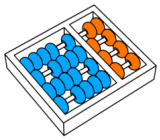

A word problem equivalent to the Collatz sequence problem

Guido C. S. Araújo *Jorge Stolfi*

Technical Report - IC-11-017 - Relatório Técnico
October - 2011 - Outubro

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

A word problem equivalent to theCollatz sequence problem

Guido C. S. Araújo* Jorge Stolfi†‡

Abstract

The Collatz (or Ulam) sequence of a positive integer n starts with $c_0 = n$, and its general term c_{i+1} for $i \geq 0$ is $c_i/2$ is c_i is even, or $3c_i + 1$ if c_i is odd. It is conjectured that the sequence of every positive integer includes a term $c_i = 1$. In this report we describe a word problem that is equivalent to the Collatz sequence problem. Namely, we define an essentially deterministic contex-sensitive grammar whose derivations from a given initial word correspond to the terms of a Collatz sequence or to intermediate states in the computation of those terms. The Collatz sequence conjeture then can be restated as saying that a derivation from any valid initial word (that encodes a positive integer) eventually leads to a specific word (that encodes the integer 1). We also explore some variants of this model.

1 Introduction

The *Collatz sequence* of a positive number n starts with $c_0 = n$, and its general term c_{i+1} for $i \geq 0$ is $c_i/2$ is c_i is even, or $3c_i + 1$ if c_i is odd. For example, the Collatz sequence of 6 is $(6, 3, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1, \ldots)$ This sequence has been called also the *Ulam*, *Kakutani*, *Thwaites*, *Hasse*, *Syracuse*, or *hailstone* sequence.

The formula for the general term can be written also as $c_{i+1} = \psi(c_i)$ for all $i \in \mathbb{N}$, where

$$\psi(m) = \begin{cases} r & \text{if } m = 2r \text{ for some } r \in \mathbb{N} \\ 6r + 4 & \text{if } m = 2r + 1 \text{ for some } r \in \mathbb{N} \end{cases} \quad (1)$$

The Collatz sequence of 1 is periodic with period length 3, $(1, 4, 2, 1, 4, 2, \ldots)$. The *Collatz conjecture* states that the sequence of *every* positive integer n eventually reaches the value 1, and therefore eventually becomes this periodic sequence.

*IC-UNICAMP guido@ic.unicamp.br
†IC-UNICAMP stolfi@ic.unicamp.br
‡This work benefitted from grants by CNQp and FAPESP

1.1 Optimized sequence

Since $6m + 4$ is always even, we can "optimize" the Collatz recurrence by combining the odd-case step with the following inevitable even-case step. Namely, we define an *optimized* Collatz sequence of a positive integer n as being $d_0 = n$ and $d_{i+1} = \phi(d_i)$ for all $i \in \mathbb{N}$, where

$$\phi(m) = \begin{cases} r & \text{if } m = 2r \text{ for some } r \in \mathbb{N} \\ 3r + 2 & \text{if } m = 2r + 1 \text{ for some } r \in \mathbb{N} \end{cases} \tag{2}$$

The optimized Collatz sequence of an integer n is identical to its standard sequence except for the omission of the even term that follows each odd term. For example, the optimized sequence of 6 is $(6, 3, 5, 8, 4, 2, 1, 2, 1, 2, 1, \ldots)$

It follows that the standard and optimized sequences are equivalent for the purposes of the conjecture; namely, the optimized sequence includes the integer 1 if and only if the standard sequence does. Therefore, we will work with the optimized recurrence (2) from now on.

2 Word formulation

2.1 Basic grammar

We now define a context-sensitive grammar G to model the Collatz recurrence function (2). The alphabet is the set $\{\texttt{0}, \texttt{1}, \texttt{<}, \texttt{>}, \texttt{A}, \texttt{B}, \texttt{C}\}$. We will call $\Delta = \{\texttt{0}, \texttt{1}\}$ the set of *bits*, $\Lambda = \{\texttt{<}, \texttt{>}\}$ the set of *delimiters*, and $\Sigma = \{\texttt{A}, \texttt{B}, \texttt{C}\}$ the set of *trits*. A *word* is any string from $(\Delta \cup \Sigma \cup \Lambda)^*$; a *valid* word is any word of the form $\texttt{<}\beta\texttt{>}$ where β is a non-empty word in $(\Delta \cup \Sigma)^*$.

2.2 Word values

Every word β of G is meant to encode a positive integer $\omega(\beta)$, according to the following rules:

$$\omega(\beta) = \begin{cases} 0 & \text{if } \beta \text{ is the empty string,} \\ \omega(\gamma) & \text{if } \beta = \texttt{<}\gamma \text{ or } \beta = \gamma\texttt{>}, \\ 2\omega(\gamma) & \text{if } \beta = \gamma\texttt{0}, \\ 2\omega(\gamma) + 1 & \text{if } \beta = \gamma\texttt{1}, \\ 3\omega(\gamma) & \text{if } \beta = \gamma\texttt{A}, \\ 3\omega(\gamma) + 1 & \text{if } \beta = \gamma\texttt{B}, \\ 3\omega(\gamma) + 2 & \text{if } \beta = \gamma\texttt{C}. \end{cases} \tag{3}$$

Thus, for example, $\omega(\texttt{110}) = 6$ (and, in general, $\omega([n]) = n$ for every n); $\omega(\texttt{110B}) = 3{\cdot}6{+}1 = 19$; and $\omega(\texttt{<B1CB>}) = 34$. The integer $\omega(\beta)$ is called the *primal value* or just the *value* of β. In general, there are many strings in $(\Sigma \cup \Delta)^*$ with the same primal value.

Observe that every string $\beta \in (\Sigma \cup \Delta)^*$ is the representation of the integer $\omega(\beta)$ in a mixed base-2 and base-3 system, where the trits `A`, `B`, and `C` are interpreted as the ternary digits 0, 1, and 2, respectively. Indeed, let $d(\beta)$ and $t(\beta)$ denote the number of bits and trits in β, respectively. Let also $\beta \dashv k$ and $k \vdash \beta$ denote the string β truncated to the last k

symbols and to the first k symbols, respectively. Then if β is a string $\tau_m\tau_{m-1}\cdots\tau_1\tau_0$, with each $\tau_i \in \Sigma \cup \Delta$, we have

$$\omega(\beta) = \sum_{i=0}^{m} \omega(\tau_i) 2^{b(\beta \dashv i)} 3^{t(\beta \dashv i)} \tag{4}$$

More generally, for any $\alpha, \beta \in (\Sigma \cup \Delta)^*$,

$$\omega(\alpha\beta) = \omega(\alpha) 2^{b(\beta)} 3^{t(\beta)} + \omega(\beta) \tag{5}$$

It follows that, for any $\beta', \beta'', \gamma \in (\Sigma \cup \Delta)^*$,

$$\omega(\beta') = \omega(\beta'') \;\Leftrightarrow\; \omega(\beta'\gamma) = \omega(\beta''\gamma) \tag{6}$$

2.3 Dual values

The *dual value* of a word β is defined by the recurrences

$$\omega^*(\beta) = \begin{cases} 0 & \text{if } \beta \text{ is the empty string,} \\ \omega^*(\gamma) & \text{if } \beta = \texttt{<}\gamma \text{ or } \beta = \gamma\texttt{>}, \\ 2\omega^*(\gamma) & \text{if } \beta = \texttt{0}\gamma, \\ 2\omega^*(\gamma) + 1 & \text{if } \beta = \texttt{1}\gamma, \\ 3\omega^*(\gamma) & \text{if } \beta = \texttt{A}\gamma, \\ 3\omega^*(\gamma) + 2 & \text{if } \beta = \texttt{B}\gamma, \\ 3\omega^*(\gamma) + 1 & \text{if } \beta = \texttt{C}\gamma. \end{cases} \tag{7}$$

In other words, the dual value is obtained by reversing the word (minus the delimiters) and reading it as a mixed-base representation of a number, with '`0`' and '`1`' as the base-2 digits, and '`A`', '`B`' and '`C`' as base-3 digits with values 0, 2, and 1, respectively. Note that the values of '`B`' and '`C`' in $\omega^*(\cdot)$ are swapped relative to its values in $\omega(\cdot)$.

The dual value has properties similar to those of the primal value, but reversed left-to-right. If β is a string $\tau_0\tau_1\cdots\tau_{m-1}\tau_m$, with each $\tau_i \in \Sigma \cup \Delta$, we have

$$\omega^*(\beta) = \sum_{i=0}^{m} \omega^*(\tau_i) 2^{b(i \vdash \beta)} 3^{t(i \vdash \beta)} \tag{8}$$

More generally, for any $\alpha, \beta \in (\Sigma \cup \Delta)^*$,

$$\omega^*(\alpha\beta) = \omega^*(\alpha) + 2^{b(\alpha)} 3^{t(\alpha)} \omega^*(\beta) \tag{9}$$

It follows that, for any $\beta', \beta'', \alpha \in (\Sigma \cup \Delta)^*$,

$$\omega^*(\beta') = \omega^*(\beta'') \;\Leftrightarrow\; \omega^*(\alpha\beta') = \omega^*(\alpha\beta'') \tag{10}$$

Let $\langle n \rangle^*$ be the shortest base-3 representation of the integer n, using digits `A` $= 0$, `B` $= 2$, and `C` $= 1$, reversed left-to-right. Then $\omega^*(\langle n \rangle^*) = n$ for all integer n.

2.4 Substitution rules

The productions (basic substitution rules) of G are

0>	→	>	1>	→	C>			
0A	→	A0	1A	→	B1	<A	→	<
0B	→	A1	1B	→	C0	<B	→	<1
0C	→	B0	1C	→	C1	<C	→	<B0

(11)

We will write $\beta \xrightarrow{*} \gamma$ if the word γ can be obtained from the word β by zero or more of these substitutions.

This grammar admits many derivations from a given initial word. For example, starting with the word <0B10> we can get, among many others, the two derivations below:

<0B10>	<0B10>
<A110>	<0B1>
<110>	<0BC>
<11>	<A1C>
<1C>	<AC1>
<C1>	<ACC>
<B01>	<CC>
<101>	<B0C>
<10C>	<BB0>
<1B0>	<BB>
<C00>	<1B>
<B000>	<C0>
<1000>	<C>
<100>	<B0>
<10>	<B>
<1>	<1>
<C>	<C>
<B0>	<B0>
<10>	<B>
<1>	<1>
<C>	<C>
<B0>	<B0>
<10>	<B>
<1>	<1>
<C>	<C>
<B0>	<B0>
⋮	⋮

(12)

The substitutions of table 11 can be justified by their effect on word values. It can be seen that, except for the two entries of the first row, every entry in the table preserves the value of the word. Namely, each application of one of those substitutions has the form

$$<\beta\delta\sigma\gamma> \;\rightarrow\; <\beta\sigma'\delta'\gamma> \tag{13}$$

or

$$\texttt{<}\sigma\gamma\texttt{>} \;\rightarrow\; \texttt{<}\beta'\gamma\texttt{>} \tag{14}$$

for words $\beta, \gamma, \beta' \in (\Sigma \cup \Delta)^*$, trits $\sigma, \sigma' \in \Sigma$, and bits $\delta, \delta' \in \Delta$. In the first case we can check that

$$\omega(\beta\sigma'\delta') = \omega(\beta\delta\sigma) \tag{15}$$

In the second case,

$$\omega(\beta') = \omega(\sigma) \tag{16}$$

Since the old and new prefixes have the same value, and the suffix γ> is unchanged, the value of the entire word is preserved. On the other hand, the two substitutions in the first row of table 11 do change the word's value. Applied to whole words, they are

$$\texttt{<}\beta\texttt{0>} \;\rightarrow\; \texttt{<}\beta\texttt{>} \tag{17}$$

and

$$\texttt{<}\beta\texttt{1>} \;\rightarrow\; \texttt{<}\beta\texttt{C>} \tag{18}$$

Let $r = \omega(\beta)$; in the first case, the (even) word value $\omega(\beta\texttt{0}) = 2\omega(\beta) = 2r$ is replaced by $\omega(\beta) = r$. In the second case, the (odd) word value $\omega(\beta\texttt{1}) = 2r + 1$ is replaced by $\omega(\beta\texttt{C}) = 3\omega(\beta) + 2 = 3r + 2$. Comparing with formulas (2), we conclude that every time one of these two rules is applied, the value of the word changes from m to $\phi(m)$.

Finally, observe that the substitutions that do not change the value either decrease the number of trits, or reduce the lexicographic rank of the word (assuming bits are sorted before trits). Therefore, any derivation in G that starts from any finite valid word must eventually use one of those two value-changing rules.

We conclude that, in any derivation of the grammar G, the values of successive words will form the Collatz sequence of the value of the starting word; except that each element of the latter may be repeated some finite number of times.

2.5 The Ztalloc sequence

One can check that every production of the form $\delta\texttt{>} \rightarrow \gamma$ or $\delta\sigma \rightarrow \gamma$, with $\delta \in \Delta$ and $\sigma \in \Sigma$, preserves the dual value of the word. On the other hand, the substitutions at the left end change the dual value m to $\psi(m)$, where

$$\psi(m) = \begin{cases} r & = & m/3 & \text{if } m = 3r \text{ for some } r \in \mathbb{N} \\ 6r + 2 & = & 2m & \text{if } m = 3r + 1 \text{ for some } r \in \mathbb{N} \\ 2r + 1 & = & (2m - 2)/3 & \text{if } m = 3r + 2 \text{ for some } r \in \mathbb{N} \end{cases} \tag{19}$$

The sequence of values obtained by this recurrence from a starting integer n is a dual of the Collatz sequence, which we will call the *Ztalloc sequence of* n for lack of a better name. The sequence of 16, for example, is $(16, 32, 21, 7, 14, 9, 3, 1, 2, 1, 2, \ldots)$.

The Collatz conjecture can be stated also for the Ztalloc sequence, and the model above shows that the two are equivalent: one sequence includes the integer 1 if and only if the other sequence does.

One can define a *double recurrence* over the pairs of positive integers, by the formula $(m, n) \mapsto (\psi(n), \psi(n))$.

3 Greedy derivation

The left column of table (12) is an example of a *greedy* derivation, where each step is determined by the following rule: if the word contains any of the trits {A, B, C}, then apply to the leftmost trit occurrence the (unique) value-preserving substitution that can be applied to it; otherwise, use the (unique) value changing substitution.

Note that a greedy derivation is completely determined by the starting word. We will write $\beta \mapsto \gamma$ if γ can be obtained from β in a single greedy step, and *beta* $\overset{*}{\mapsto} \gamma$ if it can be obtained in zero or more such steps.

If a valid word contains one or more trits, the unique greedy derivation step that can be applied to it either eliminates one trit, or preserves the number of trits and reduces the number of bits to the left of the first trit. In either case the value of the word is preserved. Therefore, starting from an arbitrary valid word β, a greedy derivation will eventually arrive to a trit-free word <[m]>, all the while preserving the word's value $m = \omega(\beta)$. Then the next greedy derivation step will either remove a final 0 from $[m]$, or replace its final 1 by the C trit; in either case changing the word's value from m to $\phi(m)$.

When using greedy derivation, it is convenient to start with a word of the form $[\![n]\!] =$ <[n]> where n is a positive integer and $[n]$ is its base-2 encoding without leading zeros. Thus, for example, $[\![1]\!] =$ <1>, $[\![2]\!] =$ <10>, and $[\![19]\!] =$ <10011>.

If a greedy derivation starts from $[\![n]\!]$, then every word in it will contain at most one trit. Let β_0, β_1, ... be the trit-free words that occur in the derivation. We conclude that the sequence $w(\beta_i)$ is the (optimized) Collatz sequence of the integer n. For example, a greedy

derivation for $n = 6$ would be

β	$\omega(\beta)$	$\omega^*(\beta)$
<01$\underline{0>}$	6	
<1$\underline{1>}$	3	
<$\underline{1C>}$	5	
$\underline{<C}$1>	5	
$\underline{<B}$01>	5	
<10$\underline{1>}$	5	
<$\underline{1C>}$	8	
<$\underline{1B}$0>	8	
$\underline{<C}$00>	8	
$\underline{<B}$000>	8	
<100$\underline{0>}$	8	
<10$\underline{0>}$	4	
<$\underline{10>}$	2	
<$\underline{1>}$	1	
$\underline{<C>}$	2	
$\underline{<B}$0>	2	
<$\underline{10>}$	2	
<$\underline{1>}$	1	
$\underline{<C>}$	2	
$\underline{<B}$0>	2	
<$\underline{10>}$	2	
<$\underline{1>}$	1	
$\underline{<C>}$	2	
$\underline{<B}$0>	2	
⋮	⋮	⋮

(20)

We remark that the last rule in the last row of table (11), <C → <B0, could be replaced by <C → <10, since the B must necessarily change to 1 at some point. This optimization would save a few steps in greedy derivations, and simplify some of the argument; however, it would considerably complicate the definition and analysis of section 4.

4 Lazy derivations

The right column of table (12) is an example of a *lazy* derivation, defined by the following rule: if the word contains any bits, use the (unique) substitution from the first two columns of table (11) that can be applied as far to the right as possible; otherwise apply the (unique) substitution from the third column that can be used to the delimiter '<'.

In a lazy derivation, if a valid word contains any bits, the next step either reduces the number of bits, or preserves the number of bits but reduces the number of trits to the right of the last bit. Therefore, starting from any valid word, a lazy derivation will eventually produce a bit-free word. The previous step (if any), that eliminated the last digi, must have used a substitution from the first row of table (11), and therefore must have changes the

word's value from some integer m to $\phi(m)$. The word values will then remain unchanged until the next bit-free word.

In view of this observation, for the study of lazy derivations it is conventient to start with a *lazy encoding* of a positive integer n, which is a valid bit-free word $\langle\langle n \rangle\rangle$ whose value is n. Namely we define $\langle\langle() \rangle\rangle n) = \texttt{<}\langle n \rangle\texttt{>}$, where $\langle n \rangle$ is the representation of n in base 3 using the trits `A`, `B`, and `C` as digits with values 0, 1, and 2, respectively, without any leading `A`. For example, $\langle\langle 1 \rangle\rangle = \texttt{<B>}$, and $\langle\langle 19 \rangle\rangle = \texttt{<BAC>}$ (since $19 = (102)_3$).

In lazy derivation of G_1 that starts with a lazy encoding $\langle\langle n \rangle\rangle$, every word will contain at most one bit. as before, let β_0, β_1, β_2, ... be the bit-free words in the derivation. Their values will be the (optimized) Collatz sequence of n. For example, the lazy derivtion of $\langle\langle 6 \rangle\rangle = \texttt{<CA>}$ is

β	$\omega(\beta)$	$\omega^*(\beta)$
<CA>	6	
<B0A>	6	
<BA0>	6	
<BA>	3	
<1A>	3	
<B1>	3	
<BC>	5	
<1C>	5	
<C1>	5	
<CC>	8	
<B0C>	8	
<BB0>	8	
<BB>	4	
<1B>	4	
<C0>	4	
<C>	2	
<B0>	2	
<B>	1	
<1>	1	
<C>	2	
<B0>	2	
<B>	1	
<1>	1	
<C>	2	
<B0>	2	
<B>	1	
<1>	1	
⋮	⋮	⋮

(21)

One can view the lazy derivation as a dual of the greedy derivation, where the bits and trits change roles, and the computation of the ϕ function proceeds in opposite directions.

5 Parallel derivation

One can increase the "efficiency" of this model by carrying out all applicable substitutions in parallel.

For this purpose we define an *alternating* word as a string with one of the forms

$$\begin{array}{lll} \texttt{<} & \delta_1\sigma_1\delta_2\sigma_2\cdots\delta_{k-1}\sigma_{k-1}\texttt{>} & [-:-] \\ \texttt{<} & \delta_1\sigma_1\delta_2\sigma_2\cdots\delta_{k-1}\sigma_{k-1}\delta_k\texttt{>} & [-:+] \\ \texttt{<}\sigma_0 & \delta_1\sigma_1\delta_2\sigma_2\cdots\delta_{k-1}\sigma_{k-1}\texttt{>} & [+:-] \\ \texttt{<}\sigma_0 & \delta_1\sigma_1\delta_2\sigma_2\cdots\delta_{k-1}\sigma_{k-1}\delta_k\texttt{>} & [+:+] \end{array} \tag{22}$$

for some $k > 0$, except <>, where each symbol δ_i is in Δ and each σ_i is in Σ. We also define the *parallel encodings* $[\![n]\!]$ and $\langle\!\langle\!\langle n \rangle\!\rangle\!\rangle$ of a positive integer n to be shortest alternating words that end with a bit or trit, respectively, whose value is n. For example:

$$\begin{array}{rclrcl} [\![1]\!] &=& \texttt{<1>} & \langle\!\langle\!\langle 1 \rangle\!\rangle\!\rangle &=& \texttt{<B>} \\ [\![6]\!] &=& \texttt{<1A0>} & \langle\!\langle\!\langle 6 \rangle\!\rangle\!\rangle &=& \texttt{<B0A>} \\ [\![19]\!] &=& \texttt{<B1A1>} & \langle\!\langle\!\langle 19 \rangle\!\rangle\!\rangle &=& \texttt{<1A0B>} \\ [\![34]\!] &=& \texttt{<C1C0>} & \langle\!\langle\!\langle 34 \rangle\!\rangle\!\rangle &=& \texttt{<1C1B>} \end{array} \tag{23}$$

A *parallel derivation* is a sequence of alternating words, where each new word is derived from the preceding one by a *parallel derivation step*; which consists of parsing the current word in one of the forms (22), applying the appropriate substitution to each of the pairs '<σ_0', 'δ_k>', and '$\delta_i\sigma_i$' for $i \in \{1..\,k-1\}$, and concatenating the resulting strings.

Note that, for each of those symbol pairs, there is exactly one rule in table (11) that can be applied. For the internal pairs $\delta_i\sigma_i$ of any form, this operation replaces a string in $\Delta\Sigma$ by a string in $\Sigma\Delta$. If the word is of the form $[+:-]$ or $[+:+]$, the initial pair <σ_0 is replaced by '<' or by a pair in <Δ or in <$\Sigma\Delta$; otherwise the initial '<' is not substituted. If the word is of the form $[-:+]$ or $[+:+]$, the last pair δ_k> is replaced by just > or by a pair in Σ>; otherwise the final '>' is not substituted.

In any case the result of a parallel derivation step is an alternating word. Moreover, if the word is of the form $[-:+]$ or $[+:+]$ its value is changed from m to $\phi(m)$; otherwise its value remains unchanged. Therefore, if we start with either $[\![n]\!]$ or $\langle\!\langle\!\langle n \rangle\!\rangle\!\rangle$, the values of the words in the parallel derivation will include all the terms of the (optimized) Collatz

sequence, with occasional repetitions. For example:

β	Type	$\omega(beta)$
<B 0 A>	$[+:-]$	6
<1 A 0>	$[-:+]$	6
<B 1>	$[+:+]$	3
<1 C>	$[-:-]$	5
<C 1>	$[+:+]$	5
<B 0 C>	$[+:-]$	8
<1 B 0>	$[-:+]$	8
<C 0>	$[+:+]$	4
<B 0>	$[+:+]$	2
<1>	$[-:+]$	1
<C>	$[+:-]$	2
<B 0>	$[+:+]$	2
<1>	$[-:+]$	1
<C>	$[+:-]$	2
<B 0>	$[+:+]$	2
<1>	$[-:+]$	1
<C>	$[+:-]$	2
<B 0>	$[+:+]$	2
⋮	⋮	⋮ ⋮

(24)

6 Macro-symbols

We can further "speed up" the model by grouping bits and trits into macro-symbols, and defining a grammar that operates on groups rather than individual bits .

First we choose a *binary suffix code* $\mathcal{B}$; that is, a finite set H of strings in Δ^+ such that every bit string in Δ^+ is a suffix of some word in $\mathcal{B}$, or has a unique suffix that is in $\mathcal{B}$. A trivial example of binary suffix code is the set Δ^k of all the bit strings with a fixed length k. Another example is the set $C = \{\texttt{00}, \texttt{010}, \texttt{110}, \texttt{1}\}$. Basically, a binary suffix code is the set of leaf-to-root paths from in any finite binary tree. Any string δ of Δ^* can be parsed into a concatenation $\alpha\delta_1\delta_2\cdots\delta_r$, for some $r \geq 0$, where each δ_i is a code word of $\mathcal{B}$ and α is a proper suffix (possibly empty) of a code word. We call this decomposition the *suffix $\mathcal{B}$-parsing* of δ.

We also choose a *ternary prefix code* $\mathcal{T}$; that is, a finite set $\mathcal{T}$ of strings in Σ^+ such that every bit string in Σ^+ a prefix of some word in $\mathcal{T}$, or has a unique prefix that is in $\mathcal{T}$. A trivial example of ternary prefix code is the set Σ^k of all the trit strings with a fixed length k. Another example is the set $C = \{\texttt{AAA}, \texttt{AAB}, \texttt{AAC}, \texttt{AB}, \texttt{AC}, \texttt{B}, \texttt{C}\}$. Basically, a ternary prefix code is the set of root-to-leaf paths from in any finite ternary tree. Any string σ of Σ^* be parsed into a concatenation $\sigma_1\sigma_2\cdots\sigma_s\beta$, where each σ_i is a code word of $\mathcal{T}$ and β is a proper prefix (possibly empty) of a code word. We call this decomposition the *prefix $\mathcal{T}$-parsing* of σ.

By definition, the individual symbols in the new grammar G' comprise the delimiters `<` and `>`, a set of *macro-bits* or *bytes* Δ', and a set of *macro-trits* or *trytes* Σ'. The bytes Δ'

are the words of $\mathcal{B}$ and all its non-empty proper suffixes; the trytes Σ' are the words of $\mathcal{T}$ and all their non-empty proper prefixes. In both cases, each word is treated as single symbol of G'. For clarity, when writing a word of G' we will use parentheses to delimit individual bytes and trytes.

For G', the notation $[n]$ is redefined to mean the suffix $\mathcal{B}$-parsing of the binary representation of n, left-padded with zeros as needed so that the first term is a complete code word; where each term of the parsing is considered a separate macro symbol. For example, if we use the code Δ^3, then [19] is the word (010)(011), with two bytes 010 and 011. If we use the code C above, [19] = (1)(00)(1)(1), [20] = (010)(1)(00), and [27] = (110)(1)(1). Similarly, we redefine $\langle n \rangle$ to the prefix $\mathcal{H}$-parsing of the base-3 representation of n, with digits A = 0, B = 1 and B = 2, with each term treated as a separate macro-symbol.

The rules of G' are then defined as follows. First, we consider each word of G of the form δ>, where δ is a byte. We apply lazy derivation in G to this (partial) word until we obtain a word σ'> where σ is in Σ^+ (a bit-free string of trits). Let $\sigma'_1\sigma'_2\cdots\sigma'_s\beta$ be the prefix $\mathcal{T}$-parsing of σ'. We then add the production (δ)> $\rightarrow (\sigma'_1)(\sigma'_2)\cdots(\sigma'_s)(\beta)$> to the rule set of G', except that the macro-symbol (β) is omitted if β is empty. Note that s may be zero in either case.

Next, we consider all strings of the form $\delta\sigma$ where δ is a byte (in Δ') and σ is a tryte (in Σ'). Again we use lazy evaluation in G to transform that string into a string $\sigma'\delta'$ where σ' is a string of trits of G and δ' is a string of bits of G (which are both non-empty, by the rules of G). Let $\alpha\delta'_1\delta'_2\cdots\delta'_r$ be the suffix $\mathcal{B}$-parsing of δ', and $\sigma'_1\sigma'_2\cdots\sigma'_s\beta$ be the prefix $\mathcal{T}$-parsing of σ'. We then add to the rules of G' the production

$$(\delta)(\sigma) \rightarrow (\sigma'_1)(\sigma'_2)\cdots(\sigma'_s)(\beta)(\alpha)(\delta'_1)(\delta'_2)\cdots(\delta'_r)$$

except that (α) is omitted if α is empty, and (β) is omitted if β is empty.

Finally, we consider every string of the form <σ where σ is one of the trytes of Σ', and use greedy derivation of G to turn it into a string <δ' where δ is a trit-free string of bits. Let $\alpha\delta'_1\delta'_2\cdots\delta'_r$ be the suffix $\mathcal{B}$-parsing of δ'. We add to the rule sof G' the production <$(\sigma) \rightarrow$ <$(\alpha)(\delta'_1)(\delta'_2)\cdots(\delta'_r)$, omitting (α) if α is empty.[1]

This construction becomes quite a bit simpler when we use fixed-length codes $\mathcal{B} = \Delta^k$ and $\mathcal{T} = \Sigma^k$ with $k \geq 2$. In that case, the lazy derivation of any string δ> for $\delta \in \Delta'$ generates a string γ> where γ has at most k trits of G, and is therefore a single tryte. The lazy derivation of any string $\delta\sigma$, where δ is a byte and σ is a tryte, generates a string $\sigma'\delta'$ where δ' is always a single byte, and σ' is a single tryte. Finally, the lazy derivation of <σ, where σ is a single tryte of G, can be adjusted to generate '<', <δ', or <$\sigma'\delta'$ where δ' is a single byte and δ' a single tryte that is lexicographically smaller than σ. Thus we can restrict the set of bytes to the code words $\mathcal{B}$ (excluding its proper suffixes).[2]

References

[1] Placeholder.

[1] This is not good, rethink. For the fixed-length case we should get alternating trytes and bytes.

[2] See `temp.tex` for further stuff.