正規表現における
非包含オペレータの提案

田中 哲
akr@fsij.org
産業技術総合研究所 情報技術研究部門

問題: C言語のコメントにマッチする正規表現を記述せよ

- /* と */ で括られた文字列
- ただし内部に */ を含まない

正解 (1)

```
\/\*[^\*]*\*+([^\*\/][^\*]*\*+)*\/
```

正解 (2)

```
\/\*((?!\*\/).)*\*\/
```

正解 (3)

```
\/\*(?>.*?\*\/)
```

近似解

```
\/\*.*?\*\/
```

類似の問題

- CR LF で終わる行
- Python の """...""" 文字列
- HTML/XML の <![CDATA[...]]>
- HTTP の CR LF CR LF によるヘッダ終端
- SMTP の CR LF . CR LF によるボディ終了
- etc.

終端文字列を含まない文字列	終端文字列

現状

- 普通のひとはどの正解にもたどりつけない
- 近似解はよく使われる
- 質問があると、だいたい近似解が紹介される

なぜこんな状況に?

- まともに正規表現を構成するのは難しすぎる
  `\/*[^*]**+([^*\/][^*]**+)*\/`
- Perl5 の機能を使うとかなりまし
  正規表現エンジンの挙動の理解が必要
  文字列の集合という考え方では済まない
  `\/*((?!*\/).)**\/`
  `\/*(?>.*?*\/)`
- それなりでよければそんなに難しくない
  でも、他の正規表現とつなげると問題が出る
  `\/*.*?*\/`

この状況をどうにかしたい

- まともなものは難し過ぎる
- Perl5 のは理論に合わない
  正規表現エンジンの挙動まで考える必要がある
- それなりじゃなくてちゃんと動いてほしい

提案

- 非包含オペレータ !r
- r にマッチする文字列を含まない文字列にマッチ
- 終端文字列にマッチする r を与えれば終端文字列を含まない文字列を表現できる
  - `!(*\/)`
  - `!(\]\]>)`
  - `!(\r\n)`
  - `!(""")`
  - `!(\r\n\r\n)`
  - `!(\r\n\.\r\n)`

C 言語のコメント

- /* と */ で括られた文字列
- ただし内部に */ は含まれない

```
\/\*!(\*\/)
 \*\/
```

非包含オペレータ

利点

- 一般的な正規表現エンジンに組み込みやすい
  一般的: バックトラッキング型正規表現エンジン
  DFA はあまり使われない
- 文字列の集合という考え方ですむ
  形式言語理論から逸脱しない
  Perl5 のは逸脱している

バックトラッキング型正規表現エンジン

- バックトラックしながら正規表現と文字列の対応を探索する
- 利点
  - 部分正規表現がどこにマッチしたかわかる
    Perl の $1, $2, ...
- 欠点
  - 最悪時間計算量が指数関数的
- 利用例: ed, sed, grep, perl, ruby, etc.

決定性有限オートマトン: DFA

- 形式言語理論を知っていれば普通 DFA を考える
- 利点
  - 時間計算量が文字列長に対して線形
  - 空間計算量が文字列長に対して定数
- 欠点
  - 部分正規表現がどこにマッチしたのかわからない
    Perl の $1, $2, ... が実装困難
    致命的
  - 正規表現サイズに対して空間・時間計算量が最悪指数関数的
- 利用例: egrep, awk

非包含オペレータの実装

- バックトラッキング型正規表現エンジンの拡張
- Ruby によるサンプル実装を行った

非包含オペレータの使用例

```
try([:cat, [:lit, "/"],
     [:cat, [:lit, "*"],
      [:cat, [:absent,
              [:cat,
               [:lit, "*"],
               [:lit, "/"]]],
       [:cat, [:lit, "*"],
        [:lit, "/"]]]]]],
["/", "*", "*", "/"], 0) {|pos|
  p pos }
```

正規表現エンジンの使用法

- `try(regexp, str, begpos) {|endpos|`
  ```
    ...
  }
  ```
- 文字列 str 内の begpos から右に regexp のマッチを試し、成功するたびにマッチが終わった位置を引数としてブロックを呼び出す
- regexp は配列で組み上げた抽象構文木
- str は1文字からなる文字列の配列

正規表現エンジンの実装: try

```
def try(re, str, pos, &b)
  case re[0]
  when :empseq; try_empseq(str, pos, &b)
  when :lit; _, ch = re; try_lit(ch, str, pos, &b)
  when :cat; _, r1, r2 = re; try_cat(r1, r2, str, pos, &b)
  when :alt; _, r1, r2 = re; try_alt(r1, r2, str, pos, &b)
  when :rep; _, r = re; try_rep(r, str, pos, &b)
  when :absent; _, r = re; try_absent(r, str, pos, &b)
  end
end
```

空文字列: try_empseq

```
def try_empseq(str, pos)
  yield pos
end
```

文字リテラル: try_lit

```
def try_lit(ch, str, pos)
  if pos < str.length && str[pos] == ch
    yield pos + 1
  end
end
```

連接: try_cat

```
def try_cat(r1, r2, str, pos, &block)
  try(r1, str, pos) {|pos2|
    try(r2, str, pos2, &block)
  }
end
```

選択: try_alt

```
def try_alt(r1, r2, str, pos, &block)
  try(r1, str, pos, &block)
  try(r2, str, pos, &block)
end
```

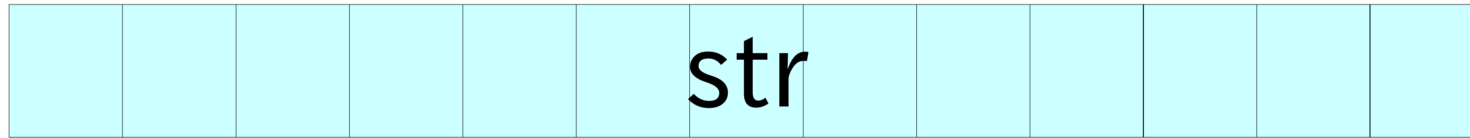
繰り返し: try_rep

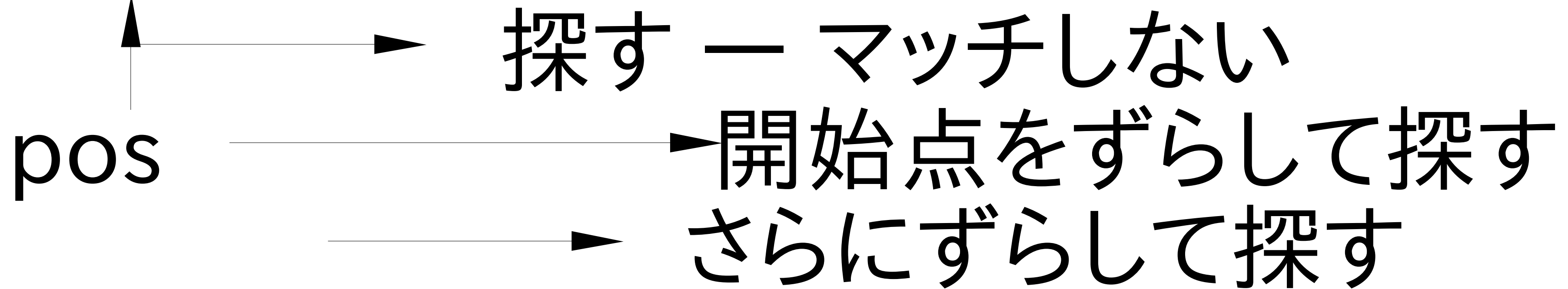
```
def try_rep(r, str, pos, &block)
  try(r, str, pos) {|pos2|
    if pos < pos2
      try_rep(r, str, pos2, &block)
    end
  }
  yield pos
end
```

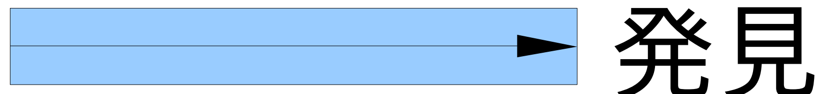
非包含オペレータ: try_absent

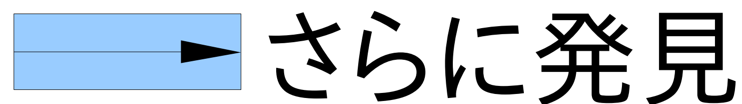
```
def try_absent(r, str, pos)
  limit = str.length; pos2 = pos
  while pos2 <= limit
    try(r, str, pos2) {|pos3|
      limit = pos3-1 if pos3-1 < limit
    }
    pos2 += 1
  end
  limit.downto(pos) {|pos4| yield pos4 }
end
```

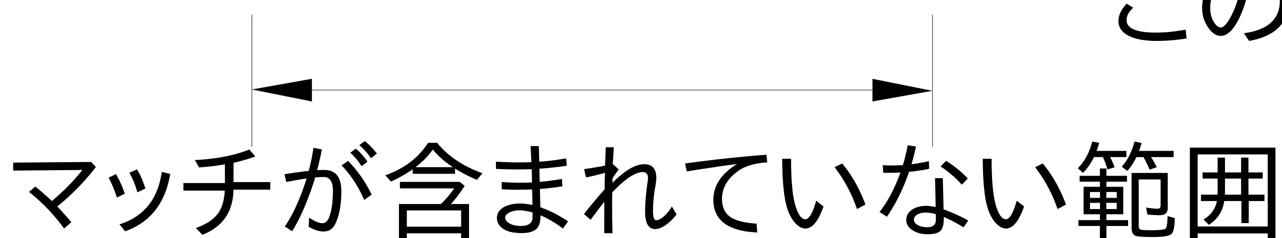
非包含オペレータの動作











非包含オペレータの動作

- 与えられた開始点 pos から右で r を探す
- pos から右へ順に探していく
- ひとつも見つからなければ pos よりも右すべてが非包含オペレータが成功する範囲
- ひとつ以上見つかれば、見つかったものの中で右端が最も左にあるのを選ぶ
- 選んだものの右端の直前までが非包含オペレータが成功する範囲
- すでに見つかったものよりも右の探索は不要

組み込みやすさ

- バックトラッキング型正規表現エンジンに問題なく組み込める
- 補集合は組み込みにくい

理論的な扱いやすさ

- 非包含オペレータは文字列の集合という考え方から逸脱しない
- Perl5 の機能は逸脱している
  - `(?!r)`　直後が `r` にマッチする空文字列にマッチ
  - `(?>r)`　`r` が最初にマッチしたものだけにマッチ

非包含オペレータの意味

- 非包含オペレータが示す言語
  `L(!r)` = Σ^* – `L(.*r.*)`
- ふつうに文字列の集合として定義できる
- 正則集合は補集合について閉じているので `L(!r)` は正則集合

Perl5の否定先読み (?!r)

- r にマッチする文字列が直後にある空文字列にマッチ
- マッチする外に依存するので文字列集合にならない

```
def try_nlookahead(r, str, pos)
  matched = false
  try(r, str, pos) {|pos2|
    matched = true
    break
  }
  yield pos if !matched
end
```

Perl5 のバックトラックの抑制 (?>r)

- r のふたつめ以降のマッチを無視
- 探索の順番に依存するので文字列集合にならない

```
def try_nobacktrack(r, str, pos)
  try(r, str, pos) {|pos2|
    yield pos2
    return
  }
end
```

まとめ

- 正規表現に対する非包含オペレータを提案した
- サンプル実装を行った
- 一般的な正規表現エンジンに組み込みやすい
- 形式言語理論から逸脱していない