Informatics 2A 2018–19
Tutorial Sheet 2 (Week 4)

Mary Cryan

This week's exercises concern some applications of regular languages and finite state machines (Lectures 6 and 7), along with the problem of determining whether a language is regular or not (Lecture 8).

Recall that there are two common notations for regular expressions, both of which are described in the lecture slides:

- The basic *mathematical notation*, as introduced at the end of Lecture 5 (and start of Lecture 6), and used e.g. in Kozen's book.
- A more extended *machine syntax*, described in Lecture 6 (and slide 5 of Lecture 7), used by the `grep` commands and in languages such as Perl and Python (expressions in this syntax are sometimes called *patterns* in the lecture slides).

For example, the language written in mathematical notation as $a^*(b+c)^* + dd^*$ could be written in machine syntax as `(a*[bc]*)|d+`. Questions 1 and 2 involve the use of both notations. Unfortunately, there are clashes between them: e.g. they differ on the meanings of both '+' and '.'

Please attempt the following three questions in advance of your tutorials next week. You should be sure to attempt at least part of Question 3, since understanding the pumping lemma and how to apply it takes a bit of practice. The last (starred) subquestion of Question 3 is an optional challenge question, and need only be attempted by those confident with the matrial.

1. A *floating-point literal* in Java (simplifying slightly) has the following form: a sequence of decimal digits, then a decimal point, then another sequence of decimal digits. *At least one* of these digit sequences must be non-empty, so `.` by itself is not allowed.

   The whole of this may then *optionally* be followed by an *exponent part*, consisting of the symbol `E` or `e`, then an optional `+` or `-` sign, then a sequence of one or more decimal digits. E.g.,

   ```
   2.    .3    3.14    6.0e23    7.E-1
   ```

   (a) Write a regular expression in mathematical notation that defines this class of floating-point literals. You may introduce abbreviations for any sets of characters you may find useful (it will be sensible to define $\underline{D}$ as a regular expression to denote the decimal digit characters).

   (b) Next write an `egrep` command that finds all substrings of a source file `foo.java` that conform to the above format for floating-point literals. It might take some time to get the detail right (it can be frustrating dealing with the idiosynchrasies of `egrep` notation) so you should test out your commands on DICE — for instance, you can use the LaTeX source file for this tutorial sheet, at `www.inf.ed.ac.uk/teaching/courses/inf2a/tutorials/tutorial2.tex`

Testing against this `.tex` file should at least return the occurrences shown in the introduction to this question (there are some commented out instances too, with text telling you you've succeeded!)

(c) Suppose we have an NFA N for the class of floating-point literals (you needn't construct one). Give an example of a string for which N will pass through a non-accepting state in between two accepting states. What is the longest sequence of non-accepting states that N may pass through between two accepting states?

(Note that when `egrep` finds nested matches to a pattern it returns only the longest supermatches. For example, if `foo.java` contains the literal `0.0`, then it also contains '`0.`' and '`.0`', but only '`0.0`' will be returned.)

2. In Java, a *string literal* is a sequence of *input characters* enclosed in double quotes, e.g. `"Catch22"`. The restrictions on the sequence of characters are that the double quote character `"` itself, if it appears in the body of the string, must be denoted by the *escape sequence* `\"`, and the character `\` itself must be denoted `\\`. Five other escape sequences denoting control characters are allowed: `\b`, `\t`, `\n`, `\f`, `\r`. However, apart from the special cases, there cannot be any stray single `\` characters in the string.

   (a) Write a regular expression in mathematical notation that defines the set of Java string literals. Again, you may introduce abbreviations for any sets of characters you find useful.

   (b) Write an `egrep` command that will print all lines of the file `foo.java` containing a valid string literal. Here you should bear in mind that `egrep` patterns have their own escape conventions: the character `"` must be written within a pattern as `\"`, while `\` must be written as `\\` and moreover must be enclosed within square brackets `[]`.

   In coming up with the expression (as in (a)) you'll need to first *exclude* `"` and `\` from your pool of characters (the "complement" operator `^` described on slide 5 of Lecture 7 will be helpful) but then add back in the legal special cases (to describe `\"`, `\\`, `\n` etc) and this is tricky because of the escape conventions described above. It will probably be helpful to do some debugging on the DICE machines.

   (c) Ignoring escape characters and comments, a line of Java code will typically be erroneous if it contains an *odd* number of occurrences of `"`. Write an `egrep` command that prints all lines from `foo.java` that have this property.

   It might not be immediately obvious, but this question is related to the problem we considered on slide 5 of Lecture 6, in designing regular expressions to recognise the language of binary strings with an even number of 0s. In this case we want an odd count (instead of even), and are concerned with `"` symbols rather than 0s, but the principle is the same.

3. Each of the following expressions defines a language over $\Sigma = \{a, b\}$. If the language is regular, give an NFA or regular expression that defines it. If it is not regular, use the pumping lemma to prove this.

   (a) The set of *palindromes*, i.e. those strings $x \in \Sigma^*$ such that $x = \mathrm{rev}(x)$, where $\mathrm{rev}(x)$ means x reversed (written backwards).

   (b) The set of strings of the form ss, i.e. those with two identical halves. (E.g. *abbabb*.)

   (c) The set of strings with an equal number of substrings ab and ba.

   (d)* The set of strings whose length is a prime number, i.e.,

$$\{x \in \Sigma^* \mid |x| \text{ is prime}\} \ .$$