Informatics 2A 2015–16
Tutorial Sheet 2 (Week 4)

John Longley

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

We draw attention to the fact that there are two common notations for regular expressions, both of which are described in the lecture slides:

- The basic *mathematical notation*, as introduced in Lecture 5, and used e.g. in Kozen's book.
- A more extended *machine syntax*, described in Lecture 6, and used by the Grep tools and in languages such as Perl and Python. Expressions in this syntax are 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 the pumping lemma is a topic that often causes some difficulty to begin with. The last (starred) subquestion of Question 3 is an optional challenge question, and need only be attempted by those confident with the material.

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.

   (b) Write an `egrep` command that finds all substrings of a source file `foo.java` that conform to the above format for floating-point literals. You may find it helpful to try out your command on the DICE machines, using a small test file containing some examples and non-examples of floating-point literals.

   (c) Extend your solution to part (b) to write an `egrep` command that matches only complete floating-point literals in `foo.java`.

   For example, if `foo.java` contains the literal `0.0`, then part (b) finds three matches '`0.`', '`.0`' and '`0.0`'; whereas part (c) only finds the complete literal '`0.0`' itself. N.B. REMOVE PART (c) IN 2015 SINCE MISGUIDED. SEE SOLUTION FOR REASON.

2. In Java, a *string literal* is a sequence of *input characters* enclosed in double quotes, e.g. `"Catch22"`. The only 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 also allowed: `\b`, `\t`, `\n`, `\f`, `\r`.

   (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 `[]`. Again, you may find it helpful to test your command 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 a `grep` command that prints all lines from `foo.java` that have this property.

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 strings $a^i b^j$ where $i < j$.

   (b) 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).

   (c) The set of strings with an equal number of substrings aa and bb. (For example, the string $aaaaaabab$ has 5 substrings aa, and 0 substrings bb — so this word is not in the language.)

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

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

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