2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems

# How to Fast Verify Collatz Conjecture by Automata

Wei Ren[1,2,3,*], Ruiyang Xiao[1]

[1]*School of Computer Science, China University of Geoseciences, Wuhan, P. R. China, 430074*

[2]*Hubei Key Laboratory of Intelligent Geo-Information Processing, China University of Geosciences, Wuhan, P. R. China*

[3]*Guizhou Provincial Key Laboratory of Public Big Data, Guizhou University, Guizhou, P. R. China*

[*]*Corresponding Author: weirencs@cug.edu.cn*

***Abstract*—To verify Collatz conjecture for a given integer, we need to justify whether or not the given starting number goes to 1 finally. Current method computes 3*x+1 and x/2 one by one. In this paper, we propose an automata-based algorithm to compute the process in a batch with m steps - (3*x+1)2 or x/2. We also propose reduced Collatz conjecture and we prove it is equivalent to Collazt conjecture. We also point out that original dynamics is the combination of reduced dynamics. (Original dynamics is a serial of occurred 3*x+1 and x/2 computation in the process from a starting number to 1; Reduced dynamics is a serial of occurred 3*x+1 and x/2 computation in the process from a starting number to the first transformed number that is less than the starting number.) We study the properties of reduced dynamics and discover that reduced dynamics for some starting numbers can be determined directly without conducting concrete 3*x+1 or x/2 computations. We build a binary tree called Collatz tree and use it as a visual tool for automata design. We find that each partition of starting number residue class decides the further computation is whether 3*x+1 or x/2. The proposed automata can output code segment with m+2 computations in terms of (3*x+1)/2 or x/2 by taking as input x=4t+3, where t is in residue class i module $2^m$ and m is a natural number. We finally propose a conjecture whose proof implies the proof of Collatz conjecture by using proposed Collatz tree.**

***Index Terms*—Collatz Conjecture; 3x+1 Problem; Discrete Mathematics; Automata; Number Theory**

## 1. Introduction

The Collatz conjecture is a mathematical conjecture that is first proposed by Lothar Collatz in 1937. It is also known as the 3x+1 conjecture, the Ulam conjecture, the Kakutani's problem, the Thwaites conjecture, or the Syracuse problem [1], [2], [3].

The conjecture can be stated simply as follows: Take any positive integer number $x$. If $x$ is even, divide it by 2 to get $x/2$. If $x$ is odd, multiply it by 3 and add 1 to get $3*x+1$. Repeat the process again and again. The Collatz conjecture is that no matter what the number (i.e., $x$) is taken, the process will always eventually reach 1.

We verified $2^{100000}-1$ can return to 1 after 481603 times of $3*x+1$ computation, and 863323 times of $x/2$ computation, which is the largest integer being verified in the world [4]. We also point out a new approach for the proof of Collatz conjecture [5]. The source code and output data by computer programs can be accessed in public repository [6], [7], [8], [9], [10], [11].

To verify the Collatz conjecture, we always take an integer and compute whether it can return 1 after finite times of computation of $3*x+1$ and $x/2$. This process can be looked as a Collatz dynamics of an inputting integer, which is a sequence of occurred $3*x+1$ or $x/2$.

Existing methods that can output dynamics mainly rely on computing $3*x+1$ or $x/2$ one by one; no algorithm by computing *in a batch* are proposed. To simplifying the computation, we concentrate on reduced dynamics, which is a serial of occurred computations such as $3*x+1$ and $x/2$ during the process from a starting number $x$ not to 1, instead, to the first transformed number less than the starting number. We proved that reduced dynamics are the building blocks of dynamics.

In this paper, we propose an automata to compute reduced dynamics. The major contributions and results of the paper are listed as follows:

1) We propose an algorithm based on automata that can generate dynamics.
2) We discover that next $m+2$ computations in terms of $(3*x+1)/2$ and $x/2$ in dynamics can be determined directly by $(x-3)/4 \in [i]_{2^m}, m \in \mathbb{N}$.
3) We propose a conjecture whose proof will imply the proof of Collatz conjecture by using proposed Collatz tree.

The rest of the paper is organized as follows. Section 2 presents backgrounds and preliminaries. Section 3 provides details on the automata design. Finally, Section 4 concludes the paper.

## 2. Preliminaries

***Definition 2.1.*** Collatz Transformation

$$CT(x) = \begin{cases} TPO(x) = 3*x+1 & (TPO) \quad (x \in [1]_2) \\ H(x) = x/2 & (H) \quad (x \in [0]_2) \end{cases} \tag{1}$$

where $[1]_2 = \{x | x \equiv 1 \mod 2, x \in \mathbb{N}\}$, $[0]_2 = \{x | x \equiv 0 \mod 2, x \in \mathbb{N}\}$. $\mathbb{N} = \{a | a \in \mathbb{Z}, a \geq 1\}$. $TPO(x)$ can be simply denoted as $TPO$, and $H(x)$ can be simply denoted as $H$. Generally, $[i]_m = \{x | x \equiv i \mod m, x \in \mathbb{N}^*, m \geq 2, m \in \mathbb{N}^*, 0 \leq i \leq m-1, i \in \mathbb{N}\}$.

***Definition 2.2.*** Collatz Conjecture. $\forall x \in \mathbb{N}$, $\exists p \in \mathbb{N}$, such that $CT^p(x) = 1$ where $CT$ is Collatz Transformation.

When $x = 1$, Collatz Conjecture is held. ($1 \to 4 \to 2 \to 1$). In the following, we mainly concern $x \geq 2$.

We propose Reduced Collatz Conjecture as follows:

***Conjecture 2.3.*** Reduced Collatz Conjecture. $\forall x \in \mathbb{N}$ and $x \geq 2$, $\exists p \in \mathbb{N}$, such that $CT^p(x) < x$ where $CT$ is Collatz Transformation.

***Proposition 2.4.*** Collatz Conjecture is equivalent to Reduced Collatz Conjecture.

**Proof** (1) If Collatz conjecture is true, $\forall x \in \mathbb{N}, x \geq 2$, $\exists p \in \mathbb{N}$, $CT^p(x) = 1 < x$. Thus, Reduced Collatz Conjecture is true.

(2) Inversely, suppose Reduced Collatz Conjecture is true. $\forall x \in \mathbb{N}, x \geq 2$, $\exists p \in \mathbb{N}$, $CT^p(x) < x$.

If $CT^p(x) > 1$, let $y_1 = CT^p(x)$, thus $\exists q_1 \in \mathbb{N}$, $CT^{q_1}(y_1) < y_1$.

Iteratively, if $y_i = CT^{q_{i-1}}(y_{q_{i-1}}) > 1$, $\exists q_i \in \mathbb{N}$, $y_{i+1} = CT^{q_i}(y_i) < y_i$.

Thus, $y_{i+1} < y_i < ... < y_1 < x$.

Besides, $y_{i+1} = CT^{p+q_1+q_2+...q_i}(x) \geq 1$.

Thus, after finite times of iteration, or $\exists n \in \mathbb{N}$, $y_n = 1$. Thus, $\exists q \in \mathbb{N}$, $CT^q(x) = 1$, where $q = p + q_1 + ... + q_{n-1} = p + \Sigma_{i=1}^{n-1} q_i$.

Thus, Conjecture Conjecture is true. ∎

**Notations**.

(1) Original Dynamics and Reduced Dynamics. Original Dynamics represents a serial of occurred Collatz transformations during the procedure from a starting number to 1. In contrast, Reduced Dynamics represents a serial of occurred Collatz transformations during the procedure from a starting number to, not 1, but the *first* transformed number that is less than the starting number. Both can be looked as a string consisting of $TPO$ and $H$.

For example, the original dynamics of 5 is $5 \to 16 \to 8 \to 4 \to 2 \to 1$. That is, the original dynamics are $TPO, H, H, H$. The reduced dynamics of 5 is $TPO, H, H$, to reach the first number less than 5, i.e., 4.

Note that, reduced dynamics is more primitive than original dynamics, due to Proposition 2.4.

(2) Starting number and Transformed number. Starting number is the integer to be checked on its dynamics. All numbers that is the result of Collatz transformation in dynamics are called transformed numbers.

In above example, 5 is the starting number and the others (i.e., 16, 8, 4, 2, and 1) are transformed numbers.

(3) Original Code and Reduced Code. To simplifying the notation and fast outputting dynamics in computer programs, we use "1" and "0" to represent $TPO$ and $H$, respectively. Hereby $TPO(x) = 3 * x + 1$ is represented by 1 and $H(x) = x/2$ is represented as 0. The original dynamics that is represented by a string consisting of "1" and "0" is called original code. The reduced dynamics that is represented by a string consisting of "1" and "0" is called reduced code. E.g., original code for 5 is 1000, and reduced code for 5 is 100.

Note that, we assume the code for 1 is 100, although the final transformed number is not less than 1 but only equal 1. In the following, we always discuss codes for $\{x | x \geq 2, x \in \mathbb{N}\}$.

(5) $\mathsf{CODE}[x]$. As the reduced code is the major concern, we introduce a short hand representation for the convenience of presentation. That is, we denote the reduced code for $x$ as $\mathsf{CODE}[x]$.

Note that, $\mathsf{CODE}[x]$ is introduced *only* as a notation mainly for representing reduced code for $x$; We do not assume the existence of $\mathsf{CODE}[x]$ for $\forall x \in \mathbb{N}$, which is exactly what Reduced Collatz Conjecture wants to prove.

$\mathsf{CODE}[x]$ represents the reduced dynamics of $x$ that is already known or outputted by our computer program. We output reduced codes for all $x \in [3, 99999999]$.

Besides, Collatz transformations (i.e, $TPO(\cdot)$ and $H(\cdot)$) are both functions. Thus, given a starting number $x$, $CT(\cdot)$ is decidable and the transformed number after each Collatz transformation is unique. A serial of $CT(\cdot)$ and all transformed numbers are both unique. Thus, reduced code for a given starting number is unique (if it exists).

*Remark*

1) If Reduced Collatz Conjecture is true, then $\forall x \in \mathbb{N}$, $\mathsf{CODE}[x]$ exists; If $\forall x \in \mathbb{N}$, $\mathsf{CODE}[x]$ exists, then Reduced Collatz Conjecture is true.
2) If $n$ is finite, the computer program that takes as input $x$ and outputs $\mathsf{CODE}[x] = \{0,1\}^n$ can be terminable; If the computer program that takes as input $x$ and outputs $\mathsf{CODE}[x] = \{0,1\}^n$ can be terminable, then $n$ is finite.

***Proposition 2.5.*** $\mathsf{CODE}[x \in [0]_2] = 0$. $\mathsf{CODE}[x \in [1]_4] = 100$.

**Proof** $x \in [0]_2$, thus $H$ occurs. $x/2 < x$, thus $\mathsf{CODE}[x \in [0]_2] = 0$.

If $x = 1$, $\mathsf{CODE}[1] = 100$. (by assumption.)

If $x \geq 2$, let $x = 4t + 1 \in [1]_2$, where $t \in \mathbb{N}$. Thus, $3 * x + 1 = 3 * (4t + 1) + 1 = 12t + 4 = 4 * (3t + 1) \in [0]_4$. $(3 * x + 1)/4 = 3t + 1 < x = 4t + 1 \quad (\because t > 0)$. In summary, $\mathsf{CODE}[x \in [1]_4] = 100$. ∎

***Theorem 2.6.*** (Together Theorem.) "10" always together occurs in original (or reduced) dynamics.

**Proof** If $x \in [1]_2$, we have $3 * x + 1 \in [0]_2$. Thus, the next transformation immediately after "$TPO$" must be "$H$". Therefore, "10" always together occurs in original (or reduced) dynamics. ∎

Due to Theorem 2.6, we introduce two notations as follows:

1) $H(\cdot)$ always occurs after $TPO(\cdot)$, thus $H(TPO(x))$ can be written together, which is denoted by $I(x)$. That is, $I(x) = H(TPO(x))$.
2) $O(\cdot)$ is used to denote $H(\cdot)$ (for easily remembering).

For example, $\mathsf{CODE}[x = 5] = 100$, so reduced dynamics sequences (i.e., transformation sequences) for $x = 5$ are $TPO$, $H$, and $H$. The transformation procedures are $TPO(x)$, $H(TPO(x))$, and $H(H(TPO(x)))$. It can also be simplified as $O(I(x)) = H(H(TPO(x)))$. Thus, "100" can be written as "$IO$". In other words, $\mathsf{CODE}[x = 5] = IO$. Also, $O(I(x))$ can be simply written as $IO(x)$. In other words, $100(x) = H(H(TPO(x))) = IO(x) = O(I(x))$, where $100(\cdot)$ and $IO(\cdot)$ are both a *composite* function. E.g., $100(5) = IO(5) = O(I(5)) = O((3*5+1)/2 = 8) = O(8) = 8/2 = 4 < 5$. In the following, we may regard $\mathsf{CODE} \in \{I, O\}^*$, $\mathsf{CODE} \in \{0, 1\}^*$, or $\mathsf{CODE} \in \{10, 0\}^*$ in different contexts.

***Proposition 2.7.*** If $\mathsf{CODE}[x \in [3]_4]$ exists, $\mathsf{CODE}[x] = \{10\}^p \| ..., p \geq 2$.

**Proof** $x = 4t+3 \in [1]_2$, $(3x+1)/2 = (12t+10)/2 = 6t+5 \in [1]_2$. $(3(6t+5)+1)/2 = 9t+8$. If $t \in [0]_2$, $9t+8 \in [0]_2$. Thus, the next transformation is "$x/2$". Thus, the first five Collatz transformations are "10100", or "$IIO$". If $t \in [1]_2$, $9t+8 \in [1]_2$. Thus, the first six Collatz transformations are "101010", or "$III$". ∎

(In the following, $\mathsf{CODE}[x] = II...O$ where $x = 4t+3, t \in [i]_m$ can be written in a short hand as $\mathsf{CODE}[x = 4t+3, t \in [i]_m] = II...O$.)

***Proposition 2.8.*** $\mathsf{CODE}[x = 4t+3, t \in \{0\} \cup [0]_4] = IIOO$.

**Proof** $x = 4t+3 \in [1]_2$.
$I(x) = (3x+1)/2 = (3(4t+3)+1)/2 = (12t+10)/2 = 6t+5 \in [1]_2$, and $I(x) = 6t+5 > 4t+3 = x$. Thus, "$I$" occurs consequently.
$II(x) = I(6t+5) = (3(6t+5)+1)/2 = 9t+8 \in [(9*0+8) \mod 4]_4 = [0]_4$. Thus, "$OO$" occurs consequently.
As $IIOO(x) = OO(9t+8) = (9t+8)/2/2 = 2.25t+2 < 3t+2 < 4t+3 = x$, the code *ends* hereby with "$IIOO$". That is, $\mathsf{CODE}[x = 4t+3, t \in \{0\} \cup [0]_4] = IIOO$. ∎

**Example** $115 \rightarrow 346 \rightarrow 173 \rightarrow 520 \rightarrow 260 \rightarrow 130 \rightarrow 65 < 115$. Thus, $\mathsf{CODE}[115] = IIOO$. It can be verified that $x = 115 \in [3]_{16}, t = (115-3)/4 = 28 \in [0]_4$.

***Proposition 2.9.*** $\mathsf{CODE}[x \in [3]_4, t = (x-3)/4 \in [2]_8] = IIOIO$.

**Proof** $x = 4t+3 \in [1]_2$.
$I(x) = (3x+1)/2 = (12t+10)/2 = 6t+5 \in [1]_2$.
$II(x) = I(6t+5) = (3(6t+5)+1)/2 = 9t+8 \in [(2*9+8) \mod 8]_8 = [2]_8 \subset [0]_2$. "$O$" occurs consequently.
$IIO(x) = (9t+8)/2 \in [2/2]_{8/2} = [1]_4 \subset [1]_2$. Thus, next transformation is "$I$". Besides, $(9t+8)/2 = 4.5t+4 > 4t+3 = x$.
$IIOI(x) = (3*\frac{9t+8}{2}+1)/2$. $3*\frac{9t+8}{2}+1 \in [(3*1+1) \mod 4]_4 = [0]_4$. $(3*\frac{9t+8}{2}+1)/2 \in [0]_2$. Thus, "$O$" occurs consequently.
$IIOIO(x) = (3*\frac{9t+8}{2}+1)/2/2 = (13.5t+13)/2/2 = (6.75t+6.5)/2 = 3.375t+3.25 = 4t+3+(0.25-0.625t) < 4t+3 = x$, the code ends with "$IIOIO$". That is, $\mathsf{CODE}(x \in [11]_{32}) = IIOIO$. ∎

***Proposition 2.10.*** $\mathsf{CODE}[x \in [3]_4, t = (x-3)/4 \in [5]_8] = IIIOO$.

**Proof** $x = 4t+3 \in [1]_2$, $t \in [5]_8$.
$I(x) = (3x+1)/2 = (12t+10)/2 = 6t+5 \in [1]_2$, thus next transformation is "$I$".
$II(x) = I(6t+5) = (3(6t+5)+1)/2 = 9t+8 \in [(9*5+8) \mod 8]_8 = [5]_8 \subset [1]_2$, thus next transformation is "$I$".
$III(x) = I(9t+8) = (3(9t+8)+1)/2$. $3(9t+8)+1 \in ([(3*5+1) \mod 8]_8 = [0]_8$. $(3(9t+8)+1)/2 \in [0]_4$, thus next transformations are double "$O$".
Check whether current transformed number is less than the starting number as follows:
$IIIO(x) = (27t+25)/2/2 = (13.5t+12.5)/2 = 6.75t+6.25t > 4t+3 = x$.
$IIIOO(x) = (6.75t+6.25)/2 = 3.375t+3.125 = 4t+3+(0.125-0.625t) < 4t+3$, as $t \geq 1$. ∎

Following equation can be obtained due to above analysis. Note that, a special case is not included in the equation (as we always assume $t \in \mathbb{N}$). That is, when $t = 0$, it can be verified that $\mathsf{CODE}[x = 3] = IIOO$.

$$\mathsf{CODE}[x] = \begin{cases} 0 & O & x \in [0]_2, \\ 100 & IO & x \in [1]_4, \\ 101000 & IIOO & x = 4t+3, t \in [0]_4, \\ 10100100 & IIOIO & x = 4t+3, t \in [2]_8, \\ 10101000 & IIIOO & x = 4t+3, t \in [5]_8. \end{cases} \quad (2)$$

## 3. Automata Design

In this section, we analyze the properties on dynamics, especially, the segment in dynamics. Indeed, we discover a "forking" form, in which the further Collatz transformation is distinct if current residue class is partitioned. We thus create a tree called Collatz tree for the better understanding. The exploration has two motivation: one is to find (the properties of) all reduced codes, which can be looked as leaf nodes; the other is to find the relation between code segments and current inputting $x$. That is, we try to find how to output a segment of dynamics in a batch according to current transformed number. This batch segment will perform as a subprocess step in our automata design.

As $\mathsf{CODE}[x \in [0]_2] = 0$ and $\mathsf{CODE}[x \in [1]_4] = 100$, we mainly analyze an integer $x$ with $x = 4t+3, t \in [i]_m$, i.e., its ongoing segments of $\mathsf{CODE}[x = 4t+3, t \in [i]_m]$ where $m \in \mathbb{N}, 0 \leq i \leq m-1, i \in \mathbb{N}$. To simplify the discussion, here we focus on $m \leq 8$.

(1) $(x-3)/4 = t \in [1]_2$.

$x = 4t+3 \in [1]_2$. Thus, $\mathsf{CODE}[x]$ should be "$10\|*$" (if it exists). We will omit "(if it exists)" in the following. Besides, $* = \{10, 0\}^{\geq 1}$.

$I(x) = (3x+1)/2 = (3(4t+3)+1)/2 = (12t+10)/2 = 6t+5 \in [1]_2$. Thus, $\mathsf{CODE}[x]$ should be "$1010\|*$".

$I^2(x) = (3(6t+5)+1)/2 = 9t+8 \in [1]_2$ due to $t \in [1]_2$. Thus, $\mathsf{CODE}[x]$ should be "$101010\|*$".

$I^3(x) = (3(9t+8)+1)/2 = (27t+25)/2$. It cannot be determined whether it is even or odd. Until now, we have $\mathsf{CODE}[x = 4t+3, t \in [1]_2] = 101010\|* = I^3\| *$. (Here $\underbrace{II...I}_{n}$ is denoted as $I^n$ in a short hand. Indeed, $\underbrace{1010...10}_{n}$ is denoted as $\{10\}^n$.)

For obtaining further dynamics, partition of $t$ should be conducted for deciding whether $I^3(x) \in [0]_2$, or $I^3(x) \in [1]_2$.

We divide $(x-3)/4 = t \in [1]_2$ into two partitions - $t \in [1]_4$ and $t \in [3]_4$, because $[1]_2 = [1]_4 \cup [3]_4$.

(1.1) (Branch $[1]_2 \to [1]_4$.) If $t \in [1]_4$, then $(27t+25)/2 \in (27[1]_4+25)/2 = ([3]_4+[1]_4)/2 = [0]_2$, Thus, $\mathsf{CODE}[x = 4t+3, t \in [1]_4] = 1010100\|* = I^3\|O\|*$.

(1.2) (Branch $[1]_2 \to [3]_4$.) If $t \in [3]_4$, then $(27t+25)/2 \in (27[3]_4+25)/2 = ([1]_4+[1]_4)/2 = [1]_2$, Thus, $\mathsf{CODE}[x = 4t+3, t \in [3]_4] = 10101010\|* = I^3\|I\|*$.

Similarly, to obtain further dynamics, we divide $t \in [1]_4$ into two partitions - $t \in [1]_8$ and $t \in [5]_8$, because $[1]_4 = [1]_8 \cup [5]_8$.

(1.1.1) (Branch $[1]_2 \to [1]_4 \to [1]_8$.) If $t \in [1]_8$, then $(27t+25)/2/2 = (27[1]_8+25)/2/2 \in ([3]_8+[1]_8)/4 = [1]_2$. Thus, $\mathsf{CODE}[x = 4t+3, t \in [1]_8] = 101010010\|* = I^3O\|I\|*$.

(1.1.2) (Branch $[1]_2 \to [1]_4 \to [5]_8$.) If $t \in [5]_8$, then $(27t+25)/2/2 = (27[5]_8+25)/2/2 \in ([7]_8+[1]_8)/4 = [0]_2$. Thus, $\mathsf{CODE}[x = 4t+3, t \in [5]_8] = 10101000 = I^3O\|O$. Recall Eq. 2, current transformed number is already less than the starting number. Thus, if we explore for reduced code, then the program exits with $\mathsf{CODE}[x = 4t+3, t \in [5]_8] = IIIOO$. If we explore for code segment, the program outputs "$IIIOO$".

Similarly, $t \in [3]_4$ is partitioned into two parts - $t \in [3]_8$ and $t \in [7]_8$, because $[3]_4 = [3]_8 \cup [7]_8$.

(1.2.1) (Branch $[1]_2 \to [3]_4 \to [3]_8$.) If $t \in [3]_8$, $(3(27t+25)/2+1)/2 \in (3(27[3]_8+25)/2+1)/2 = (3([1]_8+[1]_8)/2+1)/2 = (3[1]_4+1)/2 = [0]_4/2 = [0]_2$. Thus, $\mathsf{CODE}[x = 4t+3, t \in [3]_8] = 10101010\|0\|* = I^4\|O\|*$.

(1.2.2) (Branch $[1]_2 \to [3]_4 \to [7]_8$.) If $t \in [7]_8$, $(3(27t+25)/2+1)/2 \in (3(27[7]_8+25)/2+1)/2 = (3([5]_8+[1]_8)/2+1)/2 = (3[3]_4+1)/2 = [2]_4/2 = [1]_2$. Thus, $\mathsf{CODE}[x = 4t+3, t \in [7]_8] = 10101010\|10\|* = I^4\|I\|*$.

(2) $(x-3)/4 = t \in [0]_2$.

$x = 4t+3 \in [1]_2$. Thus, $\mathsf{CODE}[x]$ should be in the form as "$10\|*$".

$I(x) = (3x+1)/2 = (3(4t+3)+1)/2 = (12t+10)/2 = 6t+5 \in [1]_2$. Thus, $\mathsf{CODE}(x)$ should be "$1010\|*$".

$I^2(x) = (3(6t+5)+1)/2 = 9t+8 \in [0]_2$, due to $t \in [0]_2$. Thus, $\mathsf{CODE}(x)$ should be "$10100\|* = I^2O\|*$".

$I^2O(x) = (9t+8)/2$. It cannot be determined whether it is even or odd. Until now, we have $\mathsf{CODE}[x = 4t+3, t \in [0]_2] = 10100\|* = IIO\|* = I^2O\| *$.

For obtaining further dynamics, partition of $t$ should be conducted for deciding whether $I^2O(x) \in [0]_2$, or $I^2O(x) \in [1]_2$.

We divide $t \in [0]_2$ into two partitions - $t \in [0]_4$ and $t \in [2]_4$, because $[0]_2 = [0]_4 \cup [2]_4$.

(2.1) (Branch $[0]_2 \to [0]_4$.) If $t \in [0]_4$, $(9t+8)/2 \in (9[0]_4+8)/2 = [0]_4/2 = [0]_2$, Thus, $\mathsf{CODE}[x = 4t+3, t \in [0]_4] = 10100\|0 = I^2O\|O$. Recall Eq. 2, current transformed number is already less than the starting number. Thus, if we explore for reduced code, then the program exits with $\mathsf{CODE}[x = 4t+3, t \in [0]_4] = IIOO$. If we explore for code segment, the program outputs "$IIOO$".

(2.2) (Branch $[0]_2 \to [2]_4$.) If $t \in [2]_4$, $(9t+8)/2 \in (9[2]_4+8)/2 = [2]_4/2 = [1]_2$, Thus, $\mathsf{CODE}[x = 4t+3, t \in [2]_4] = 10100\|10\|* = IIO\|I\|* = I^2O\|I\|*$.

Similarly, we divide $t \in [2]_4$ into two partitions - $t \in [2]_8$ and $t \in [6]_8$, because $[2]_4 = [2]_8 \cup [6]_8$.

(2.2.1) (Branch $[0]_2 \to [2]_4 \to [2]_8$.) If $t \in [2]_8$, $(9t+8)/2 \in (9[2]_8+8)/2 = [2]_8/2 = [1]_4$, Thus, $\mathsf{CODE}[x = 4t+3, t \in [2]_8] = 10100\|100 = IIO\|IO = I^2O\|IO$. Recall Eq. 2, current transformed number is already less than the starting number. Thus, if we explore for reduced code, then the program exits with $\mathsf{CODE}[x = 4t+3, t \in [2]_8] = IIOIO$. If we explore for code segment, the program outputs "$IIOIO$".

(2.2.2) (Branch $[0]_2 \to [2]_4 \to [6]_8$.) If $t \in [6]_8$, $(9t+8)/2 \in (9[6]_8+8)/2 = [6]_8/2 = [3]_4$, Thus, $\mathsf{CODE}[x = 4t+3, t \in [6]_8] = 10100\|10\|* = IIO\|I\|*$.

The extension of binary branches is depicted in Fig. 1. We call it Collatz Tree.

In summary, according to above analysis, we can give $\mathsf{CODE}[x = 4t+3, t \in [i]_8], i = 0, 1, ..., 7$ in Eq. 3. It includes some reduced codes such as $\mathsf{CODE}(t \in [0]_4) = 101000 = IIOO$, $\mathsf{CODE}(t \in [2]_8) = 10100100 = IIOIO$, and $\mathsf{CODE}(t \in [5]_8) = 10101000 = IIIOO$. It also includes code segments ($*$ presenting further dynamics). Indeed, for outputting original dynamics by code segments, $x = 4t+3, t \in [0]_8$ and $x = 4t+3, t \in [4]_8$ should be either $IIOOI$ or $IIOOO$.

$$\mathsf{CODE}[x = 4t+3] = \begin{cases} 101000 & IIOO & t \in [0,4]_8, \\ 101010010* & IIIOI* & t \in [1]_8, \\ 10100100 & IIOIO & t \in [2]_8, \\ 101010100* & IIIIO* & t \in [3]_8, \\ 10101000 & IIIOO & t \in [5]_8, \\ 1010010* & IIOII* & t \in [6]_8, \\ 1010101010* & IIIII* & t \in [7]_8. \end{cases} \quad (3)$$

Similarly, we can also give $\mathsf{CODE}[x = 4t+3, t \in [i]_4]$ $(i = 0, 1, 2, 3)$ and $\mathsf{CODE}[x = 4t+3, t \in [i]_2]$ $(i = 0, 1)$ as follows:

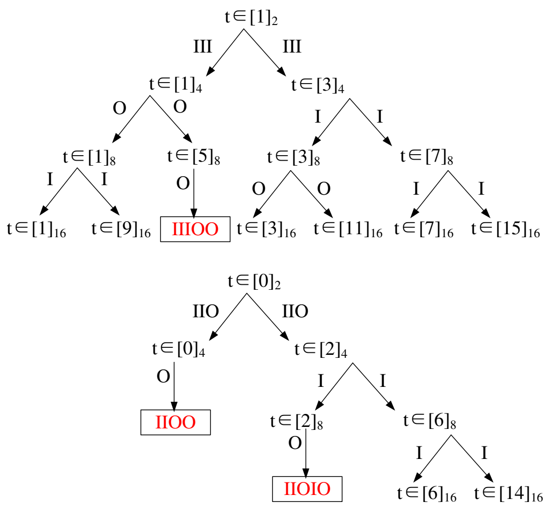

Figure 1. Collatz Tree. The branches denote the dynamics forking for $x = 4t + 3, t \in [i]_8, i = 0, 1, ..., 7$. If $(x - 3)/4 = t \in [0]_2$, $\mathsf{CODE}[x] = IIO*$. It depends on $t \in [2]_4$ or $t \in [0]_4$ whether "*" is $I$ or $O$. If $x \in [2]_4$, $\mathsf{CODE}[x] = IIOI*$. It depends on $t \in [6]_8$ or $t \in [2]_8$ whether "*" is $I$ or $O$. The rectangular leaves means that current transformed number has already been less than the starting number, which are so-called reduced codes. More specifically, $c$ is a reduced code if $\forall x \in \mathbb{N}, c(x) < x$. Certainly, we do not need to distinguish reduced codes with other intermediate segments for outputting original dynamics. It is a binary tree, and each half partition of father node's associated residue class will result in either $I$ or $O$ at next Collatz contrasformaiton.

$$\mathsf{CODE}[x = 4t + 3] = \begin{cases} 101000 & IIOO \quad t \in [0]_4, \\ 1010100* & IIIO* \quad t \in [1]_4, \\ 1010010* & IIOI* \quad t \in [2]_4, \\ 10101010* & IIII* \quad t \in [3]_4. \end{cases} \tag{4}$$

$$\mathsf{CODE}[x = 4t + 3] = \begin{cases} 10100* & IIO* \quad t \in [0]_2, \\ 101010* & III* \quad t \in [1]_2. \end{cases} \tag{5}$$

As the output of $\mathsf{CODE}[x \in [0]_2]$ and $\mathsf{CODE}[x \in [1]_4]$ is trivial, we concentrate on the dynamics of $x \in [3]_4$.

***Proposition 3.1.*** If Collatz tree is extended with sufficient width or depth, the dynamics of a starting number can be determined directly by $(x - 3)/4 = t \in [i]_m$ without conducting concrete computations such as $3 * x + 1$ or $x/2$.

**Proof** (Sketch.) If we construct Collatz tree via above analysis method, we can find that there are some leaves on Collatz tree that are reduced codes. For example, code $c$ is a reduced code when $\forall x \in \mathbb{N}, c(x) < x$. In other words, if the path to the node is $\{I, O\}^*$, we have $(\{I, O\}^*)(x) < x$, where $\{I, O\}^*(\cdot)$ is looked as a function. Each leaf node is associated a residue class $[i]_m$, where $m = 2^k$, $k$ is the layer of its father, $0 \le i < m$. If the tree grows broad or deep enough by (manual) analysis, for given $x$, $\mathsf{CODE}[x]$ can be directly determined by deciding $t \in [i]_m$, and the leaf node (i.e., terminal code) associated to $[i]_m$ can be outputted directly. For example, given $x = 23$, we can determine its dynamics directly by $(23 - 3)/4 = 5 \in [5]_8$, and output leaf node $\mathsf{CODE}[x = 4t + 3, t \in [5]_8] = IIIOO$. Given $x = 11$, we can determine its dynamics by locating leaf node (reduced code) that is $\mathsf{CODE}[x = 11, t = (11-3)/4 \in [2]_8] = IIOIO$ directly. ∎

Note that, if above $x$ is a transformed number instead of a starting number, all code segment results should be added "*" to denote that further code segments may follow. It provides a foundation for an automata (design) that takes as input $t \in [i]_m$ and outputs corresponding code segment.

Obviously, above Eq. 3, 4, and 5 can also be derived from the tree structure in Fig. 1. It can be looked as a path from the root to leaves.

According to above analysis, we can design an algorithm that takes as input $t = (x - 3)/4$ and output corresponding code segments consisting of "$I$" and "$O$".

We firstly design a Finite State Automata to output the dynamics of starting number $x$. If the automata is terminal, the program will exist and $\mathsf{CODE}[x]$ will be outputted.

Fig. 2 depicts the Finite State Automata for outputting reduced dynamics from starting number $x$ to the first transformed number that is less than $x$. Indeed, it provides a basic component for outputting original dynamics - it can be invoked as a subfunction for outputting original dynamics, as reduced dynamics is the building block of original dynamics.

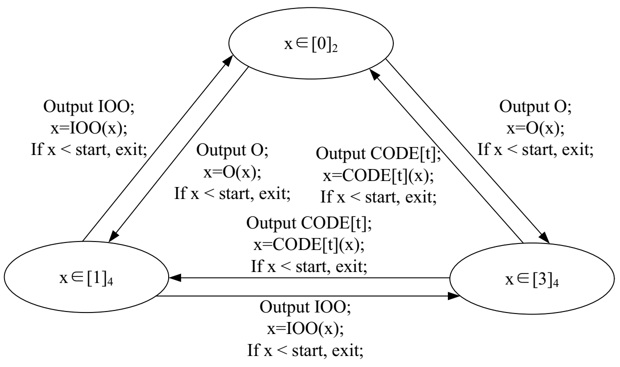

Figure 2. The Finite State Automata for outputting reduced dynamics. If $\mathsf{CODE}[x]$ exists, the automata can be terminal. The outputting individual segments depend on branch conditions in terms of $(x - 3)/4 = t \in [i]_m$.

Next, an algorithm based on above automata can be proposed as follows:

**Remark** 1) $Initialize()$ is a function in which code segment table is constructed. That is, assign code segment to $\mathsf{CODE}[[i]_m]$ according to Eq. 3, 4, and 5. In the implementation, $m$ could be 2, 4, 8, and larger, depending on the available analysis strength for code segments corresponding to $[i]_m$ (e.g., Collatz tree width and depth. Depth is related to $m$; Width is related to $i$).

```
Data: $x$
Result: $code = \mathsf{CODE}[x]$.
$code = NULL$;
$Initialize()$;
$start \Leftarrow x$;
while $x > start$ do
    if $x \in [0]_2$ then
        $c \Leftarrow CODE[x \in [0]_2]$;
    end
    if $x \in [1]_4$ then
        $c \Leftarrow CODE[x \in [1]_4]$;
    end
    if $x \in [3]_4$ then
        $t = (x-3)/4$;
        $i \Leftarrow t \mod m$;
        $c \Leftarrow CODE[x = 4t+3, t \in [i]_m]$;
    end
    $code \Leftarrow code \| c$;
    $x \Leftarrow c(x)$;
end
return $code$;
```

**Algorithm 1:** Input $x$. Output $\mathsf{CODE}[x]$. $Initialize()$ creates a table with two columns - $[i]_m$ and code segments for $[i]_m$. (E.g., Eq. 3 can be used for constructing the table.) $c(\cdot)$ is a function to compute a transformed number from current $x$. E.g., $c \Leftarrow \mathsf{CODE}[x \in [1]_4] = IOO$, thus $c(x) = \mathsf{CODE}[x \in [1]_4](x) = IOO(x) = (3x+1)/2/2$.

For example, according to Eq. 5, $\mathsf{CODE}[x = 4t+3, t \in [0]_2] = IIO$, $\mathsf{CODE}[x = 4t+3, t \in [1]_2] = III$.
For another example, according to Eq. 4, $\mathsf{CODE}[x = 4t+3, t \in [0]_4] = IIOO$, $\mathsf{CODE}[x = 4t+3, t \in [1]_4] = IIIO$, $\mathsf{CODE}[x = 4t+3, t \in [2]_4] = IIOI$, $\mathsf{CODE}[x = 4t+3, t \in [3]_4] = IIII$.

2) The loop condition in the algorithm is "While $x > start$" where $start$ is starting number and $x$ is current transformed number, which outputs reduced codes. Instead, it can be changed to "While $x > 1$", in which the outputting code represents original code (the dynamics from starting number to 1).
3) As the algorithm can output further code segment with $m+2$ computations (i.e., $\{I, O\}^{m+2}$) in one time (step) according to $(x-3)/4 = t \in [i]_m$, the outputting time of dynamics is shorten to about $1/(m+2)$ in one loop and in overall as well.

Fig. 3 depicts the flow chart of the algorithm.

Finally, a conjecture is given for the proof of Collatz conjecture as follows:

***Conjecture 3.2.*** (Collatz Tree Conjecture). The union of residue classes associated to leaf nodes in Collatz tree covers $x \in [3]_4$. That is, $\bigcup [i]_m = 1$, where $[i]_m$ are all residue classes associated to leaf nodes. (As $\bigcap [i]_m = \emptyset$, thus "cover" is also "only cover".)

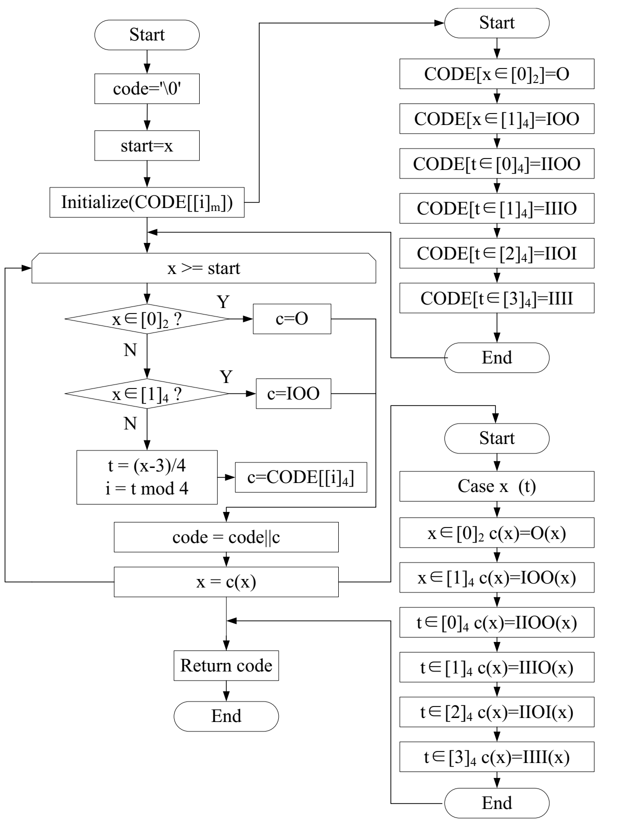

Figure 3. The flow chart for the algorithm. $[i]_m$ could be other parameters such as $m = 8$. $c(x)$ is current transformed number. $O(x) = x/2$, $IOO(x) = ((3x+1)/2)/2/2 = 3/8x + 1/8$, $IIOO(x) = ((3(3x+1)/2+1)/2)/2/2 = (3(3/2x+1/2)+1)/2/4 = (9/4x+7/4)/4 = 9/16x + 7/16$, and so on.

***Proposition 3.3.*** If Collatz Tree Conjecture is true, then Collatz conjecture is true.

**Proof** Straightforward. Collatz Tree Conjecture is true, thus $\forall x \in [3]_4$, $t = (x-3)/4 \in [i]_m$ can be associated to a leaf node or reduced code $\mathsf{CODE}[x = 4t+3, t \in [i]_m]$. Thus, $\mathsf{CODE}[x]$ exists. Thus, together $\mathsf{CODE}[x \in [0]_2] = 0$ and $CODE[x \in [1]_4] = 100$, we have that Reduced Collatz Conjecture is true. Thus, if Collatz Tree Conjecture is true, then Reduced Collatz conjecture is true. Besides, if Reduced Collatz Conjecture is true, then Collatz Conjecture is true. Thus, Collatz Tree Conjecture is true, then Collatz Conjecture is true. ∎

## 4. Conclusion

In this paper, we analyze the properties of dynamics in terms of Collatz transformations in verifying Collatz conjecture. We propose an automata-based algorithm to compute the dynamics in a batch way (i.e., outputting code segment with form $\{I, O\}^*$ in one step). We also discover that (reduced) dynamics for some starting numbers can be determined directly without conducting concrete computations such as $3*x+1$ or $x/2$ via our constructed binary tree called Collatz tree. The proposed automata can output a code segment consisting of $m+2$ Collatz transformations in terms of $I$ and $O$ by taking as input $(x-3)/4 \in [i]_{2^m}, m \in \mathbb{N}$. We finally propose a conjecture whose proof implies the proof of Collatz conjecture by using proposed Collatz tree.

## Acknowledgment

The research was financially supported by Major Scientific and Technological Special Project of Guizhou Province (20183001), the Open Funding of Guizhou Provincial Key Laboratory of Public Big Data (2018BDKFJJ009,2017BDKFJJ006), and Open Funding of Hubei Provincial Key Laboratory of Intelligent Geo-Information Processing (KLIGIP2016A05).

## References

[1] Tomas Oliveira e Silva, *Maximum excursion and stopping time record-holders for the 3x+1 problem: computational results*, *Mathematics of Computation*, vol. 68, no. 225, pp. 371-384, 1999.

[2] Livio Colussi. *The convergence classes of Collatz function* Theoretical Computer Science, vol. 412, no. 39, pp. 5409-5419, 2011

[3] Hew, Patrick Chisan. *Working in binary protects the repetends of 1/3h: Comment on Colussi's 'The convergence classes of Collatz function'*. Theoretical Computer Science, vol. 618, pp. 135-141, 2016

[4] Wei Ren, Simin Li, Ruiyang Xiao and Wei Bi, *Collatz Conjecture for* $2^1 00000 - 1$ *is True - Algorithms for Verifying Extremely Large Numbers*, In Proc. of IEEE UIC 2018, Oct. 2018, Guangzhou, China, 411-416, 2018

[5] Wei Ren, *A New Approach on Proving Collatz Conjecture*. Journal of Mathematics, Hindawi, April 2019, https://www.hindawi.com/journals/jmath/2019/6129836/.

[6] Wei Ren, *Exploring properties in reduced Collatz dynamics*, IEEE Dataport, 2018. http://dx.doi.org/10.21227/ge9h-8a77 (2018).

[7] Wei Ren, *Verifying whether extremely large integer guarantees Collatz conjecture (can return to 1 finally)*, IEEE Dataport, http://dx.doi.org/10.21227/fs3z-vc10 (2018).

[8] Wei Ren, *Exploring the ratio between the count of x/2 and the count of (3*x+1)/2 in original dynamics for extremely large starting integers asymptotically*, IEEE Dataport. http://dx.doi.org/10.21227/rxx6-8322 (2018).

[9] Wei Ren, *Exploring the inverse mapping from a dynamics to a residue class - inputting a reduced dynamics or partial dynamics and outputting a residue class*, IEEE Dataport. http://dx.doi.org/10.21227/qmzw-gn71 (2018).

[10] Wei Ren, *Reduced Collatz Dynamics for Integers from 3 to 999999*, IEEE Dataport. http://dx.doi.org/10.21227/hq8c-x340 (2018).

[11] Wei Ren, *Collatz Automata and Compute Residue Class from Reduced Dynamics by Formula*, IEEE Dataport. http://dx.doi.org/10.21227/7z84-ms87 (2018).

## Appendix

The source code can be complied and executed as follows:

```
txpo23.exe 27 > txpo23-27

txpo23-27:

t in [2]_4, x=47
t in [3]_4, x=242
x in [0]_2, x=121
x in [1]_4, x=91
t in [2]_4, x=155
t in [2]_4, x=263
t in [1]_4, x=445
x in [1]_4, x=334
x in [0]_2, x=167
t in [1]_4, x=283
t in [2]_4, x=479
t in [3]_4, x=2429
x in [1]_4, x=1822
x in [0]_2, x=911
t in [3]_4, x=4616
x in [0]_2, x=2308
x in [0]_2, x=1154
x in [0]_2, x=577
x in [1]_4, x=433
x in [1]_4, x=325
x in [1]_4, x=244
x in [0]_2, x=122
x in [0]_2, x=61
x in [1]_4, x=46
x in [0]_2, x=23
CODE(27)=--0-----0-0--0---0----0-00---0-
-0------00----000-0-0-000-00
u=37, d=59, ratio=(d-u)/u=0.3728814
```

Source code that can output $CODE[x]$ via finite state automata algorithm is given as follows (In output "$-$" denotes "$I$", "0" denotes "$O$". Source code can be downloaded from IEEE Dataport [11]):

```
//////////////////////////////////////////////////////////////////////////////
//Copyright 2016, Dr. Wei Ren, China University of Geosciences, Wuhan, China
//---------------------------------------------------------------------------
//Function: Output CODE(x) by Finite State Automata.
//Input: x
//Output: CODE(x)
//Usage Example:  tpxo23.exe 27 > txpo23-27
//////////////////////////////////////////////////////////////////////////////

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <time.h>

int main(int argc, char *argv[]) {
int starting_number, x, t, i;
int u=0,d=0;                        // for counting (U,D) pair
char CODE[500]="\0";
float ratio;

starting_number = atoi(argv[1]);
//starting_number = 27;

x = starting_number;

while( x >= starting_number)
{
if(x%2==0)
{
strcat(CODE,"0");
x = x/2;
printf("x in [0]_2, x=%d\n",x); //debug

d = d+1;
continue;
}

if(x%4==1)
{
strcat(CODE,"-0");
x = (3*x+1)/2;
x = x/2;
printf("x in [1]_4, x=%d\n",x);    //debug
```

```
u = u+1;
d = d+2;
continue;
}

if(x%4==3)
{
t = (x-3)/4;
i = t%4;

switch(i)
{
case 0:
strcat(CODE,"--00");
x = (3*x+1)/2;
x = (3*x+1)/2;
x = x/2;
x = x/2;
printf("t in [0]_4, x=%d\n",x); //debug

u = u+2;
d = d+4;
break;

case 1:
strcat(CODE,"---0");
x = (3*x+1)/2;
x = (3*x+1)/2;
x = (3*x+1)/2;
x = x/2;
printf("t in [1]_4, x=%d\n",x); //debug

u = u+3;
d = d+4;
break;

case 2:
strcat(CODE,"--0-");
x = (3*x+1)/2;
x = (3*x+1)/2;
x = x/2;
x = (3*x+1)/2;
printf("t in [2]_4, x=%d\n",x); //debug

u = u+3;
d = d+4;
break;

case 3:
strcat(CODE,"----");
x = (3*x+1)/2;
x = (3*x+1)/2;
x = (3*x+1)/2;
x = (3*x+1)/2;
printf("t in [3]_4, x=%d\n",x); //debug

u = u+4;
d = d+4;
break;
}

continue;
}

} //end of while

ratio = (float)(d-u)/d;
printf("CODE(%d)=%s\nu=%d, d=%d, ratio=(d-u)/u=%9.7f", starting_number, CODE, u, d, ratio);   //debug

return 1;
}
```