Mathematisch-Naturwissenschaftliche Fakultät

Eldar Sultanow, Denis Volkov, Sean Cox

Introducing a Finite State Machine for processing Collatz Sequences

Second Version

Preprint published at the Institutional Repository of the Potsdam University:
http://nbn-resolving.de/urn:nbn:de:kobv:517-opus4-404738

Universität Potsdam

Contents

1. Introduction

The present work will introduce a Finite State Machine (FSM) that processes any Collatz Sequence; further, we will endeavor to investigate its behavior in relationship to transformations of a special infinite input. Moreover, we will prove that the machines word transformation is equivalent to the standard Collatz number transformation and then discuss the possibilities for utilizing this approach for solving similar problems. The benefit of this approach is that the investigation of the word transformation performed by the Finite State Machine is less complicated than the traditional number-theoretical transformation.

1.1 Motivation

The Collatz conjecture is a number theoretical problem, which has puzzled countless researchers using myriad approaches. Presently, there are scarcely any methodologies to describe and treat the problem from the perspective of the Algebraic Theory of Automata. Such an approach is promising with respect to facilitating the comprehension of the Collatz sequences "mechanics". The systematic technique of a state machine is both simpler and can fully be described by the use of algebraic means.

The current gap in research forms the motivation behind the present contribution. The present authors are convinced that exploring the Collatz conjecture in an algebraic manner, relying on findings and fundamentals of Automata Theory, will simplify the problem as a whole.

1.2 Related Research

The Collatz conjecture is one of the unsolved mathematical Millennium problems [1]. When Lothar Collatz began his professorship in Hamburg in 1952, he mentioned this problem to his colleague Helmut Hasse. From 1976 to 1980, Collatz wrote several letters but missed referencing that he first proposed the problem in 1937. He introduced a function $g : \mathbb{N} \to \mathbb{N}$ as follows:

$$g(x) = \begin{cases} 3x+1 & 2 \nmid x \\ x/2 & \text{otherwise} \end{cases} \tag{1.1}$$

This function is surjective, but it is not injective (for example $g(3) = g(20)$) and thus it is not reversible.

In his book The Ultimate Challenge: The 3x+1 Problem [2], along with his annotated bibliographies [3], [4] and other manuscripts like an earlier paper from 1985 [5], Lagarias has

reseached and put together different approaches from various authors intended to describe and solve the Collatz conjecture.

For the integers up to 2,367,363,789,863,971,985,761 the conjecture holds valid. For instance, see the computation history given by Kahermanes [6] that provides a timeline of the results which have already been achieved.

Inverting the Collatz sequence and constructing a Collatz tree is an approach that has been carried out by many researchers. It is well known that inverse sequences [7] arise from all functions $h \in H$, which can be composed of the two mappings $q, r : \mathbb{N} \to \mathbb{N}$ with $q : m \mapsto 2m$ and $r : m \mapsto (m-1)/3$: $H = \{h : \mathbb{N} \to \mathbb{N} \mid h = r^{(j)} \circ q^{(i)} \circ \ldots, i, j, h(1) \in \mathbb{N}\}$

An argumentation that the Collatz Conjecture cannot be formally proved can be found in the work of Craig Alan Feinstein [8], who presents the position that any proof of the Collatz conjecture must have an infinite number of lines and thus no formal proof is possible. However, this statement will not be acknowledged in depth within this study.

Treating Collatz sequences in a binary system can be performed as well. For example, Ethan Akin [9] handles the Collatz sequence with natural numbers written in base 2 (using the Ring $\mathbb{Z}_2$ of two-adic integers), because divisions by 2 are easier to deal with in this method. He uses a shift map σ on $\mathbb{Z}_2$ and a map τ:

$$\sigma(x) = \begin{cases} (x-1)/2 & 2 \nmid x \\ x/2 & \text{otherwise} \end{cases} \qquad \tau(x) = \begin{cases} (3x+1)/2 & 2 \nmid x \\ x/2 & \text{otherwise} \end{cases}$$

The shift map's fundamental property is $\sigma(x)_i = x_{i+1}$, noting that $\sigma(x)_i$ is the i-th digit of $\sigma(x)$. This property can easily be comprehended by an example $x = 5 = 1010000\ldots = x_0x_1x_2\ldots$, containing $\sigma(x) = 2 = 0100000\ldots$

Akin then defines a transformation $Q : \mathbb{Z}_2 \to \mathbb{Z}_2$ by $Q(x)_i = \tau^i(x)_0$ for non-negative integers i which means $Q(x)_i$ is zero if $\tau^i(x)$ is even and then it is one in any other instance. This transformation is a bijective map that defines a conjugacy between τ and σ: $Q \circ \tau = \sigma \circ Q$ and it is equivalent to the map denoted Q_∞ by Lagarias [5] and it is the inverse of the map Φ introduced by Bernstein [10]. Q can be described as follows: Let x be a 2-adic integer. The transformation result $Q(x)$ is a 2-adic integer y, so that $y_n = \tau^{(n)}(x)_0$. This means, the first bit y_0 is the parity of $x = \tau^{(0)}(x)$, which is one, if x is odd and otherwise zero. The next bit y_1 is the parity of $\tau^{(1)}(x)$, and the bit after next y_2 is parity of $\tau \circ \tau(x)$ and so on. The conjugancy $Q \circ \tau = \sigma \circ Q$ can be demonstrated by transforming the expression as follows: $(\sigma \circ Q(x))_i = Q(x)_{i+1} = \tau^{(i+1)}(x)_0 = \tau^{(i)}(\tau(x))_0 = Q(\tau(x))_i$

A simulation of the Collatz function by Turing machines has been presented by Michel [11]. He introduces Turing machines that simulate the iteration of the Collatz function, where he considers them having 3 states and 4 symbols. Michel examines both turing machines, those that never halt and those that halt on the final loop.

A function-theoretic approach this problem has been provided by Berg and Meinardus [12], [13] as well as Gerhard Opfer [14], who consistently relies on the Bergs and Meinardus idea. Opfer tries to prove the Collatz conjecture by determining the kernel intersection of two linear operators U, V that act on complex-valued functions. First he determined the kernel of V, and then he attempted to prove that its image by U is empty. Benne de Weger [15] contradicted Opfers attempted proof.

Reachability Considerations based on a Collatz tree exist as well. It is well known that the inverted Collatz sequence can be represented as a graph; to be more specific, they can be

depicted as a tree [16], [17]. It is acknowledged that in order to prove the Collatz conjecture, then one needs to demonstrate that this tree covers all (odd) natural numbers.

The Stopping Time theory has been introduced by Terras [18], [19], [20]. He introduces another notation of the Collatz function $T(n) = (3^{X(n)}n + X(n))/2$, where $X(n) = 1$ when n is odd and $X(n) = 0$ when n is even, and defined the stopping time of n, denoted by $\chi(n)$, as the least positive k for which $T^{(k)}(n) < n$, if it exists, or otherwise it reaches infinity. Let L_i be a set of natural numbers, it is observable that the stopping time exhibits the regularity $\chi(n) = i$ for all n fulfilling $n \equiv l(mod 2^i)$, $l \in L_i$, $L_1 = \{4\}$, $L_2 = \{5\}$, $L_4 = \{3\}$, $L_5 = \{11, 23\}$, $L_7 = \{7, 15, 59\}$ and so on. As i increases, the sets L_i, including their elements, become significantly larger. Sets L_i are empty when $i \equiv l(mod 19)$ for $l = 3, 6, 9, 11, 14, 17, 19$. Additionally, the largest element of a non-empty set L_i is always less than 2^i.

Many other approaches exist as well. From an algebraic perspective Trümper [21] analyzes The Collatz Problem in light of an Infinite Free Semigroup. Kohl [22] generalized the problem by introducing residue class-wise affine, in short, by utilizing rcwa mappings. A polynomial analogue of the Collatz Conjecture has been provided by Hicks et al. [23] [24] and there are also stochastical, statistical and Markov chain-based and permutation-based approaches to proving this elusive theory.

2. Dependent Threads State Machine

2.1 Introducing a Dependent Threads State Machine (DTSM)

Let us regard a Finite State Machine $(\Sigma, S, s_0, \delta, F \in S)$, Σ as the input alphabet, S a set of states, s_0 the starting state, F a set of final states, and $\delta : S \times \Sigma \to S$ the transition function. We may concisely write $\delta_0(x) = \delta(x, 0) : S \to S$.

Definition 2.1 A DTSM (Dependent Threads State Machine) is a finite state machine that has the following properties:

1. $\Sigma = \{0, 1\}$, the input alphabet consists of two elements called bits. It is a binary alphabet.
2. $F = \{f_0\}$, the DTSM has only one final state.
3. $\delta_0(s_0) = s_0$, the DTSM remains in its starting state when inputting zero.
4. $\forall s \in S \setminus \{s_0\} : \exists n \geq 0, \delta_0^{(n)}(s) = f_0$, if the DTSM is in any state except s_0, a continuous input of zero leads to guaranteed f_0.
5. A transition $S \times \Sigma \to S$ is considered synonymous with a directed edge. Any bit that is an input of the function δ and thus a value of the corresponding edge, we call a δ-bit. Additionally, we label each edge with an ϵ-bit using a function $\epsilon : S \times \Sigma \to \Sigma$. The meaning of this labeling will be explored later.

It may be noted that the term "State Machine" can be treated as synonymous to the term "Automaton" (plural Automata). Hence, a Finite State Machine (FSM) may also be denoted as Finite State Automaton (FSA). In accordance with the automata theory, a FSM belongs to a special set of the Turing machines respective to all Turing machines.

2.2 The Collatz DTSM

The Collatz DTSM is an example of a DTSM and defined by four states $S = \{s_0, a, b, c\}$ and the functions δ, ϵ provided by table 2.1. The positions that are inputs of both functions δ and ϵ are the source positions, not the target positions.

δ	**0**	**1**
s_0	s_0	c
a	a	b
b	a	c
c	b	c

ϵ	**0**	**1**
s_0	0	0
a	0	1
b	1	0
c	0	1

Table 2.1: Definition of the both functions δ and ϵ

We can represent the δ and ϵ functions in a more compact way. Let's focus on the edges which connect the nodes s_0, a, b and c. Every edge has it's δ-bit and it's ϵ-bit. In such a way, we have a graph with double colored edges. The $\delta\epsilon$-adjuncency matrix is provided by table 2.2.

	s_0	a	b	c
s_0	0,0	-	-	1,0
a	-	0,0	1,1	-
b	-	0,1	-	1,0
c	-	-	0,0	1,1

Table 2.2: δ and ϵ as colorings of the transition edges

The graph of the Collatz DTSM is exhibited by Figure 2.1. The edges of the graph presented in Figure 2.1 are labeled with their $\delta\epsilon$-colorings. The node s_0 is the starting node. The node b is highlighted in blue, since all of it's outgoing edges have uniquely unequal δ- and ϵ-bits. Additionally, this node represents the center of symmetry.

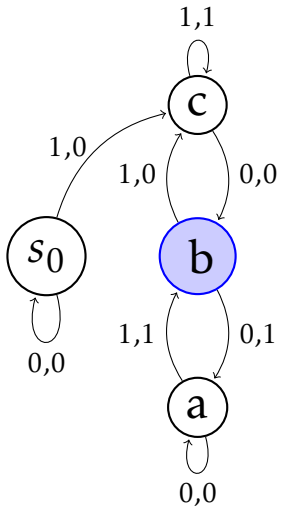


Figure 2.1: Graph Representation of the Collatz DTSM

2.3 Series of State Transition Sequences

Allow us to regard a binary sequence $(d_k)_{k\in\mathbb{N}_0}$ defined by a mapping $D : \mathbb{N} \to \Sigma$ that has a finite preimage $D^{-1}(1)$. In other words, a natural k exists, for which all $m \geq k$ are mapped to zero $D(m) = 0$ and thus all sequence members d_m are zero. This binary sequence describes the DTSM's state transitions starting from s_0. Hence the sequence members correspond to the δ-bits. In accordance to the DTSM definition, this sequence must end up and remain eternally in the state a. The following example illustrates the state transitions of the DTSM.

Example 2.1 Assume we have a sequence $(d_k) = (1,0,1,0,1,0,1,1,0,1,0,0,0\ldots)$. This sequence generates a sequence of DTSM positions $(p_k) = (c,b,c,b,c,b,c,c,b,c,b,a,a,\ldots)$.

It is important to point out that for an input bit d_k the corresponding position p_k is the target position into which the token moves to and not the source position from which the tokens moves from. Hence we consider the starting position s_0 to have a negative index -1, which in our notation means $p_{-1} = s_0$.

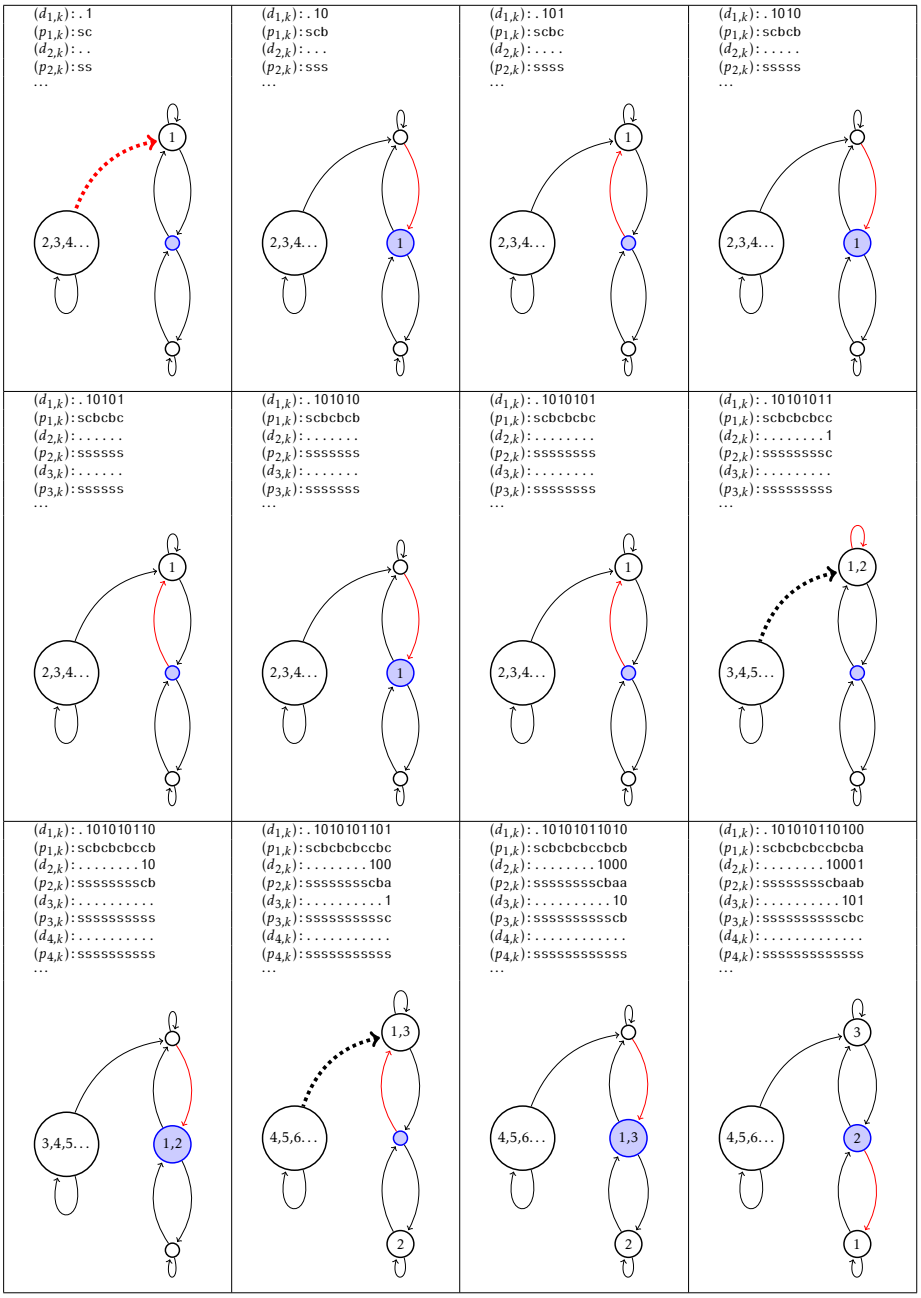
The ϵ-bit, an edge is labeled with, belongs to a sequence of ϵ-bits that result from the DTSM's state transitions. A sequence of δ-bits describes a sequence of state transitions through the DTSM's edges beginning at the starting node s_0. The sequence of ϵ-bits is defined by the order of passed edges in the walk through, which each naturally are specified by (labeled with) an ϵ-bit. A sequence of ϵ-bits forms the sequence of δ-bits describing the state transitions for the next walk through. This continuing principle is illustrated by table 2.3, which provides a simulative description (state by state) of these consecutive walk throughs up to 10101011010000000000.

The latter binary sequence is the δ-bit sequence of the first walk through. All other δ-bit sequences arise from the ϵ-bits of those edges that are traversed during this and subsequent walkthroughs. For example, the ϵ-bits of traversed edges by the second walk through form the sequence of δ-bits for the third walk through and beyond.

Along the δ-bit sequences, table 2.3 portrays the corresponding sequences of DTSM positions, which have been walked through up to their respective states.

In table 2.3 we can reproduce and verify the repeated application of ϵ-bit sequences. If we take a closer look at the example $(p_{1,k})$ which is the sequence of DTSM positions describing the first token's walk through, that generates the ϵ-bit sequence 00000001000100000000. This bit sequence corresponds to the δ-bit sequence $(d_{2,k})$ that defines the walk through of the second token.

Red colored edges indicate the first token's walk through. The edge between the starting node s_0 and the node c is highlighted dotted, when a new token becomes active by leaving the starting node, respectively moving to c. For greater readability, we symbolize the starting state simply with s.


$(d_{1,k})$: . 1
$(p_{1,k})$: sc
$(d_{2,k})$: . .
$(p_{2,k})$: ss
…
1
2,3,4…
$(d_{1,k})$: . 10
$(p_{1,k})$: scb
$(d_{2,k})$: . . .
$(p_{2,k})$: sss
…
1
2,3,4…
$(d_{1,k})$: . 101
$(p_{1,k})$: scbc
$(d_{2,k})$: . . . .
$(p_{2,k})$: ssss
…
1
2,3,4…
$(d_{1,k})$: . 1010
$(p_{1,k})$: scbcb
$(d_{2,k})$: . . . . .
$(p_{2,k})$: sssss
…
1
2,3,4…
$(d_{1,k})$: . 10101
$(p_{1,k})$: scbcbc
$(d_{2,k})$: . . . . . .
$(p_{2,k})$: ssssss
$(d_{3,k})$: . . . . . .
$(p_{3,k})$: ssssss
…
1
2,3,4…
$(d_{1,k})$: . 101010
$(p_{1,k})$: scbcbcb
$(d_{2,k})$: . . . . . .
$(p_{2,k})$: sssssss
$(d_{3,k})$: . . . . . .
$(p_{3,k})$: sssssss
…
1
2,3,4…
$(d_{1,k})$: . 1010101
$(p_{1,k})$: scbcbcbc
$(d_{2,k})$: . . . . . .
$(p_{2,k})$: ssssssss
$(d_{3,k})$: . . . . . .
$(p_{3,k})$: ssssssss
…
1
2,3,4…
$(d_{1,k})$: . 10101011
$(p_{1,k})$: scbcbcbcc
$(d_{2,k})$: 1
$(p_{2,k})$: sssssssss
$(d_{3,k})$: . . . . . .
$(p_{3,k})$: sssssssss
…
1,2
3,4,5…
$(d_{1,k})$: . 101010110
$(p_{1,k})$: scbcbcbccb
$(d_{2,k})$: 10
$(p_{2,k})$: sssssssscb
$(d_{3,k})$: . . . . . .
$(p_{3,k})$: ssssssssss
$(d_{4,k})$: . . . . . .
$(p_{4,k})$: ssssssssss
…
1,2
3,4,5…
$(d_{1,k})$: . 1010101101
$(p_{1,k})$: scbcbcbccbc
$(d_{2,k})$: 100
$(p_{2,k})$: ssssssssscba
$(d_{3,k})$: 1
$(p_{3,k})$: sssssssssssc
$(d_{4,k})$: . . . . . .
$(p_{4,k})$: ssssssssssss
…
1,3
4,5,6…
2
$(d_{1,k})$: . 10101011010
$(p_{1,k})$: scbcbcbccbcb
$(d_{2,k})$: 1000
$(p_{2,k})$: sssssssscbaa
$(d_{3,k})$: 10
$(p_{3,k})$: sssssssssscb
$(d_{4,k})$: . . . . . .
$(p_{4,k})$: ssssssssssss
…
1,3
4,5,6…
2
$(d_{1,k})$: . 101010110100
$(p_{1,k})$: scbcbcbccbcba
$(d_{2,k})$: 10001
$(p_{2,k})$: sssssssscbaab
$(d_{3,k})$: 101
$(p_{3,k})$: sssssssssscbc
$(d_{4,k})$: . . . . . .
$(p_{4,k})$: ssssssssssss
…
3
4,5,6…
2
1

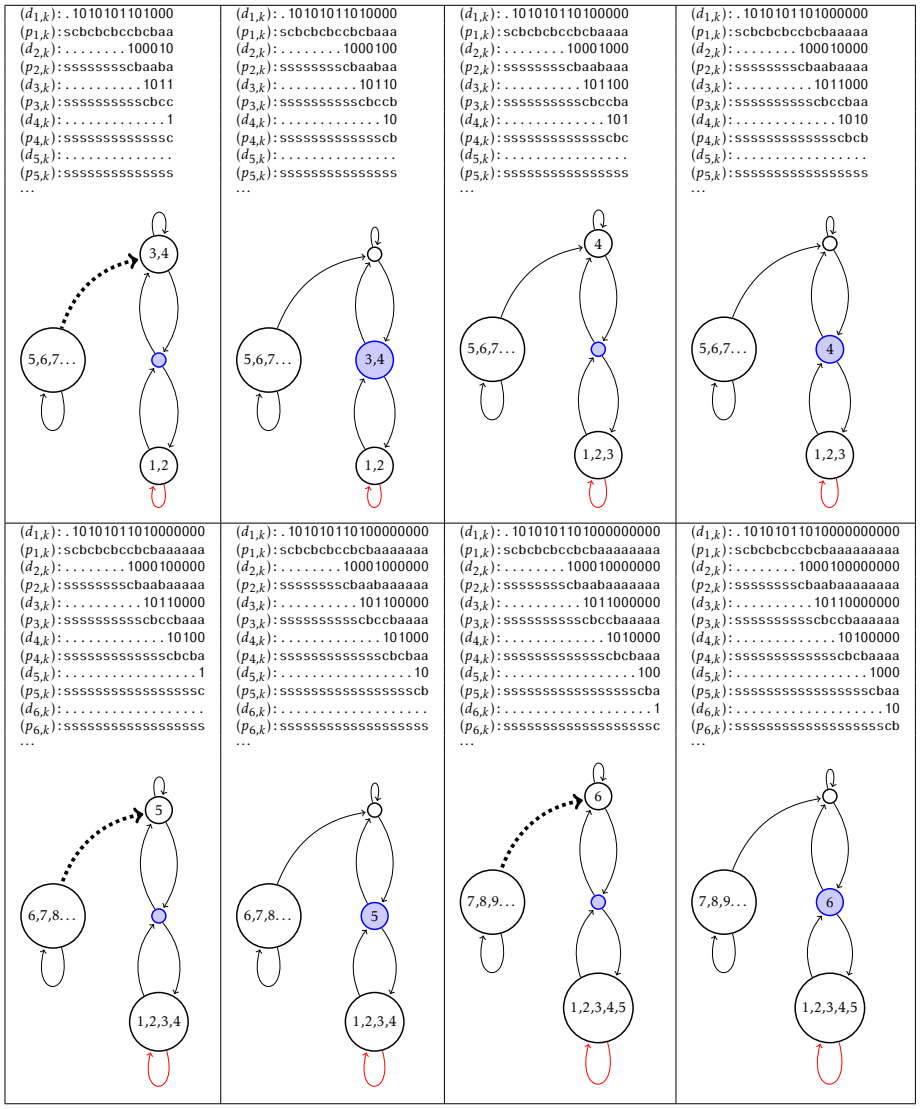


Table 2.3: Simulation of the Collatz DTSM's walk through

If we consider the δ-bit sequences $(d_{1,k}),(d_{2,k}),\ldots,(d_{6,k})$ shown in table 2.3 as the inverse binary representation of odd numbers (from the first 1 to the last 1) we are given the number sequence provided by table 2.4.

Sequence	Inverse binary	Binary	Decimal
$(d_{1,k})$	1010101101	1011010101	725
$(d_{2,k})$	10001	10001	17
$(d_{3,k})$	1011	1101	13
$(d_{4,k})$	101	101	5
$(d_{5,k})$	1	1	1
$(d_{6,k})$	1	1	1

Table 2.4: Collatz numbers that result from the walk throughs

This example corresponds to the Collatz sequence started at 725 (binary 1011010101, inverse binary 1010101101), which is illustrated by table 2.5.

725							
2176	1088	544	272	136	68	34	**17**
52	26	**13**					
40	20	10	**5**				
16	8	4	2	**1**			

Table 2.5: Collatz sequence started at 725

Theorem 2.1 All members of a Collatz sequence are equivalent to the sequences of δ-bits.

Proof. Let d be a positive odd number, the binary representation is accomplished through the bit sequence $(d_0, d_1, \ldots, d_{n-1})$ as follows: $d = d_0 \cdot 2^0 + d_1 \cdot 2^1 + \ldots + d_{n-1} \cdot 2^{n-1}$. Since d is odd, we know that $d_0 = 1$. Additionally we fill this sequence infinitely with zero bits on the right side, which means, an index exists n, for which all bits that are indexed with an $k \geq n$ are zero $d_{k \geq n} = 0$.

An operation $x \mapsto 3x + 1$ on d can be described as $(2d + 1) + d$, where the multiplication with 2 is equivalent to a right shift of the bit sequence. The addition with an integer 1 means to insert a bit 1 at the front of the bit sequence: $2d + 1 = (1, d_0, d_1, d_2, \ldots)$. The sum $(2d + 1) + d$ can be represented as follows:

$$
\begin{aligned}
&2(d_0 \cdot 2^0 + d_1 \cdot 2^1 + \ldots + d_{n-1} \cdot 2^{n-1}) + 1 + d \\
&= 1 + d_0 \cdot 2^1 + d_1 \cdot 2^2 + \ldots + d_{n-1} \cdot 2^n + d \\
&= 1 + d_0 \cdot 2^1 + d_1 \cdot 2^2 + \ldots + d_{n-1} \cdot 2^n + d_0 \cdot 2^0 + d_1 \cdot 2^1 + \ldots + d_{n-1} \cdot 2^{n-1} \\
&= 1 + d_0 + (d_0 + d_1) \cdot 2^1 + (d_1 + d_2) \cdot 2^2 + \ldots + (d_{n-1} + d_n) \cdot 2^n \\
&= 1 + d'_0 \cdot 2^0 + d'_1 \cdot 2^1 + \ldots + d'_n \cdot 2^n
\end{aligned}
$$

We initiate a sequence $(e_k)_{k \in \mathbb{N}_0}$ defined by the recurrence $e_k = \lfloor (e_{k-1} + d_{k-1} + d_k)/2 \rfloor$. And we define another (non-recursive) sequence $(d'_k)_{k \in \mathbb{N}_0}$ by $d'_k = mod(e_{k-1} + d_{k-1} + d_k, 2)$. For the sake of completeness we define $e_{-1} = 1$ and $d_{-1} = 0$, which can be understood as predecessor of a first sequence element, but which technically is not a member of the sequence. Because we have already stated that d is odd, we know that $d_0 = 1$ and thus $d'_0 = 0$:

$$
\begin{aligned}
d'_0 &= mod(1 + 0 + d_0, 2) = 0 & e_0 &= \lfloor (1 + 0 + d_0)/2 \rfloor = 1 \\
d'_1 &= mod(e_0 + d_0 + d_1, 2) & e_1 &= \lfloor (e_0 + d_0 + d_1)/2 \rfloor \\
d'_2 &= mod(e_1 + d_1 + d_2, 2) & e_2 &= \lfloor (e_1 + d_1 + d_2)/2 \rfloor \\
&\vdots & &\vdots \\
d'_n &= mod(e_{n-1} + d_{n-1} + d_n, 2) & e_n &= \lfloor (e_{n-1} + d_{n-1} + d_n)/2 \rfloor
\end{aligned}
\tag{2.1}
$$

A member of the sequence $(e_k)_{k \in \mathbb{N}_0}$ is the overflow bit that results from binary adding two successive members of $(d_k)_{k \in \mathbb{N}_0}$, which is illustrated with the example of $(2d + 1) + d = 27 + 13 = 40$:

$$
\begin{array}{cccccccc}
 & 0 & 0 & 0 & 1 & 1 & 1 \\
\hline
 & 0 & 1 & 1 & 0 & 1 & 0 \\
+ & 0 & 0 & 1 & 0 & 0 & 1 \\
\hline
 & 1 & 0 & 1 & 0 & 0 & 0
\end{array}
\quad = \quad
\begin{array}{cccccccc}
 & e_4 & e_3 & e_2 & e_1 & e_0 & 1 \\
\hline
 & d_4 & d_3 & d_2 & d_1 & d_0 & 0 \\
+ & d_5 & d_4 & d_3 & d_2 & d_1 & d_0 \\
\hline
 & d'_5 & d'_4 & d'_3 & d'_2 & d'_1 & d'_0
\end{array}
$$

We assume that the sequence $(d_k)_{k\in\mathbb{N}_0}$ is an odd input into the state machine controlling the n-th token. To prove the theorem it would be sufficient to demonstrate that the sequence $(d'_k)_{k\in\mathbb{N}_0}$ controls the next token which is the same as saying this sequence is equal to the sequence of that generated ϵ-bits as given later by (2.3). Let the n-th token be in a state $p_{n,k}$, that is $k+1$ steps have been taken by processing the sequence $(d_0,\ldots,d_k)$. Therefore $p_{n,-1} = s_0$ and $p_{n,0} = c$, since $d_0 = 1$. The state machine's mechanics are given by the following explicit definition of the n-th token's target position $p_{n,k}$ (shorthand noted as p_k) that depends on the sum $d_k + e_k$. At this point, we refer to example 2.1 to remind that p_k continually refers to the target (and not to the source) position.

$$p_k = \begin{cases} a & d_k + e_k = 0 \\ b & d_k + e_k = 1 \\ c & d_k + e_k = 2 \end{cases} \tag{2.2}$$

The sequence of ϵ-bits, generated by the walk of the n-th token is exactly equal to the sequence $(d'_k)_{k\in\mathbb{N}_0}$. We remember that in compliance with table 2.1 the input position of the ϵ-function is the token's source and not target position:

$$\epsilon(p_{n,k-1}, d_{n,k}) = d'_{n,k} \tag{2.3}$$

Using induction over k we prove that both the DTSM's mechanics (2.2) and the equality of the bit sequences stated in (2.3).

Start of induction: The n-th token is in the DTSM's starting node s, furthermore $k = 0$, $d_0 = 1$ (the input is odd) and as given by (2.1) we are also aware that $e_0 = 1$. The first bit $d_0 = 1$ of the input sequence causes the token to move from s_0 into node c (see figure 2.1). For this reason the construct (2.2) describing the DTSM's mechanics are correct. Because of the input's oddness we recognize as validated in (2.1) that $d'_0 = 0$. The edge connecting s_0 with c is labeled with the ϵ-bit 0, which conforms to the equality $\epsilon(p_{-1}, d_0) = \epsilon(s, 1) = d'_0 = 0$ stated by (2.3).

Induction steps: We assume that the statements (2.2) and (2.3) are valid for all integers up to $k-1$ (induction hypothesis) and we corroborate the validity of these statements for k. We will now examine the three cases defined by (2.2) separately.

Case 1: $p_{k-1} = c$. According to the induction hypothesis applied to (2.2) we require that the sum $d_{k-1} + e_{k-1} = 2$, which leads to the only possible bit-variable setting $d_{k-1} = e_{k-1} = 1$. By taking a closer look at the definition of the recurrence $e_k = \lfloor (e_{k-1} + d_{k-1} + d_k)/2 \rfloor$ given in (2.1), we recognize that e_k only accepts the value 1, no matter what binary value d_k has: $e_k = \lfloor (2 + d_k)/2 \rfloor$.

So far we have substantiated that $e_k = 1$ when a token has moved into the target position $p_{k-1} = c$ within the context of our induction hypothesis. In the course of proving the legitimacy of (2.2) for the next target position p_k, the node c is our source position. Conformant with the conditions given by (2.2) the token's next target position p_k is b for an input bit $d_k = 0$ ($d_k + e_k = 0 + 1 = 1$) and it remains c for an input $d_k = 1$ ($d_k + e_k = 1 + 1 = 2$). This is consistent with the DTSM's mechanics, see figure 2.1.

Now we must validate the correctness of the statement (2.3). In this case we have $d'_k = \epsilon(p_{k-1}, d_k) = \epsilon(c, d_k)$ and as of (2.1) we have $d'_k = mod(e_{k-1}+d_{k-1}+d_k, 2)$. Utilizing the induction hypothesis we know the sum $d_{k-1} + e_{k-1} = 2$ and thus $\epsilon(c, d_k) = mod(2 + d_k, 2) = d_k$. Since the ϵ-bit of each edge outgoing from c is equal to the σ- bit as shown in figure 2.1 the statement (2.3) is correct in this case as well.

Case 2: $p_{k-1} = b$. Analoguous to the first case, we apply the induction hypothesis to (2.2) and thus assume the sum $d_{k-1} + e_{k-1} = 1$, which leads us to $e_k = \lfloor (1 + d_k)/2 \rfloor = d_k$ by substituting this assumed sum into (2.1).

Now we prove the exactness of (2.2) for the next target position p_k, whereby b becomes the source position. In compliance with the DTSM's mechanics posed in figure 2.1, an input $d_k = 0$ causes the token to move to a. The condition defined by (2.2) for the target position $p_k = a$ is the sum $d_k + e_k = 0$, which is correct in our current case $d_k = e_k = 0$. An input $d_k = 1$ causes the token's movement from b to c. Here again the condition $d_k + e_k = 2$ specified by (2.2) matches our case $d_k = e_k = 1$.

Now we have to affirm the statement (2.3). For this we proceed in the same way as in the first case referring to (2.1) and substitute again into $d'_k = mod(e_{k-1} + d_{k-1} + d_k, 2)$ the sum $d_{k-1} + e_{k-1} = 1$ with the result that $d'_k = mod(1 + d_k, 2)$. The consequence is that d'_k has the inverse value of d_k. In fact each edge outgoing from b is labeled with a σ- and ϵ-bit that are mutually inverse (see figure 2.1). For this reason (2.3) is valid in this case too.

Case 3: $p_{k-1} = a$. The induction hypothesis will now be applied for this third case to (2.2) with the result $d_{k-1} + e_{k-1} = 0$, that is $d_{k-1} = e_{k-1} = 0$. With reference to (2.1) we are given $e_k = \lfloor d_k/2 \rfloor = 0$. Simply put e_k is in each case zero, notwithstanding which value d_k has.

So far we have authenticated that $e_k = 0$ when a token has moved into the target position $p_{k-1} = a$ within the context of our induction hypothesis. In the course of proving the legitimacy of (2.2) for the next target position p_k, the node a is our source position. As illustrated in 2.1 an input bit $d_k = 1$ will cause the tokens move to node b and an input $d_k = 0$ will cause the token to remain at position a. The formula (2.2) that explicitely defines the tokens target position p_k complies to this behaviour in its conditions $d_k + e_k = 1 + 0 = 1$ (token's movement to b) and $d_k + e_k = 0 + 0 = 0$ (token's stay in position a). Hence (2.2) is credible in this third case.

Finally we have to substantiate the statement (2.3) in this third case. Substituting the sum $d_{k-1} + e_{k-1} = 0$ into (2.1) we are given $d'_k = mod(d_k, 2) = d_k$ and thus $\epsilon(a, d_k) = d'_k = d_k$. Both edges that are outgoing from a have an ϵ- and σ-bit which are the same. Finally (2.3) is accurate as well in this third case.

The equality of the sequence $(d'_k)_{k \in \mathbb{N}_0}$ to the sequence of generated ϵ-bits has been validated and due to the fact that the sequence of ϵ-bits forms the sequence of σ-bits that describes the next token's walk through, we have proven the theorem 2.1.

□

3. Free Binary String Monoid

3.1 Defining a Free Binary String Monoid

Let us express a monoid M that is freely generated by the set of two elements p and q. The monoid's operator is the concatenation and it's elements are *strings* (similarly called *words*) made of the 2-letter alphabet $\{p,q\}$, where each letter represents an ϵ-bit of an integer that is a Collatz sequence member (p represents zero, q represents one). For instance, 5 is represented by qpq.

The complete specification of this monoid, including all of its characteristics, is given by the following definition 3.1

Definition 3.1 We define a Binary String Monoid as an algebraic structure $(M,+,e)$, which possesses the following properties:

1. M is freely generated over $A \subset M$, $A = \{p,q\}$. The subset A is the called *generator* or *base* of M. It satisfies $e \cup A \cup A^2 \cup \cdots \cup A^n \cup \cdots = M$ and if $u_1 \cdots u_n = v_1 \cdots v_m$, where $u_1,\ldots,u_n,v_1,\ldots,v_m \in A$, then $m = n$ and $u_i = v_i$, $i = 1,\ldots,n$ (see definition of free monoids [25, p. 5]). In other words, each string in M is unambiguously represented by the elements of A. Since A is finite, A is also symbolized by *alphabet*.

2. $+ : M \times M \to M$ is an associative operator that signifies the concatenation of elements of M, in our case the concatenation of the binary strings.

3. The neutral element e is and empty string, which demonstrably fulfills $e + u = u + e = u$ for all elements $u \in M$.

We can now describe a homomorphism $\phi : M \to S_3$ which maps M to the symmetric group S_3 on the set $S \setminus \{s_0\}$ of states defined by our DTSM, (see definition 2.1) excluding the starting state. Hence the image of ϕ contains the permutations of the DTSM's states $\{a,b,c\}$. The image of the string p is $(a\ c\ b)$ and the image of q is $(b\ a\ c)$. For better readability we have written permutations in one-line notation. We defined the image under ϕ of all elements of the generator $A = \{p,q\}$, which means that we have implicitly defined the image of all elements in M, due to the nature that any multi-letter string inside M is a concatenation of p and q (freely generated over A) and $\phi(u_1 + u_2) = \phi(u_1) \circ \phi(u_2)$, where $\circ$ is the operator for permutation composition in S_3.

Preimages of the identity permutation, more precisely the identity element $(a\ b\ c) \in S_3$, are all the strings that can be transformed into an empty string e by striking out substrings such as pp, qq, $pqpqpq$, whose image under ϕ is the identity element in S_3. One example is

the string $pqqp$, because we can strike out the substring qq from the middle and then gain the resulting string pp.

The mapping from M to S_3 by ϕ comprising all six elements of S_3 is revealed in table 3.1. It is worth noting, that, because ϕ is surjective, ϕ is an epimorphism and the preimage of each element in S_3 is non-empty.

string in M	**permutation in S_3**	**one-line notation**	**order**
e	$\begin{pmatrix} a & b & c \\ a & b & c \end{pmatrix}$	$(a\ b\ c)$	1
p	$\begin{pmatrix} a & b & c \\ a & c & b \end{pmatrix}$	$(a\ c\ b)$	2
q	$\begin{pmatrix} a & b & c \\ b & a & c \end{pmatrix}$	$(b\ a\ c)$	2
pq	$\begin{pmatrix} a & b & c \\ c & a & b \end{pmatrix}$	$(c\ a\ b)$	3
qp	$\begin{pmatrix} a & b & c \\ b & c & a \end{pmatrix}$	$(b\ c\ a)$	3
pqp, qpq	$\begin{pmatrix} a & b & c \\ c & b & a \end{pmatrix}$	$(c\ b\ a)$	2

Table 3.1: Definition of the mapping from M to S_3 by ϕ

The fourth column of table 3.1 contains the order of each element $s \in S_3$, which is defined as the smallest natural number n such that s^n is the identity element of S_3: $s^n = (a\ b\ c)$. In this context we define the order of an element $u \in M$ as the order of its image under ϕ.

Both operations, the concatenation of strings as well as the permutations chains are composed from right to left. To this end, we need to read any string $u \in M$ from right to left and the same applies to interpreting permutation chains. In contrast, sequences of bits or positions we read exactly as usual - from left to right.

We construct a function $f_c : M \to M$, which maps every string $u \in M$ to another string $v \in M$ in line with the following principle: The output string v results from the δ-bits, which a token at state c needs to traverse in order to produce an ϵ-bit sequence that is represented by the input string u. The δ-bit sequence behind v describes the token's walk through in concert with the permutation given by $\phi(u)$. We should like to return to example 2.1 and illustrate this principle by the following example 3.1.

Example 3.1 Let $u, v \in M$ and $f_c(u) = v = pppqpqqpqpqp$. The string v is the reverse representation of the bit sequence $(0,1,0,1,0,1,1,0,1,0,0,0)$, which is a subsequence of the sequence $(d_k)_{k\in\mathbb{N}_0}$ that we previously investigated in example 2.1. The sole difference consists in the fact that the succeeding does not contain the first δ-bit d_0 that initiated the token's move from the starting state s_0 to the state c. Indeed the function f_c, envisaged here, refers to a token's walk beginning at the state c. The image of u under ϕ is the composition of permutations: $(a\ c\ b) \circ (b\ a\ c) \circ (a\ c\ b)^3 \circ (b\ a\ c) \circ (a\ c\ b)^6 = \phi(pqpppqpppppp) = \phi(u)$.

This token residing in the state c will then accordingly move to b and back to c and will continue moving so that it is consistent with example 2.1 and the resulting sequence of DTSM positions $(b,c,b,c,b,c,c,b,c,b,a,a)$. Concurrently, we may suggest that the input string u that we pass to f_c is the reverse representation of the ϵ-bits generated by this walk through. We observed this walk through, which is defined by the bit sequence given through v, from the token's position at state c. Finally, u correctly relates to the binary form of the Collatz number 17 as demonstrated in table 2.4.

To surmise, the strings belonging to the set M, reversely written they represent bit sequences. These sequences are inverse representations of Collatz numbers. The nature of the Collatz sequences might tempt the reader to assume that the function f_c would not be injective, because of the two different input sequences; that is, the sequence taken from example 2.1 which is the inverse binary representation of 725 and the sequence which is the inverse binary representation of 181 both induce a walk through that generates an ϵ-bit sequence which is the inverse binary representation of one and the same Collatz number, namely of 17 - the successor of 725 and of 181. Nevertheless, the input strings u that are arguments of f_c for both cases differ from each other as demonstrated in table 3.2.

	case of input 725	**case of input 181**
δ-bit sequence	1010101101000	1010110100000
subsequence without first δ-bit	010101101000	010110100000
string $\boldsymbol{v}$ rep. the subsequence (reversly written)	pppqpqqpqpqp	pppppqpqqpqp
position sequence starting in c	bcbcbccbcbaa	bcbccbcbaaaa
permutation chain behind position sequence	$(acb)(bac)(acb)^3(bac)(acb)^6$	$(acb)^3(bac)(acb)^3(bac)(acb)^4$
string $\boldsymbol{u}$ rep. permutation chain	pqpppqpppppp	pppqpppqpppp
ϵ-bit sequence rep. by string $\boldsymbol{u}$ (reversly written)	000000100010	000010001000

Table 3.2: Processing the Collatz numbers 725 and 181 compared

Consequently, we define the functions f_a and f_b, where the token's walk begins at the DTSM state a and b. The set $F = \{f_a, f_b, f_c\}$ form together with the identity function id_F as a neutral element and the operator $\circ$ that acts as the function composition a monoid $(F, \circ, Id_F)$. And here again, the operator is a right-to-left composition.

Theorem 3.1 The functions f_a, f_b, f_c are all bijective.

Proof. The bijectivity of the function f_c can be proven in two steps: First we prove f_c is injective and then we prove it is surjective.

Injectivity: Each string in the preimage of f_C has its own unique matching string in the image. We need to prove that every string of the function's codomain (the functions image - also referred to as the target set), is the image of at most one string of it's domain (preimage). Let us recall, the function f_c when inputting a string $u_n \cdots u_2 u_1$ (that represents the ϵ-bit sequence of the token's walk through starting in state c) outputs the related δ-bit sequence, which induces this exact walk through.

Let $u, w \in M, u \neq w$ be two non-equal strings and let $i \in \mathbb{N}$ be the index that indicates the position of the first letter, in which both strings differ: $u_i \neq w_i$ and $\forall j < i : u_j = w_j$. The token resides first in state c, then the token will move in synchronization with the substring $u_{i-1} \cdots u_2 u_1 = w_{i-1} \cdots w_2 w_1$, and after the token has finished its walk through (defined by this substring) it will be located in a state $s \in \{a, b, c\}$. The δ-bits, that were the necessary

input of our DTSM for this walk through, are (in reverse order) represented by the image of $u_{i-1}\cdots u_2u_1$ under f_c and markedly in the same way by the image of $w_{i-1}\cdots w_2w_1$ under f_c.

As per definition, our DTSM has no state, from which a token, when inputting the same δ-bit, has two different movement possibilities (directions) that would generate a different ϵ-bit. In other words, no state defined by our DTSM has two outgoing edges that are labeled with a same δ-bit and a different ϵ-bit (see figure 2.1). Quite to the contrary, each state has two outgoing edges, which differ in their δ-bit. Even beyond the states a, b and c have solely outgoing edges (naturally two) that differ both in their δ-bit and ϵ-bit. This leads to the fact that due to $u_i \neq w_i$ the images $f_s(u_i)$ and $f_s(w_i)$ are unequal $f_s(u_i) \neq f_s(w_i)$ and thus $f_c(u) \neq f_c(w)$. Formally, the compositions may be expressed as follows:

$$\begin{aligned} f_c(u_i\cdots u_2u_1) &= f_s(u_i) \circ f_c(u_{i-1}\cdots u_2u_1) \\ \neq f_c(w_i\cdots w_2w_1) &= f_s(w_i) \circ f_c(w_{i-1}\cdots w_2w_1) \end{aligned}$$

Surjectivity: Each string in the image of f_c has at least one matching string in the preimage. In other words, there will not be a string in the codomain (target set) that is left out. Every string $v_n\cdots v_2v_1$ that represents a δ-bit sequence written in a reverse order, moves a token residing at state c through the positions c, b and a. The ϵ-bits generated by this walk form a sequence that is (in reverse order) represented by the string $u_n\cdots u_2u_1$. The latter string is precisely the preimage of $v_n\cdots v_2v_1$ under f_c: $f_c(u_n\cdots u_2u_1) = v_n\cdots v_2v_1$. Hence we have proved that every bit sequence (represented by a string that is an element of M and thus composed of letters p and q) has a preimage under f_c and therefore surjectivity is proven.

The argumentation used for proving that the function f_c is bijective is similarly applied for the functions f_a and f_b. Ultimately the functions f_a, f_b and f_c are all bijections.

□

4. Conclusion and Outlook

4.1 Summary

We have defined a structure in Algebraic Automata Theory, which we call Dependent Threads State Machine (DTSM). Building on this, we introduced an example called Collatz DTSM, which utilizes Collatz sequences (in binary form) as an input alphabet and is able to traverse these sequences. Finally, we proved that any binary coded Collatz sequence is equivalent to the sequence of the Collatz DTSM's input bits (δ-bits).

4.2 Outlook

By introducing the DTSM we defined an algebraic structure of automata that forms a basis for further research.

Bibliography

[1] S. W. Williams. “Million Buck Problems”. In: *National Association of Mathematicians Newsletter* 31.2 (2000), pp. 1–3.

[2] J. C. Lagarias. *The Ultimate Challenge: The 3x+1 Problem*. Providence, RI: American Mathematical Society, 2010. ISBN: 978-0821849408.

[3] J. C. Lagarias. “The 3x + 1 Problem: An Annotated Bibliography (1963-1999)”. In: *ArXiv Mathematics e-prints* (2011). eprint: math/0309224v13.

[4] J. C. Lagarias. “The 3x + 1 Problem: An Annotated Bibliography, II (2000-2009)”. In: *ArXiv Mathematics e-prints* (2012). eprint: math/0608208v6.

[5] J. C. Lagarias. “The 3x + 1 Problem and Its Generalizations”. In: *The American Mathematical Monthly* 92.1 (1985), pp. 3–23.

[6] S. Kahermanes. *Collatz Conjecture*. Tech. rep. Math 301 Term Paper. San Francisco State University, 2011.

[7] M. Klisse. “Das Collatz-Problem: Lösungs- und Erklärungsansätze für die 1937 von Lothar Collatz entdeckte (3n+1)-Vermutung”. 2010.

[8] C. A. Feinstein. “The Collatz 3n+1 Conjecture is Unprovable”. In: *Global Journal of Science Frontier Research Mathematics and Decision Sciences* 12.8 (2012), pp. 13–15.

[9] E. Akin. “Why is the 3x + 1 Problem Hard?” In: *Chapel Hill Ergodic Theory Workshops*. Ed. by I. Assani. Vol. 356. Contemporary Mathematics. Providence, RI: American Mathematical Society, 2004, pp. 1–20. DOI: http://dx.doi.org/10.1090/conm/364.

[10] D. J. Bernstein and J. C. Lagarias. “The 3x + 1 Conjugacy Map”. In: *Canadian Journal of Mathematics* 48 (1996), pp. 1154–1169.

[11] P. Michel. “Simulation of the Collatz 3x + 1 function by Turing machines”. In: *ArXiv Mathematics e-prints* (2014). eprint: 1409.7322v1.

[12] L. Berg and G. Meinardus. “Functional Equations Connected With The Collatz Problem”. In: *Results in Mathematics* 25.1 (1994), pp. 1–12. DOI: 10.1007/BF03323136.

[13] L. Berg and G. Meinardus. "The 3n+1 Collatz Problem and Functional Equations". In: *Rostocker Mathematisches Kolloquium*. Vol. 48. Rostock, Germany: University of Rostock, 1995, pp. 11–18.

[14] G. Opfer. "An analytic approach to the Collatz 3n + 1 Problem". In: *Hamburger Beiträge zur Angewandten Mathematik* 2011-09 (2011).

[15] B. de Weger. *Comments on Opfer's alleged proof of the 3n + 1 Conjecture*. Tech. rep. Eindhoven University of Technology, 2011.

[16] S. Andrei and C. Masalagiu. "About the Collatz conjecture". In: *Acta Informatica* 35.2 (1998), pp. 167–179. DOI: 10.1007/s002360050117.

[17] S. Kak. *Digit Characteristics in the Collatz 3n+1 Iterations*. Tech. rep. Oklahoma State University, 2014.

[18] R. Terras. "A stopping time problem on the positive integers". In: *Acta Arithmetica* 30.3 (1976), pp. 241–252.

[19] T. Oliveira e Silva. "Maximum Excursion and Stopping Time Record-Holders for the 3x + 1 Problem: Computational Results". In: *Mathematics of Computation* 68.225 (1999), pp. 371–384.

[20] M. A. Idowu. "A Novel Theoretical Framework Formulated for Information Discovery from Number System and Collatz Conjecture Data". In: *Procedia Computer Science* 61 (2015), pp. 105–111.

[21] M. Trümper. "The Collatz Problem in the Light of an Infinite Free Semigroup". In: *Chinese Journal of Mathematics* (2014), pp. 105–111. DOI: http://dx.doi.org/10.1155/2014/756917.

[22] S. Kohl. "On conjugates of Collatz-type mappings". In: *International Journal of Number Theory* 4.1 (2008), pp. 117–120. DOI: http://dx.doi.org/10.1142/S1793042108001237.

[23] K. Hicks et al. "A Polynomial Analogue of the 3n + 1 Problem". In: *The American Mathematical Monthly* 115.7 (2008), pp. 615–622.

[24] B. Snapp and M. Tracy. "The Collatz Problem and Analogues". In: *Journal of Integer Sequences* 11.4 (2008).

[25] H. Xie. *Grammatical Complexity and One-Dimensional Dynamical Systems*. Vol. 6. Directions in Chaos. Singapore: World Scientific Publishing, 1996. ISBN: 978-981-02-2398-4.