CMPSCI 250 Discussion #11: Designing Regular Expressions
Individual Handout

David Mix Barrington
17 April 2013

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 three 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.
- The language No-aba of strings that never have aba as a (consecutive) substring.

The language Yes-aba has the simple regular expression $\Sigma^* aba \Sigma^*$, where Σ is an abbreviation for $a + b$. It's harder to get an expression for No-aba — our basic idea is to find classes of strings that don't have aba's, and union them together until we have everything possible. We can start with a^* and b^*, since a string needs both a's and b's to have an aba.

We need to look at each b in the string and make sure that it is not part of an aba. Each b must be the first letter, be the last letter, or have a b either before it or after it. So any b except for the first or last letter must come in a group of two or more b's. The language of groups of two or more b's is bbb^* (not $(bbb)^*$) and so $(a + bbb^*)^*$ is the set of all strings where the b's come in such groups. We could have any string of this type, with an optional single b before and/or after. A correct regular expression for No-aba is thus $(b + \lambda)(a + bbb^*)^*(b + \lambda)$, where λ is an abbreviation for $\emptyset^*$. We can check that a^* and b^* are contained within this language, so we don't have to add them in separately.

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.

We've broken this problem into subproblems. 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 substring* 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 zero, two, four, and six letters are in EEP? (Hint: There are one zero-letter string, two two-letter strings, four four-letter strings, eight six-letter strings, and 16 eight-letter strings in EEP. Make sure you don't include strings that can be factored into smaller ones in EE.)

- Construct a regular expression for EEP, and explain why this solves the main problem. (If you don't find a pattern in the six-letter strings, try the eight-letter ones...)