Alternation with The Vertical Bar or Pipe Symbol

I already explained how you can use character classes to match a single character out of several possible characters. Alternation is similar. You can use alternation to match a single regular expression out of several possible regular expressions.

If you want to search for the literal text `cat` or `dog`, separate both options with a vertical bar or pipe symbol: `cat|dog`. If you want more options, simply expand the list: `cat|dog|mouse|fish`.

The alternation operator has the lowest precedence of all regex operators. That is, it tells the regex engine to match either everything to the left of the vertical bar, or everything to the right of the vertical bar. If you want to limit the reach of the alternation, you need to use parentheses for grouping. If we want to improve the first example to match whole words only, we would need to use `\b(cat|dog)\b`. This tells the regex engine to find a word boundary, then either `cat` or `dog`, and then another word boundary. If we had omitted the parentheses then the regex engine would have searched for a word boundary followed by `cat`, or, `dog` followed by a word boundary.

Remember That The Regex Engine Is Eager

I already explained that the regex engine is eager. It stops searching as soon as it finds a valid match. The consequence is that in certain situations, the order of the alternatives matters. Suppose you want to use a regex to match a list of function names in a programming language: Get, GetValue, Set or SetValue. The obvious solution is `Get|GetValue|Set|SetValue`. Let's see how this works out when the string is `SetValue`.

The regex engine starts at the first token in the regex, `G`, and at the first character in the string, `S`. The match fails. However, the regex engine studied the entire regular expression before starting. So it knows that this regular expression uses alternation, and that the entire regex has not failed yet. So it continues with the second option, being the second `G` in the regex. The match fails again. The next token is the first `S` in the regex. The match succeeds, and the engine continues with the next character in the string, as well as the next token in the regex. The next token in the regex is the `e` after the `S` that just successfully matched. `e` matches `e`. The next token, `t` matches `t`.

At this point, the third option in the alternation has been successfully matched. Because the regex engine is eager, it considers the entire alternation to have been successfully matched as soon as one of the options has. In this example, there are no other tokens in the regex outside the alternation, so the entire regex has successfully matched `Set` in `SetValue`.

Contrary to what we intended, the regex did not match the entire string. There are several solutions. One option is to take into account that the regex engine is eager, and change the order of the options. If we use `GetValue|Get|SetValue|Set`, `SetValue` is attempted before `Set`, and the engine matches the entire string. We could also combine the four options into two and use the question mark to make part of them optional: `Get(Value)?|Set(Value)?`. Because the question mark is greedy, `SetValue` is be attempted before `Set`.

The best option is probably to express the fact that we only want to match complete words. We do not want to match Set or SetValue if the string is `SetValueFunction`. So the solution is `\b(Get|GetValue|Set|SetValue)\b` or `\b(Get(Value)?|Set(Value)?)\b`. Since all options have the same end, we can optimize this further to `\b(Get|Set)(Value)?\b`.

Text-Directed Engine Returns the Longest Match

Alternation is where regex-directed and text-directed engines differ. When a text-directed engine attempts `Get|GetValue|Set|SetValue` on `SetValue`, it tries all permutations of the regex at the start of the string. It does so efficiently, without any backtracking. It sees that the regex can find a match at the start of the string, and that the matched text can be either `Set` or `SetValue`. Because the text-directed engine evaluates the regex as a whole, it has no concept of one alternative being listed before another. But it has to make a choice as to which match to return. It always returns the longest match, in this case `SetValue`.

POSIX Requires The Longest Match

The POSIX standard leaves it up to the implementation to choose a text-directed or regex-directed engine. A BRE that includes backreferences needs to be evaluated using a regex-directed engine. But a BRE without backreferences or an ERE can be evaluated using a text-directed engine. But the POSIX standard does mandate that the longest match be returned, even when a regex-directed engine is used. Such an engine cannot be eager. It has to continue trying all alternatives even after a match is found, in order to find the longest one. This can result in very poor performance when a regex contains multiple quantifiers or a combination of quantifiers and alternation, as all combinations have to be tried to find the longest match.

The Tcl and GNU flavors also work this way.

Make a Donation

Did this website just save you a trip to the bookstore? Please make a donation to support this site, and you'll get a **lifetime of advertisement-free access** to this site!

Copyright © 2003-2020 Jan Goyvaerts. All rights reserved.