CS 3719 (Theory of Computation and Algorithms) – Lecture 2

Antonina Kolokolova*

January 7, 2011

1 Preliminaries: Alphabets and languages

We will start by introducing some notation.

- Here, define an *alphabet* to be any finite non-empty set. Usually, we denote an alphabet by the greek letter capital sigma Σ. Note that this is the same symbol as is often used for summation, but it should be understandable from the context which meaning of Σ to use.
  - Unary alphabet: an alphabet which contains exactly one letter.
  - Binary alphabet: alphabet with two symbols, usually called $\{0, 1\}$
  - English alphabet: for example, $\Sigma = \{a, b, c, d, \ldots, z\}$.
- A *string* over an alphabet is a finite sequence of letters from that alphabet. E.g., 0110 is a string over binary alphabet, and *aaabaa* can be a string over $\{a, b\}$ or, say, $\{a, b, c, d\}$. The special *empty string* denoted ϵ can be a string over any alphabet. The length of a string is a number of symbols in it; the length of ϵ is 0.
- A set of all strings over a given alphabet Σ is denoted Σ^* ("sigma star"). We will explain this notation at the end of this lecture. For example, $00110 \in \{0, 1\}^*$. Sometimes we are only interested in strings of length at most n; then we write a set of them as Σ^n.
- A *language* is a set of strings over some alphabet. For example, English language (ignoring capitalization) is a subset of strings over English alphabet (together with symbol $-$). Prime numbers $\{2, 3, 5, 7, 11, 13, 17...\} \subseteq \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}^*$. Two

*The material in this set of notes came from many sources, in particular "Introduction to Theory of Computation" by Sipser and course notes of U. of Toronto CS 364.

special languages are the empty language $\emptyset$ and the language Σ^* of all strings over an alphabet Σ. Since our alphabets are finite sets, our languages will be countable sets (possibly finite, although interesting ones are infinitely countable).

Usually, when we talk about solving problems we think about a solution as a long string: e.g., a path between two points in a graph. But the simplest kind of solution is just an answer to a yes/no question: "is graph G connected?", "is number n prime?". Moreover, we can convert the first type of the problem into the second by asking many yes/no questions about the solution: "is the first bit of the solution 1? Is it the last bit? Is the second bit 1?...".

With this as motivation, for the purpose of this course we will concentrate problems where solutions are just yes/no answers. The first convenience of such assumption is that we can speak about problems as *membership in a language* problems. For example, "is the number n prime?" becomes "is n a member of the set of all prime numbers? Most of the course we will be talking about classes of languages of certain complexity: that is, classes of problems that have a that complexity when viewed as language membership problems.

2 Regular expressions

The first class of languages we will consider is the languages which can be described by regular expressions. Most of you have done Google searches for expressions like "MUN AND (CS3719 OR COMP3719)" to find pages that contain the string MUN and either the string CS3719 or the string COMP3719. Also, most of you have searched for a file with, say, 3719 in the name on a UNIX system by doing `ls ''*3719*''`. In both cases you were using regular expressions to describe a set of strings you needed.

Formally, regular expressions are defined by the following recursive definition: R is a regular expression (over Σ) if R is one of the following:

1) A letter from an alphabet: $a \in \Sigma$, for some a (here, a is a generic name for a letter, does not have to be the symbol "a").

2) An empty string ϵ.

3) The empty set $\emptyset$.

4) A union of two regular expressions $(R_1 \cup R_2)$ (that is, all strings that either match R_1 or match R_2, or both).

5) A concatenation of two regular expressions $R_1 \circ R_2$ (that is, all strings that can be formed by writing a string of R_1 followed by a string of R_2).

6) A *Kleene star* R_1^*, which is zero or more concatenated strings from R_1. This is the same $*$ as in Σ^*, where it means zero or more letters from Σ.

And only strings formed in this way are regular expressions. We would say a string "matches" a regular expression if it is a member of a language of this regular expression.

For example, suppose we want to define a set of all binary numbers (that is, $\Sigma = \{0, 1\}$) divisible by 4 by a regular expression. To make it more interesting, say we only want to list numbers starting with a 1 (e.g., we only consider strings starting with 1 as valid natural numbers for this example). Then we can do it as follows.

- 0 is a regular expression; so is 1, by rule 1.
- Then 00 is a regular expression by rule 5
- And $(0 \cup 1)$ is a regular expression by rule 4.
- By rule 6, $(0 \cup 1)^*$ is a regular expression (in fact, it matches all strings in Σ^*.)
- Finally, $1(0 \cup 1)^*00$ is a regular expression by two applications of rule 5

You can check yourself that the strings that are matched by this regular expression are binary numbers starting with 1 and ending with 00, that is, divisible by 4. For example, 100 is matched, and so is 10010101010100, but not 01100 or 1001. This language is infinitely countable: there are infinitely many numbers divisible by 4. Or, alternatively, you can see that there is a string in the regular expression for any string in $\Sigma^* = \{0 \cup 1\}^*$.