Bonus problem 2: No repetitions challenge

You may submit a solution to the following problem for bonus marks.

For any $k \geq 1$, define the alphabet $\Sigma_k = \{1, 2, \ldots, k\}$ and let L_k be the language over Σ_k that includes every nonempty string which has no consecutive symbols that are identical. For example,

$$L_3 = \{1, 2, 3, 12, 13, 23, 121, 123, 131, 132, 212, \ldots\}.$$

What is the shortest regular expression that you can construct for L_k?

Here are some of the known results for this problem. See which ones you can match or beat!

(i) It is not too difficult to show that L_k has a regular expression of length polynomial in k.
(ii) Prof. Shallit, who proposed this problem, obtained a regular expression of length $O(k^{2.585})$.
(iii) Bai Li and Ron Meng, students from an earlier offering of CS 365, obtained a regular expression of length $O(n^{2.388})$.
(iv) Luke Schaeffer, a former student here and now a Ph.D. candidate at MIT, obtained a regular expression of length $O(k^{2.441})$ and later improved this result to have length only $O(k^{2.17})$.
(v) It is conjectured that the shortest regular expression for this language has length $O(k^2)$.

Note. For simplicity, you may assume that k is a power of 2 for this question.

No Repetition Regex in $O(2^{2.331})$

Aditya Jayaprakash (20579038) and Milap Sheth*(20579040)
{ajayapra,m2sheth}@edu.uwaterloo.ca

April 23, 2017

Bonus problem 2: No repetitions challenge

For any $k \geq 1$, define the alphabet $\Sigma_k = \{1, 2, \ldots, k\}$ and let L_k be the language over Σ_k that includes every nonempty string which has no consecutive symbols that are identical. For example,

$$L_3 = \{1, 2, 3, 12, 13, 23, 121, 123, 131, 132, 212, \ldots\}$$

What is the shortest regular expression that you can construct for L_k?

Solution

The simplest regular expression for this problem is when $\Sigma_1 = \{1\}$ and $r_1 = 1$.

When $\Sigma_2 = \{1, 2\}$, $r_2 = (2|\epsilon)(12)^*1|(1|\epsilon)(21)^*2$. We will first show that the regex for L_2 indeed recognizes only nonempty strings that have no consecutive identical symbols.

1 $(2|\epsilon)(12)^*1|(1|\epsilon)(21)^*2$ is a regular expression for L_2

For L_2 to accept a string, it needs to either end with a 2 or 1 since it doesn't accept any non-empty strings. Let w be the string:

Case 1: $|w| = 1$*:* The only possible strings recognized by r_2 are 1 and 2 and the bolded parts of $(2|\epsilon)(12)^*\mathbf{1}|(1|\epsilon)(21)^*2$ and $(2|\epsilon)(12)^*1|(1|\epsilon)(21)^*\mathbf{2}$ are used to recognize it.

Case 2: $|w|$ *is odd:* The only strings recognized by L_2 are 121,12121,... or 212,21212, ... and the bolded parts of $(2|\epsilon)(\mathbf{12})^*\mathbf{1}|(1|\epsilon)(21)^*2$ and $(2|\epsilon)(12)^*1|(1|\epsilon)(\mathbf{21})^*\mathbf{2}$ is used to recognize it.

*Milap Sheth isn't a student of CS365 W17

Case 3: $|w|$ is even: The only strings recognized by L_2 are 21,2121,... or 12,1212,... and the bolded parts of $(2|\epsilon)$ $(\mathbf{12})^*1|(1|\epsilon)(21)^*2$ and $(2|\epsilon)(12)^*1|(1|\epsilon)\mathbf{21}^*2$ is used to recognize it.

Hence $r_2 = (2|\epsilon)(12)^*1|(1|\epsilon)(21)^*2$

The problem states that r_k for every k needs to be non-empty. Let us define a new language, r_k^ϵ which recognizes strings that has no consecutive symbols, but also accepts the empty string.

$r_1^\epsilon = 1|\epsilon$ and $r_2^\epsilon = (1|\epsilon)(21)^*(2|\epsilon)$. We could also define r_k^ϵ as $r_k^\epsilon = \epsilon|r_k$.

2 Generalizing L_2 for any a,b $\in \mathbb{N}$

For the above cases, we have considered $\Sigma = \{1, 2\}$, but what if we wanted to create a language for recognizing non-consecutive symbols for any $a, b \in \mathbb{N}$. Let $\Sigma_{ab} = \{a, b\}$, $\Sigma_a = \{a\}$ and $\Sigma_b = \{b\}$ where a, b are positive integers. We define

$$\begin{aligned} r_{\Sigma_{ab}} &= (b|\epsilon)(ab)^*a|(a|\epsilon)(ba)^*b \\ r_{\Sigma_a} &= a \\ r_{\Sigma_b} &= b \end{aligned}$$

We could also rewrite $r_{\Sigma_{ab}}$ as

$$r_{\Sigma_{ab}} = (r_{\Sigma_b}|\epsilon)(r_{\Sigma_a}r_{\Sigma_b})^*r_{\Sigma_a}|(r_{\Sigma_a}|\epsilon)(r_{\Sigma_b}r_{\Sigma_a})^*r_{\Sigma_b}$$

3 Generating a regular expression for L_k

So far, we've restricted Σ to only contain only two numbers, but let $\Sigma_k = \{1, ..., k\}$. We wish to generate a regular expression for the language L_k such that it recognizes strings that don't have consecutive symbols. To do so, we make an crucial observation about our solution for L_2. r_2 was created by splitting the alphabet into two symbols such that no two consecutive symbols are the same. This can be generalized to splitting the alphabet into two groups such that a word in L_k can be divided into consecutive maximal sub-words that are from different groups and each sub-word is part of the language $L_{k'}$ where k' is the size of the group alphabet. no two consecutive symbols are from the same group. This gives us a recursive method to a build a regular expression for L_k using our previously defined regex for L_2.

Lemma 1. *For $k \geq 2$, there exists an onto function f that maps a string in L_k to a string in L_2.*

Proof. Let $w \in L_k$, $\Sigma^1 \subset \Sigma_k$ such that $|\Sigma^1| = \frac{k}{2}$ and $\Sigma^2 = \Sigma_k - \Sigma^1$ w can now be divided into alternating strings that only contain symbols from Σ^1 or Σ^2 in the following way,

To construct a function $f : L_k \mapsto L_2$, we substitute a sub-word in w for a symbol in $\{1, 2\}$ as follows:

Let $a \in \{1, 2\}$, $b \in \{1, 2\} - a$ such that $w = uw'$ where $\Sigma(u) \subseteq \Sigma^a$ ($u \neq \epsilon$ since w is non-empty). Replace u with a. Now proceed with w' as follows:

Case 1: $w' = \epsilon$: In this case, we replace it with ϵ itself.

Case 2: $w' = vw''$: In this case $\Sigma(v) \subseteq \Sigma_k - \Sigma^a$ such that w'' is either ϵ or starts with a symbol in Σ^a. So we can replace v with b. Now, we can recursively replace w''.

Thus, we have constructed a non-empty string of symbols a, b mapped from w. This shows that f is well-defined. Now to show that f is onto, i.e, $\forall y \in L_2, \exists x \in L_k, f(x) = y$. This is true since, $\exists\, L_2' \subset L_k$ where L_2' is defined over the alphabet $\{a, b\}$, $a \in \Sigma_U, b \in \Sigma_V$. and thus we have $\forall\, w \in L_2', f(w) = w$. ■

Let $\Sigma_U \subset \Sigma_k$ such that $|\Sigma_U| > 0$ and $\Sigma_V \subset \Sigma_k - \Sigma_U$.

Theorem 1. *A regular expression for L_2, r_2, is also a regular expression for L_k under the map g which substitutes a for a regular expression for L_{Σ_U} and b for a regular expression for L_{Σ_V} where $\Sigma_2 = \{a, b\}$ and $\Sigma_U \dot\cup \Sigma_V = \Sigma_k$.*

Proof. We use induction on the claim to prove it.

Base Case: $k = 1$
Then let $a \in \Sigma_1$. $r_1 = a$ trivially.

Inductive Case: $k > 1$
Assume that this holds for all $1 \leq i < k$. Now, we use the function f from Lemma 1. Let $w \in L_k$ and $w' = f(w) \in L_2$. f divides the alphabet Σ_k into two groups Σ_U and Σ_V that correspond to two symbols in Σ_2. Each sub-word substituted in w contains symbols from either Σ_U or Σ_V. Thus, each sub-word t is either in L_{Σ_U} or L_{Σ_V}. Thus, we can use a regular expression for either of these languages to recognize t. Now, for $\{a, b\} \in \Sigma_2$, we substitute a with r_{Σ_U} and b with r_{Σ_V}. This gives a regular expression for r_k under this map g from r_2. ■

Using Theorem 1 on r_2, we have,

$$r_{\Sigma_k} = \begin{cases} a & k = 1,\, a \in \Sigma_k \\ (r_{\Sigma_V}|\epsilon)(r_{\Sigma_U} r_{\Sigma_V})^* r_{\Sigma_U} | (r_{\Sigma_U}|\epsilon)(r_{\Sigma_V} r_{\Sigma_U})^* r_{\Sigma_V} & \text{otherwise} \end{cases}$$

4 Analysis of r_k

Now, we can analyze the length of regular expression we obtained through the above method. It's obvious that it depends only on the size of alphabet, k. r_k is composed of 8 regular expressions defined on Σ' and Σ'' along with some constant number of symbols.
Thus, length of r_k, $A(k)$ is,

$$A(k) = 4A\left(\left\lfloor \frac{k}{2} \right\rfloor\right) + 4A\left(\left\lceil \frac{k}{2} \right\rceil\right) + O(1) \approxeq 8A\left(\left\lfloor \frac{k}{2} \right\rfloor\right) + O(1)$$

Using Master's theorem, we obtain,

$$A(k) \in \Theta(k^3)$$

We can see that the length is polynomial in k although it does not improve upon any of the good existing solutions. Thus, we now try to optimize our regular expression.

5 Looking at another language

One might observe as to how the condition of nonempty string in the language restricts our regular expression. Removing this restriction can actually shorten our regular expression. Another insight is that r_k allows for ϵ to be recognized in some places. Thus, one can optimize the length by using a regular expression for this new language instead of L_{Σ_U} or L_{Σ_V} when ϵ is allowed. Let this new language be $L_k^\epsilon = L_k \cup \epsilon$. We first obtain a regular expression r_2^ϵ for L_2^ϵ.
Let w be a string in L_2^ϵ:

Case 1: $|w| = 0$*:* It clearly accepts the empty string because every part of the regex can recognize empty and if all parts recognize empty, the regex also recognizes empty.

Case 2: $|w| = 1$*:* It either accepts 1 or 2 which occur when $(\mathbf{1}|\epsilon)(21)^*(2|\epsilon)$ or $(1|\epsilon)(21)^*(\mathbf{2}|\epsilon)$ is used to recognize the string.

Case 3: $|w|$ *is even:* It either uses $(\mathbf{1}|\epsilon)(21)^*(\mathbf{2}|\epsilon)$ or $(1|\epsilon)(\mathbf{21})^*(2|\epsilon)$ or $(\mathbf{1}|\epsilon)(\mathbf{21})^*(\mathbf{2}|\epsilon)$ which accepts all even length strings i.e., 12,21,1212,2121,...

Case 4: $|w|$ *is odd:* It either uses $(1|\epsilon)(\mathbf{21})^*(\mathbf{2}|\epsilon)$ or $(\mathbf{1}|\epsilon)(\mathbf{21})^*(2|\epsilon)$ which accepts all string that ends either 1 or 2 and that are odd. e.g., 121,212,12121,21212,...

Hence, $r_2^\epsilon = (1|\epsilon)(21)^*(2|\epsilon)$

6 Generalizing L_2 and L_2^ϵ for any a,b $\in \mathbb{N}$

We again generalize the languages for recognizing non-consecutive symbols for any $a, b \in \mathbb{N}$. Let $\Sigma_{ab} = \{a, b\}$, $\Sigma_a = \{a\}$ and $\Sigma_b = \{b\}$ where a, b are positive integers. Define

$$
\begin{aligned}
r_{\Sigma_{ab}} &= (b|\epsilon)(ab)^*a|(a|\epsilon)(ba)^*b \\
r^\epsilon_{\Sigma_{ab}} &= (a|\epsilon)(ba)^*(a|\epsilon) \\
r^\epsilon_{\Sigma_b} &= b|\epsilon \\
r^\epsilon_{\Sigma_a} &= a|\epsilon \\
r_{\Sigma_a} &= a \\
r_{\Sigma_b} &= b
\end{aligned}
$$

We could also rewrite $r_{\Sigma_{ab}}$ and $r^\epsilon_{\Sigma_{ab}}$ as

$$r_{\Sigma_{ab}} = r^\epsilon_{\Sigma_b}(r_{\Sigma_a}r_{\Sigma_b})^*r_{\Sigma_a}|r^\epsilon_{\Sigma_a}(r_{\Sigma_b}r_{\Sigma_a})^*r_{\Sigma_b}$$

$$r^\epsilon_{\Sigma_{ab}} = r^\epsilon_{\Sigma_a}(r_{\Sigma_b}r_{\Sigma_a})^*r^\epsilon_{\Sigma_b}$$

7 A shorter regular expression for L_2

Finding a shorter regular expression for L_2 could indeed lower the length of a regular expression for L_k. In the regex we found for L_k in the previous section, r_k consists of eight regexes. There are two regexes which accepts the empty string and six regex which don't. We also also know that $|r^\epsilon_k| < |r_k|$. One could build a shorter regex for r_k by coming up with a regex with a lower number of child regexes that accept non-empty strings.

After some thought, a shorter regex for recognizing all strings in $L_{\Sigma_{ab}}$ where $\Sigma_{ab} = \{a, b\}$ is

$$r_{\Sigma_{ab}} = (a|\epsilon)b(ab)^*(a|\epsilon)|a$$

$(a|\epsilon)b(ab)^*(a|\epsilon)|a$ recognizes the exact same strings that $(b|\epsilon)(ab)^*a|(a|\epsilon)(ba)^*b$ recognizes. The two regular expressions are equivalent in power. Let w be a string in $L_{\Sigma_{ab}}$, then:

Case 1: $|w| = 1$*:* a and b are the only strings in $L_{\Sigma_{ab}}$ and it's recognized by $(a|\epsilon)\mathbf{b}(ab)^*(a|\epsilon)|a$ or $(a|\epsilon)b(ab)^*(a|\epsilon)|\mathbf{a}$

Case 2: $|w| = 2$*:* ab and ba are the only strings in $L_{\Sigma_{ab}}$ and it's recognized by $(a|\epsilon)b(\mathbf{ab})^*(a|\epsilon)|a$ and $(a|\epsilon)\mathbf{b}(ab)^*(\mathbf{a}|\epsilon)|a$

Case 3: $|w|$ *is even:* *abab*,*ababab*,... and *baba*,*bababa*,... are strings of even length in $L_{\Sigma_{ab}}$ and it's recognized by $(a|\epsilon)b(\mathbf{ab})^*(a|\epsilon)|a$ and $(a|\epsilon)\mathbf{b}(\mathbf{ab})^*(\mathbf{a}|\epsilon)|a$

Case 4: $|w|$ is odd: aba,ababa,... and *bab,babab,...* are strings of odd length in $L_{\Sigma_{ab}}$ and it's recognized by $(a|\epsilon)b(\mathbf{ab})^*(\mathbf{a}|\epsilon)|a$ and $(a|\epsilon)\mathbf{b}(\mathbf{ab})^*(a|\epsilon)|a$

We can further use this to find a regex for L_k. Our new regex for L_k is

$$r_{\Sigma_k} = r^{\epsilon}_{\Sigma_V} r_{\Sigma_U} (r_{\Sigma_V} r_{\Sigma_U})^* r^{\epsilon}_{\Sigma_V} | r_{\Sigma_V}$$

$$r^{\epsilon}_{\Sigma_k} = r^{\epsilon}_{\Sigma_U} (r_{\Sigma_V} r_{\Sigma_U})^* r^{\epsilon}_{\Sigma_V}$$

where r_{Σ_U} and r_{Σ_V} are smaller regular expressions for the alphabets Σ_U and Σ_V.

8 Analysis of the new r_k

Let $A(k) = |r_k|$ and $A'(k) = |r^{\epsilon}_k|$.

$$A'(k) = 2A\left(\frac{k}{2}\right) + 2A'\left(\frac{k}{2}\right) + x \tag{1}$$

$$A(k) = 4A\left(\frac{k}{2}\right) + 2A'\left(\frac{k}{2}\right) + y \tag{2}$$

From the above two recurrences, we can get the following,

$$A(k) = 2A\left(\frac{k}{2}\right) + A'(k) + x - y \quad (2) - (1) \tag{3}$$

$$2A'(k) = A(k) + 2A'\left(\frac{k}{2}\right) + y \quad (1) - (3) \tag{4}$$

Using, recurrence (4), we can keep on substituting in recurrence (2) to get only $A(k)$ terms in it.
Thus, we get

$$\begin{aligned} A(k) &= 4A\left(\frac{k}{2}\right) + \sum_{i=1}^{\log_2 k} \left(A\left(\frac{k}{2^i}\right) + y\right) + y \\ &= 4A\left(\frac{k}{2}\right) + \sum_{i=1}^{\log_2 k} A\left(\frac{k}{2^i}\right) + y(\log_2 k + 1) \end{aligned}$$

Theorem 2. $\exists x \geq 1, A(k) \in O(k^x)$

Proof. We prove this by induction.
Base case: $k = 1$: $A(1) = 1 \leq 1^x = 1$

$$
\begin{aligned}
A(k) &= 4A\left(\frac{k}{2}\right) + \sum_{i=1}^{\log_2 k} A\left(\frac{k}{2^i}\right) + 12(\log_2 k + 1) \\
&\leq 4\left(\frac{k}{2}\right)^x + \sum_{i=1}^{\log_2 k} c\left(\frac{k}{2^i}\right)^x + 12(\log_2 k + 1) \\
&\leq 4\left(\frac{k}{2}\right)^x + k^x \sum_{i=1}^{\log_2 k} \left(\frac{1}{(2^x)^i}\right) + 12(\log_2 k + 1) \\
&\leq 4\left(\frac{k}{2}\right)^x + k^x \frac{1}{2^x}\left(\frac{1 - \frac{1}{2^x}^{\log_2 k}}{1 - \frac{1}{2^x}}\right) + 12(\log_2 k + 1) \quad \text{sum of geometric series} \\
&\leq \left(\frac{k}{2}\right)^x \left(4 + \frac{1 - \frac{1}{k^x}}{1 - \frac{1}{2^x}}\right) + 12(\log_2 k + 1) \\
&\leq \left(\frac{k}{2}\right)^x \left(4 + \frac{2^x(k^x - 1)}{k^x(2^x - 1)}\right) + 12(\log_2 k + 1) \\
&\leq 4\left(\frac{k}{2}\right)^x + \frac{k^x - 1}{2^x - 1} + 12(\log_2 k + 1) \\
&\leq \left(4\left(\frac{1}{2}\right)^x + \frac{1}{2^x - 1}\right) k^x - \frac{1}{2^x - 1} + 12(\log_2 k + 1) \\
&\leq \left(4\left(\frac{1}{2}\right)^x + \frac{1}{2^x - 1}\right) k^x + \epsilon k^x \quad \text{for } 12(\log_2 k + 1) \leq \epsilon k^x, 0 < \epsilon < 1 \\
&\leq \left(4\left(\frac{1}{2}\right)^x + \frac{1}{2^x - 1} + \epsilon\right) k^x
\end{aligned}
$$

This works since $\forall\, x > 0$, $k^x \in \omega(\log_2(k))$ and thus we can always pick an $0 < \epsilon < 1$ for a given x that holds for all $k \geq k_0 > 0$. To show that $A(k) \leq k^x$, we need to show that the coefficient of k^x is less than or equal to 1. Thus,

$$
\begin{aligned}
4\left(\frac{1}{2}\right)^x + \frac{1}{2^x - 1} + \epsilon &\leq 1 \\
4\left(\frac{1}{2}\right)^x + \frac{1}{2^x - 1} &< 1 \\
4(2^x - 1) + 2^x &< 2^x(2^x - 1)
\end{aligned}
$$

Let $y = 2^x \implies 4(y-1) + y < y(y-1) \implies y^2 - 6y + 4 > 0$. Thus,

$$
y > \frac{6 + \sqrt{36 - 16}}{2} = 3 + \sqrt{5}
$$

Now, $y > 3 + \sqrt{5}$ since the other case is impossible given $y = 2^x \geq 1$

$\implies 2^x > 3 + \sqrt{5}$

$$\boxed{x > \log_2 (3 + \sqrt{5}) \approxeq 2.388}$$

■

9 Optimal split

In our previous definition of Σ_k, $|\Sigma_U| = |\Sigma_V| = \frac{k}{2}$. Both have the same number of elements, but in our shortest regex for r_k, one can observe that there are four regexes related to Σ_U and only two regexes related to Σ_V, so it makes more sense to split the the two sets unevenly. Let $kt_1 = |\Sigma_U|$ and $kt_2 = |\Sigma_V|$ where $1 = t_1 + t_2$. However, for the regex that allows empty, we will leave it to have an even split since the number of regexes related to Σ_U and Σ_V are equal.

$$A'(k) = 2A\left(\frac{k}{2}\right) + 2A'\left(\frac{k}{2}\right) + u \tag{5}$$

$$A(k) = 2A(t_1 k) + 2A(t_2 k) + 2A'(t_1 k) + v \tag{6}$$

10 Analysis of the optimal split regex for r_k

$$\begin{aligned} A'(k) &= 2A\left(\frac{k}{2}\right) + 2A'\left(\frac{k}{2}\right) + u \\ &= 2A\left(\frac{k}{2}\right) + 2\left(2A\left(\frac{k}{2^2}\right) + 2A'\left(\frac{k}{2^2}\right) + u\right) + u \end{aligned}$$

Expanding all A' terms gives us,

$$= \sum_{i=1}^{\log_2 k} \left(2^i A\left(\frac{k}{2^i}\right) + u2^{i-1}\right)$$

Substituting in $A(k)$, we get,

$$\begin{aligned} A(k) &= 2A(t_1 k) + 2A(t_2 k) + 2A'(t_1 k) + v \\ &= 2A(t_1 k) + 2A(t_2 k) + \sum_{i=1}^{\log_2 k} \left(2^i A\left(\frac{k}{2^i}\right) + u2^{i-1}\right) + v \end{aligned}$$

We use the similar induction technique as before to obtain the optimal value of the optimal split

Theorem 3. $\exists\, x \geq 2, A(k) \in O(k^x)$

Proof. Base case: $k = 1$: $A(1) = 1 \leq 1^x = 1$

Inductive hypothesis: $\forall\, 1 \leq i < k$, $A(i) \leq k^x$

$$
\begin{aligned}
A(k) &= 2A(t_1k) + 2A(t_2k) + \sum_{i=1}^{\log_2 t_1 k} \left(2^i A\left(\frac{t_1 k}{2^i}\right) + u2^{i-1}\right) + v \\
&\leq 2(t_1k)^x + 2(t_2k)^x + \sum_{i=1}^{\log_2 t_1 k} \left(2^i \left(\frac{t_1 k}{2^i}\right)^x + u2^{i-1}\right) + v \\
&\leq 2(t_1k)^x + 2(t_2k)^x + \sum_{i=1}^{\log_2 t_1 k} \left((t_1k)^x \left(\frac{1}{2^{x-1}}\right)^i + u2^{i-1}\right) + v \\
&\leq 2(t_1k)^x + 2(t_2k)^x + (t_1k)^x \sum_{i=1}^{\log_2 t_1 k} \left(\frac{1}{2^{x-1}}\right)^i + u \sum_{i=1}^{\log_2 t_1 k} 2^{i-1} + v \\
&\leq 2(t_1k)^x + 2(t_2k)^x + \left(\frac{(t_1k)^x}{2^{x-1}}\right) \frac{1 - \left(\frac{1}{2^{x-1}}\right)^{\log_2 t_1 k}}{1 - \frac{1}{2^{x-1}}} + u\frac{2^{\log_2 t_1 k} - 1}{2 - 1} + v \\
&\leq 2(t_1k)^x + 2(t_2k)^x + \left(\frac{(t_1k)^x}{2^{x-1}}\right) \frac{1 - \left(\frac{1}{(t_1k)^{x-1}}\right)}{1 - \frac{1}{2^{x-1}}} + u(t_1k - 1) + v \\
&\leq 2(t_1k)^x + 2(t_2k)^x + \left(\frac{(t_1k)^x}{2^{x-1}}\right) \frac{1 - \left(\frac{1}{(t_1k)^{x-1}}\right)}{1 - \frac{1}{2^{x-1}}} + u(t_1k - 1) + v \\
&\leq 2(t_1k)^x + 2(t_2k)^x + \frac{(t_1k)^x - t_1k}{2^{x-1} - 1} + u(t_1k - 1) + v \\
&\leq 2\left[t_1^x + t_2^x + \frac{t_1^x}{2^{x-1} - 1}\right] k^x + \left(u - \frac{1}{2^{x-1} - 1}\right) t_1 k - u + v \\
&\leq 2\left[t_2^x + t_1^x \left(1 + \frac{1}{2^{x-1} - 1}\right)\right] k^x + \left(u - \frac{1}{2^{x-1} - 1}\right) t_1 k - u + v \\
&\leq 2\left[t_2^x + t_1^x \left(1 + \frac{1}{2^{x-1} - 1}\right)\right] k^x + \left(u - \frac{1}{2^{x-1} - 1}\right) t_1 k - u + v \\
&\leq 2\left[t_2^x + t_1^x \left(1 + \frac{1}{2^{x-1} - 1}\right) + \epsilon\right] k^x
\end{aligned}
$$

Since, $g(k) = \left(u - \frac{1}{2^{x-1}-1}\right) t_1 k - u + v \in o(k^x), \exists\, \epsilon > 0$ such that $g(k) \leq \epsilon k^x$.

As before the coefficient of k^x has to be less than or equal to 1 for induction hypothesis to hold. We also know that $t_2 = 1 - t_1$ Thus,

$$
\begin{aligned}
2\left[t_2^x + t_1^x\left(1 + \frac{1}{2^{x-1}-1}\right) + \epsilon\right] &\leq 1 \\
2\left[t_2^x + t_1^x\left(\frac{2^x}{2^x-2}\right)\right] &< 1 \\
2\left[t_2^x(2^x-2) + t_1^x 2^x\right] &< 2^x - 2 \\
(1-t_1)^x(2^x-2) + (2t_1)^x &< 2^{x-1} - 1
\end{aligned}
$$

We can verify our method works for r_2 with a 50-50 split by substituting $t_1 = \frac{1}{2}$.

$\left(\frac{1}{2}\right)^x (2^x - 2) + 1^x < 2^{x-1} - 1 \implies 1 - \frac{1}{2^{x-1}} + 1 < 2^{x-1} - 1 \implies 2^{x-1} + \frac{1}{2^{x-1}} > 3$
Let $y = 2^{x-1}$,

$$
\begin{aligned}
y + \frac{1}{y} &> 3 \\
\implies y^2 - 3y + 1 &> 0 \\
\implies y &> \frac{3+\sqrt{5}}{2} \\
\implies 2^{x-1} &> \frac{3+\sqrt{5}}{2} \\
\implies \boxed{x > \log_2 3 + \sqrt{5}}
\end{aligned}
$$

Thus, we get the same growth rate as expected for a 50-50 split. Thus, we can minimize x based on this inequality. We show a better growth rate for $t_1 = 0.407$

$$
\begin{aligned}
(1-t_1)^x(2^x-2) + (2t_1)^x &< 2^{x-1} - 1 \\
(0.6)^x(2^x-2) + (0.8)^x &< 2^{x-1} - 1 \\
\boxed{x > 2.33093}
\end{aligned}
$$

$\therefore$ We've obtained a regular expression for L_k of length $O(2^{2.331})$. ■

11 Conclusion

Hence, we have a regular expression for L_k of length $O(k^{2.331})$. One might note that this can be improved upon by minimizing for x to get optimal split. Empirical analysis shows us that a $40 - 60$ split was close to optimal.
To obtain shorter regular expressions, one can try to find a smaller base regular expression for r_3 and use that to generalize to r_k using a three-way group split of the alphabet.