Parabola Volume 56, Issue 3 (2020)

Computational and Mathematical Analysis of a Convergent Alphanumeric Reduction Algorithm

Arqam Patel[1]

1 Introduction

Consider the following algorithm.

```
Algorithm 1: Convergent Alphanumeric Reduction Algorithm (CARA)
  1. Take any number $x$.
  2. Spell it and count the number of letters (excluding spaces and 'and's).
  3. Repeat Step 2 with the number obtained thus.
```

For example, applying CARA to the number 15, we get:

$$15 \text{ (fifteen)} \to 7 \text{ (seven)} \to 5 \text{ (five)} \to 4 \text{ (four)} \to 4 \text{ (four)} \to \cdots$$

Let $f_n(x)$ be the nth iteration of this algorithm on x. For instance, we can see from the above example that $f_1(15) = 7$, $f_2(15) = 5$, and $f_n(15) = 4$ for all $n \geq 3$.

The aim of this paper is to present and prove the following result.

Theorem 1. $\lim\limits_{n\to\infty} f_n(x) = 4$.

This theorem is roughly analogous to how the $3n+1$ function reduces every number to 1 according to the Collatz Conjecture [1].

The paper present a computational verification as well as a mathematical proof for the convergence of all whole numbers to 4 upon successive application of CARA.

Since counting the letters in the spelling of a number is quasi-mathematical, the behaviour of the function is erratic and seemingly arbitrary, which makes it challenging to prove our theorem. However, in the following paper, this is alleviated by taking into account an observed pattern in the behaviour of certain numbers ('n-illion'), and using it to make an assumption on the spelling length of all numbers belonging to the group. Further, all possible cases are considered and separately treated when necessary. This allows us to complete the proof for all natural numbers.

While the result itself is presumably of little practical significance, the paper demonstrates how the behaviour of unpredictable and seemingly arbitrary functions may be analysed using generalisations and assumptions from initial conditions.

[1] Arqam Patel is a senior student at Rahul International School, India

2 Definitions

To help make the proofs reader-friendly, some functions and sets have been defined in the subsections below. Further, we will define a useful system used for spelling numbers.

2.1 Functions

The following functions are used in the proof and other parts of the paper to denote the operations defined here. For each natural number x, define the function $f_n(x)$ by

$f_0(x) = x\,;$
$f_1(x)$ is the number of digits in the spelling of $x\,;$
$f_n(x) = f_1(f_{n-1}(x))\,.$

Also, define $g(x) = f_0(x) - f_1(x)$.

2.2 Special sets

Define the sets

$$D = \{20, 30, 40, 50, 60, 70, 80, 90\}\,;$$
$$T = \{10, 11, 12, 13, 14, 15, 16, 17, 18, 19\}\,;$$
$$L = \{1, 2, 3, 4\}\,.$$

The set D consists of tens digit place holders which are quite ubiquitous in numbers greater than ten. The set T consists of numbers which cannot be named using conventional rules since they have distinct names incorporating both units and tens places simultaneously. The set L consists of numbers x whose numerical value are smaller than or equal to the length of their spelling; that is, $f_0(x) \leq f_1(x)$. The f_0, f_1 and g values of the numbers in D, T, and L are given below.

f_0	Spelling	f_1	g
20	Twenty	6	14
30	Thirty	6	24
40	Forty	5	35
50	Fifty	5	45
60	Sixty	5	55
70	Seventy	7	63
80	Eighty	6	74
90	Ninety	6	84

f_0	Spelling	f_1	g
10	Ten	3	7
11	Eleven	6	5
12	Twelve	6	6
13	Thirteen	8	5
14	Fourteen	8	6
15	Fifteen	7	8
16	Sixteen	7	9
17	Seventeen	9	8
18	Eighteen	8	10
19	Nineteen	8	11

f_0	Spelling	f_1	g
1	One	3	-2
2	Two	3	-1
3	Three	5	-2
4	Four	4	0
5	Five	4	1
6	Six	3	3
7	Seven	5	2
8	Eight	5	3
9	Nine	4	5

2.3 A system for spelling numbers

There are infinitely many numbers but the need is rarely felt for regularly using words for numbers beyond 10^{12} (one trillion). However, for the purpose of the proof, an extrapolation of the current English short scale system [3] has been used to denote every number, however large, in words.

It may be noted that compound place value indicators like million trillion etc. are not used in the algorithm. Capital letters are used to denote digits and numerals before/after them indicate a single number, not their product. For instance, 1M can be 10, 11, 17 etc.

In the present system, all numbers are treated as a sum of a series of three digit numbers ('triads') which are each multiplied by $10^{3(n-1)}$ (for the n^{th} triad). For example,

$$\begin{aligned} 333,333,123 &= 333 \times 10^6 + 333 \times 10^3 + 123 \times 10^0 \\ &= 333 \times 10^{3(3-1)} + 333 \times 10^{3(2-1)} + 123 \times 10^{3(1-1)} \,. \end{aligned}$$

The term $10^{3(n-1)}$ may be called n-illion (though the word for n-illion is often derived from the Latin name for n-2).

The following table contains the names and values of n-illion numbers for each value of n:

Name	n	Value
-	1	10^0
Thousand	2	10^3
Million	3	10^6
Billion	4	10^9
Trillion	5	10^{12}
Quadrillion	6	10^{15}
Quintillion	7	10^{18}
Hextillion	8	10^{21}
Septillion	9	10^{24}
Octillion	10	10^{27}
Nonillion	11	10^{30}
Decillion	12	10^{33}
Undecillion	13	10^{36}
Duodecillion	14	10^{39}
Tredecillion	15	10^{42}
Quattuordecillion	16	10^{45}
Quindecillion	17	10^{48}

Name	n	Value
Hexdecillion	18	10^{51}
Septendecillion	19	10^{54}
Octodecillion	20	10^{57}
Novemdecillion	21	10^{60}
Vigintillion	22	10^{63}
Unvigintillion	23	10^{66}
Duovigintillion	24	10^{69}
Trevigintillion	25	10^{72}
Quattourvigintillion	26	10^{75}
Quinvigintillion	27	10^{78}
Hexvigintillion	28	10^{81}
Septenvigintillion	29	10^{84}
Octovigintillion	30	10^{87}
Novemvigintillion	31	10^{90}
Trigintillion	32	10^{93}
Untrigintillion	33	10^{96}
Duotrigintillion	34	10^{99}

Similarly, the spellings of each group $T_r \times 10^{3(n-1)}$ are also broken down, as the spelling of the three-digit number suffixed with the corresponding n-illion word, as expressed below:

$$f_1(T_r \times 10^{3(r-1)}) = f_1(T_r) + f_1(10^{3(r-1)}) \,.$$

The triads, or three digit groups are further broken down, as follows. Let KLM be a three digit number where K, L and M represent, respectively, the hundreds', tens' and ones' digits. X' signifies the number X spelled out.

The spelling of a number KLM can be written as

"K' hundred L'-ty M'"

where L'-ty is the word for the corresponding number from set D.

If $L = 1$, then the spelling of $K1M$ will be of the form "K' hundred M'-teen" where M'-teen is the word for $1M$, the corresponding number from the set T, so

$$f_1(KLM) = f_1(K00) + f_1(LM) = f_1(K) + f_1(100) + f_1(LM) = f_1(K) + 7 + f_1(LM)\,.$$

If $L \neq 1$ and $L0 \in D$, then

$$f_1(LM) = f_1(L0) + f_1(M) = f_1(1M)$$

where $L = 1$ and $1M \in T$, and

$$\begin{aligned} g(KLM) &= f_0(KLM) - f_1(KLM) \\ &= f_0(K00) + f_0(LM) - f_1(K00) - f_1(LM) \\ &= g(K00) + g(LM)\,. \end{aligned}$$

In conclusion, any natural number x can be expressed as

$$x = \sum (T_r \times 10^{3(r-1)})\,,$$

and its spelling x' can be expressed as

x' = "T'_n n-illion $\cdots$ T'_4 billion T'_3 million T'_2 thousand T'_1"

where each T_r is a triad.

For example, the spelling of 23,456,789 is the concatenation of the spellings of 23, 456, and 789, each suffixed by the respective n-illion word (empty in case of $n = 1$):

"Twenty three *million* four hundred fifty six *thousand* seven hundred eighty nine".

Finally, the functions $f_1(x)$ and $g(x)$ can be expressed as follows:

$$f_1(x) = \sum f_1(T_r \times 10^{3(r-1)}) \qquad \text{and} \qquad g(x) = \sum g(T_r \times 10^{3(r-1)})\,.$$

3 Computational verification

3.1 Method

A program [5] was built in Python 3 to verify the applicability of CARA for numbers in a given range.

Each number within the range is used as the initial input for an iteration of the algorithm function $f_n(x)$ where n is the number of iterations before the value repeats itself. Subsequently, the process is repeated for the new number thus obtained, as a nested loop.

When the loop reaches a value which yields itself upon further iteration, that is, a number which has a spelling length equal to its numerical value (which would otherwise lead to an infinite loop), the program stops and checks whether this value is four. If so, then the program records an increment of one in the number of values satisfying convergence.

3.2 Source code

```
def number_to_word(number):
    number_string = str(number)

# Remove extra zeroes from beginning (e.g. 000001 -> 1)
    number_string.lstrip('0')

# Define required sets of strings for naming numbers
    suffix = ['' , 'thousand', 'million', 'billion', 'trillion',
     ↪  'quadrillion', 'quintillion', 'sextillion', 'septillion',
     ↪  'octillion', 'nonillion', 'decillion', 'undecillion',
     ↪  'duodecillion', 'tredecillion', 'quattuordecillion',
     ↪  'quindecillion', 'hexdecillion', 'septendecillion',
     ↪  'octodecillion', 'novemdecillion', 'vigintillion',
     ↪  'unvigintillion', 'duovigintillion', 'trevigintillion',
     ↪  'quattourvigintillion', 'quinvigintillion', 'hexvigintillion',
     ↪  'septenvigintillion', 'octovigintillion', 'novemvigintillion',
     ↪  'trigintillion', 'untrigintillion', 'duotrigintillion']

    singledigits = ['','one', 'two', 'three', 'four', 'five', 'six',
     ↪  'seven', 'eight', 'nine' ]

    teen = ['ten', 'eleven', 'twelve', 'thirteen', 'fourteen', 'fifteen',
     ↪  'sixteen', 'seventeen', 'eighteen', 'nineteen']

    ty = ['','','twenty', 'thirty', 'forty', 'fifty', 'sixty', 'seventy',
     ↪  'eighty', 'ninety']
```

```
# Converting the number into triad composition form (e.g. 1345 -> 001345)
    m = len(number_string) % 3
    if   m == 1:
        number_string = '00' + number_string
    elif m == 2:
        number_string = '0'  + number_string

# Counting number of triads (123456 : 2)
    n = len(number_string)/3
    no_of_triads = int(n)

# Extracting all triads into a list (123456 -> ['123' , '456'])
    triad_list = []

    for i in range(no_of_triads):
        k = number_string[3*i:3*i+3]
        triad_list.append(k)

# Naming each triad [123 -> one hundred twenty three]

# Matching a single digit to its name (e.g. 5 -> five)
    def digitname(n):
        return singledigits[n]

# Naming the hundreds place of triad
    def hundreds(first_digit):
        if first_digit == 0:
            return ""
        else:
            return digitname(first_digit) + ' ' + 'hundred '

# Naming the tens and units places of triad
    def tens_and_units(second_digit,third_digit):
        if   second_digit == 0:
            return digitname(third_digit)
        elif second_digit == 1:
            return teen[third_digit]
        else:
            tens_place = ty[second_digit]
            units_place = digitname(third_digit)
            return tens_place + ' ' +  units_place

# Naming a triad
    def triad_name(triad):
        hundred = triad[0]
        ten     = triad[1]
        unit    = triad[2]
        hundreds_place = hundreds(int(hundred))
        tens_units_place = tens_and_units(int(ten),int(unit))
        return hundreds_place + tens_units_place
```

```
# Suffixing a triad place value (In 123000, 123-> one hundred twenty three
↪  thousand)
    number_spelling = ''
    for i in range(no_of_triads):
        k = triad_list[i]
        if triad_list[i] != '000':
            number_spelling += (triad_name(k) + ' ' + suffix[no_of_triads -
            ↪  i -1] + ' ')
    return number_spelling

# Counting length of spelling
def namelength(s):
    k = number_to_word(s)
    spelling_length = len(k) - k.count(' ')
    return spelling_length

# For each number, finds spelling length (after converting to spelling
↪  first) and then proceeds with it as next number in loop
# Outputs the final number obtained after algorithm is repeated
↪  sufficiently (i.e. before entering an infinite loop)
def cara_loop(n):
    while namelength(n) != n:
        n = namelength(n)
    return n

# Taking input for the upper bound of the range of natural numbers to test
test_range = int(input('Range? ')) + 1
true_outputs = 0

# Loop checking convergence, i.e. whether output of the loop is 4, for each
↪  integer and counting the number of integers satisfying it
for i in range(1,test_range):
    if cara_loop(i) == 4:
        true_outputs +=1

# Displaying output (number of natural numbers in the range satisfying the
↪  condition)
print('CARA is convergent for {} numbers in the given
↪  range.'.format(true_outputs))
```

3.3 Results

The following represent trials of the program:

Input: `Range upper bound? 100000000`
Output: `CARA is convergent for 100000000 numbers in the given range.`

A variation of the program takes the base 10 logarithm of the upper bound as input, to simplify input for verifying for a large range:

Input: `Exponent of 8 in upper bound? 8`
Output: `CARA is convergent for 100000000 numbers in the given range.`

CARA was tested and observed to hold for the first hundred million natural numbers.

4 Proof of Theorem 1

A mathematical proof of Theorem 1 will now be given. It proceeds step-wise, proving that certain numbers get reduced to 4 and then that all other numbers in turn must reduce to one of those numbers. Thus, all numbers must eventually reduce to 4.

The proof is divided into four sub-proofs:

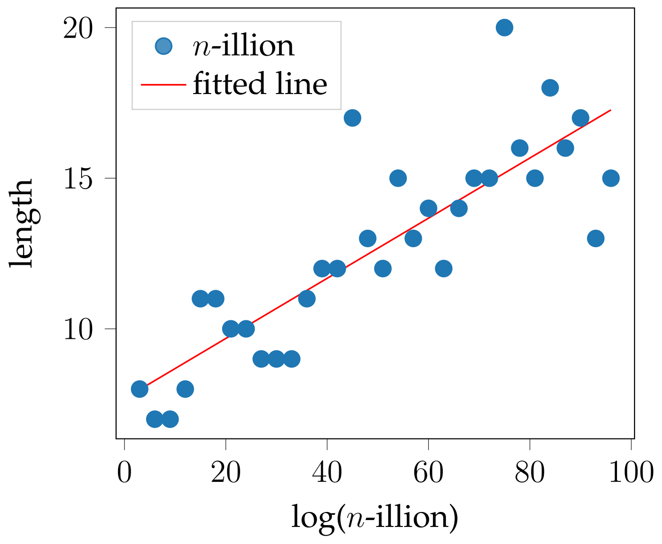
- Proof that all single-digit numbers reduce to 4
- Proof that all double-digit numbers reduce to 4
- Proof that all three-digit numbers reduce to 4
- Proof that all larger numbers reduce to 4

4.1 Assumption

The sole assumption taken for the purpose of the proof is that $g(10^{3n}) > 2$; that is, that the number of letters in the spelling of n-illion is at least 2 less than the numerical value of n-illion itself. This may be interpreted as intuitively true, as words having such a large number of letters are not a part of conventional lexicon (the longest English non-technical word [4] is just 29 letters long, though there are chemical names up to 189819 letters long).

However, a more rigorous justification may also be provided using extrapolation from the set of well-defined n-illion names (Table 1), taken till 10^{99}.

A plot of $\log_{10}(n - \text{illion})$ versus length of the corresponding spelling was obtained. Subsequently, using linear regression, the best fitting line was identified. The slope was almost exactly 0.1 and intercept slightly more than 7.68.



The initial samples may thus be modeled as

$$f_1(x) \approx 0.1 \log_{10}(x) + 7.68\,.$$

The coefficient of determination (R^2) was found to be nearly 0.71, indicating that this would be a reasonably good approximation to model the growth of the spelling length in relation with the size of n-illion numbers. If extrapolated, this yields the conclusion that, to a large extent, f_1 varies logarithmically with f_0.

Assuming that early trends are followed (or that the rate of increase is slower than the early trends, as indicated by the last few cases in the present sample space), this yields the conclusion that the rate of growth of spelling lengths of decimal place holding names is likely to be much smaller than that of their numerical values .

This implies that for every thousandfold increase in numerical size, the spelling length would increase much less than 1000 times for n-illion numbers, and since the initial values of g are well above 5, the assumption would likely never fail.

4.2 All single-digit numbers reduce to 4

The following sequences, obtained by applying CARA to all single digit numbers, all end in 4.

1 (one) $\rightarrow$ 3 (three) $\rightarrow$ 5 (five) $\rightarrow$ 4 (four)
2 (two) $\rightarrow$ 3 (three) $\rightarrow$ 5 (five) $\rightarrow$ 4 (four)
3 (three) $\rightarrow$ 5 (five) $\rightarrow$ 4 (four)
4 (four) $\rightarrow$ 4 (four)
5 (five) $\rightarrow$ 4 (four)
6 (six) $\rightarrow$ 3 (three) $\rightarrow$ 5 (five) $\rightarrow$ 4 (four)
7 (seven) $\rightarrow$ 5 (five) $\rightarrow$ 4 (four)
8 (eight) $\rightarrow$ 5 (five) $\rightarrow$ 4 (four)
9 (nine) $\rightarrow$ 4 (four)

4.3 All two-digit numbers reduce to 4

From 1 to 9, 1 ($g = -2$), 2 ($g = -1$) and 3 ($g = -2$) are the only numbers which have $f_0 < f_1$ and will thus produce a number larger than themselves on application of the function. However, there are no numbers with 1 or 2 letters in their spellings so these appear in the algorithm only if these numbers are chosen. The number 3 occurs only in case of 6 and 1, both of which were shown above to lead to 4 by application of CARA.

From the set T, there is no number with $g < 0$; the minimum value of g among numbers in T is $g(11) = g(13) = 5$. Similarly, from D, there is no number with $g < 0$: the minimum value of g among numbers in D is $g(20) = 20 - f_1(20) = 20 - 6 = 14$.

Now consider a number of the form LM where $L > 1$. Then, from the comments above,

$$g(LM) = g(L0) + g(M) \geq 6 - 2 = 4 > 0$$

Indeed, for such numbers, the minimum value of g is 12, achieved by 21 and 23.

Thus, for any two-digit number x, it is proved that $g(x) > 0$, so $f_1 < f_0$. Hence, the number of letters spelling x is strictly less than the value of x. CARA therefore reduces all two-digit numbers to single-digit numbers and then to 4.

4.4 All three-digit numbers reduce to 4

"Seventy" is the longest string arising from the tens D. Also, 3, 7 and 8 have 5 letters, the most letters among single-digit numbers. The three-digit numbers with the longest spellings are therefore

$$777 \quad 373 \quad 878 \quad 378 \quad 873 \quad 773 \quad 778 \quad 377 \quad 877\,,$$

each of which is spelled with $f_1 = 24$ letters. Hence, for any three-digit number x, $f_0 \geq 100$ and $f_1 \leq 24$, so $g(x) \geq 76 > 0$.

It follows that CARA reduces each three-digit number to a smaller number, and eventually to a double- or single-digit number, and from there to 4.

4.5 All numbers reduce to 4

Any number $x = \sum(T_r \times 10^{3(r-1)})$ can be written in words as

$$x' = \text{"}T_n\ n\text{-illion} \cdots T_4 \text{ billion } T_3 \text{ million } T_2 \text{ thousand } T_1\text{"}\,.$$

Define $R = \{r \in \{1, \dots, n\} \ : \ T_r > 0\}$. Then

$$\begin{aligned} g(x) = x - f_1(x) &= \sum_{r\in R} T_r \times 10^{3(r-1)} - f_1\Big(\sum_{r\in R} T_r \times 10^{3(r-1)}\Big) \\ &= \sum_{r\in R} T_r \times 10^{3(r-1)} - \sum_{r\in R} f_1\big(T_r \times 10^{3(r-1)}\big) \\ &= \sum_{r\in R} (T_r \times 10^{3(r-1)} - f_1\big(T_r \times 10^{3(r-1)}\big) \\ &= \sum_{r\in R} g\big(T_r \times 10^{3(r-1)}\big)\,. \end{aligned}$$

Suppose that $x \geq 1000$; then R contains at least one number r greater or equal to 2. By assumption, $g(10^{3(r-1)}) > 2$ for each $r \geq 2$. Since $g(x) \geq -2$ for each number $x = T_1 < 1000$,

$$g(x) = \sum_{r\in R} g\big(T_r \times 10^{3(r-1)}\big) > \sum_{r\in R: r\geq 2} 2 - 2 \geq 0\,.$$

Thus, the length of spelling of any number $x \geq 1000$ will be smaller than the value of x. Therefore, CARA will reduce such numbers to numbers less than 1000 - which, as already proved, eventually reduce to 4.

This proves that CARA does indeed reduce all positive integers to 4. □

5 Conclusion

We have proved that each sequences generated by CARA eventually ends in 4. The proof also informs us of more properties of these sequences. In particular, each number bigger than 4 is strictly reduced by CARA.

This rules out the existence of any infinite self-repeating sequences, except for the terminating sequence

$$4 \to 4 \to 4 \to \cdots .$$

This also rules out the existence of more complex repeating patterns such as

$$x \to y \to z \to x \to y \to z \to \cdots .$$

Thus, all natural numbers larger than four are successively converted to smaller numbers by CARA until eventually reaching a number from the set $\{1, 2, 3, 4\}$, more specifically either 3 or 4 since there are no numbers with one or two letter spellings, and then yield 4 on further iteration.

This proof ultimately depends on the validity of the assumption that $g(10^{3(r-1)}) > 2$ for each $r \geq 2$. Taking into account the pattern suggested by early trends, the assumption and thus the proof can be said to hold. For future research, one could investigate this assumption more formally.

Acknowledgements

I would like to thank Dr Narasimhan Chari, Dr Amitava Bhattacharya and Mr Amit M Agarwal for their insightful suggestions which helped improve this paper.

References

[1] J.P. van Bendegem, The Collatz Conjecture: A case study in mathematical problem solving, *Logic and Logical Philosophy* **14** (2005), 7–23.

[2] L. Collatz, On the motivation and origin of the $(3n + 1)$-Problem, [*J. Qufu. Norm. Univ. Nat. Sci. Ed.* **12** (1986), 9–11], in *The Ultimate Challenge: The $3x+1$ Problem*, pages 241–247, Amer. Math. Soc., Providence, RI, 2010.

[3] Wikipedia, *Long and Short Scale Systems*, `https://en.wikipedia.org/wiki/Long_and_short_scales`, last accessed on 2021-01-14.

[4] Wikipedia, *Longest Word in English*, `https://en.wikipedia.org/wiki/Longest_word_in_English`, last accessed on 2021-01-14.

[5] A. Patel, *CARA Verification Program*, `https://github.com/arq62/CARA`, last accessed on 2021-01-14.

[6] A. Bellos, *So You Think you've Got Problems?: Surprising and Rewarding Puzzles to Sharpen Your Mind*, Guardian Faber Publishing, 2019.