

# Finding Regular Expressions Given a Set of Strings

Brendan Farrell

March 8, 2022

## 1 Main Idea

We view a regular expression as a description for a set of strings and propose an approach to determine a regular expression for a given set of strings. An analogy for this task to be determine a curve around a set of points where one must balance two criteria: the size of the set should not be too large, yet the boundary should not be too complicated.

**Definition 1.** An *alphabet* is a set of characters, and a *string* or *word* is a finite sequence of characters. A *regular expression* is a description of a set of strings according to a certain format. We use the following formats, which are from the python language but do not include all options available in python:

1. Our alphabet is the ASCII characters 32-126, i.e. digits, uppercase and lowercase letters and punctuations.
2. A *class* is a set of characters that is written in one of the following ways:
  - (a) any character is denoted by a period (".").
  - (b) a single character, e.g. 'b'
  - (c) a set of ranges within the ASCII ordering, e.g. [a-m!-%] is characters 97-109 and 33-37.
  - (d) the specific classes \d (all digits), \D (all non-digits), \w (all word characters: a-z,A-Z,0-9,-) , \W (all non-word characters, i.e. punctuation)
3. A class is followed by a *quantifier*. When no quantifier is written, the number of elements from the class is 1. The quantifier \* means zero or more; ? means zero or one; + means one or more; {k,} means at least k; {,k} means up to k; and {k,l} means at least k and no more than l.
4. The statement  $RE_1 | \dots | RE_k$  means  $RE_1 \dots \text{or} \dots RE_k$  and will be called an *or statement*. Parentheses are put around an or statement when it is concatenated with another statement.

5. We will not use any other regex symbols, e.g. the symbol  $\wedge$  (not),  $\b$  (beginning or end of a word), or  $\s$  (any white space), etc.

**Definition 2.** A regular expression is *simple* if it is a class followed by a quantifier or if it is an or statement.

**Definition 3.** The *entropy* of a regular expression is the  $\log_2$  of the number of strings that satisfy the regular expression and is denoted  $ent$ . The *complexity* of the regular expression is the length of the regular expression, i.e. the total number of characters required to write the regular expression and is denoted  $K$ . The function  $\phi$  on regular expressions is

$$\phi(RE) = ent(RE) + \alpha K(RE), \quad (1)$$

where  $\alpha \in (0, \infty)$  (generally  $(0, 1]$ ) is a weight.

Given a set of strings, we will determine a regular expression that all of the strings satisfy and that has minimal  $\phi$ -value. The entropy is the size of the regular expression, and the complexity is how complicated it is. The complexity is based on Kolmogorov entropy, hence the letter  $K$ , but is somewhat different. Kolmogorov complexity of a string is the length of the shortest computer program that can write the string. Here, we only consider the shortest regular expression that can describe a set; that is we only allow specific ways of describing the sets.

## 2 Details

### 2.1 The Length 1 case

We start by looking at strings of length 1. Let  $S$  be the set of all characters appearing in a set of words of length 1. Given  $\alpha$ , a function exists that returns a  $\phi$ -minimizing regular expression for any set  $S$ . For example, if  $S = \{0, 2, 4, 6, 8\}$ , then two possible regular expressions are  $r_1 = [02468]$  and  $r_2 = \backslash d$ . The  $\phi$  values of these are  $\phi(r_1) = \log_2(5) + \alpha 7$  and  $\phi(r_2) = \log_2(10) + \alpha 2$ , so that the  $\phi$ -minimal regular expression depends on the value of  $\alpha$ .

We only work with ASCII characters 32-126, so a set of characters can be represented as a vector of zeros and ones of length 94 and passed to a function with the parameter  $\alpha$  to return a  $\phi$ -minimizing regular expression.