.

- updown is a quite different mode. For each odd number x such that $x_{\min} \leq x \leq x_{\max}$, determine two things. First, the number of steps it takes for x to become some number x_i such that $x_i < x$. Second, the number of steps it takes for another number x_g to grow to x if such an x_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, x_i, the number of steps it takes for x to turn into x_i, and, if x_g exists, the number x_g and the number of steps it takes for x_g to grow into x.

- avoidingmodgrowth is a mode like baseavoid. However, it prints tables showing progressively growing records. This is the mode we used to generate the hardness results.

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 [16]. We then combine the records of these 8 tables by hand using the defined record criterion.

We run sequences of numbers and have an option to avoid recomputing odd numbers already part of a priod 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. So the space efficient approach was chosen for this project.

4.3 Single Base Avoidance 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 the log of all records r versus $H(x, A)$ and $P(x, A)$, respectively. We also added in Collatz Variant 3 as a reference for both of these measures as a control case.

4.3.1 Hardness Function Results and Analysis

Figure 4.1 shows the results of $H(x, A)$ versus the number of bits ($\log_2 x$). Total steps for each mod number were also added in as well.

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 than the unproven variants.

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 around

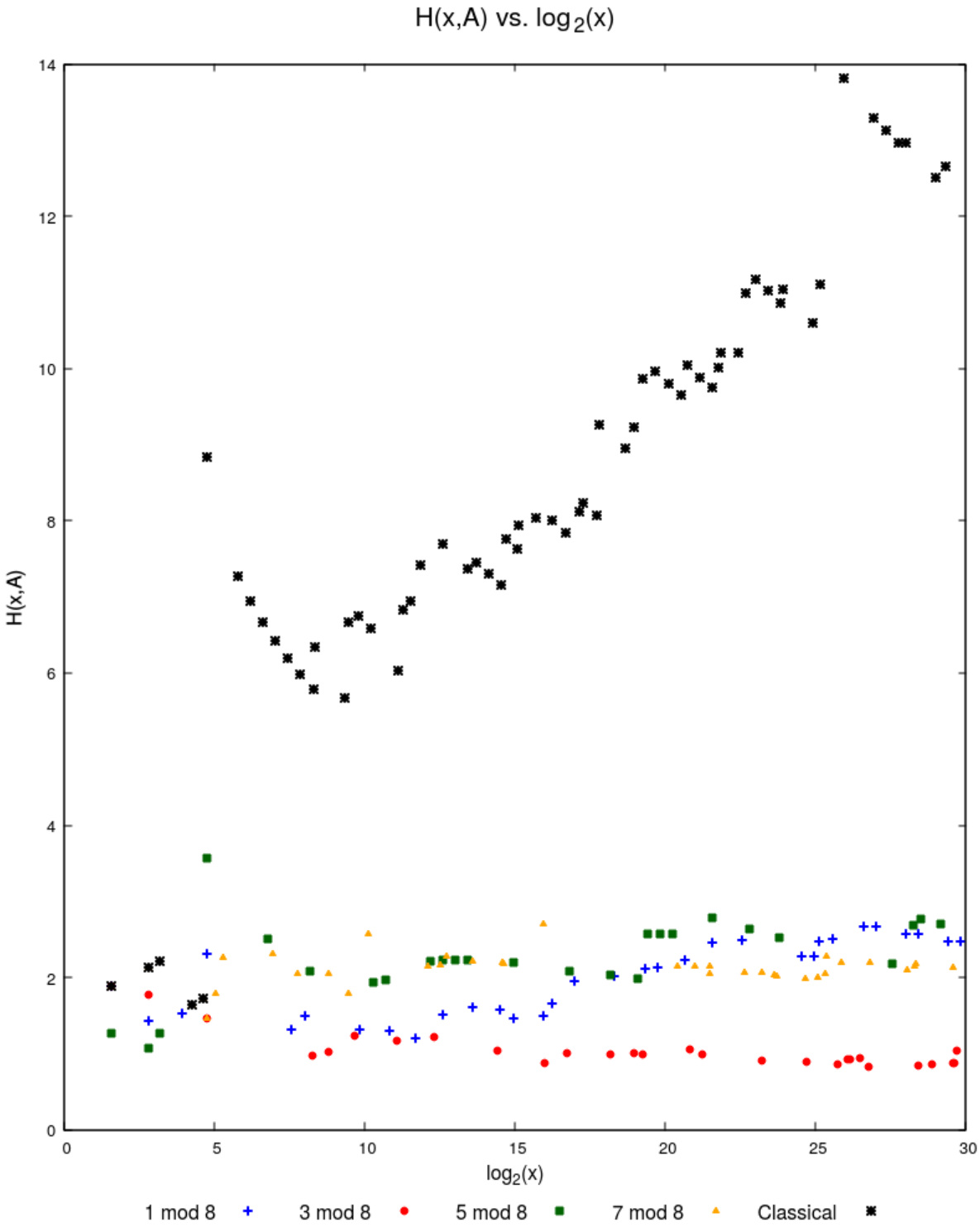

Figure 4.1: This graph visualizes how the H values for Collatz Variants 1, 3, 5, and 7 compare to each other, and to classical hardness. The log of the record holding numbers, or number of bits needed, is the x-axis, and the hardness measure H as defined in subsection 4.1 is the y-axis.

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 around 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 cleraly 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 trend the same way as we add more bits into the computation. Classical Hardness actually tends to grow linearly against the log scale, meaning that as the input number increases, determining if the input number converges to 1 gets logarithimcally harder. This contrasts to all three unproven variants, which tend to stay between values of 1.5 and 3 for H for numbers below 1 billion, 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 the results of $P(x, A)$ versus the number of bits.

$P(x, A)$, as discussed earlier, is just calculating what percentage of the record sequence contributes to the overall decay of that sequence to 1.

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).

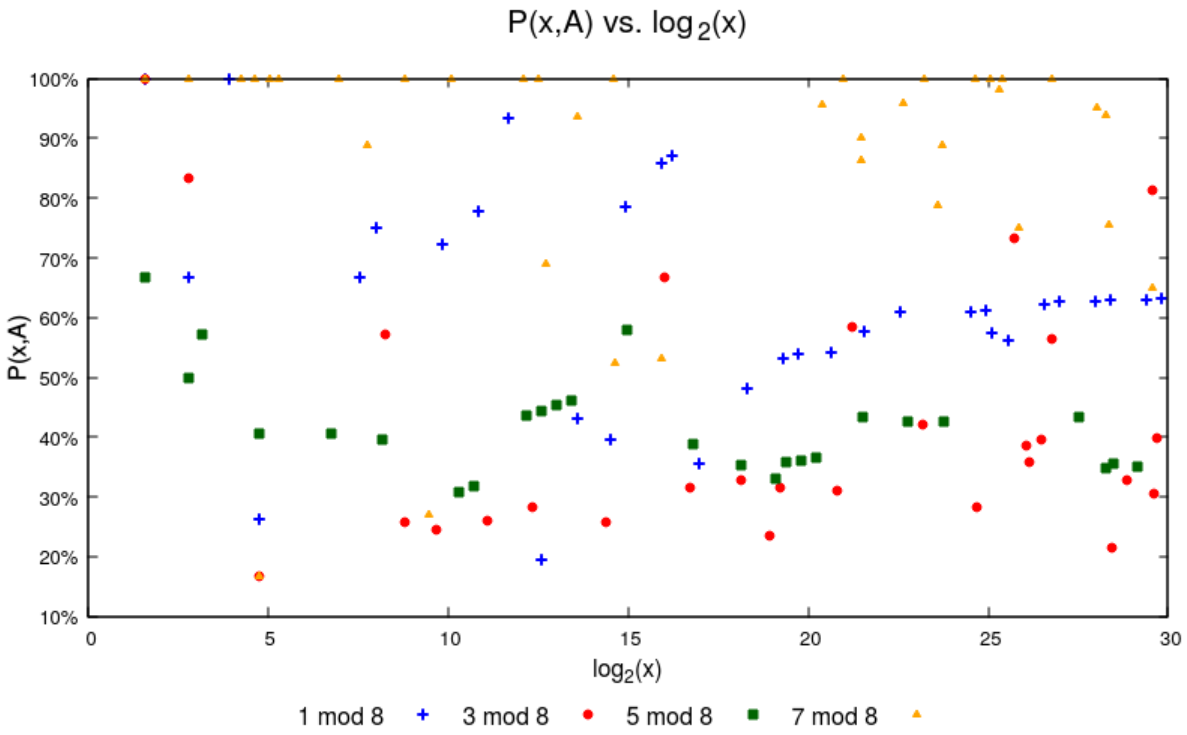

Figure 4.2: This graph visualizes how the P values for Collatz Variants 1, 5, and 7 compare to each other. The log of the record holding numbers, or number of bits needed, is the x-axis, and the percentage measure P as defined in subsection 4.1 is the y-axis.

Variant 5 tends to have a low percentage, and is the least erratic of all four variants, meaning the standard deviation is lower. Variant 5 avoids the 0 self-cycle, which causes many divisions by 2. Numbers having a long sequence that prevent termination of variant 5 tend to have a very large number when they finally reach a number that is 5 (mod 8), meaning that often, many more steps in the $3x + 1$ mapping must 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 between approximately 50% and 60% at around 20 bits. This is likely a consequence of the sequence similarity that is seen in larger record for variant 1, which will be analyzed in subsubsection 4.3.3. Further, as mentioned in the cycle analysis for the $4 \to 6 \to 3 \to 2 \to 5 \to 0 \to 4$ cycle (allowing $6 \to 7 \to 6$, but avoiding 1), avoiding 1 (mod 8) causes a clash between the decay of the 0 self-cycle and the growth of the $6 \to 7 \to 6$ cycle, which may explain some of the erratic behavior for variant 1, aside from the small part with chain similarity. Variant 3 tends toward the lowest percentage of all odd numbers, even lower than variant 5, but also has erratic behavior. 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 $4 \to 2 \to 5 \to 0 \to 4$ cycle with some number of 0 self-cycles. The first cycle causes growth, whereas all three 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 here.

- Variant 5: This had few matches and was the most changing of the records, so chain similarity appears not to have did not have as much to do with the low standard deviation for P we saw for variant 5.

4.4 Paired Base Avoidance Analysis

Because it appears to be difficult to figure out why Collatz Variants 1, 5, and 7 must terminate, we also analyzed the unknown variant $\{5, 7\}$. This section will analyze what happens to $H(x, A)$ and $P(x, A)$ where A is equal to three different base sets: the unknown variant $A = \{5, 7\}$, and the known variants where $A = \{1, 5\}$, and $A = \{1, 7\}$. The known variant $\{1, 5\}$ has not been proven yet but a quick proof is shown here:

Lemma 4.4.1. *The Collatz Variant* $\{1, 5\}$ *must terminate.*

Proof. We already know that Collatz Variant 2 must terminate. Looking at the base 8 graph, 2 must traverse to either node 1 or 5. Clearly, variant $\{1, 5\}$ must terminate for any N. □

The termination of variant $\{1, 7\}$ was discussed in 3.4.2. We decide to include both variants to compare it to variant $\{5, 7\}$.

4.4.1 Hardness Function Results and Analysis

Figure 4.3 shows the results of $H(x, A)$ versus $\log_2 x$. Total steps for each mod number were also added in as well.

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

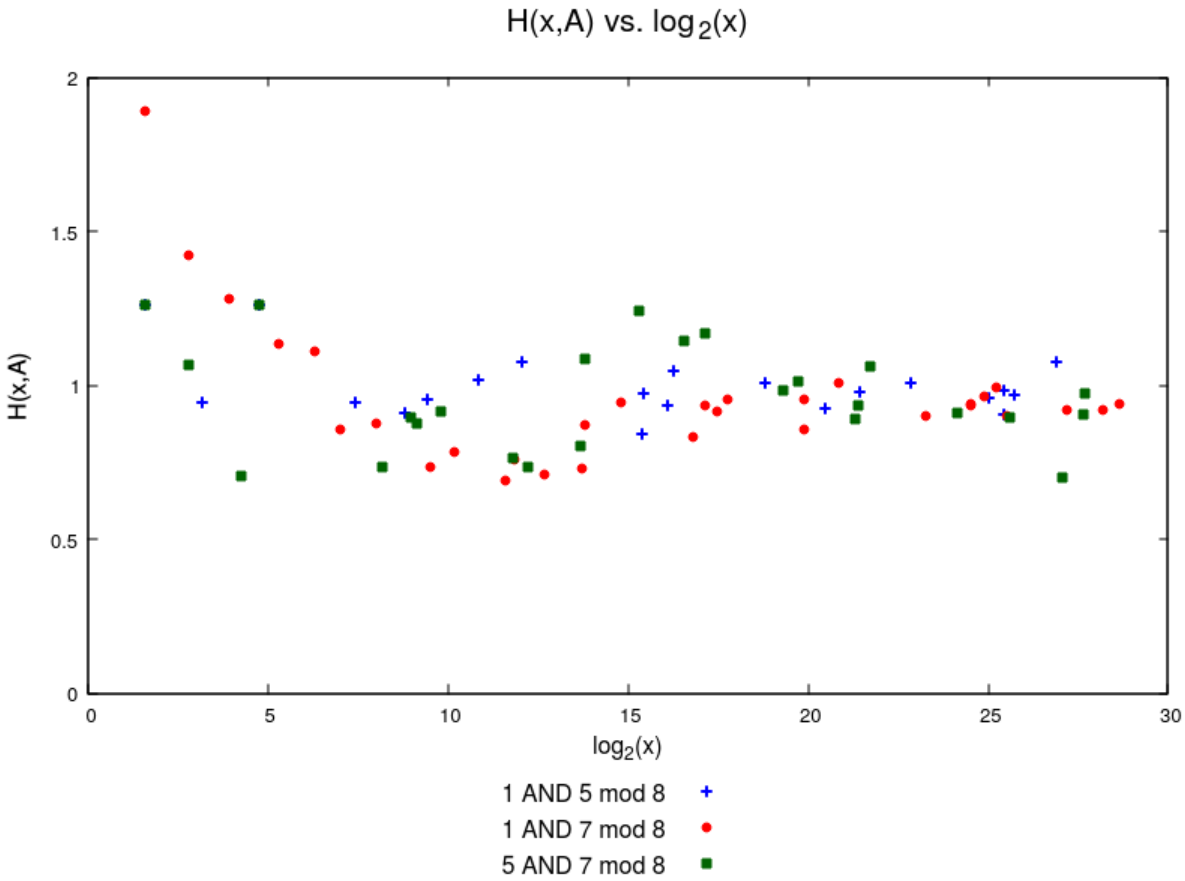

Figure 4.3: This graph visualizes the H measure for the three Collatz Variants $\{1, 5\}$, $\{1, 7\}$, and $\{5, 7\}$. The log of the record holding numbers, or number of bits needed, is the x-axis, and the hardness measure H as defined in subsection 4.1 is the y-axis. Classical Hardness was omitted from this graph to eliminate distortion.

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 the results of $P(x, A)$ versus $\log_2 x$.

The variant $\{1, 7\}$ makes up the highest percentage of the sequence compared to the other two cases, because avoiding both the $6 \to 7 \to 6$ and the $4 \to 6 \to 3 \to 2 \to 1$ cycles only allows for the sequence to go through the $4 \to 2 \to 5 \to 0 \to \ldots \to 4$ cycle, causing fast decay, like with variant 7.

Both variants $\{1, 5\}$ and $\{5, 7\}$ are much closer to each other in percentage of total sequence, although as the numbers grow past 17 bits in size, variant $\{5, 7\}$ comprises of the higher percentage of the sequence. A possible explanation is the fact that the $6 \to 7 \to 6$ cycle allowed in variant $\{1, 5\}$ allows causes a number to grow larger than the $4 \to 6 \to 3 \to 2 \to 1$ cycle that variant $\{5, 7\}$ allows, and since variant $\{1, 5\}$ allows for larger numbers, and the fact that larger numbers generally (but not always) take more steps to decline, that allowing for more growth should mean that the long sequence avoiding termination of variant $\{1, 5\}$ should make up a lower percentage of all odd numbers.

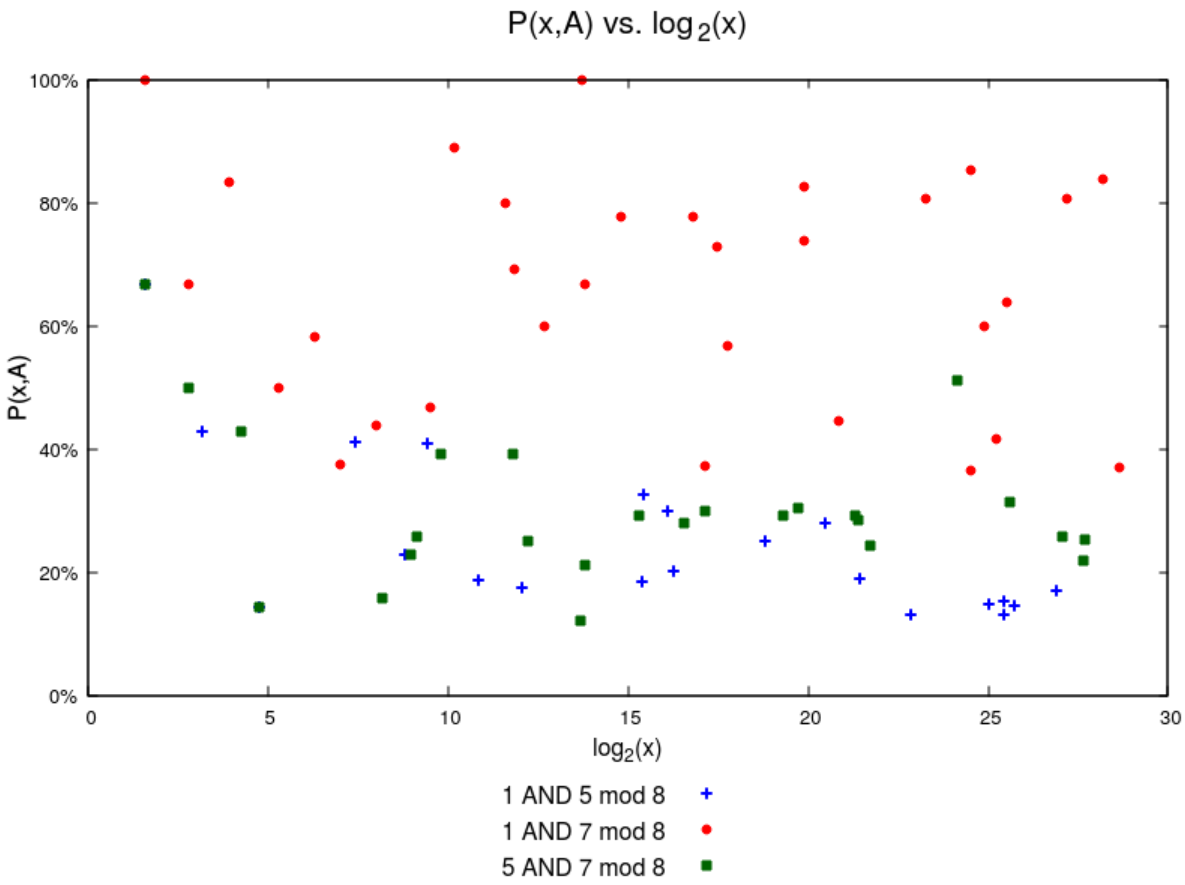

Figure 4.4: This graph visualizes the P measure for variants $\{1, 5\}$, $\{1, 7\}$, and $\{5, 7\}$. The log of the record holding numbers, or number of bits needed, is the x-axis, and the hardness measure P as defined in subsection 4.1 is the y-axis.

Chapter 5

Using SAT Solvers with String Rewriting Systems

In this chapter, we talk briefly about SAT solvers, string rewrite systems, how we can use SAT solving to determine if a string rewrite systems terminates for any input, and a string rewrite system that Aaronson built, which, if this system terminates, we believe it is equivalent to showing that the Collatz Conjecture holds. We then present results of an attempt that Heule built using Aaronson's rewrite system.

5.1 SAT Solvers

SAT solvers are powerful devices 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 derived from SAT that is NP-Complete when $k > 2$, meaning that if $P \neq NP$, the worst case runtime is exponential. However, with clever heuristics, we can actually make any propositional logic formulas run in linear time for many interesting cases. Knowing SAT solving background is not important for this Thesis, but there is a great deal of literature talking about SAT solving background, so one can check, for instance, [2].

5.2 String Rewrite Systems

A string rewriting system (SRS), at a high level, takes a input string of an alphabet, and applies string rewriting rules (SRRs) in an arbitrary order on the input string to see if the string can be transformed further. An example SRS is given.

Definition 5.2.1. 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 [9], using SRS A, asks the following question:

Question 5.2.1. 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 to show that this problem terminates for all input strings should be easy to show. Surprisingly, both humans and computers struggled to come up with a proof for this problem when initially presented. However, Hofbuaer and Waldmann came up with a proof for determining that it terminates [9], and from this, found that, for any rewriting rules in any rewrite system, if they are transformed into functions that causes all inputs to decrease, then the rewrite system will terminate. [10].

The matrices are built using SAT solving to determine if a $d \times d$ matrix is large enough to ensure that the matrix functions for the set of rewrite rules always decrease for all inputs. The process of building these matrices is described in a paper by Endrullis, Waldmann and Zantema [5]. An example using Zantema's Other Problem will be explained here.

Here are the matrices computed by SAT from [9]:

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

We will show an example string to show that 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, while the other $d-1$ elements must either be the same or decrease.

Let our sample string be *bbaa*. Here is a possible set of rules that can be applied to it as well as the vector representations of the symbols:

$$\begin{array}{ccccccccc} bb\underline{aa} & \to & b\underline{bb}c & \to & ba\underline{cc} & \to & b\underline{aa}b & \to & \underline{bb}cb \\ \begin{pmatrix} 18 \\ 14 \\ 6 \\ 0 \end{pmatrix} & \succ & \begin{pmatrix} 17 \\ 14 \\ 6 \\ 0 \end{pmatrix} & \succ & \begin{pmatrix} 15 \\ 14 \\ 6 \\ 0 \end{pmatrix} & \succ & \begin{pmatrix} 14 \\ 14 \\ 6 \\ 0 \end{pmatrix} & \succ & \begin{pmatrix} 13 \\ 14 \\ 6 \\ 0 \end{pmatrix} \end{array}$$

$$\begin{array}{ccccccccc} \to a\underline{cc}b & \to & aa\underline{bb} & \to & a\underline{aa}c & \to & ab\underline{cc} & \to & abab \\ \succ \begin{pmatrix} 7 \\ 14 \\ 5 \\ 0 \end{pmatrix} & \succ & \begin{pmatrix} 6 \\ 14 \\ 5 \\ 0 \end{pmatrix} & \succ & \begin{pmatrix} 4 \\ 14 \\ 3 \\ 0 \end{pmatrix} & \succ & \begin{pmatrix} 3 \\ 0 \\ 3 \\ 0 \end{pmatrix} & \succ & \begin{pmatrix} 2 \\ 0 \\ 3 \\ 0 \end{pmatrix} \end{array}$$

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 rewriting system

Aaronson built an SRS representing the Collatz Conjecture [6]. 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:

$$a(x) = 2x, b(x) = 2x + 1, c(x) = 1, d(x) = x, e(x) = 3x, f(x) = 3x + 1, g(x) = 3x + 2$$

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 rewrite system 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:

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

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

Each of the rules denotes how to handle the symbols $a - g$, or the combination of binary and placeholder strings. The D rules represent the two steps taken by the Collatz Conjecture in a binary system. D_1 is how we handle an even number. It computes division of x 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 $3x + 1$, 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 do $gad \to gd$. Since $3x + 1$ always results in an even number, we can just make rule D_2 compute $(3x + 1)/2$ 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 the 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 $eb = b \circ e = b(e(x)) = 2(3x) + 1 = 6x + 1$.
- $\boldsymbol{ag \to fa}$: $ag = g \circ a = g(a(x)) = 3(2x) + 2 = 6x + 2$, and $fa = a \circ f = a(f(x)) = 2(3x + 1) = 6x + 2$.
- $\boldsymbol{be \to fb}$: $be = e \circ b = e(b(x)) = 3(2x+1) = 6x+3$, and $fb = b \circ f = b(f(x)) = 2(3x + 1) + 1 = 6x + 3$.
- $\boldsymbol{bf \to ga}$: $bf = f \circ b = f(b(x)) = 3(2x + 1) + 1 = 6x + 4$, and $ga = a \circ g = a(g(x)) = 2(3x + 2) = 6x + 4$.
- $\boldsymbol{bg \to gb}$: $bg = g \circ b = g(b(x)) = 3(2x + 1) + 2 = 6x + 5$, 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 corresponding number x. 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, and the proofs showing this is the case for each of these rules are 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 $3x + 1$ 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. Until we have the string cd:
    - Apply the appropriate D rule.
    - If D_2 is applied, 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 rule D_2 is applied, then it handles the ternary symbol immediately by applying A, B, and C rules until it is converted into binary symbol(s). The number is not changed during application of these rules, making the ordering correct. This ordering was used in building the system that investigated the number of steps needed in the rewrite system, which will be talked about in section 6.

If we take this SRS and model matrix functions for all symbols that cause all inputs to decrease, then we believe we can prove the Collatz Conjecture. We don't have an absolute link, but 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 would Algorithm 1 operate on positive integers.

5.4 Proving Collatz Conjecture Results

This section discusses some of the results mentioned in [6]. Heule and Aaronson have not been able to prove termination of the Collatz SRS for all 11 rules, which is why this investigation for Collatz subproblems came about. This section describes the results thus far.

Heule tried to run the Collatz SRS on state-of-the-art matrix interpretation solvers like AProVE. However, 4 of the 11 rules needed to be removed before the system could be solved. So a custom matrix interpretation solver was built by Heule specifically for this problem. Unfortunately, Heule could not prove the entire 11 rule system with matrix interpretation. However, any combination of 10 of the 11 rules

can be proven. Some were very easy (example: omitting rule D_2), others much more challenging.

Hence, the motivation for simplifying the problem by creating the Collatz Variants derived from 2, came about, as did the motivation for writing this thesis. Chapter 6 investigates alterations of the SRR's for the Collatz SRS, which we call to try and make the problem simpler for the matrix interpretation solver, and see how difficult these alternate forms ought to be to prove.

Chapter 6

Hardness of Application of Rewrite Rules

Now we have explained the background for Rewrite Systems, as well as the motivation for them, we now repeat the algebraic hardness measure computation done in chapter 4 with a couple of changes: First, we do computations with slight modifications to the Collatz SRS, and second, we only compute record sequences from our previous algebraic analysis. In this section, we first introduce a modified version of the SRS, then we define measures, talk about the computation, then present our results.

6.1 Modified Base 8 Rewrite System

Recall the Collatz SRS, and the D rules: the rules that handle the even and odd numbers:

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

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

D_1 handles 0 (mod 2) (even numbers) by effectively dividing by 2, while D_2 handles 1 (mod 2) (odd numbers) by effectively computing $3x+1/2$. Also note that all input

strings for these rules are just one bit, since the placeholder d is not a digit. However, we can expand this input to be 3 bits and come up with 8 corresponding SRRs:

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

It is easy to see how these rules all correspond to a node in graph G_8: an input string strictly with symbols a, b, c, and d corresponds to a binary number. The inputs, in order, are numbers congruent moduly to 0-7 (mod 8), and the outputs are the result of dividing by 2, if even, or multiplying by 3 and adding 1. All of the odd rules, 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 number 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. Hence, we use the following SRRs in

the base 8 modification of the 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 equal to the original Collatz SRS. However, they do add an extra dimension not present before. We can remove one of the rules to make it easier to prove that the derived SRS will terminate. We present a sample 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 implies termination of Collatz Variant 1, as removing a rule causes any string with this input to terminate the system. Note how removing an SRR is equivalent to adding the corresponding termination condition in Algorithm **??**. Any derived Collatz SRS that alters the base 8 modification of the Collatz SRS will henceforth be referred to as a Collatz Subproblem A, where A will bethe same base avoidance set

as used in $\text{Col}_{\text{mod}}(N, A, b)$. For singleton sets A, we just write the number. Collatz Subprolem A denotes which rewrite rule(s) are dropped, and that subproblem A would imply termination of Collatz Variant A.

The rest of this chapter describes the investigation we took to determine the number of steps that Collatz Subproblems 1, 5, and 7 need before terminating.

6.2 Defining Measures

Instead of defining hardness by number of odd numbers, for the SRS, we define hardness based off of the total steps applied. This is because an odd number adds a significant more number of rewrite steps... $\Theta(m)$ for the odd number, compared to just 1 for an even number. Define the following numbers, given some input number x:

- $f_r(x)$: The total number of rewrite steps in the sequence for x before it converges to 1.
- A: The base avoidance set, same as used in Algorithm **??** and chapter 4. As before, $A \subseteq \{1, 5, 7\}$ and $A \neq \varnothing$. Dropping an SRR that corresponds to avoiding $a \pmod 8$ effectively adds a to A.
- Record Sequence for $A \pmod b$: Same exact definition as in chapter 4. We only run rewrite systems for the record sequences we computed with the algebraic Collatz method, as running computation for strictly the rewrite system would take an extremely long time.

- $R(x, A, b)$: The number of rewrite steps that the record sequence from Collatz Variant $\mathrm{Col}_{\mathrm{mod}}(x, A, b)$ takes in an SRS based off of Collatz Subproblem A.

We define only one hardness measure: H_{SRS}, where $H_{SRS} = \frac{R(x,A,b)}{\log_2 x}$. This effectively computes the hardness of the SRRs that corresponds to determining termination of Collatz Variant A.

6.3 Computation

The program we wrote simulates the Collatz SRS in Java. It takes two different keys of input: 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 is equivalent to the rule $input \rightarrow output$. The # character is a comment, meaning if the first character of a rewrite rule is #, we ignore that line. This is a convenience to comment out a rule to create SRRs that correspond to Collatz subproblems.

The program converts an input number into a binary rewrite string with characters a, b, c, and d. The rewrite term is stored in a "sliding" array, because in Aaronson's SRS, a number can only add string length from the c rules. When we apply rule D_{8_1}, 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 trailing d terms.

As discussed in section **??**, we don't apply SRRs in arbitrary order. Given a rewrite string completely in binary, 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 character is generated by it. If so, we apply the A and B rules to move the ternary character index-by-index until we can apply a C rule, which removes the ternary character.

The input is first number from record sequences computed algebraically that *does not* cause $Col(N, A, b)$ to terminate. We run the SRS for subproblem A until it terminates. The output is, for one number, the terms that result from the application of the SRRs until termiation, as well as which number the intermediate terms correspond to. For a batch of numbers, each individual number is output in a separate file, and there is an overall file that outputs the input number, the final number, and the number of rewrite steps.

6.4 Single SRR removal analysis

Figure 6.1 shows the anaylsis of hardness for the modified SRS with removal on one of three different rules. Note that the hardness tends to grow for all three cases, as opposed to the analysis for $H(x, A)$ for each of these three cases, which tend to stay flat. This shows, that as the number of bits increases, the number of steps for the rewrite system tend to increase logarithmically. The best explanation for why this is the case is because for each odd rule, we add $\Theta(\log n)$ steps, so as discussed in the algebraic case, hardness is determined by odd numbers. However, it is clear that all three cases don't have the same slope of increase. Subproblem 7 has the most gradual growth of all three cases, followed by subproblem 1 and subproblem 5, which has not only the most growth, but the highest standard deviation of all three

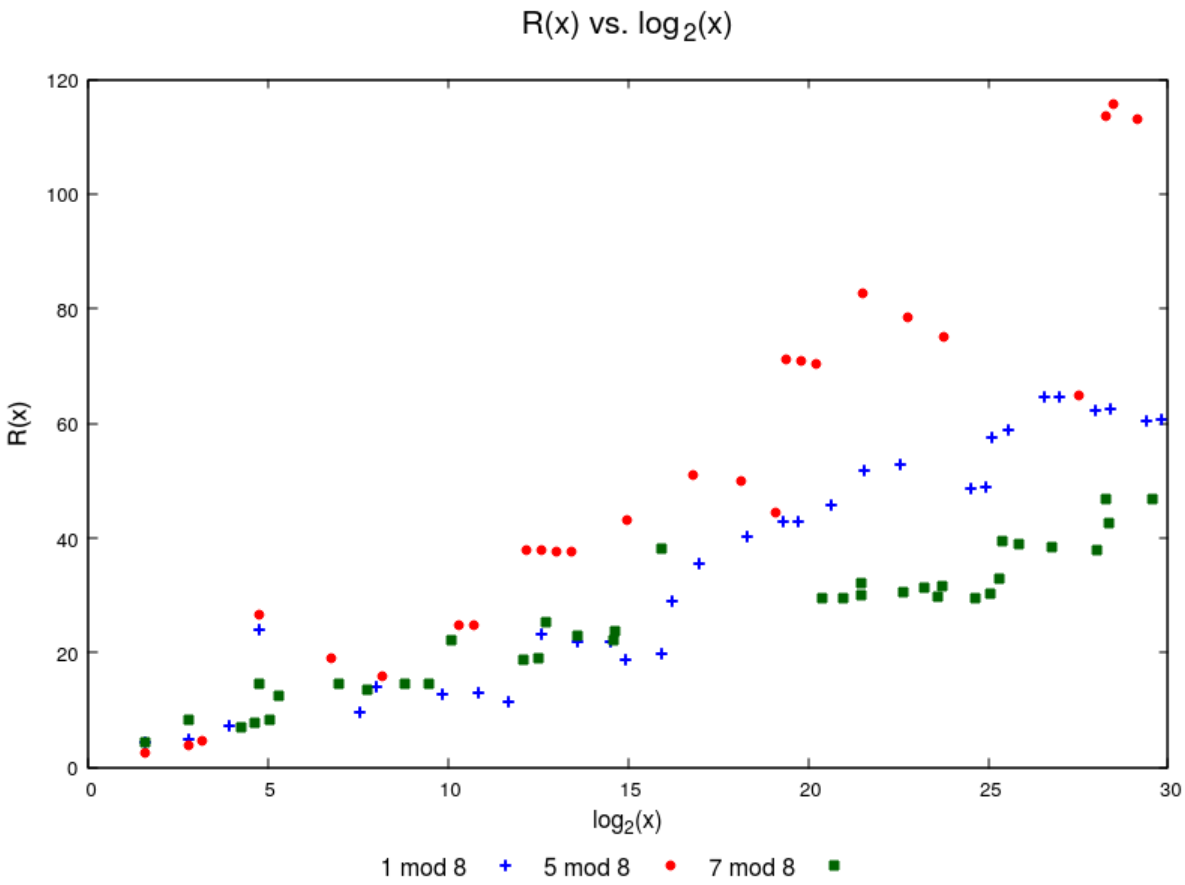

Figure 6.1: This graph visualizes how the R values for subproblems 1, 5, and 7 compare to each other. The log of the record holding numbers, or number of bits needed, is the x-axis, and the hardness measure R as defined in section 6.2 is the y-axis.

cases. These can be explained by the following observations:

- Record sequences for subproblem 7 eliminate the growth of the $6 \to 7 \to 6$ cycle, meaning the numbers tend to get smaller and need less bits to encode as a rewrite string.
- Record sequences for subproblem 5 eliminate the decay of the 0 self-cycle, meaning numbers tend to grow more often than not, so the numbers here are larger.
- Record sequences for subproblem 1 is in between the other two cases, since both the 0 self-cycle of decay and the $6 \to 7 \to 6$ of growth can occur.

Chapter 7

Conclusion

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

We also investigated the Collatz SRS that Aaronson came up with in [6] and that Heule tried to prove with Matrix Interpretation. Even though this methodology was not able to solve the Collatz Conjecture, we still think it has promise. We also built a simple program that ran the Collatz SRS and determine the hardness for variants, finding out that the hardness varies more than in the algebraic case.

As described in [6], we believe this approach for trying to solve the Collatz Conjecture, as well as Collatz Variants, has merit and needs further investigation, so we hope to continue to do so in the future.

Bibliography

[1] K. Appel and W. Haken. Every planar map is four colorable. *Illinois J. Math.*, 21(3):429–490, 09 1977.

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

[3] J.H. Conway. Unpredicatable iterations. In *Proceedings of the 1972 Number Theory Conference: University of Colorado*, pages 49–52, University of Colorado, Boulder, CO, 1972. University of Colorado.

[4] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, July 1962.

[5] Jörg Endrullis, Johannes Waldmann, and Hans Zantema. *Matrix Interpretations for Proving Termination of Term Rewriting*, pages 574–588. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[6] Marijn J. H. Heule and Scott Aaronson. Spx: Mapamap: Massively parallel solving of math problems. Unpublished.

[7] Marijn J. H. Heule, Oliver Kullmann, and Victor W. Marek. Solving and verifying the boolean pythagorean triples problem via cube-and-conquer, 2016.

[8] Marijn J. H. Heule, Oliver Kullmann, Siert Wieringa, and Armin Biere. Cube and conquer: Guiding cdcl sat solvers by lookaheads. In Kerstin Eder, João Lourenço, and Onn Shehory, editors, *Hardware and Software: Verification and Testing*, pages 50–65, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[9] Dieter Hofbauer and Johannes Waldmann. Termination of $aa \to bc, bb \to ac, cc \to ab$. *Inf. Process. Lett.*, 98(4):156–158, May 2006.

[10] Dieter Hofbauer and Johannes Waldmann. *Termination of String Rewriting with Matrix Interpretations*, pages 328–342. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[11] J. C. Lagarias. The 3x+1 problem: An annotated bibliography (1963–1999) (sorted by author), September 2003.

[12] J. C. Lagarias. The 3x+1 Problem: An Annotated Bibliography, II (2000-2009), August 2006.

[13] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: engineering an efficient sat solver. In *Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232)*, pages 530–535, 6 2001.

[14] Eric Roosendaal. On the 3x+1 problem, 2017.

[15] João P. Marques Silva. The impact of branching heuristics in propositional satisfiability algorithms. In *Proceedings of the 9th Portuguese Conference on Artificial Intelligence: Progress in Artificial Intelligence*, EPIA '99, pages 62–74, London, UK, UK, 1999. Springer-Verlag.

[16] Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed computing in practice: The condor experience: Research articles. *Concurr. Comput. : Pract. Exper.*, 17(2-4):323–356, February 2005.

[17] G.S. Tseitin. On the complexity of derivation in propositional calculus. Presented at the Leningrad Seminar on Mathematical Logic held in September 1966., 1966.

[18] A. M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265, 1936.

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 and Handset 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 become a full-time student.

Email address: mad4672@cs.utexas.edu

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.