Quantifiers On Recursion

The introduction to recursion shows how `a(?R)?z` matches `aaazzz`. The quantifier `?` makes the preceding token optional. In other words, it repeats the token between zero or one times. In `a(?R)?z` the `(?R)` is made optional by the `?` that follows it. You may wonder why the regex attempted the recursion three times, instead of once or not at all.

The reason is that upon recursion, the regex engine takes a fresh start in attempting the whole regex. All quantifiers and alternatives behave as if the matching process prior to the recursion had never happened at all, other than that the engine advanced through the string. The regex engine restores the states of all quantifiers and alternatives when it exits from a recursion, whether the recursion matched or failed. Basically, the matching process continues normally as if the recursion never happened, other than that the engine advanced through the string.

If you're familiar with procedural programming languages, regex recursion is basically a recursive function call and the quantifiers are local variables in the function. Each recursion of the function gets its own set of local variables that don't affect and aren't affected by the same local variables in recursions higher up the stack. Quantifiers on recursion work this way in all flavors, except Boost.

Let's see how `a(?R){3}z|q` behaves (Boost excepted). The simplest possible match is `q`, found by the second alternative in the regex.

The simplest match in which the first alternative matches is `aqqqz`. After `a` is matches, the regex engine begins a recursion. `a` fails to match `q`. Still inside the recursion, the engine attempts the second alternative. `q` matches `q`. The engine exits from the recursion with a successful match. The engine now notes that the quantifier `{3}` has successfully repeated once. It needs two more repetitions, so the engine begins another recursion. It again matches `q`. On the third iteration of the quantifier, the third recursion matches `q`. Finally, `z` matches `z` and an overall match is found.

This regex does not match `aqqz` or `aqqqqz`. `aqqz` fails because during the third iteration of the quantifier, the recursion fails to match `z`. `aqqqqz` fails because after `a(?R){3}` has matched `aqqq`, `z` fails to match the fourth `q`.

The regex can match longer strings such as `aqaqqqzqz`. With this string, during the second iteration of the quantifier, the recursion matches `aqqqz`. Since each recursion tracks the quantifier separately, the recursion needs three consecutive recursions of its own to satisfy its own instance of the quantifier. This can lead to arbitrarily long matches such as `aaaqqaqqqzzaqqqzqzqaqqaaqqqzqqzzz`.

How Boost Handles Quantifiers on Recursion

Boost has its own ideas about how quantifiers should work on recursion. Recursion only works the same in Boost as in other flavors if the recursion operator either has no quantifier at all or if it has `*` as its quantifier. Any other quantifier may lead to very different matches (or lack thereof) in Boost 1.59 or prior versus Boost 1.60 and later versus other regex flavors. Boost 1.60 attempted to fix some of the differences between Boost and other flavors but it only resulted in a different incompatible behavior.

In Boost 1.59 and prior, quantifiers on recursion count both iteration and recursion throughout the entire recursion stack. So possible matches for `a(?R){3}z|q` in Boost 1.59 include `aaaazzzz`, `aaaqzzz`, `aaqqzz`, `aaqzqz`, and `aqaqzzz`. In all these matches the number of recursions and iterations add up to 3. No other flavor would find these matches because they require 3 iterations during each recursion. So other flavors can match things like `aaqqqzaqqqzaqqqzz` or `aqqaqqqzz`. Boost 1.59 would match only `aqqqz` within these strings.

Boost 1.60 attempts to iterate quantifiers at each recursion level like other flavors, but does so incorrectly. Any quantifier that makes the recursion optional allows for infinite repetition. So Boost 1.60 and later treat `a(?R)?z` the

same as `a(?R)*z`. While this fixes the problem that `a(?R)?z` could not match `aaazzz` entirely in Boost 1.59, it also allows matches such as `aazazz` that other flavors won't find with this regex. If the quantifier is not optional, then Boost 1.60 only allows it to match during the first recursion. So `a(?R){3}z|q` could only ever match `q` or `aqqqz`.

Boost's issues with quantifiers on recursion also affect quantifiers on parent groups of the recursion token. They also affect quantifiers on subroutine calls and quantifiers on groups that contain a subroutine call to a parent group of the group with the quantifier.

Quantifiers on Other Tokens in The Recursion

Quantifiers on other tokens in the regex behave normally during recursion. They track their iterations separately at each recursion. So `a{2}(?R)z|q` matches `aaqz`, `aaaaqzz`, `aaaaaaqzzz`, and so on. `a` has to match twice during each recursion.

Quantifiers like these that are inside the recursion but do not repeat the recursion itself do work correctly in Boost.

Make a Donation

Page URL: https://www.regular-expressions.info/recurserepeat.html
Page last updated: 30 March 2020
Site last updated: 08 April 2020