COMS W3261
Computer Science Theory
Chapter 3 Notes

Alexander Roth

2014-09-09

Regular Expressions

Regular expressions can define exactly the same languages that the various forms of automata describe: the regular languages. However, regular expressions offer something that automata do not: a declarative way to express the strings we want to accept. Thus regular expressions serve as the input language for many systems that process strings.

The Operators of Regular Expressions

Regular expressions denote languages. There are three operations on languages that the operators of regular expressions represent. These operators are:

1. The *union* of two languages L and M, denoted by $L \cup M$, is the set of strings that are in either L or M or both.
2. The *concatenation* of languages L and M is the set of strings that can be formed by taking any string in L and concatenating it with any string in M. The concatenation operator is frequently called "dot".
3. The *closure* (or *star*, or *Kleene closure*) of a language L is denoted L^* and represents the set of those strings that can be formed by taking any number of strings from L, possibly with repetitions and concatenating all of them.

Applications of Regular Expressions

We shall consider two important classes of regular-expression-based applications: lexical analyzers and text search.

Regular Expressions in UNIX

Before seeing the applications, we shall introduce the UNIX notation for extended regular expressions. This notation gives us a number of additional capabilities. In fact, the UNIX extensions include certain features, especially the ability to name and refer to previous strings that have matched a pattern, that actually allow nonregular languages to be recognized.

The first enhancement to the regular-expression notation concerns the fact that most real applications deal with the ASCII character set. UNIX regular expressions allow us to write *character classes* to represent large sets of characters as succinctly as possible. The rules for character classes are:

- The symbol `.` (dot) stands for "any character".
- The sequence $[a_1 a_2 \cdot a_k]$ stands for the regular expression

  $$a_1 + a_2 + \cdots + a_k$$

  The notation saves about half the characters, since we don't have to write the $+-$ signs. For example, we could express the four characters used in C comparison operators by `[<>=!]`.
- Between the square braces we can put a range of the form x-y to mean all the characters from x to y in the ASCII sequence. Since the digits have codes in order, as do the upper-case letters and the lower-case letters, we can express many of the classes of characters that we really care about with just a few keystrokes. If we want to include a minus sign among a list of characters, we can place it first or last, so it is not confused with its use to form a character range. Square brackets, or other characters that have special meanings in UNIX regular expressions can be represented as characters by preceding them with a backslash (\).
- There are special notations for server of the most common classes of characters. For instance:

  a) `[:digit:]` is the set of ten digits, the same as `[0-9]`.

  b) `[:alpha:]` stands for any alphabetic character, as does `[A-Za-z]`.

  c) `[:alnum:]` stands for the digits and letters (alphabetic and numeric characters), as does `[A-Za-z0-9]`.

In addition, there are several other operators that are used in UNIX regular expressions. None of these operators extended what languages can be expressed, but they sometimes make it easier to express what we want.

- The operator — is used in place of + to denote union.
- The operator `?` means "zero or one of." Thus, $R?$ in UNIX is the same as $\epsilon + R$ in our notation.

- The operator $+$ means "one or more of." Thus, $R+$ in UNIX is shorthand for RR^* in our notation.
- The operator $\{n\}$ means "n copies of." Thus, $R\{5\}$ in UNIX is shorthand for $RRRRR$.