Backreferences to Failed Groups

The previous topic on backreferences applies to all regex flavors, except those few that don't support backreferences at all. Flavors behave differently when you start doing things that don't fit the "match the text matched by a previous capturing group" job description.

There is a difference between a backreference to a capturing group that matched nothing, and one to a capturing group that did not participate in the match at all. The regex `(q?)b\1` matches `b`. `q?` is optional and matches nothing, causing `(q?)` to successfully match and capture nothing. `b` matches `b` and `\1` successfully matches the nothing captured by the group.

In most flavors, the regex `(q)?b\1` fails to match `b`. `(q)` fails to match at all, so the group never gets to capture anything at all. Because the whole group is optional, the engine does proceed to match `b`. The engine now arrives at `\1` which references a group that did not participate in the match attempt at all. This causes the backreference to fail to match at all, mimicking the result of the group. Since there's no `?` making `\1` optional, the overall match attempt fails.

One of the few exceptions is JavaScript. According to the official ECMA standard, a backreference to a non-participating capturing group must successfully match nothing just like a backreference to a participating group that captured nothing does. In other words, in JavaScript, `(q?)b\1` and `(q)?b\1` both match `b`. XPath also works this way.

Dinkumware's implementation of std::regex handles backreferences like JavaScript for all its grammars that support backreferences. Boost did so too until version 1.46. As of version 1.47, Boost fails backreferences to non-participating groups when using the ECMAScript grammar, but still lets them successfully match nothing when using the basic and grep grammars.

Backreferences to Non-Existent Capturing Groups

Backreferences to groups that do not exist, such as `(one)\7`, are an error in most regex flavors. There are exceptions though. JavaScript treats `\1` through `\7` as octal escapes when there are fewer capturing groups in the regex than the digit after the backslash. `\8` and `\9` are an error because 8 and 9 are not valid octal digits.

Java treats backreferences to groups that don't exist as backreferences to groups that exist but never participate in the match. They are not an error, but simply never match anything.

.NET is a little more complicated. .NET supports single-digit and double-digit backreferences as well as double-digit octal escapes without a leading zero. Backreferences trump octal escapes. So `\12` is a line feed (octal 12 = decimal 10) in a regex with fewer than 12 capturing groups. It would be a backreference to the 12th group in a regex with 12 or more capturing groups. .NET does not support single-digit octal escapes. So `\7` is an error in a regex with fewer than 7 capturing groups.

Forward References (useful only in repeated groups)

Many modern regex flavors, including JGsoft, .NET, Java, Perl, PCRE, PHP, Delphi, and Ruby allow forward references. They allow you to use a backreference to a group that appears later in the regex. Forward references are obviously only useful if they're inside a repeated group. Then there can be situations in which the regex engine evaluates the backreference after the group has already matched. Before the group is attempted, the backreference fails like a backreference to a failed group does.

If forward references are supported, the regex `(\2two|(one))+` matches `oneonetwo`. At the start of the string, `\2` fails. Trying the other alternative, `one` is matched by the second capturing group, and subsequently by the first group. The first group is then repeated. This time, `\2` matches `one` as captured by the second group. `two` then matches `two`. With two repetitions of the first group, the regex has matched the whole subject string.

JavaScript does not support forward references, but does not treat them as an error. In JavaScript, forward references always find a zero-length match, just as backreferences to non-participating groups do in JavaScript. Because this is not particularly useful, XRegExp makes them an error. In std::regex, Boost, Python, Tcl, and VBScript forward references are an error.

Nested References

A nested reference is a backreference inside the capturing group that it references. Like forward references, nested references are only useful if they're inside a repeated group, as in `(\1two|(one))+`. When nested references are supported, this regex also matches `oneonetwo`. At the start of the string, `\1` fails. Trying the other alternative, `one` is matched by the second capturing group, and subsequently by the first group. The first group is then repeated. This time, `\1` matches `one` as captured by the last iteration of the first group. It doesn't matter that the regex engine has re-entered the first group. The text matched by the group was stored into the backreference when the group was previously exited. `two` then matches `two`. With two repetitions of the first group, the regex has matched the whole subject string. If you retrieve the text from the capturing groups after the match, the first group stores `onetwo` while the second group captured the first occurrence of `one` in the string.

The JGsoft, .NET, Java, Perl, and VBScript flavors all support nested references. PCRE does too, but had bugs with backtracking into capturing groups with nested backreferences. Instead of fixing the bugs, PCRE 8.01 worked around them by forcing capturing groups with nested references to be atomic. So in PCRE, `(\1two|(one))+` is the same as `(?>(\1two|(one)))+`. This affects languages with regex engines based on PCRE, such as PHP, Delphi, and R.

JavaScript and Ruby do not support nested references, but treat them as backreferences to non-participating groups instead of as errors. In JavaScript that means they always match a zero-length string, while in Ruby they always fail to match. In std::regex, Boost, Python, and Tcl, nested references are an error.

Make a Donation

Page URL: https://www.regular-expressions.info/backref2.html
Page last updated: 22 November 2019
Site last updated: 08 April 2020