Onigmoの非包含オペレータに存在するバグの詳細

tonco-miyazawa 2022/04/05
https://github.com/tonco-miyazawa/regex_etc

2017/01/16 にリリースされた Onigmo バージョン 6.1.0 に搭載された非包含オペレータは “ある文字列を含まない文字列” にマッチするという動作をする画期的なオペレータだが、含んではいけないはずの文字列を含む文字列にマッチしてしまうというバグが存在する。

このバグは現行バージョンのOnigmo 6.2.0 でも発生し、これより少し前のバージョンの Onigmo 6.1.3 を搭載している ruby の現行バージョンの ruby 3.1.1 もこのバグの影響を受ける。

・ruby での簡単な例

```
------------------------------
p /^(?~a{1,2})/.match("aa")
------------------------------
```

結果: #<MatchData "a">

この例では “a{1,2}” がマッチ可能な文字列である “aa” と “a” が含まれない文字列にマッチすべきところだが、“a” にマッチしてしまっている。

この文書ではこのバグの原因となっている内部動作を手短かに解説し、バグが起きる仕組みの理解とバグが起きないパターンの書き方の習得の手助けをする。

【1】 非包含オペレータの内部動作

ここでは解説の都合で極端な例を紹介する。
非包含オペレータに入れる正規表現を “¥d*” とし、検索対象となる文字列を “0123456789” とする。

・ruby でのコード

```
----------------------------
p /(?~¥d*)/.match('0123456789')
----------------------------
```

結果: #<MatchData "01234">

結果を見ると “01234” にマッチしている。なんとも不思議な結果になっているが、バグを抱えた非包含オペレータがどのようにこの結果を導き出したのかというと、その処理の流れは以下のようになっている。

(1) “¥d*” が “0” から “9” までマッチする

対象テキストの先頭から検索が開始される。 “*” は貪欲の繰り返しなので “9” までマッチする。

(2) “0” から “8” までを仮の答えとしてキープする

マッチした文字列から1文字削ればマッチしない文字列になるため、(1) のマッチ結果から最後の1文字である “9” を削除。

(3) “¥d*” が “1” から “8” までマッチする

検索開始位置が1つ進められ、”1” から検索が始まる。
検索対象となる範囲は仮の答えの終端である “8” まで。

(4) “0” から “7” までを仮の答えとしてキープする

最後の1文字を削る。 以降、同じ過程を繰り返していく。

(5) “¥d*” が “2” から “7” までマッチする

検索開始位置が1つ進む。

(6) “0” から “6” までを仮の答えとしてキープする

最後の1文字を削る。

(7) “¥d*” が “3” から “6” までマッチする

検索開始位置が1つ進む。

(8) “0” から “5” までを仮の答えとしてキープする

最後の1文字を削る。

(9) “¥d*” が “4” から “5” までマッチする

検索開始位置が1つ進む。

(10) “0” から “4” までを仮の答えとしてキープする

最後の1文字を削る。

(11) “¥d*” が “4” と “5” の間の “” にマッチする

“5” の位置から検索が始まる順番だが、(10) によって仮の答えは “4” までとなっている。 このため検索は “4” と “5” の間にある空文字列 “” に対して行われ、“¥d*” は “” にマッチする。

(12) “0” から “4” までを仮の答えとしてキープする

(11) ではゼロ幅マッチをしている。ゼロ幅マッチをした場合は仮の答えは1文字削られるのではなく、ゼロ幅マッチをした位置が仮の答えの終端となる。

何故1文字削る必要が無いのかと言うと、“4” と “5” の間の空文字列 “” は仮の答えである “0” から “4” までの範囲には含まれないと見なすことが出来るからだ。

以上の過程により全体の検索が終了し、仮の答えとして残った “01234” が非包含オペレータがマッチする文字列となる。

※ 補足 1

ある検索開始位置からの検索でマッチするものが見つからなかった場合、仮の答えは削られずに処理が次の検索開始位置に進められる。仮の答えはマッチするものが見つかった場合にのみ削られる。

※ 補足 2

仮の答えの初期値は検索対象文字列である。
すべての位置からの検索を通してマッチするものが1つも見つからなかった場合、仮の答えは検索対象文字列のままとなり、検索対象文字列の末端までが非包含オペレータのマッチ可能範囲となる。

※ 補足 3

検索対象文字列の先頭の位置でゼロ幅マッチをした場合、
非包含オペレータは空文字列“”を含むマッチが許されなくなる。
そのため非包含オペレータはマッチに失敗する。

【2】 バグの原因となる処理

ここでは【1】の過程の中の (1) と (2) の処理について解説する。

(1) と (2) はどちらも検索開始位置が先頭の位置で行われる処理であり、この2つの処理で1セットである。この処理が終わると検索開始位置が1つ進められ、2つ目のセットである (3) と (4) に処理が進む。

(1) では“¥d*”がマッチする文字列“0123456789”を見つけ出し、
(2) ではマッチした文字列から最後の1文字である“9”を削って“012345678”を仮の答え、すなわち“¥d* を含まない文字列”として導き出している。しかしこれがおかしいのは明らかだ。

何故なら“012345678”は“¥d* を含まない文字列”ではないからだ。
この矛盾が非包含オペレータのバグの原因となっている。

本来の非包含オペレータの動作は各検索開始位置ごとに
”文字数が最短となるマッチ” を探し出さなくてはならない。
でなければ含んではいけない文字列を含んでしまう可能性があるのだ。

しかし (1) ではあろうことか “文字数が最長となるマッチ” を返してしまっている。そのため “¥d*” がマッチする文字だらけになっているのだ。

このように Onigmo の実装ではマッチするものを一度見つけたらそれが ”文字数が最短となるマッチ” かどうかに関わらず、その検索開始位置からの検索を打ち切り、次の検索開始位置に処理を進めてしまう。

※ 補足3

“¥d*” はゼロ回の繰り返しも許されるので文字数が最短となるマッチは “0123456789” ではなく “” となるのが正しい動作である。

【3】 非包含オペレータの論文

web検索したところ、非包含オペレータの論文は 2008年1月8 〜 10日に箱根ホテル小涌園 (2018年1月10日閉館) で行われた “第49回プログラミング・シンポジウム“ の講演にて論文の著者である 田中 哲 氏によって発表されたもののようだ。

第49回プログラミング・シンポジウム プログラム
https://prosym.org/49/49program.html

情報処理学会 - 電子図書館
https://ipsj.ixsq.nii.ac.jp/ej/index.php?active_action=repository_view_main_item_detail&page_id=13&block_id=8&item_id=91521&item_no=1

非包含オペレータの論文には自分の知る限り以下の2種類がある。

正規表現における非包含オペレータの提案
https://staff.aist.go.jp/tanaka-akira/pub/prosym49-akr-paper.pdf

正規表現における非包含オペレータの提案
https://staff.aist.go.jp/tanaka-akira/pub/prosym49-akr-presen.pdf

後者は講演でスクリーンに映し出すために作ったものだろうか、
本命の論文は前者であるのでここではこちらを使う。

＜ 論文の切り抜き (コピペ) ＞

論文の中で ruby や Onigmo に関わる点を切り抜き、以下にまとめた。

1、Perl, Ruby 等の既存の正規表現エンジンで採用されている
バックトラッキングを用いるアルゴリズム
（ 1ページ 概要 ）

2、既存のバックトラック型の正規表現エンジン上で容易に効率よく
実装でき、かつ、形式言語理論による適切な意味付けが可能な
非包含オペレータ
（ 1ページ 概要 ）

3、 非包含オペレータは記述が容易であり、かつ、正規表現の
組合せの部品として問題なく用いることができる。
（ 7ページ 左側 下部 ）

=> 著者は Perl や Ruby の正規表現の組合せの部品として
非包含オペレータを問題なく用いることが出来ると考えていた。

4、理論的な取扱いも問題のない正規表現の拡張として
非包含オペレータを提案する。
（ 2ページ 左側 中ほど ）

5、 非包含オペレータは形式言語理論との対応に問題がない。
（ 6ページ 左側 中ほど ）

=> 著者は非包含オペレータが形式言語理論から逸脱しないものと
考えていた。

6、バックトラックの抑制はバックトラックの戦略に強く依存するため、
　理論的な扱いに問題が生じる。
（ 7ページ 右側 中ほど ）

7、否定先読みは明らかに理論から逸脱している。
（ 8ページ 左側 下部 ）

=> バックトラックの抑制や否定先読みには理論的な扱いに問題がある。

その一方で、非包含オペレータは理論的な扱いに問題が無いと
しているので非包含オペレータの内側にバットクラックの抑制や
否定先読みを入れることは論文では想定されていない。

しかし、Onigmo は非包含オペレータの内側にこれらを入れて
検索が行える。また、Onigmo には再帰マッチを可能にする
“部分呼び出し” など他にも理論から逸脱するものが存在する。

=> これらを使った場合、Onigmo の非包含オペレータは形式言語
理論から逸脱してしまう。 これらを非包含オペレータに入れて
検索した場合の動作については論文では言及されていない。

【4】 論文のサンプルコード

この論文には簡単な正規表現エンジンのサンプルコードが記載されて
おり、ruby 上で動かすことが出来る。これについての説明は論文に
分かりやすく書いてあるのでそちらを読んで頂きたい。

＜ サンプルコードの説明文の切り抜き (コピペ) ＞

その説明の中で特に重要な点を切り抜き、まとめた。

1、try が正規表現エンジンの中心となる手続きである。
これは文字列中の特定の位置で始まる部分文字列で
あたえられた正規表現から導出可能なものをすべて検出する。
（3ページ 左側 下部）

2、try により、ある文字列のある位置から始まる部分文字列で、
ある正規表現から導出可能なものをすべて得ることができる。
（3ページ 右側 中ほど）

3、(a|b)* という正規表現を “aba” という文字列の先頭から適用し、
正規表現から導出可能な部分文字列 “aba”, “ab”, “a”, “” の
各終端となる 3, 2, 1, 0 を順に表示する。
（3ページ 右側 下部）

=> サンプルコードは “aba”, “ab”, “a”, “” の4つのマッチを見つける。
それに対し、Onigmo の非包含オペレータは最初にマッチする
“aba” しか見つけない。 この違いが Onigmo のバグの原因である。
これは検索開始位置が先頭の位置での話である。

4、複数の箇所が見つかる可能性があるが、各箇所の右端の中で
最も左にある位置を求める。
（5ページ 右側 中ほど）

=> サンプルコードではある検索開始位置からの検索でマッチが複数
見つかる場合を想定しており、その中から最短一致を見つける。

5、その最左位置のさらにひとつ左 (limit) までの文字列には
r から導出可能な文字列は含まれていない。
（5ページ 右側 中ほど）

=> 最短一致でマッチした文字列から最後の1文字を削った文字列は
正規表現 r にマッチする文字列を含まない。

【5】 Onigmo の非包含オペレータは実用可能

Onigmo の非包含オペレータにはバグがあるものの、実用可能だ。
バグの原因となっている“文字数が最短となるマッチを見逃す”という
事態が起こらないような使い方をすれば期待通りに動作してくれるのだ。

非包含オペレータが期待通りに動作する条件には以下のようなものがある。
この条件の中のどれか1つを満たせば期待通りに動作する。

＜ バグが起きない条件 ＞

[条件1]
マッチする文字列の文字数が固定長となる正規表現を使う時
=> すべてのマッチが最短一致のマッチとなるため見逃しがない

(ただし、異なる検索開始位置同士のマッチであればマッチする
文字数が異なることが許される。)

[条件2]
各検索開始位置からの検索で、文字数の異なる複数通りの
マッチが不可能な時
=> 1通りのマッチだけが可能ならそれが最短一致のマッチとなる

[条件3]
各検索開始位置からの検索で最初に見つかるマッチが必ず
その位置での最短一致となる正規表現を使うとき

[条件4]
バグを発生させるマッチが起こらない検索対象文字列が用意された時
=> 最短一致でないマッチがマッチ出来ないならばバグは起こらない。

■ 条件1 に該当する正規表現

例えば C言語のコメントにマッチする正規表現の場合、非包含オペレータに入れる正規表現は “¥*¥/” である。 これにマッチする文字列は “*/” の2文字で固定長である。そのため常に最短一致となりバグが起こらない。

¥/¥*(?~¥*¥/)¥*¥/

・ruby でのサンプルコード

```
----------------------------------
p '/*  /*  */  */  /*  */  */'.gsub(/¥/¥*(?~¥*¥/)¥*¥//, '[¥0]')
----------------------------------
```

・結果

```
[/*  /*  */] */  [/*  */] */
```

=> 2つのコメントアウトに正しくマッチしている。

※ 補足 4

Onigmo の非包含オペレータは改行文字を跨いだマッチが可能だ。
“.” とは違い、/m オプションを有効にしなくても複数行にマッチする。

・ruby でのサンプルコード

```
----------------------------------
p /¥A(?~abc)¥z/.match("1¥n2¥n3")
----------------------------------
```

・結果

```
#<MatchData "1¥n2¥n3">
```

=> 非包含オペレータがマッチした文字列に改行文字が含まれている。

■ 条件2 に該当する正規表現

(?~12345|234) はバグの起きない正規表現である。
“12345“ がマッチする位置では “234” は絶対にマッチ出来ない。

“12345“ がマッチするためには検索開始位置からの1文字目が
“1” である必要があるが、”234” がマッチするためには 1文字目は
“2” でなければならない。

つまりどちらかがマッチする検索開始位置ではもう一方は必ず
マッチに失敗する。そのため各位置での複数通りのマッチは
不可能となり、最短一致でないマッチは起こり得ない。

■ 条件3 に該当する正規表現

(?~123|1234) は “1234” がマッチ可能な位置では必ず “123” が最初に
マッチするので各検索開始位置からのマッチは必ず最短一致となる。

=> 両者の位置を逆にしただけでバグの起きる正規表現になってしまう。

■ 条件4 に該当する正規表現

(?~12345|123) は検索対象文字列が “123456” の場合、
先頭の位置からのマッチで “12345” が最初にマッチするため、
同じ位置でマッチ可能な “123” を見逃してバグが発生する。
よってこれはバグの起きる正規表現だ。

しかし、“12345” が含まれない検索対象文字列に対して検索する場合
であれば “12345” にマッチすることは不可能だ。そのため “123” が
マッチするケースを見逃すことが無くなり、バグが発生しなくなる。

このように非包含オペレータのバグは内側に入れる正規表現だけでなく
検索対象となる文字列にも左右される。

【6】HTMLタグへの実用例

　最後に［条件2］に該当する実用例を紹介する。
この例では非包含オペレータの内側には“<¥/?div¥b”を入れる。
この“<¥/?div¥b”は“<div”と“</div”の2通りのマッチが可能な正規表現だが、この2つは同じ検索開始位置からの検索で両者がマッチ可能になることは無いのでバグの起きない正規表現である。

　HTMLには DIV タグや TABLEタグのように入れ子が可能なタグが存在する。 子要素を持たない末端のタグの組み合わせにマッチさせたい場合には非包含オペレータを利用すると便利である。

・検索対象となるHTMLソース

```
----------------------------------
<div>AAA</div>
<div>
  <div>
    <div>BBB</div>
  </div>
  <div>CCC</div>
  <div>DDD</div>
</div>
----------------------------------
```

このHTMLソースの中から AAA、BBB、CCC、DDD を挟むDIVタグの組み合わせにマッチするサンプルコードが以下のもの。

・ ruby でのサンプルコード

```
----------------------------------
html = <<EOS
<div>AAA</div>
<div>
  <div>
    <div>BBB</div>
  </div>
```

```
  <div>CCC</div>
  <div>DDD</div>
</div>
EOS

p html.gsub( /<div>(?~<¥/?div¥b)<¥/div>/ , '[¥0]')
--------------------------------
```

・結果

```
--------------------------------
"[<div>AAA</div>]¥n<div>¥n  <div>¥n      [<div>BBB</div>]¥n  </div>¥n
[<div>CCC</div>]¥n  [<div>DDD</div>]¥n</div>¥n"
--------------------------------
```

=> 期待通りに入れ子末端のタグの組み合わせにマッチしている。
    / .*<¥/div>/ などと違い誤爆する可能性が無いので便利だ。

※ 補足 5

このサンプルコードでは非包含オペレータの中に “¥b” を入れている。
これは形式言語理論から逸脱する “先読み” や “戻り読み” の性質を
持つメタ文字だが、今まで試した限りでは問題のある動作をしたことは
無いのでこのサンプルコードでも使用している。

※ 補足 6

“¥d” や “¥w” や “¥b” などはオプションによってマッチするものが
変わる油断ならないメタ文字だ。使用の際は必ずオプションの確認を。

Onigmo RE.ja : 孤立オプション
https://github.com/k-takata/Onigmo/blob/master/doc/RE.ja#L237

【7】 まとめ

　Onigmo の非包含オペレータにはバグがあるが、固定長のマッチをする正規表現を中に入れる場合は安全に使うことが出来る。

だが、可変長のマッチとなる正規表現を中に入れる場合はバグを
回避するための制約が大きく、実用的に使える場面は限られてしまう。

バグを回避する方法は理解が難しく、あまり正規表現になじみの無い
人が使うにはリスクが高すぎると言わざるを得ない。
そのためほとんどの人は perl5 などで使われている否定先読みを使った
従来の方法である (?:(?!abc).)* を使うべきである。

　また、箱根ホテル小涌園は非包含オペレータの講演が行われた
シンポジウムの最終日からちょうど10年後に閉館している、バルス。

最後に、論文の著者である田中 哲 氏、Onigmo の作者である K-Takata 氏、
oniguruma の作者である K.Kosako 氏 のお三方の多大なる貢献に感謝。

おわり

【 余談 】

　もし非包含オペレータの講演の映像を持っている方がいらっしゃい
ましたら良かったら頂けないでしょうか、大変貴重な映像だと思うので
拝見させて頂けたら嬉しいです。よろしくお願いします。

ついでに非包含オペレータの論文のパロディのつもりで作ったネタを。

正規表現における1行定義オペレータの提案
https://github.com/tonco-miyazawa/regex_etc/blob/master/OneLineDefineOperator.txt

提案から実装まで9年かかっちゃう?!　2030年をお楽しみに!!