CPSC 490 Final Report: Counting Regular Languages

Hui Yang
advised by Timos Antonopoulos and Ruzica Piskac

Abstract

We explore the notion of similarity between regular languages: it may be the case that two regular expressions are radically different yet recognize very similar languages. With this in mind, one may ask how to rigorously define and quickly compute notions of distance between regular languages. A good answer allows us to then attack the problem of approximating two regular expressions: can one quickly compute smaller regular expressions from larger ones that still recognize similar languages?

In this report we focus on the problem of counting the size of a regular language. Such an approach allows us to then rapidly compute a distance metric introduced in previous work whose naive implementation would otherwise be computationally expensive. Specifically, we want to compute the number of strings of a certain length from a given regular expression. We introduce different notions of distinguishability criteria to classify regular expression forms, and prove combinatorial bounds for regular expressions that satisfy such criteria. We first prove an equality for regular expressions in which the letters with kleene star are distinct. We then propose two distinguishability criteria and prove that for regular expressions that satisfy the two distinguishability criteria, the number of strings of a given length satisfy an upper bound. Furthermore, we present algorithms that reduce sufficiently nice regular expressions to ones with form that satisfy these distinguishability criteria. Lastly, we propose two more distinguishability criteria and prove that for regular expressions that satisfy the first two criteria and either of the last two criteria, the upper bound becomes tight. We also provide code samples for generating strings of a regular language.

1 Introduction

Regular languages are one of the simplest models in formal language theory that have captured the curiosity of linguists, computer scientists, and mathematicians alike since its initial inception. A regular expression $r$ is a finite sequence of characters that defines a pattern. This pattern can then be used search and match for strings, which are themselves finite sequences of characters. For instance, consider the regular expression $a(b + c)d^*$. This regular expression matches the following strings

\[
\begin{align*}
\text{ab} & \quad \text{ac} & \quad \text{ab} & \quad \text{ad} & \quad \text{ab} & \quad \text{add} & \quad \text{abdd} & \quad \text{...}
\end{align*}
\]
We say that a regular expression recognizes a string if it can match on the string. In the above example, the regular expression recognizes strings consisting of the character a followed by either b or c, followed by any number of d’s.

The syntax for constructing a regular expression may seem daunting, but the pattern, as we will formalize later, is quite straight-forward. Roughly put, consecutive characters denote string concatenation, parentheses act as grouping delimiters, the plus symbol + denotes a choice of text from a group, while the star symbol * represents a sequence of one more words on the group that is starred, with single characters representing their own group. Additionally, use ε to denote the empty strings, which is a string that contains no characters: we are careful to distinguish this from the empty set. We present a few more examples to illustrate this and acquire some intuition:

\[
\begin{align*}
 r_1 &= abcdedg \\
 L(r_1) &= \{abcdedg\} \\
 r_2 &= (ab)^* \\
 L(r_2) &= \{\varepsilon, ab, abab, ababab, \ldots\} \\
 r_3 &= ((ab)^* + c)d \\
 L(r_3) &= \{d, cd, abd, ababd, abababd, \ldots\}
\end{align*}
\]

The language, or set of strings, that a regular expression \( r \) recognizes is denoted \( L(r) \). It is entirely possible for a language to contain infinitely many strings. We assume the strings of a language to be finite in length, although generalized extensions to infinite length exist [1].

We now consider the question of interest: how can we quantify the similarity between two regular expressions that recognize similar, though possibly not identical, regular languages? To approach this question we extend previous work done along the lines of endowing the space of strings with a probability distribution [5, 4], as well as more generalized spectral techniques [2]. Prior work largely focuses on modelling how embedding may be defined; our primary contributions target the more practical side: we instead turn our attention to combinatorial techniques for counting and bounding the number of strings of a regular language, which is necessary for rapidly computing some of these more abstract notions.

This report is formatted as follows: Section 2 provides a quick background overview of regular expressions and a more in-depth overview is available in [3]; Section 3 first highlights a simple combinatorial result on a very specialized case to illustrate how we may attack the counting problem; Section 4 introduces notions of Type I and Type II distinguishability that generalize away from specialized cases and in addition we prove combinatorial bounds; Section 5 describes algorithms to transform regular expressions containing only stars and letters into those satisfying our distinguishability criterion; Section 6 further introduces Type III and Type IV distinguishability that allow us to tighten previous bounds derived on Type I and Type II distinguishability; Section 7 discusses more generalized regular expressions; and finally we conclude and provide code samples.

## 2 Background

An alphabet \( \Sigma \) is a finite set of distinct symbols (also called characters or letters) from which we can build strings, also known as words. In general, a string \( w \) of length \( n \) is a finite concatenation of characters from the alphabet \( \sigma_1 \sigma_2 \cdots \sigma_n \), where each \( \sigma_j \in \Sigma \).
for $1 \leq j \leq n$ is not necessarily distinct. Denote the length of a string via $|w|$. A set of strings is called a language. The set of all finite strings is denoted $\Sigma^*$.

**Definition 1 (Regular Expression).** A regular expression over the finite alphabet $\Sigma$ and the languages they describe are recursively defined as follows:

- $r, s := \emptyset$ (empty set)
- $\epsilon$ (empty string)
- $a \in \Sigma$ (concatenation)
- $r \cdot s$ (Kleene-closure)
- $r^*$ (alternation (logical OR))
- $r \& s$ (logical AND)
- $\neg r$ (complement)

The set of strings that a regular expression $r$ accepts is denoted $L(r)$, and is called the *regular language* corresponding to $r$, or the language of the regular expression $r$. $L(r)$ can be defined recursively based on the following rules:

- $L(\emptyset) = \emptyset$
- $L(\epsilon) = \{\epsilon\}$
- $L(a) = \{a\}$
- $L(r \cdot s) = \{m \cdot n \mid m \in L(r) \text{ and } n \in L(s)\}$
- $L(r^*) = \{\epsilon\} \cup L(r \cdot r^*)$
- $L(r + s) = L(r) \cup L(s)$
- $L(r \& s) = L(r) \cap L(s)$
- $L(\neg r) = \Sigma^* \setminus L(r)$

In the following sections, we only consider regular expressions consisting of * and letters. Nested operations are not allowed. If a letter in $r$ has * immediately after it, we denote the letter as starred. Otherwise, the letter is unstarred. By construction, $r$ contains only starred and unstarred letters. We assume $r$ has $v$ starred letters and $w$ unstarred letters.

### 3 A Simple Combinatorial Result

We first state and prove a simple combinatorial result on regular expressions that consist of only starred and unstarred letters and all starred letters are distinct. The result will motivate us to explore more generalized cases in later sections.

**Lemma 1.** In a regular expression $r$ with $w$ unstarred letters and $v$ starred letters and all $v$ starred letters being distinct, then $|L(r)^i| = \left(\binom{i-w+v-1}{v-1}\right)$ for $i \geq w$. 

---

3
Proof. To prove the above lemma, we use induction on \( v \).

**Base case:** \( v = 1 \). With only one starred letter in \( r \), it has to repeat \( i - w \) times to take up all the remaining slots not taken up by \( w \) unstarred letters. Thus there is only one string of length \( i \) from \( r \). \(|L(r)| = \binom{i-w}{0} = 1 \) for \( i \geq w \).

**Inductive step:** assume the above hypothesis (lemma 1) is true, we consider a regular expression \( r' \) with \( w \) unstarred letters and \( v + 1 \) starred letters and all \( v + 1 \) starred letters are distinct. We claim \(|L(r')| = \binom{i-w+v}{v} \) for \( i \geq w \).

Without loss of generality, we assume that \( r' \) starts with a starred letter. If this is not the case, let’s assume there are \( k \) unstarred letters before the first starred letters. We consider another regular expression, \( r'' \), that is derived from \( r' \) without the first \( k \) unstarred letter. We claim that the problem, finding the number of strings of length \( i \) from \( r' \), and the problem, finding the number of strings of length \( i - k \), from \( r'' \), are equivalent. The one to one correspondence is obvious. Every string in the first problem can be mapped to a string in the second problem and vice versa. Furthermore, the solution to the first problem is \( \binom{i-w+v}{v} \) and the solution to the second problem is \( \binom{i-k-(w-k)+v}{v} \) according to the hypothesis. Hence we could always assume that \( r' \) starts with a starred letter.

We assume the first starred letter to be \( a \) and it can repeat up to \( i - w \) times because that is the maximum number of available slots. If \( a \) repeat \( j \) times, where \( 0 \leq j \leq i - w \), we consider another regular expression \( r''' \) that is derived from \( r' \) without the first starred letter. We consider the number of strings from \( r''' \) of length \( i - j \). Each of the string formed from \( r''' \) together with \( a \) repeated \( j \) times form a string of length \( i \). Here \( r''' \) has \( w \) unstarred letters, \( v \) starred letters (instead of \( v + 1 \) in the case of \( r' \)). The number of strings from \( r''' \) of length \( i - j \) is \( \binom{i-j-w+v-1}{v-1} \) using the inductive hypothesis.

Before continuing, we need to prove that if the first starred letter appears different times, the resultant strings will be different.

In two strings, \( m, n \) such that in \( m \) the first starred letter, \( a \), appears \( j_1 \) times and in \( n \) the first starred letter appear \( j_2 \) times where \( j_1 \neq j_2 \), then \( m \neq n \). We prove this by contradiction. In \( r''' \), there is no other starred \( a \) since all starred letters are distinct. Assume in \( r''' \), there might be \( l \) unstarred \( a \). Hence in \( m \), the total number of \( a \) will be \( j_1 + l \) and in \( n \), the total number of \( a \) will be \( j_2 + l \). Thus \( j_1 + l \neq j_2 + l \) and \( m \neq n \).

We can then continue to sum over all the different possibilities of \( j \) from 0 to \( i - w \). We have \( \sum_{j=0}^{i-w} \binom{i-j-w+v-1}{v-1} \) and we want to prove that this is equal to \( \binom{i-w+v}{v} \), that is the number of strings for \( r' \). For simplicity, let \( i - w = i' \) and \( v - 1 = v' \). We will use an identity \( \sum_{j=0}^{i} \binom{j}{k} = \binom{i+1}{k+1} \):
\[ \sum_{j=0}^{i-w} \binom{i-j-w+v-1}{v-1} \]

\[ = \sum_{j=0}^{i'} \binom{i'-j+v'}{v'} \]

\[ = \binom{i'+v'}{v'} + \binom{i'+v'-1}{v'} + \binom{i'+v'-2}{v'} + \cdots + \binom{1+v'}{v'} + \binom{v'}{v'} \]

\[ = \binom{i'+v'}{v'} + \binom{i'+v'-1}{v'} + \binom{i'+v'-2}{v'} + \cdots + \binom{1+v'}{v'} + \binom{v'}{v'} + \binom{v'-1}{v'} + \cdots + \binom{0}{v'} \]

\[ = \sum_{j=0}^{i'+v'} \binom{j}{v'} - \sum_{j=0}^{v'-1} \binom{j}{v'} \]

\[ = \binom{i'+v'+1}{v'+1} - \binom{v'-1+1}{v'+1} \]

\[ = \binom{i-w+v-1+1}{v-1+1} - \binom{v-1-1+1}{v-1+1} \]

\[ = \binom{i-w+v}{v} - \binom{v-1}{v} = \binom{i-w+v}{v} \]

This completes the proof lemma 1.

\[ \square \]

4 Type I and Type II of Distinguishability

An extremely restricted class of regular expressions allows us to get exact counting on the number of strings. However, such restrictions may not always be reasonable. In this section we introduce more generalized classifications that still allow us to retain nice combinatorial properties.

Definition 2 (Type I). A pair of starred characters in a regular expression are Type I distinguishable if they are distinct.

Definition 3 (Type II). A pair of starred characters in a regular expression are Type II distinguishable if they are identical, but are separated by at least one unstarred letter distinct from both.

Definition 4 (Distinguishability Criterion). A regular expression is said to satisfy the distinguishability criterion if all pairs of starred characters are either Type I or Type II distinguishable.

Based on this distinguishability criterion, we are able to derive upper bounds on the string counting problem.
Lemma 2. In a regular expression \( r \) with \( w \) unstarred letters and \( v \) starred letters and \( r \) satisfies the distinguishability criterion, then \(|L(r)|^i \leq \binom{i-w+v-1}{v-1} \) for \( i \geq w \).

Proof. \(|L(r)|^i \leq \binom{i-w+v-1}{v-1} \) gives an upper bound for the number of strings of length \( i \). We use induction on \( v \) to prove the lemma.

Base case: \( v = 1 \). \( r \) with only one starred letter trivially satisfy the distinguishability criterion. The starred letter has to repeat \( i - w \) times to take up all the remaining slots not taken up by \( w \) unstarred letters. Thus there is only one string of length \( i \) from \( r \). \(|L(r)|^i = 1 \leq \binom{i-w}{0} = 1 \) for \( i \geq w \).

Inductive hypothesis: assume the above hypothesis (lemma 2) is true, we consider a regular expression \( r' \), with \( v + 1 \) starred letters and \( w \) unstarred letters and \( r' \) satisfies distinguishability criterion. We claim \(|L(r')|^i \leq \binom{i-w+v}{v} \) for \( i \geq w \). For \( r' \) to satisfy the distinguishability criterion, every pair of starred letters in \( r' \) is either Type I or Type II distinguishable. If every pair of starred letters is Type I distinguishable, then all starred letters are distinct and from lemma 1, the the number of strings with length \( i \) from \( r' \) will be \( \binom{i-w+v}{v} \).

Assume there exists at least a pair of starred letters that is Type II distinguishable, then the pair of starred letters is identical. We locate any unstarred letter that is between the pair and distinct from the pair and we break the expression into two parts before and after the letter. Assume the pair of identical starred letter to be \( a^* \) and the distinct unstarred letter to be \( b \). For simplicity, we have \( r' = r_1 a^* r_2 b r_3 a^* r_4 \) where \( r_1, r_2, r_3, r_4 \) are some regular expressions such that the whole \( r' \) satisfies the distinguishability criterion. We thus break \( r' \) into \( r_1 a^* r_2 \) and \( r_3 a^* r_4 \). Assume the first part has \( v_1 \) starred letters and \( w_1 \) unstarred letters and the second part has \( v_2 \) starred letters and \( w_2 \) unstarred letters with \( v_1 + v_2 = v + 1 \) and \( w_1 + w_2 = w - 1 \).

We assume a string formed by \( r' \) is \( m = m_1 \cdot b \cdot m_2 \). \( b \) must appear in \( m \) since \( r' \) contains an unstarred \( b \). We fix the location index of \( b \) to be \( j + 1 \). \( m_1 \) is formed by \( r_1 a^* r_2 \) and \( m_2 \) is formed by \( r_3 a^* r_4 \). Since \( r_1 a^* r_2 \) has \( v_1 \leq v \) starred letters (there is at least one starred letter after \( b \)) and \( w_1 \) unstarred letters, we could find an upper bound on the number of strings of length \( j \) where \( 0 \leq j \leq i \). Similarly, we could find an upper bound on the number of strings of length \( i - 1 - j \) formed by \( r_3 a^* r_4 \). Furthermore, we know the strings formed by \( r_1 a^* r_2 \) are all different and strings formed by \( r_3 a^* r_4 \) are all different, we could multiply the two to get an upper bound on the number of strings of length \( i \) when the location of index \( b \) is fixed at \( j + 1 \). We then sum across different indices of \( j \) from \( 0 \leq j \leq i \) to get an upper bound on the total number of strings of length \( i \) that could be formed by \( r' \). We are getting an upper bound instead of an equality because we could not prove that when the location index of \( b \) is different, all the strings formed are different. Furthermore, when the location index of \( b \) is at \( j + 1 \) and if we know the number of strings of length \( j \) from \( r_1 a^* r_2 \) is \( k_1 \) and the number of strings of length of \( i - 1 - j \) from \( r_3 a^* r_4 \) is \( k_2 \), we know exactly that then number of strings of length \( i \) from \( r' \) with location index of \( b \) being \( j + 1 \) is \( k_1 \times k_2 \). However, there might be cases such that \( m = m_1 \cdot b \cdot m_2 \) where the location index of \( b \) is \( j_1 + 1 \) and \( m' = m_1' \cdot b \cdot m_2' \) where the location index of \( b \) is \( j_2 + 1 \) and \( j_1 \neq j_2 \), but we have \( m = m' \). We will give an counterexample later. Thus an upper bound on the number
of strings of length $i$ from $r'$ will be
\[
\sum_{j=0}^{i} \binom{j-w_1+v_1-1}{v_1-1} \times \binom{i-1-j-w_2+v_2-1}{v_2-1}
\]
and we want to show that
\[
\sum_{j=0}^{i} \binom{j-w_1+v_1-1}{v_1-1} \times \binom{i-1-j-w_2+v_2-1}{v_2-1}
\]
\[
= \binom{i-(w_1+w_2+1)+v_1+v_2-1}{v_1+v_2-1}
\]
\[
= \binom{i-w+v}{v}
\]
To simplify the proof for the above identity, we consider the following change of variables:
\[
x_1 = v_1 - 1, x_2 = v_2 - 1.
\]
The above equation becomes:
\[
\sum_{j=0}^{i} \binom{j-w_1+x_1}{x_1} \times \binom{i-1-j-w_2+x_2}{x_2}
\]
\[
= \binom{i-w_1-w_2-1+x_1+1+x_2+1-1}{x_1+x_2+1}
\]
Next we realize in the summation, when $j < w_1$, $(\frac{j-w_1+x_1}{x_1}) = 0$ and when $j > i-1-w_2$, $(\frac{i-1-j-w_2+x_2}{x_2}) = 0$. Hence the above summation simplifies to be:
\[
\sum_{j=w_1}^{i-1-w_2} \binom{j-w_1+x_1}{x_1} \times \binom{i-1-j-w_2+x_2}{x_2}
\]
\[
= \binom{i-w_1-w_2-1+x_1+1+x_2+1-1}{x_1+x_2+1}
\]
We use again a change of variable: $k = j - w_1$ and the above equation becomes:
\[
\sum_{k=0}^{i-1-w_1-w_2} \binom{k+x_1}{x_1} \times \binom{i-1-w_1-w_2-k+x_2}{x_2}
\]
\[
= \binom{i-w_1-w_2-1+x_1+1+x_2+1-1}{x_1+x_2+1}
\]
We use another change of variable: $t = i-1-w_1-w_2$ and the above equation becomes:
\[
\sum_{k=0}^{t} \binom{k+x_1}{x_1} \times \binom{t-k+x_2}{x_2} = \binom{t+x_1+1+x_2}{x_1+x_2+1}
\]
We use two identities in Mathematics.
\[
\binom{n}{k} = (-1)^k \binom{k-n-1}{k}, \quad \sum_{k=0}^{t} \binom{r}{k} \binom{s}{t-k} = \binom{r+s}{t}
\]
The first identity is the definition of binomial coefficient for negative integers while the second identity is known as Chu-Vandermonde identity. We will not prove the two identities here.

\[
\sum_{k=0}^{t} \binom{k + x_1}{x_1} \times \binom{t - k + x_2}{x_2} = \binom{t + x_1 + 1 + x_2}{x_1 + x_2 + 1}
\]

is the same as

\[
\sum_{k=0}^{t} \binom{k + x_1}{k} \times \binom{t - k + x_2}{t - k} = \binom{t + x_1 + 1 + x_2}{t}
\]

since \( \binom{a}{b} = \frac{a!}{b!(a-b)!} = \binom{a}{a-b} \). Using the first identity, we have

\[
\binom{k + x_1}{k} = (-1)^k \binom{k - (k + x_1) - 1}{k} = (-1)^k \binom{-x_1 - 1}{k}
\]

and

\[
\binom{t - k + x_2}{t - k} = (-1)^{t-k} \binom{t - k - (t - k + x_2) - 1}{t - k} = (-1)^{t-k} \binom{-x_2 - 1}{t - k}
\]

Thus

\[
\sum_{k=0}^{t} \binom{k + x_1}{k} \times \binom{t - k + x_2}{t - k}
\]

\[
= \sum_{k=0}^{t} (-1)^k \times \binom{-x_1 - 1}{k} \times (-1)^{t-k} \times \binom{-x_2 - 1}{t - k}
\]

\[
= \sum_{k=0}^{t} (-1)^t \binom{-x_1 - 1}{k} \times \binom{-x_2 - 1}{t - k}
\]

\[
= (-1)^t \sum_{k=0}^{t} \binom{-x_1 - 1}{k} \times \binom{-x_2 - 1}{t - k}
\]

\[
= (-1)^t \binom{-x_1 - x_2 - 2}{t}
\]

and

\[
\binom{t + 1 + x_1 + x_2}{t} = (-1)^t \binom{t - (t + 1 + x_1 + x_2) - 1}{t} = (-1)^t \binom{-x_1 - x_2 - 2}{t}
\]

This completes the proof of Lemma 2. □

As we will also see later, Type I and Type II distinguishability are general enough in the sense that regular expressions with only stars and letters can be transformed to satisfy them.
5 Transforming to Distinguishable Expressions

We define a flowchart and two algorithms that transform a given regular expression $r$ to a set of regular expressions $\{r_i\}$ that satisfy the distinguishability criteria.

![Flowchart]

Algorithm 1 Reduction

1: create a dictionary $\text{star}$
2: for every distinct starred letter, $a$, in $r$
3: \text{star} stores the location indices of all incidences $a^*$ in $r$
4: for every key in $\text{star}$
5: $\text{star}[\text{key}]$ gives all location indices of starred letter $\text{key}$
6: for every pair elements, $j, k$ of $\text{star}[\text{key}]$
7: \text{if} (in $r$ between location $j, k$ (exclusive), all letters are unstarred and
8: identical to $\text{key}$)
9: then
10: \{remove the starred letter in location $k$\}
We continuously run the reduction algorithm on \( r \) until no further changes, finally we get \( r' \). We claim that the number of strings of length \( i \) from \( r \) is the same as the number of strings of length \( i \) from \( r' \). To prove this, we note every string that could be generated by \( r \) could also be generated by \( r' \) and vice versa.

The algorithm essentially finds a pattern of \( a^*aa...aaa^* \) in \( r \) (there are zero or more than one identical unstarred letter in between) and remove the second \( a^* \). Here, we only prove that in \( r = ...a^*a...aa^*... \) and \( r' = ...a^*a...a..., \) the number of strings of length \( i \) will be the same. For clearer notation, we define \( r = r_1a^*r_2a^*r_3 \) and \( r' = r'_1a^*r'_2r'_3 \) where \( r_1 \) and \( r_3 \) are any regular expressions with only starred and unstarred letters and \( r_2 \) is a sequence of \( k \) a where \( 0 \leq k \).

To prove that in \( r \) and \( r' \), the number of strings with length \( i \) are the same, we prove that every string that is generated by \( r \) could also be generated by \( r' \) and vice versa. We consider a string, \( m \) that is generated by \( r = r_1a^*r_2a^*r_3 \). We could decompose \( m \) into \( m_1 \cdot m_2 \cdot m_3 \) where \( m_1, m_2, m_3 \) are sub-strings of \( m \) and after concatenation, \( m = m_1 \cdot m_2 \cdot m_3 \). We define \( m_1, m_2, m_3 \) in such a way that \( m_1 \) is generated by \( r_1 \), \( m_2 \) is generated by \( a^*r_2a^* \) and \( m_3 \) is generated by \( r_3 \). By this construction, we know immediately that this string can also be formed by \( r' = r_1a^*r_2r_3 \). Here the only thing we need to prove is that \( m_2 \) needs to be generated by \( a^*r_2 \). Since \( m_2 \) is generated by \( a^*r_2a^* \) and \( r_2 \) is a sequence of \( k \) a, \( m_2 \) is a sequence of \( k' \) a where \( k' \geq k \geq 0 \). It is trivial that \( m_2 \) is generated by \( a^*r_2 \).

The above proof shows that finding a pattern of \( a^*a...aa^* \) and remove the second \( a^* \) does not change the number of strings of length \( i \). The reduction algorithm finds all such patterns in \( r \) and reduce each instance.

After \( r \) passes through the reduction algorithm with no change, the construction algorithm inspects for another pattern. First of all, we prove that in the 10\textsuperscript{th} line, the identified first starred letter between \( j, k \) location will be different from the \textbf{key}. If this is not the case, then assume the first starred letter to be \textbf{key} and it has location index \( j_1 \). Assume \textbf{key} is \( a \). Then between \( j, j_1 \), there are only unstarred letters and they are all unstarred \( a \). Furthermore, the starred letters at \( j \) and \( j_1 \) are both \( a^* \). Then this section of the \( r \) will be \( a^*a...aa^* \), a contradiction of the result of reduction algorithm.

---

**Algorithm 2 Construction**

1. create a dictionary \textbf{star}
2. for every distinct starred letter, \( a \), in \( r \)
3. \textbf{star} stores the location indices of all incidences \( a^* \) in \( r \)
4. for every \textbf{key} in \textbf{star}
5. \textbf{star}[\textbf{key}] gives all location indices of starred letter \textbf{key}
6. for every pair elements, \( j, k \) of \textbf{star}[\textbf{key}]
7. if (in \( r \) between location \( j, k \) (exclusive), all unstarred letters are identical to \textbf{key})
8. then
9. \{identify the first starred letter between \( j,k \) and assume it is \( b^* \) and
10. form \( r_1, r_2 \)
11. \( r_1 \) is from original \( r \) after removing the identified \( b^* \)
12. \( r_2 \) is from original \( r \) after inserting an unstarred \( b \) in front of \( b^* \)}
This shows that the identified first starred letter must be distinct from key.

For simplicity, we represent \( r = r_3 a^* r_4 b^* r_5 a^* r_6 \) where \( r_3, r_6 \) are any regular expressions with only starred and unstarred letters. \( r_4 \) is a sequence of (zero or more) \( a \). \( r_5 \) contains some starred letters and unstarred \( a \). The construction algorithm essentially finds all such patterns and creates two new regular expressions: \( r_1 = r_3 a^* r_4 r_5 a^* r_6 \) and \( r_2 = r_3 a^* r_4 b b^* r_5 a^* r_6 \). We claim the number of strings of length \( i \) from \( r \) is the same as the sum of the number of strings of length \( i \) from \( r_1 \) and \( r_2 \). To prove this, we demonstrate that 1) every string that is generated by \( r \) could be generated by either \( r_1 \) or \( r_2 \); 2) every string that is generated by \( r_1 \) is also generated by \( r \); 3) every string that is generated by \( r_1 \) is also generated by \( r \); 4) strings generated by \( r_1 \) and \( r_2 \) are distinct.

We prove (1) and (4) here. (2) and (3) are proved similarly. Assume a string, \( m \), is generated by \( r = r_3 a^* r_4 b^* r_5 a^* r_6 \). \( m \) could be decomposed to \( m_1 \cdot m_2 \cdot m_3 \cdot m_4 \cdot m_5 \cdot m_6 \cdot m_7 \) where \( m_1 \) is generated by \( r_3 \), \( m_2 \) is generated by \( a^* \) and so on. We claim \( m \) could be generated by either \( r_1 \) or \( r_2 \). If \( m_4 \) is \( \epsilon \), then \( m \) is generated by \( r_1 \); otherwise, \( m \) is generated by \( r_2 \).

In the former case, \( m_4 \) is \( \epsilon \) and we have \( m = m_1 \cdot m_2 \cdot m_3 \cdot m_5 \cdot m_6 \cdot m_7 \) where each sub-string is generated by corresponding parts in \( r_1 \). In the later case, \( m_4 \) is generated by \( b^* \) in \( r \) and is not \( \epsilon \), implying that \( m_4 \) is a sequence of (one or more) \( b \). It is thus generated by \( b b^* \) in \( r_2 \). This completes the proof that strings generated by \( r \) could be generated by either \( r_1 \) or \( r_2 \).

We know strings generated by \( r_1 \) and \( r_2 \) are distinct because strings generated by \( r_1 \) do not contain any \( b \) between \( m_1 \cdot m_2 \cdot m_3 \) generated by \( r_3 a^* r_4 \) and \( m_5 \cdot m_6 \cdot m_7 \) generated by \( r_5 a^* r_6 \). On the other hand, strings generated by \( r_2 \) contains a sequence of \( b \) between \( m_1 \cdot m_2 \cdot m_3 \) generated by \( r_3 a^* r_4 \) and \( m_5 \cdot m_6 \cdot m_7 \) generated by \( r_5 a^* r_6 \). Thus we know that the number of strings generated by \( r \) is the same as the sum of the number of strings generated by \( r_1 \) and \( r_2 \).

Given any regular expression \( r \), we execute the above flowchart and get a family of regular expressions \( \{ r_i \} \). However, we need to prove that the flowchart terminates. The reduction process will terminate since at every step, the number of starred letters reduces by one and there are finite number of starred letters in the input. Hence there will not be any regular expressions that are trapped at the self-loop at reduction node forever.

The case for construction process is slightly more complicated. A change happens at construction step when two new regular expressions are formed. This occurs if there exists a pair of identical starred letters with all unstarred letters in between them being identical to the starred letter and there exists at least one more distinct starred letter. For simplicity, we again represent \( r = r_3 a^* r_4 b^* r_5 a^* r_6 \) where \( r_3 \) and \( r_6 \) are any regular expressions containing only starred and unstarred letters and \( r_4 \) is a sequence of (zero or more) \( a \) and \( r_5 \) contains some starred letters and unstarred \( a \). Assume the first \( a^* \) is at location index \( j \) and the second \( a^* \) is at location index \( k \). Only if a regular expression of such pattern will change in construction step and form \( r_1 = r_3 a^* r_4 r_5 a^* r_6 \) and \( r_2 = r_3 a^* r_4 b b^* r_5 a^* r_6 \). We note for \( r \to r_2 \), between \( a^* \) at location \( j, k \) (or probably \( k+1 \) since we inserted a new \( b \)), not all unstarred letters are \( a \) since we have inserted a \( b \). Hence the construction algorithm would not need to examine this particular pair of identical starred letters. Since a regular expression has finite number of starred letters
and thus finite pairs of identical starred letters, this branch of construction step will terminate. We note for $r \rightarrow r_1$, between $a^*$ at location $j,k$ (or probably $k−1$ since we removed $b^*$), the number of starred letters reduce by one. Since there are finite number of starred letters in between two $a^*$, all starred letters will be removed and we will eventually have $r_1' = r_3a^*r_7a^*r_6$ where $r_7$ is a sequence of zero or more $a$. $r_1'$ will change in reduction step and form $r' = r_3a^*r_7r_6$. This branch will also terminate since the construction algorithm could now move on to examine other pairs of identical starred letters. Hence the flowchart will terminate and output a a family of regular expressions $\{r_i\}$

I claim that for any $r_i$ in $\{r_i\}$ from $r$, all pairs of starred letters satisfy either Type I or Type II distinguishability criteria. We analyze the different cases. For any pair of starred letters in $r_i$, they are either different or identical. If they are different, then they satisfy Type I distinguishability. If they are identical, either they are separated by at least one unstarred letter that is different or not. In the former case, they satisfy Type II distinguishability. We consider what if they are identical and all unstarred letters are also the same as the starred letters. This contradicts the result of the construction algorithm and such regular expressions will not be generated as the end product of the flowchart.

We now give an upper bound for any regular expression $r$ of only starred and unstarred letters.

**Lemma 3.** For any regular expression $r$ consisting of starred and unstarred letters, we run the flowchart and get a set of $n$ final $\{r_j\}$ and each $r_j$ satisfies the distinguishability criterion. The set of $\{r_j\}$ has $\{v_j\}$ starred letters and $\{w_j\}$ unstarred letters each. The number of strings of length $i$ from $r$ is $|L(r)|^i \leq \sum_{j=1}^{n} \binom{t−w_j+w_j−1}{v_j−1}$.

The result follows immediately from the previous discussion and Lemma 2.

### 6 Type III and Type IV Distinguishability

In this section, we introduce two more distinguishability criteria that are stronger than Type I and Type II but weaker than the all-starred-letters-are-distinct criterion. We would be able to prove an equality on the number of strings of length $i$.

A block of starred letters is a consecutive sequence of letters such that all letters are starred while a block of unstarred letters is a consecutive sequence of letters such that all letters are unstarred. A regular expression consisting of only starred and unstarred letters could be decomposed into a sequence of alternate blocks of starred and unstarred letters. Again without loss generality, we assume $r$ starts with a block of starred letters by a similar argument as before. Hence $r = g_1f_1g_2f_2...g_7f_7g_{t+1}$ where $g_j$ is a block of starred letters while $f_j$ is a block of unstarred letters.

**Definition 5 (Type III).** 1) A regular expression $r = g_1f_1g_2f_2...g_tf_tg_{t+1}$ is Type III distinguishable if 1) for $1 \leq j \leq t$, $f_j \notin L(g_j)$ 2) for $1 \leq j \leq t$ and for $1 \leq k \leq |f_j|−1$, if $f_j[[:]k] = f_j[[:]−k]$; $f_j[[:]−k] \notin L(g_j)$ where $f_j[[:]k]$ is the first $k$ letters in $f_j$ and $f_j[[:]−k]$ is the last $k$ letters in in $f_j$ and $f_j[[:]−k]$ is the first $|f_j|−k$ letters in $f_j$.

**Definition 6 (Type IV).** 1) A regular expression $r = g_1f_1g_2f_2...g_tf_tg_{t+1}$ is Type IV distinguishable if 1) for $1 \leq j \leq t$, $f_j \notin L(g_j+1)$ 2) for $1 \leq j \leq t$ and for $1 \leq k \leq$
$|f_j| - 1$, if $f_j[: k] = f_j[-k :]$, $f_j[k :] \notin L(g_{j+1})$ where $f_j[: k]$ is the first $k$ letters in $f_j$ and $f_j[-k :]$ is the last $k$ letters in $m$ and $f_j[k :]$ is the last $|f_j| - k$ letters in $f_j$.

Given a regular expression $r$ that satisfies distinguishability criterion that we defined earlier on, we claim that if $r$ is either Type III distinguishable or Type IV distinguishable, then the number of strings of length $i = (i^{-w+v-1})$ where $v$ is the number of starred letters while $w$ is the number of unstarred letters in $r$.

Before proceeding to prove our result, we first prove a lemma that helps us in later proof.

**Lemma 4.** If a string $m$ is generated by $r$ consisting of only starred distinct letters, then any substrings of $m$ could also be generated by $r$.

**Proof.** We prove an even stronger version of this, that is if a string $m$ is generated by $r$ consisting of only starred distinct letters, then any subsequence of $m$ could also be generated by $r$.

A string $m$ that is generated by $r$ of only starred distinct letters will have the form $a_1g_{1a_1}...a_2g_{2a_2}...a_ng_{n}...$ where $a_i \neq a_j$ if $i \neq j$ and each $a_j$ repeated zero times or more than once and assume $r$ has $n$ distinct starred letters. Assume each $a_j$ appears $k_j$ times where $0 \geq k_j$ and might be infinite. A subsequence by $m$ will take the form $m' = a_1a_1...a_2a_2...a_3a_3...a_na_n...$ where each $a_j$ appears $l_j$ times where $0 \geq l_j \geq k_j$. It is clear that $m'$ could also be generated by $r$. Thus any subsequence of $m$ could also be generated by $r$. Since a substring is a subsequence (a result that we will not prove here), any substrings of $m$ could also be generated by $r$. \qed

**Lemma 5.** A regular expression $r$ with $w$ unstarred letters and $v$ starred letters and $r$ satisfies the distinguishability criterion and $r$ could be decomposed into $r = g_1f_1g_2f_2...g_tf_tg_{t+1}$ where $g_j$ is a block of starred letters while $f_j$ is a block of unstarred letters. If $r$ is either type III distinguishable or type IV distinguishable, then $|L(r)_i| = (i^{-w+v-1})$ for $i \leq w$.

**Proof.** We use induction on the number of blocks $t$ to prove the lemma. We only give the proof of when $r$ is Type III distinguishable. The proof for Type IV distinguishability is similar.

**Base case:** $t = 1$ and $r = g_1f_1g_2$ and $f_1 \notin L(g_1)$ and for $1 \leq k \leq |f_1| - 1$, if $f_1[: k] = f_1[-k :]$, $f_1[: -k] \notin L(g_1)$. I first prove that in $g_1$, all letters are starred and distinct. All letters are starred because by definition $g_1$ is a block of starred letters. Assume some letters in $g_1$ are identical and assume one pair of identical letter to be $a$. Then we have $g_1 = r_1a^*r_2a^*r_3$ where $r_1, r_2$ and $r_3$ are all blocks of starred letters and they could be the empty string. This is in contradiction with the fact that $r$ satisfies the previously defined distinguishability criterion. If $r_2 = \epsilon$, then the reduction algorithm would reduce $g_1$ to be $r_1a^*r_3$. If $r_2 \neq \epsilon$, then again the pair of starred $a$ are not separated by at least one unstarred distinct letter. Hence we could never have a pair of identical starred letters in $g_1$. Thus all letters in $g_1$ are starred and distinct. By a similar proof, all letters in $g_2$ are starred and distinct. Assume $g_1$ has $v_1$ letters and $g_2$ has $v_2$ letters and $v_1 + v_2 = v$.

Hence $f_1$ are the only letters in $r$ that are unstarred and $|f_1| = w$. We consider the process of forming strings of length $i$. Since $f_1$ is unstarred, $f_1$ will appear somewhere
and appear exactly once. Assume $f_1$ takes up the position from index $j + 1$ to $j + w$ (without further notice, we assume the initial and ending position are inclusive), so $g_1$ will form a string of length $j$ and $g_2$ will form a string of length $i - j - w$. From Lemma 4.1, the number of strings of length $j$ for $g_1$ is \((j - 0w)^{v_1-1}\) and the number of strings of length $i - j - w$ for $g_2$ is \((i - j - w - 0w)^{v_1-1}\). The number of strings of length $i - j - w$ from index $j + 1$ to $j + w$ is \((j - 0w)^{v_1-1}\) \times \((i - j - w - 0w)^{v_1-1}\) \times \((j + v_1-1)^{-2j}\).

We then consider all the possible position that $f_1$ could take up. The first letter of $f_1$ could take the position from 1, 2, 3 until $i + w + 1$. We want to sum up all the possibilities. For this to to valid, we need to prove that a string $m_1$ formed when $f_1$ takes the position from index $j + 1$ to $j + w$ and another string $m_2$ formed when $f_1$ takes the position from index $j + 2$ to $j + w + 1$ will be different if $j_1 \neq j_2$. We prove this by contradiction. Assume we could find two such strings $m_1$ and $m_2$ such that $m_1 = m_2$ even if $j_1 \neq j_2$. With loss of generality, we assume $j_2 > j_1$ and we define $k = |f_1| - (j_2 - j_1)$. Now we consider two cases where $k \leq 0$ and $k > 0$. First consider $k > 0$. Consider the sequence in both strings from position $j + 1$ to $j + w$ (this is valid because $j_1 + w - (j_2 + 1)+1 = j_1 - j_2 + w = j_1 - j_2 + |f_1| = |f_1| - (j_2 - j_1) = k > 0$). When we count the number of letters, we use the formula ending position - initial position + 1. $f_1$ in $m_1$ takes up the position from index $j + 1$ to $j + w$ and thus sequence from position index of $j_2 + 1$ to $j_1 + w$ is the last $k$ letters of $f_1$, also represented by $f_1[-k]$. Also we note $k < w$ since $w - (j_2 - j_1)$ and $j_2 > j_1$. Similarly, $f_1$ in $m_2$ takes up the position from index $j_2 + 1$ to $j_2 + w$ and thus the sequence from index $j_2 + 1$ to $j_1 + w$ is the first $k$ letters of $f_1$, also represented by $f_1[+k]$. Again we note $0 < k < w$. For $m_1 = m_2$, then the sequence from index of $j + 1$ to $j + w$ must be equal in both $m_1$ and $m_2$. Thus $f_1[+k] = f_1[-k]$. However we know from the definition of Type III distinguishability criterion, at all such $k$ when $f_1[+k] = f_1[-k]$ occurs, $f_1[-k] \notin L(g_1)$. Consider $m_2$, we represent $m_2 = m_2 \cdot f_1 \cdot m_3$ where $m_3$ is generated by $g_1$ and $m_4$ is generated by $g_2$. Then consider the sequence of letters from index $j_1 + 1$ to $j_2$ in $m_2$, it is the same as the first $w - k$ letters in $f_1$, also represented as $f_1[-k]$. We know this because in $m_1$ the sequence of letters from index $j_1 + 1$ to $j_2$ is the first $w - k$ letters and $m_1 = m_2$. We note $j_2 - (j_1 + 1) + 1 = j_2 - j_1 = w - k$ since $k = w - (j_2 - j_1)$. Here, we have a contradiction since in $m_2$, the substring $m_3$ must be generated by $g_1$. But a substring of $m_3$, that is the letters from position index from $j_1 + 1$ to $j_2$ in $m_2$, also the last $w - k$ letters in $m_3$, also the letters from position index from $j_1 + 1$ to $j_2$ in $m_1$, also the first $w - k$ letters in $f_1$, also represented by $f_1[-k]$ cannot be generated by $g_1$ by definition of Type III distinguishability criterion. This is valid contradiction because by Lemma 4, if a string could be generated by $r$ of only distinct starred letters, all its substring could also be generated by $r$. The fact that one substring could not be generated by $r$ implies that $m_3$ could not be generated by $g_1$, making $m_1 \neq m_2$ by contraposition.

Now we need to consider the case $k \leq 0$ - implying overlap of $m_1$ and $m_2$, $k = w - (j_2 - j_1)$ and $-k = j_2 - j_1 - w = j_2 - 1 - (j_1 + w) + 1 \geq 0$ and we let $k' = -k$ and $k'$ represents the sequence of letters from index $j_1 + w + 1$ to $j_2$ in $m_1$ and $m_2$. Again let $m_2 = m_3 \cdot f_1 \cdot m_4$ and $m_3$ must be generated by $g_1$. $m_3$ represents the substring of index from 1 to $j_2$. In $m_1$ for the substring of index from 1 to $j_2$, we consider its substring of index from $j_1 + 1$ to $j_2 - k' = j_2 - (j_2 - 1 - (j_1 + w) + 1) = j_2 - (j_2 - j_1 - w) = j_1 + w,$
essentially \( f_1 \) (\( f_1 \) takes up the position from \( j_1 + 1 \) to \( j_1 + w \)). However, \( f_1 \) could not be generated by \( g_1 \). Thus by Lemma 4 and by contraposition, we have a contradiction and \( m_1 \neq m_2 \). This successfully proves that when \( f_1 \) takes up different positions index from \( j_1 + 1 \) to \( j_1 + w \), the strings formed can never be the same. This allows us to sum up all the possibilities.

We have

\[
\sum_{j=0}^{i-w} \binom{j+v_1-1}{v_1-1} \times \binom{i-j-w+v_2-1}{v_2-1}
\]

Using \( i' = i - w \), we have

\[
\sum_{j=0}^{i'} \binom{j+v_1-1}{v_1-1} \times \binom{i'-j+v_2-1}{v_2-1}
\]

We use the two Mathematics identities before:

\[
\binom{n}{k} = (-1)^k \binom{k-n-1}{k}
\]

\[
\sum_{k=0}^{t} \binom{r}{k} \binom{s}{t-k} = \binom{r+s}{t}
\]

\[
\binom{j+v_1-1}{v_1-1} = \binom{j+v_1-1}{j} = (-1)^j \binom{j-(j+v_1-1)-1}{j} = (-1)^j \binom{-v_1}{j}
\]

\[
\binom{i'-j+v_2-1}{v_2-1} = \binom{i'-j+v_2-1}{i'} = (-1)^{i'-j} \binom{i'-j-(i'-j+v_2-1)-1}{i'} = (-1)^{i'-j} \binom{-v_2}{i'-j}
\]
\[
\sum_{j=0}^{i'} (-1)^j \binom{-v_1}{j} \times (-1)^{i'-j} \binom{-v_2}{i'-j} \\
= \sum_{j=0}^{i'} (-1)^j \binom{-v_1}{j} \times \binom{-v_2}{i'-j} \\
= (-1)^{i'} \binom{-v_1 - v_2}{i'} \\
= (-1)^{i-w} \binom{-v_1 - v_2}{i-w}
\]

\[
\left( i - w + v - 1 \right)\left( v - 1 \right) = \left( i - w + v - 1 \right)\left( i - w \right) \\
= (-1)^{i-w} \left( i - w - (i - w + v - 1) - 1 \right) \\
= (-1)^{i-w} \binom{-v}{i-w} \\
= (-1)^{i-w} \binom{-v_1 - v_2}{i-w}
\]

This completes the base case proof.

**Induction hypothesis:** assume lemma 5 is true for a regular expression \( r = g_1 f_1 g_2 f_2 \ldots g_t f_t g_{t+1} \) that is Type III distinguishable. Consider another regular expression \( r' = g_1 f_1 g_2 f_2 \ldots g_t f_t g_{t+1} f_{t+1} g_{t+2} \) that is Type III distinguishable, we claim the number of strings of length \( i \) over \( r \) is \( |L(r')| = \binom{i-w+v-1}{v-1} \) for \( i \geq w \) where \( v \) is the number of starred letters in \( r' \) and \( w \) is the number of unstarred letters in \( r' \).

To prove this, we have \( r' = g_1 f_1 r_2 \) where \( r_2 = g_2 f_2 \ldots g_t f_t g_{t+1} f_{t+1} g_{t+2} \). Assume \( |f_1| = w_1 \leq w \) and \( |g_1| = v_1 \leq v \). Thus \( r_1 \) has only starred and unstarred letters and is Type III distinguishable and has \( v - v_1 \) starred letters and \( w - w_1 \) unstarred letters.

Here, again the key is the prove that as \( f_1 \) takes up position of index from \( j + 1 \) to \( j + w_1 \), the strings formed will be different for different \( j \). The number of strings of length \( j \) for \( g_1 \) will be \( \binom{j-v_1-1}{v_1-1} \) and the number of strings of length \( i - w_1 - j \) for \( r_1 \) will be \( \binom{i-w_1-j-w_2+v_2-1}{v_2-1} \) by inductive hypothesis. The argument for why, string \( m_1 \) formed from \( r' \) when \( f_1 \) takes up the position \( j_1 \) to \( j_1 + w_1 \) and string \( m_2 \) formed from \( r' \) when \( f_1 \) takes up the position \( j_2 \) to \( j_2 + w_1 \) will be different if \( j_1 \neq j_2 \), is extremely
similar to the proof in the base case. Hence we will not elaborate here.

This is exactly the same calculation as before. This completes the proof for induction.

The proof for Type IV distinguishability criterion is similar to the proof with Type III distinguishability criterion and we will not elaborate here. Either distinguishability criterion gives us the desired equality.

7 More Generalized Regular Expressions

We consider a slightly more complicated problem: a regular expression of starred and unstarred letters and + (concatenation). Assume a regular expression \( r = r_1 + r_2 + \ldots + r_n \) where \( r_j \) consists of only starred and unstarred letters. We use lemma 4.3 on each \( r_j \) to find an upper bound and \( |L(r)|^i \leq \sum_{j=1}^n |L(r_j)|^i \). You might expect us to do better than this, but that is extremely difficult. We consider an example where \( n = 2 \) so \( r = r_1 + r_2 \). Due to set properties, we have either \( L(r)^i = L(r_1)^i + L(r_2)^i - L(r_1 \cap r_2)^i \) or \( L(r)^i = L(r_1)^i + L(r_1 \setminus r_2)^i \) where \( r_1 \cap r_2 \) is the intersection of \( r_1 \) and \( r_2 \) and \( r_1 \setminus r_2 \) is the complement of \( r_2 \) in \( r_1 \) and both are regular expressions.

There are well-known algorithms to find \( r_1 \cap r_2 \) and \( r_1 \setminus r_2 \) using Deterministic Finite Automata (DFAs) and Non-deterministic Finite Automata (NFAs). However the resultant regular expressions are usually complicated, involving + and () and making it impossible to apply the prior formula.

We consider another slightly more complicated problem: regular expressions of starred and unstarred letters and (). For simplicity, we do not allow nested stars i.e. \((a^*b)^*\). To be more specific, star either appears directly after a letter, or after a pair of parentheses within which there is a block of unstarred letters. Furthermore, we assume whenever there is a pair of parentheses, a star always follows. We try to solve for a very crude upper bound. Assume we are interested in strings of length \( i \). In the regular expression, a letter could be a singlet if it is not within a pair of parentheses, otherwise it is in a block. For each block of letters enclosed by a pair of parentheses,
let the number of letters in the block be $k_j$ and there are $p$ of them. For letters of singlets, assume there are $w$ unstarred letters and $v$ starred letters. The number of strings of length $i$ from the expression is bounded by the number of non-negative integer solutions to the equation $k_1x_1 + k_2x_2 + \ldots + k_px_p + y_1 + \ldots + y_v + w = i$ where $x_1, \ldots, x_p, y_1, \ldots, y_v$ are non-negative integers. There is no elegant formula for the problem.

8 Conclusion

We presented a method to find the number of strings of given length from a regular expression. We showed that if a regular expression $r$ consists of only starred and unstarred letters and all starred letters are distinct, the number of strings of length $i$ formed from $r$ is $(i - w + v - 1) v^{-1}$ where $v$ is the number starred letters and $w$ is the number of unstarred letters in $r$.

We then suggested a framework to produce a family of regular expressions $\{r_i\}$ from any given regular expression $r$ consisting of only starred and unstarred letters such that each $r_i$ satisfies a distinguishability criterion that we defined earlier. We show that each $r_i$ has an upper bound given by $(i - w + v - 1) v^{-1}$.

Lastly, we further proposed Type III distinguishability and Type IV distinguishability criterion and claim that $r$ satisfying either one will satisfy the equality that the number of strings of length $i$ is $(i - w + v - 1) v^{-1}$.

In the future, we hope to study more generalized regular expressions, consisting of not only starred and unstarred letters but also other operators, including nested operators. We hope to propose a formula to count the number of strings of a certain length quickly.

9 Code and Examples

We now give two examples. The first one satisfies the equality while the second one satisfies the strict inequality.

We consider $r = b^*a^*b^*ba^*ab^*$. It passes through the reduction process without change. There are two instances of $a^*$ and between them there is unstarred $b$. There are three instances of $b^*$. Between the first and third, and the second and the third, there is an unstarred $a$. However, between the first and second, there is no unstarred letter that is distinct. Thus we create $r_1 = b^*b^*ba^*ab^*$ which becomes $r_1 = b^*ba^*ab^*$ in the reduction process. We also have $r_2 = b^*aa^*b^*ba^*ab^*$ which does not change in the reduction process. We use a python program to find all the strings of length 9 generated from $r, r_1, r_2$.

$r = b^*a^*b^*ba^*ab^*$ first gives 330 strings by decomposing $9 - 2 = 7$ into the sum of 5 different integers. After removing double count, we have 246 distinct strings of length 9 from $r$.

$r_1 = b^*ba^*ab^*$ satisfies the distinguishability criterion and generates $36 = \binom{9-2+3-1}{3-1}$ distinct strings. All strings generated are attached at the end of the report.

$r_2 = b^*aa^*b^*ba^*ab^*$ satisfies the distinguishability criterion and generates $210 = \binom{9-3+5-1}{5-1}$ distinct strings. All strings generated are attached below.

18
In total, the number of strings generated by $r$ is the same as the sum of the number of strings generated by $r_1$ and $r_2$.

Now we give another counter-example to show that the equality does not hold all the time. Consider $r = a^*baba^*b^*$. It satisfies the distinguishability criterion. However, it generates a total of $9 < \binom{4}{2} + \binom{4}{1} = 10$ distinct strings of length 4. Hence we have a strict inequality. This is because there is a double counting of string $abab$, which could be generated by $a^*b^*ab$ or $aba^*b^*$.

Strings of length 9 generated by $b^*ba^*ab^*$:

\[
\begin{align*}
&\text{Strings of length 9 generated by } b^*ba^*ab^*:
\end{align*}
\]

In total, the number of strings generated by $r$ is the same as the sum of the number of strings generated by $r_1$ and $r_2$.

Now we give another counter-example to show that the equality does not hold all the time. Consider $r = a^*baba^*b^*$. It satisfies the distinguishability criterion. However, it generates a total of $9 < \binom{4}{2} + \binom{4}{1} = 10$ distinct strings of length 4. Hence we have a strict inequality. This is because there is a double counting of string $abab$, which could be generated by $a^*b^*ab$ or $aba^*b^*$.

Strings of length 9 generated by $b^*ba^*ab^*$:

\[
\begin{align*}
&\text{Strings of length 9 generated by } b^*ba^*ab^*:
\end{align*}
\]

In total, the number of strings generated by $r$ is the same as the sum of the number of strings generated by $r_1$ and $r_2$.

Now we give another counter-example to show that the equality does not hold all the time. Consider $r = a^*baba^*b^*$. It satisfies the distinguishability criterion. However, it generates a total of $9 < \binom{4}{2} + \binom{4}{1} = 10$ distinct strings of length 4. Hence we have a strict inequality. This is because there is a double counting of string $abab$, which could be generated by $a^*b^*ab$ or $aba^*b^*$.

Strings of length 9 generated by $b^*ba^*ab^*$:

\[
\begin{align*}
&\text{Strings of length 9 generated by } b^*ba^*ab^*:
\end{align*}
\]

In total, the number of strings generated by $r$ is the same as the sum of the number of strings generated by $r_1$ and $r_2$.

Now we give another counter-example to show that the equality does not hold all the time. Consider $r = a^*baba^*b^*$. It satisfies the distinguishability criterion. However, it generates a total of $9 < \binom{4}{2} + \binom{4}{1} = 10$ distinct strings of length 4. Hence we have a strict inequality. This is because there is a double counting of string $abab$, which could be generated by $a^*b^*ab$ or $aba^*b^*$.

Strings of length 9 generated by $b^*ba^*ab^*$:

\[
\begin{align*}
&\text{Strings of length 9 generated by } b^*ba^*ab^*:
\end{align*}
\]
Strings of length 4 generated by $a^*b^*aba^*b^*$:

bbab, abaa, abab
abbb, babb, aaab
aabb, aaba, baba

The following is the code used to generate distinct strings of given length from any given regular expression of starred and unstarred letters.

```python
import sys

file = open("output.txt","w")
r = sys.argv[1]
i = int(sys.argv[2])
v = 0
for l in r:
    if l == ":
        v += 1
w = len(r) - 2*v
i = i - w
arr = [0] * v
j = v-1
all = []
count = 0
while True:
    arr[j] += 1
    if sum(arr) == i:
        count += 1
        all.append(arr.copy())
        while j >= 0 and arr[j] == i:
            j -= 1
    if j == -1:
        break
    if j != v-1:
        arr[j] += 1
        k = j + 1
        while k < v:
            arr[k] = 0
            k += 1
        j = v-1
        if sum(arr) == i:
            count += 1
            all.append(arr.copy())
final = []
for a in all:
    j = 0
```
str = ""
c = 0
while c < len(r)-1:
    if r[c+1] == "*":  
        str += r[c]*a[j]
        c += 2
        j += 1
    else:
        str += r[c]
        c += 1
if c == len(r)-1:
    str += r[c]
final.append(str)
for s in set(final):
    file.write(s)
    file.write("\n")
file.close()
p

10 Acknowledgement

I want to thank Timos Antonopoulos and Anton Xue for the generous help and valuable discussion. Without their help, this paper would not be possible.

References


