NAME

PCRE2 - Perl-compatible regular expressions (revised API)

PCRE2 MATCHING ALGORITHMS

This document describes the two different algorithms that are available in PCRE2 for matching a compiled regular expression against a given subject string. The "standard" algorithm is the one provided by the **pcre2_match()** function. This works 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 **pcre2jit** documentation is compatible with this function.

An alternative algorithm is provided by the **pcre2_dfa_match()** function; it operates in a different way, and is 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 PCRE2.

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 alternations and the greedy or 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 backreferences.

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 the output vector 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 the size of vector needed to contain all the results depends on the number of simultaneous matches, not on the number of parentheses in the pattern. Using **pcre2_match_data_create_from_pattern()** to create the match data block is therefore not advisable when doing DFA matching.

Note also 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 is 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.

PCRE2'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 PCRE2_NO_AUTO_POSSESS option when compiling.

There are a number of features of PCRE2 regular expressions that are not supported or behave differently in the alternative matching function. Those that are not supported cause an error if encountered.

1. Because the algorithm finds all possible matches, the greedy or ungreedy nature of repetition quantifiers is not relevant (though it may affect auto-possessification, as just described). During matching, 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 PCRE2'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, backreferences within the pattern are not supported.

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. Again for the same reason, script runs are not supported.

6. 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.

7. Callouts are supported, but the value of the *capture_top* field is always 1, and the value of the *capture_last* field is always 0.

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

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

10. The PCRE2_MATCH_INVALID_UTF option for **pcre2_compile()** is not supported by **pcre2_dfa_match()**.

ADVANTAGES OF THE ALTERNATIVE ALGORITHM

The main advantage of the alternative algorithm is that 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 at the same point using the standard algorithm, you have to do kludgy things with callouts.

Partial matching is possible with this algorithm, though it has some limitations. The **pcre2partial** 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, backreferences, script runs, and matching within invalid UTF string are not supported.

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

4. JIT optimization is not supported.

AUTHOR

Philip Hazel
Retired from University Computing Service
Cambridge, England.

REVISION

Last updated: 28 August 2021