COMPSCI 250 Discussion #8: Designing Regular Expressions

Individual Handout
David Mix Barrington and Ghazaleh Parvini
15 November 2023

In lecture we have faced the problem of taking a particular language and constructing a regular expression for it. Our goal is to get a regular expression that represents all the strings that are *in* the language, but no strings that are *not in* the language. There is a basic strategy for this problem — find a regular expression that represents *only* desirable strings, see whether it represents all the desired strings, and if not construct another regular expression for some of the strings that it misses.

In the discussion and/or lecture we'll look at two examples:

- The language of strings with an even number of a's, which has expression $(b + ab^*a)^*$,
- The language $F = (a + ab)^*$, equal to the set of strings that do not start with b and have no bb substring.

Writing Exercise:

Construct a regular expression for the set EE ("even-even") of strings in $\{a, b\}^*$ that have both an even number of a's and an even number of b's. Justify your answer carefully – explain why your expression generates only even-even strings and why it generates *all* even-even strings.

Note that all even-even strings have even length, so you may think of the whole string as being broken up into two-letter blocks.

Here are some more hints. You are not required to use them to solve the main problem, but they will probably be useful.

Define the language EEP ("even-even-primitive") of nonempty strings that are in EE and have *no proper prefix* in EE. (That is, if $w \in EEP$ and $w = uv$ with both u and v in EE, then either $u = \lambda$ or $v = \lambda$.) It turns out that while EEP is harder than EE to describe in English, it has a simpler regular expression.

- Explain why $EE = (EEP)^*$.
- Which strings of up to six letters are in EEP?
- Construct a regular expression for EEP, and explain why this solves the main problem.