NAME

PCRE - Perl-compatible regular expressions

PCRE MATCHING ALGORITHMS

This document describes the two different algorithms that are available in PCRE for matching a compiled regular expression against a given subject string. The "standard" algorithm is the one provided by the **pcre_exec()**, **pcre16_exec()** and **pcre32_exec()** functions. These work in the same as as Perl's matching function, and provide a Perl-compatible matching operation. The just-in-time (JIT) optimization that is described in the **pcrejit** documentation is compatible with these functions.

An alternative algorithm is provided by the **pcre_dfa_exec()**, **pcre16_dfa_exec()** and **pcre32_dfa_exec()** functions; they operate in a different way, and are not Perl-compatible. This alternative has advantages and disadvantages compared with the standard algorithm, and these are described below.

When there is only one possible way in which a given subject string can match a pattern, the two algorithms give the same answer. A difference arises, however, when there are multiple possibilities. For example, if the pattern

```
^<.*>
```

is matched against the string

```
<something> <something else> <something further>
```

there are three possible answers. The standard algorithm finds only one of them, whereas the alternative algorithm finds all three.

REGULAR EXPRESSIONS AS TREES

The set of strings that are matched by a regular expression can be represented as a tree structure. An unlimited repetition in the pattern makes the tree of infinite size, but it is still a tree. Matching the pattern to a given subject string (from a given starting point) can be thought of as a search of the tree. There are two ways to search a tree: depth-first and breadth-first, and these correspond to the two matching algorithms provided by PCRE.

THE STANDARD MATCHING ALGORITHM

In the terminology of Jeffrey Friedl's book "Mastering Regular Expressions", the standard algorithm is an "NFA algorithm". It conducts a depth-first search of the pattern tree. That is, it proceeds along a single path through the tree, checking that the subject matches what is required. When there is a mismatch, the algorithm tries any alternatives at the current point, and if they all fail, it backs up to the previous branch point in the tree, and tries the next alternative branch at that level. This often involves backing up (moving to the left) in the subject string as well. The order in which repetition branches are tried is controlled by the greedy or ungreedy nature of the quantifier.

If a leaf node is reached, a matching string has been found, and at that point the algorithm stops. Thus, if there is more than one possible match, this algorithm returns the first one that it finds. Whether this is the shortest, the longest, or some intermediate length depends on the way the greedy and ungreedy repetition quantifiers are specified in the pattern.

Because it ends up with a single path through the tree, it is relatively straightforward for this algorithm to keep track of the substrings that are matched by portions of the pattern in parentheses. This provides support for capturing parentheses and back references.

THE ALTERNATIVE MATCHING ALGORITHM

This algorithm conducts a breadth-first search of the tree. Starting from the first matching point in the subject, it scans the subject string from left to right, once, character by character, and as it does this, it

remembers all the paths through the tree that represent valid matches. In Friedl's terminology, this is a kind of "DFA algorithm", though it is not implemented as a traditional finite state machine (it keeps multiple states active simultaneously).

Although the general principle of this matching algorithm is that it scans the subject string only once, without backtracking, there is one exception: when a lookaround assertion is encountered, the characters following or preceding the current point have to be independently inspected.

The scan continues until either the end of the subject is reached, or there are no more unterminated paths. At this point, terminated paths represent the different matching possibilities (if there are none, the match has failed). Thus, if there is more than one possible match, this algorithm finds all of them, and in particular, it finds the longest. The matches are returned in decreasing order of length. There is an option to stop the algorithm after the first match (which is necessarily the shortest) is found.

Note that all the matches that are found start at the same point in the subject. If the pattern

```
cat(er(pillar)?)?
```

is matched against the string "the caterpillar catchment", the result will be the three strings "caterpillar", "cater", and "cat" that start at the fifth character of the subject. The algorithm does not automatically move on to find matches that start at later positions.

PCRE's "auto-possessification" optimization usually applies to character repeats at the end of a pattern (as well as internally). For example, the pattern "a\d+" is compiled as if it were "a\d++" because there is no point even considering the possibility of backtracking into the repeated digits. For DFA matching, this means that only one possible match is found. If you really do want multiple matches in such cases, either use an ungreedy repeat ("a\d+?") or set the PCRE_NO_AUTO_POSSESS option when compiling.

There are a number of features of PCRE regular expressions that are not supported by the alternative matching algorithm. They are as follows:

1. Because the algorithm finds all possible matches, the greedy or ungreedy nature of repetition quantifiers is not relevant. Greedy and ungreedy quantifiers are treated in exactly the same way. However, possessive quantifiers can make a difference when what follows could also match what is quantified, for example in a pattern like this:

```
^a++\w!
```

This pattern matches "aaab!" but not "aaa!", which would be matched by a non-possessive quantifier. Similarly, if an atomic group is present, it is matched as if it were a standalone pattern at the current point, and the longest match is then "locked in" for the rest of the overall pattern.

2. When dealing with multiple paths through the tree simultaneously, it is not straightforward to keep track of captured substrings for the different matching possibilities, and PCRE's implementation of this algorithm does not attempt to do this. This means that no captured substrings are available.

3. Because no substrings are captured, back references within the pattern are not supported, and cause errors if encountered.

4. For the same reason, conditional expressions that use a backreference as the condition or test for a specific group recursion are not supported.

5. Because many paths through the tree may be active, the \K escape sequence, which resets the start of the match when encountered (but may be on some paths and not on others), is not supported. It causes an error if encountered.

6. Callouts are supported, but the value of the *capture_top* field is always 1, and the value of the *capture_last* field is always -1.

7. The \C escape sequence, which (in the standard algorithm) always matches a single data unit, even in UTF-8, UTF-16 or UTF-32 modes, is not supported in these modes, because the alternative algorithm moves through the subject string one character (not data unit) at a time, for all active paths through the tree.

8. Except for (*FAIL), the backtracking control verbs such as (*PRUNE) are not supported. (*FAIL) is supported, and behaves like a failing negative assertion.

ADVANTAGES OF THE ALTERNATIVE ALGORITHM

Using the alternative matching algorithm provides the following advantages:

1. All possible matches (at a single point in the subject) are automatically found, and in particular, the longest match is found. To find more than one match using the standard algorithm, you have to do kludgy things with callouts.

2. Because the alternative algorithm scans the subject string just once, and never needs to backtrack (except for lookbehinds), it is possible to pass very long subject strings to the matching function in several pieces, checking for partial matching each time. Although it is possible to do multi-segment matching using the standard algorithm by retaining partially matched substrings, it is more complicated. The **pcrepartial** documentation gives details of partial matching and discusses multi-segment matching.

DISADVANTAGES OF THE ALTERNATIVE ALGORITHM

The alternative algorithm suffers from a number of disadvantages:

1. It is substantially slower than the standard algorithm. This is partly because it has to search for all possible matches, but is also because it is less susceptible to optimization.

2. Capturing parentheses and back references are not supported.

3. Although atomic groups are supported, their use does not provide the performance advantage that it does for the standard algorithm.

AUTHOR

Philip Hazel
University Computing Service
Cambridge CB2 3QH, England.

REVISION

Last updated: 12 November 2013