NAME

PCRE - Perl-compatible regular expressions

PARTIAL MATCHING IN PCRE

In normal use of PCRE, if the subject string that is passed to a matching function matches as far as it goes, but is too short to match the entire pattern, PCRE_ERROR_NOMATCH is returned. There are circumstances where it might be helpful to distinguish this case from other cases in which there is no match.

Consider, for example, an application where a human is required to type in data for a field with specific formatting requirements. An example might be a date in the form *ddmmmyy*, defined by this pattern:

```
^\d?\d(jan|feb|mar|apr|may|jun|jul|aug|sep|oct|nov|dec)\d\d$
```

If the application sees the user's keystrokes one by one, and can check that what has been typed so far is potentially valid, it is able to raise an error as soon as a mistake is made, by beeping and not reflecting the character that has been typed, for example. This immediate feedback is likely to be a better user interface than a check that is delayed until the entire string has been entered. Partial matching can also be useful when the subject string is very long and is not all available at once.

PCRE supports partial matching by means of the PCRE_PARTIAL_SOFT and PCRE_PARTIAL_HARD options, which can be set when calling any of the matching functions. For backwards compatibility, PCRE_PARTIAL is a synonym for PCRE_PARTIAL_SOFT. The essential difference between the two options is whether or not a partial match is preferred to an alternative complete match, though the details differ between the two types of matching function. If both options are set, PCRE_PARTIAL_HARD takes precedence.

If you want to use partial matching with just-in-time optimized code, you must call **pcre_study()**, **pcre16_study()** or **pcre32_study()** with one or both of these options:

```
PCRE_STUDY_JIT_PARTIAL_SOFT_COMPILE
PCRE_STUDY_JIT_PARTIAL_HARD_COMPILE
```

PCRE_STUDY_JIT_COMPILE should also be set if you are going to run non-partial matches on the same pattern. If the appropriate JIT study mode has not been set for a match, the interpretive matching code is used.

Setting a partial matching option disables two of PCRE's standard optimizations. PCRE remembers the last literal data unit in a pattern, and abandons matching immediately if it is not present in the subject string. This optimization cannot be used for a subject string that might match only partially. If the pattern was studied, PCRE knows the minimum length of a matching string, and does not bother to run the matching function on shorter strings. This optimization is also disabled for partial matching.

PARTIAL MATCHING USING pcre_exec() OR pcre[16|32]_exec()

A partial match occurs during a call to **pcre_exec()** or **pcre[16|32]_exec()** when the end of the subject string is reached successfully, but matching cannot continue because more characters are needed. However, at least one character in the subject must have been inspected. This character need not form part of the final matched string; lookbehind assertions and the \K escape sequence provide ways of inspecting characters before the start of a matched substring. The requirement for inspecting at least one character exists because an empty string can always be matched; without such a restriction there would always be a partial match of an empty string at the end of the subject.

If there are at least two slots in the offsets vector when a partial match is returned, the first slot is set to the offset of the earliest character that was inspected. For convenience, the second offset points to the end of the subject so that a substring can easily be identified. If there are at least three slots in the offsets vector, the third slot is set to the offset of the character where matching started.

For the majority of patterns, the contents of the first and third slots will be the same. However, for patterns

that contain lookbehind assertions, or begin with \b or \B, characters before the one where matching started may have been inspected while carrying out the match. For example, consider this pattern:

```
/(?<=abc)123/
```

This pattern matches "123", but only if it is preceded by "abc". If the subject string is "xyzabc12", the first two offsets after a partial match are for the substring "abc12", because all these characters were inspected. However, the third offset is set to 6, because that is the offset where matching began.

What happens when a partial match is identified depends on which of the two partial matching options are set.

PCRE_PARTIAL_SOFT WITH pcre_exec() OR pcre[16|32]_exec()

If PCRE_PARTIAL_SOFT is set when **pcre_exec()** or **pcre[16|32]_exec()** identifies a partial match, the partial match is remembered, but matching continues as normal, and other alternatives in the pattern are tried. If no complete match can be found, PCRE_ERROR_PARTIAL is returned instead of PCRE_ERROR_NOMATCH.

This option is "soft" because it prefers a complete match over a partial match. All the various matching items in a pattern behave as if the subject string is potentially complete. For example, \z, \Z, and $ match at the end of the subject, as normal, and for \b and \B the end of the subject is treated as a non-alphanumeric.

If there is more than one partial match, the first one that was found provides the data that is returned. Consider this pattern:

```
/123\w+X|dogY/
```

If this is matched against the subject string "abc123dog", both alternatives fail to match, but the end of the subject is reached during matching, so PCRE_ERROR_PARTIAL is returned. The offsets are set to 3 and 9, identifying "123dog" as the first partial match that was found. (In this example, there are two partial matches, because "dog" on its own partially matches the second alternative.)

PCRE_PARTIAL_HARD WITH pcre_exec() OR pcre[16|32]_exec()

If PCRE_PARTIAL_HARD is set for **pcre_exec()** or **pcre[16|32]_exec()**, PCRE_ERROR_PARTIAL is returned as soon as a partial match is found, without continuing to search for possible complete matches. This option is "hard" because it prefers an earlier partial match over a later complete match. For this reason, the assumption is made that the end of the supplied subject string may not be the true end of the available data, and so, if \z, \Z, \b, \B, or $ are encountered at the end of the subject, the result is PCRE_ERROR_PARTIAL, provided that at least one character in the subject has been inspected.

Setting PCRE_PARTIAL_HARD also affects the way UTF-8 and UTF-16 subject strings are checked for validity. Normally, an invalid sequence causes the error PCRE_ERROR_BADUTF8 or PCRE_ERROR_BADUTF16. However, in the special case of a truncated character at the end of the subject, PCRE_ERROR_SHORTUTF8 or PCRE_ERROR_SHORTUTF16 is returned when PCRE_PARTIAL_HARD is set.

Comparing hard and soft partial matching

The difference between the two partial matching options can be illustrated by a pattern such as:

```
/dog(sbody)?/
```

This matches either "dog" or "dogsbody", greedily (that is, it prefers the longer string if possible). If it is matched against the string "dog" with PCRE_PARTIAL_SOFT, it yields a complete match for "dog". However, if PCRE_PARTIAL_HARD is set, the result is PCRE_ERROR_PARTIAL. On the other hand, if the pattern is made ungreedy the result is different:

```
/dog(sbody)??/
```

In this case the result is always a complete match because that is found first, and matching never continues after finding a complete match. It might be easier to follow this explanation by thinking of the two patterns like this:

```
/dog(sbody)?/    is the same as  /dogsbody|dog/
/dog(sbody)??/   is the same as  /dog|dogsbody/
```

The second pattern will never match "dogsbody", because it will always find the shorter match first.

PARTIAL MATCHING USING pcre_dfa_exec() OR pcre[16|32]_dfa_exec()

The DFA functions move along the subject string character by character, without backtracking, searching for all possible matches simultaneously. If the end of the subject is reached before the end of the pattern, there is the possibility of a partial match, again provided that at least one character has been inspected.

When PCRE_PARTIAL_SOFT is set, PCRE_ERROR_PARTIAL is returned only if there have been no complete matches. Otherwise, the complete matches are returned. However, if PCRE_PARTIAL_HARD is set, a partial match takes precedence over any complete matches. The portion of the string that was inspected when the longest partial match was found is set as the first matching string, provided there are at least two slots in the offsets vector.

Because the DFA functions always search for all possible matches, and there is no difference between greedy and ungreedy repetition, their behaviour is different from the standard functions when PCRE_PARTIAL_HARD is set. Consider the string "dog" matched against the ungreedy pattern shown above:

```
/dog(sbody)??/
```

Whereas the standard functions stop as soon as they find the complete match for "dog", the DFA functions also find the partial match for "dogsbody", and so return that when PCRE_PARTIAL_HARD is set.

PARTIAL MATCHING AND WORD BOUNDARIES

If a pattern ends with one of sequences \b or \B, which test for word boundaries, partial matching with PCRE_PARTIAL_SOFT can give counter-intuitive results. Consider this pattern:

```
/\bcat\b/
```

This matches "cat", provided there is a word boundary at either end. If the subject string is "the cat", the comparison of the final "t" with a following character cannot take place, so a partial match is found. However, normal matching carries on, and \b matches at the end of the subject when the last character is a letter, so a complete match is found. The result, therefore, is *not* PCRE_ERROR_PARTIAL. Using PCRE_PARTIAL_HARD in this case does yield PCRE_ERROR_PARTIAL, because then the partial match takes precedence.

FORMERLY RESTRICTED PATTERNS

For releases of PCRE prior to 8.00, because of the way certain internal optimizations were implemented in the **pcre_exec()** function, the PCRE_PARTIAL option (predecessor of PCRE_PARTIAL_SOFT) could not be used with all patterns. From release 8.00 onwards, the restrictions no longer apply, and partial matching with can be requested for any pattern.

Items that were formerly restricted were repeated single characters and repeated metasequences. If PCRE_PARTIAL was set for a pattern that did not conform to the restrictions, **pcre_exec()** returned the error code PCRE_ERROR_BADPARTIAL (-13). This error code is no longer in use. The PCRE_INFO_OKPARTIAL call to **pcre_fullinfo()** to find out if a compiled pattern can be used for partial matching now always returns 1.

EXAMPLE OF PARTIAL MATCHING USING PCRETEST

If the escape sequence \P is present in a **pcretest** data line, the PCRE_PARTIAL_SOFT option is used for the match. Here is a run of **pcretest** that uses the date example quoted above:

```
  re> /^\d?\d(jan|feb|mar|apr|may|jun|jul|aug|sep|oct|nov|dec)\d\d$/
data> 25jun04\P
 0: 25jun04
 1: jun
data> 25dec3\P
Partial match: 23dec3
data> 3ju\P
Partial match: 3ju
data> 3juj\P
No match
data> j\P
No match
```

The first data string is matched completely, so **pcretest** shows the matched substrings. The remaining four strings do not match the complete pattern, but the first two are partial matches. Similar output is obtained if DFA matching is used.

If the escape sequence \P is present more than once in a **pcretest** data line, the PCRE_PARTIAL_HARD option is set for the match.

MULTI-SEGMENT MATCHING WITH pcre_dfa_exec() OR pcre[16|32]_dfa_exec()

When a partial match has been found using a DFA matching function, it is possible to continue the match by providing additional subject data and calling the function again with the same compiled regular expression, this time setting the PCRE_DFA_RESTART option. You must pass the same working space as before, because this is where details of the previous partial match are stored. Here is an example using **pcretest**, using the \R escape sequence to set the PCRE_DFA_RESTART option (\D specifies the use of the DFA matching function):

```
  re> /^\d?\d(jan|feb|mar|apr|may|jun|jul|aug|sep|oct|nov|dec)\d\d$/
data> 23ja\P\D
Partial match: 23ja
data> n05\R\D
 0: n05
```

The first call has "23ja" as the subject, and requests partial matching; the second call has "n05" as the subject for the continued (restarted) match. Notice that when the match is complete, only the last part is shown; PCRE does not retain the previously partially-matched string. It is up to the calling program to do that if it needs to.

That means that, for an unanchored pattern, if a continued match fails, it is not possible to try again at a new starting point. All this facility is capable of doing is continuing with the previous match attempt. In the previous example, if the second set of data is "ug23" the result is no match, even though there would be a match for "aug23" if the entire string were given at once. Depending on the application, this may or may not be what you want. The only way to allow for starting again at the next character is to retain the matched part of the subject and try a new complete match.

You can set the PCRE_PARTIAL_SOFT or PCRE_PARTIAL_HARD options with PCRE_DFA_RESTART to continue partial matching over multiple segments. This facility can be used to pass very long subject strings to the DFA matching functions.

MULTI-SEGMENT MATCHING WITH pcre_exec() OR pcre[16|32]_exec()

From release 8.00, the standard matching functions can also be used to do multi-segment matching. Unlike the DFA functions, it is not possible to restart the previous match with a new segment of data. Instead, new data must be added to the previous subject string, and the entire match re-run, starting from the point where the partial match occurred. Earlier data can be discarded.

It is best to use PCRE_PARTIAL_HARD in this situation, because it does not treat the end of a segment as the end of the subject when matching \z, \Z, \b, \B, and $. Consider an unanchored pattern that matches dates:

```
  re> /\d?\d(jan|feb|mar|apr|may|jun|jul|aug|sep|oct|nov|dec)\d\d/
data> The date is 23ja\P\P
Partial match: 23ja
```

At this stage, an application could discard the text preceding "23ja", add on text from the next segment, and call the matching function again. Unlike the DFA matching functions, the entire matching string must always be available, and the complete matching process occurs for each call, so more memory and more processing time is needed.

Note: If the pattern contains lookbehind assertions, or \K, or starts with \b or \B, the string that is returned for a partial match includes characters that precede the start of what would be returned for a complete match, because it contains all the characters that were inspected during the partial match.

ISSUES WITH MULTI-SEGMENT MATCHING

Certain types of pattern may give problems with multi-segment matching, whichever matching function is used.

1. If the pattern contains a test for the beginning of a line, you need to pass the PCRE_NOTBOL option when the subject string for any call does start at the beginning of a line. There is also a PCRE_NOTEOL option, but in practice when doing multi-segment matching you should be using PCRE_PARTIAL_HARD, which includes the effect of PCRE_NOTEOL.

2. Lookbehind assertions that have already been obeyed are catered for in the offsets that are returned for a partial match. However a lookbehind assertion later in the pattern could require even earlier characters to be inspected. You can handle this case by using the PCRE_INFO_MAXLOOKBEHIND option of the **pcre_fullinfo()** or **pcre[16|32]_fullinfo()** functions to obtain the length of the longest lookbehind in the pattern. This length is given in characters, not bytes. If you always retain at least that many characters before the partially matched string, all should be well. (Of course, near the start of the subject, fewer characters may be present; in that case all characters should be retained.)

From release 8.33, there is a more accurate way of deciding which characters to retain. Instead of subtracting the length of the longest lookbehind from the earliest inspected character (*offsets[0]*), the match start position (*offsets[2]*) should be used, and the next match attempt started at the *offsets[2]* character by setting the *startoffset* argument of **pcre_exec()** or **pcre_dfa_exec()**.

For example, if the pattern "(?<=123)abc" is partially matched against the string "xx123a", the three offset values returned are 2, 6, and 5. This indicates that the matching process that gave a partial match started at offset 5, but the characters "123a" were all inspected. The maximum lookbehind for that pattern is 3, so taking that away from 5 shows that we need only keep "123a", and the next match attempt can be started at offset 3 (that is, at "a") when further characters have been added. When the match start is not the earliest inspected character, **pcretest** shows it explicitly:

```
  re> "(?<=123)abc"
data> xx123a\P\P
Partial match at offset 5: 123a
```

3. Because a partial match must always contain at least one character, what might be considered a partial

match of an empty string actually gives a "no match" result. For example:

```
  re> /c(?<=abc)x/
data> ab\P
No match
```

If the next segment begins "cx", a match should be found, but this will only happen if characters from the previous segment are retained. For this reason, a "no match" result should be interpreted as "partial match of an empty string" when the pattern contains lookbehinds.

4. Matching a subject string that is split into multiple segments may not always produce exactly the same result as matching over one single long string, especially when PCRE_PARTIAL_SOFT is used. The section "Partial Matching and Word Boundaries" above describes an issue that arises if the pattern ends with \b or \B. Another kind of difference may occur when there are multiple matching possibilities, because (for PCRE_PARTIAL_SOFT) a partial match result is given only when there are no completed matches. This means that as soon as the shortest match has been found, continuation to a new subject segment is no longer possible. Consider again this **pcretest** example:

```
  re> /dog(sbody)?/
data> dogsb\P
 0: dog
data> do\P\D
Partial match: do
data> gsb\R\P\D
 0: g
data> dogsbody\D
 0: dogsbody
 1: dog
```

The first data line passes the string "dogsb" to a standard matching function, setting the PCRE_PARTIAL_SOFT option. Although the string is a partial match for "dogsbody", the result is not PCRE_ERROR_PARTIAL, because the shorter string "dog" is a complete match. Similarly, when the subject is presented to a DFA matching function in several parts ("do" and "gsb" being the first two) the match stops when "dog" has been found, and it is not possible to continue. On the other hand, if "dogsbody" is presented as a single string, a DFA matching function finds both matches.

Because of these problems, it is best to use PCRE_PARTIAL_HARD when matching multi-segment data. The example above then behaves differently:

```
  re> /dog(sbody)?/
data> dogsb\P\P
Partial match: dogsb
data> do\P\D
Partial match: do
data> gsb\R\P\P\D
Partial match: gsb
```

5. Patterns that contain alternatives at the top level which do not all start with the same pattern item may not work as expected when PCRE_DFA_RESTART is used. For example, consider this pattern:

```
1234|3789
```

If the first part of the subject is "ABC123", a partial match of the first alternative is found at offset 3. There is no partial match for the second alternative, because such a match does not start at the same point in the subject string. Attempting to continue with the string "7890" does not yield a match because only those alternatives that match at one point in the subject are remembered. The problem arises because the start of the

second alternative matches within the first alternative. There is no problem with anchored patterns or patterns such as:

```
1234|ABCD
```

where no string can be a partial match for both alternatives. This is not a problem if a standard matching function is used, because the entire match has to be rerun each time:

```
  re> /1234|3789/
data> ABC123\P\P
Partial match: 123
data> 1237890
 0: 3789
```

Of course, instead of using PCRE_DFA_RESTART, the same technique of re-running the entire match can also be used with the DFA matching functions. Another possibility is to work with two buffers. If a partial match at offset *n* in the first buffer is followed by "no match" when PCRE_DFA_RESTART is used on the second buffer, you can then try a new match starting at offset *n+1* in the first buffer.

AUTHOR

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

REVISION

Last updated: 02 July 2013