COMS 381, Summer 2005
Supplementary Handout 3
Quick introduction to the Unix regular expressions
Tuesday, June 7th

As mentioned earlier in the course, regular expresssions are commonly used in real-world settings. They are built-in in some programming languages such as Perl, and they are used extensively in a variety of UNIX utilities. Some examples of the latter are `grep`, a tool that searches given files for occurences of a given pattern, text editors such as `vi` and `emacs`, and many others. Unfortunately, there is no one standard syntax for regular expressions across programming languages, or even across the UNIX world. However, what follows is a sampling of regular expression syntax features that you are likely to meet in many UNIX programs. The information below corresponds roughly to what is sometimes called the extended regular expressions in UNIX.

1 Most common operators

. Matches any single character

a Matches just the letter 'a'. Same for any other character, except these which are special characters (part of the regexp syntax). Those have to be *escaped* by prefixing with a backslash if you mean them literally: for instance, to match a literal ?, type \?.

ab Concatenation; matches a single 'a' followed by a single 'b'

a | b Alternation, like + in the theoretical regular expression syntax. Matches a single a or a single b.

a+ Matches one or more occurrences of a

a? Matches zero or one occurrences of a

a* Matches zero or more occurrences of a

[abcd] Matches any one of a, b, c, d

[a-d] Matches any one of a, b, c, d ("a through d")

[^a] Matches any character *except* a

^ Matches the start of a line

$ Matches the end of a line

( ) Can be used to package together a subexpression as a group so that operators like ? or * can be applied to it; for instance, **a(bc)*** matches **a, abc, abcbc** and so on

\n where n is a digit from 1 to 9; this is the *backreferencing* construct, which matches whatever the nth group matched. For example, (**bana**)**na\1bo\1** matches **bananabanabobana**.

2 Examples

Some shamelessly stolen from the Web:

re[**aei**]**d** Any of 'read', 'reed', 'reid'
^$ A blank line (match the start, nothing, and the end).
^[**a** − **zA** − **Z**]$ One word lines.
^[**0** − **9**]$ Lines with exactly one number.
d.**n** Any three characters starting with a 'd' and ending with an 'n'.
d[**aeiou**]**n** 'dan', 'den', 'din', 'don', or 'dun'.
^[**0** − **9a** − **zA** − **Z**] + @[**0** − **9a** − **zA** − **Z**] + \.[**a** − **zA** − **Z**][**a** − **zA** − **Z**][**a** − **zA** − **Z**]$

A very very simplified email address regexp.

3 The irregularity of backreferences

Note that the backreferencing construct allows us to express languages that are not regular. For instance, the language $\{ww \mid w \in \Sigma^*\}$ is matched by the UNIX regular expression $(.*)\backslash\mathbf{1}$ Thus, "regular expressions" really need not mean the same thing in the theoretical world and in the programming world!

4 For more info

You can find plenty of information on regular expressions on the Web. Or just read `man regexp`, `info regex` or `man grep` :).

For examples of real-world regular expressions, check out http://regexlib.com/. Search for "email", for instance, and see what expressions people come up with!

On the irregularity of backreferences, a theoretical paper is K. S. Larsen. Regular expressions with nested levels of back referencing form a hierarchy. Inf. Process. Lett., 65(4):169-172, 27 Feb. 1998.