REX – The Regular Expression Filter

This is a text filter designed in a manner similar to the UNIX utility grep that processes text files using Regular Expressions. The method used is to incrementally construct a DFA for the given regular expression one state at a time as the input is being processed. For regular expressions with small DFA's construction of new states will be complete early on with the result that most of the input file will be processed relatively fast without having to calculate new states. For regular expressions with large DFA's but smaller inputs, the construction of new states will be kept to a minimum. Overall the algorithm used is O(n) in the size of the input file.

The options are the same as the GNU [e]grep utility. Proviso: not all the bells and whistles may ding and whizz the same, since I based the program in my interpretation of the local GREP manual section (included here for reference), and only a cursory run-time analysis of the program itself.

Much of REX is just a grafting from DFA with minor changes to make the regular expression syntax look a little bit more like [E]GREP. Even the algebraic reductions, CatExp(), BinExp() are still present, though in reality they serve little purpose in this application.

Syntax

The syntax used for regular expressions is similar to the EGREP syntax, but is significantly more expansive. It includes not only the full range of regular expression operators, A B (concatenation), A | B (alteration), A* (iteration), but also boolean operators: A & B (intersection), A - B (relative complement), and another very powerful operator, A ^ B, called the Interleave Product.

The Interleave Product of two regular expressions, A, B, is the set of all the interleavings of the expressions A and B. For instance:

a^b = ab|ba
ab^ab = aabb|abab = a(a^b)b
a^b^c = abc|acb|bac|bca|cab|cba

The automaton for the interleave of expressions E1 and E2 can be formed by taking the Cartesian Product of the automata for E1 and E2, so it has all the properties of the Cartesian Product.

The following characters are assigned special meanings:

{ } < > -------- Beginning and end of line and word respectively
^ & - | + * ? -- Regular expression operators
[ ] ---------- Set brackets.
\ ---------- The escape character.

The special meanings of these characters can be cancelled by preceding them by the escape character. For example, \{ will match a literal bracket, whereas { matches the beginning of the line.

An example should illustrate just how powerful this notation, particularly the interleave product, is. The regular expression

<(n^a^p^o^]^e^o^n)>

will match any anagram of Napoleon when REX is called with the -i option (to make matching case insensitive). This works in reverse as well. One could take an anagram (like aponenol) and run REX on a dictionary with it in order to find all of its matches.

The one difference from [E]GREP syntax is that the characters { and } are used instead of ^ and $. The operator syntax is the same as for the companion DFA utility provided.

Future Versions

The future may hold the creation of a filter for Context Free Expressions.