テキスト処理 第13回 (2006-07-18)

田中哲
産業技術総合研究所
情報技術研究部門
akr@isc.senshu-u.ac.jp
http://staff.aist.go.jp/tanaka-akira/textprocess/

今日の内容

- 前回のレポートの説明
- backreferenceの機能
- backreferenceの実装
- いろんな数に対するマッチ
- 試験と評価

backreference

- キャプチャした文字列と同じ文字列にマッチする
- キャプチャしていないときにはマッチしない
- \1, \2, \3, ... で、番号がキャプチャの番号を示す
- Ruby 1.9 の名前つきキャプチャに対応する backreference は \k<name>

backreferenceの例

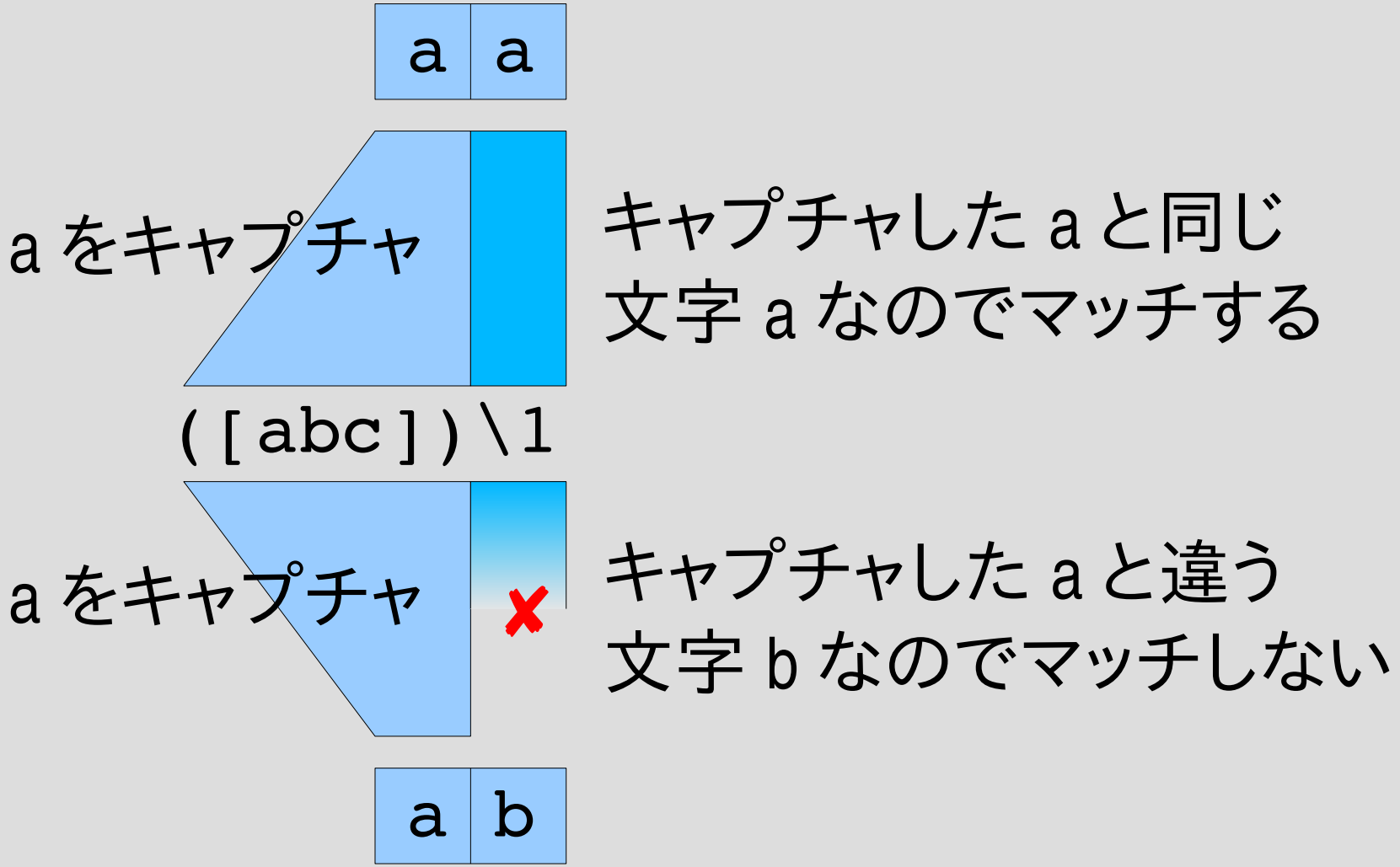
- /([abc])\1/ は aa, bb, cc にマッチする

```
p /([abc])\1/ =~ "aa"   #=> 0
p /([abc])\1/ =~ "bb"   #=> 0
p /([abc])\1/ =~ "ab"   #=> nil
```

- /\1(a)/ は \1 に到達した時点でキャプチャされていないのでマッチしない

```
p /\1(a)/ =~ "aa"       #=> nil
```

backreference の動作

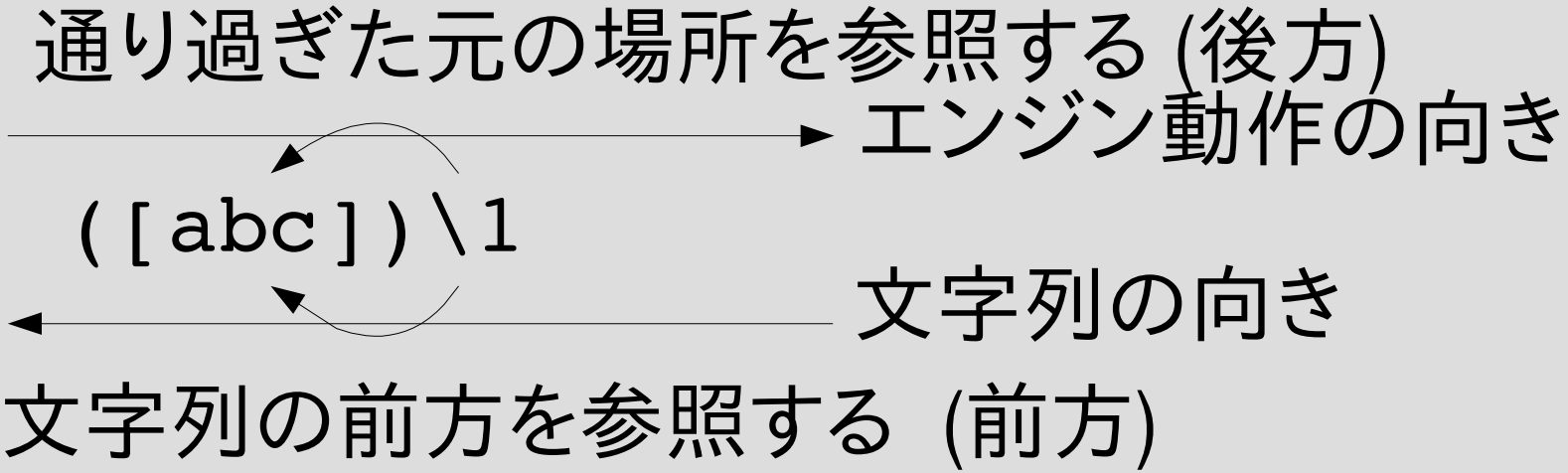


backreference の用途

- "..." と '...' の両方にマッチする
  (内部のエスケープを考えるとあまり使えない)
  /(["']).*\1/
- HTML の対応するタグにマッチする
  (完全には程遠い)
  /<([a-zA-Z0-9_]+)>.*?<\/\1>/

backreference の日本語訳

- 後方参照と訳されることがある
- 前方参照と訳されることもある



backreference の実装

- [:backref, name] で表現する
- Ruby 1.9 の \k<name> に対応する
- キャプチャに番号をつけていないので 番号による backreference (\1, \2, ...) は扱わない

- /(?<n>a)\k<n>/ は以下のように表現する

```
[:cat, [:capture, :n, [:lit, "a"]],
       [:backref, :n]]
```

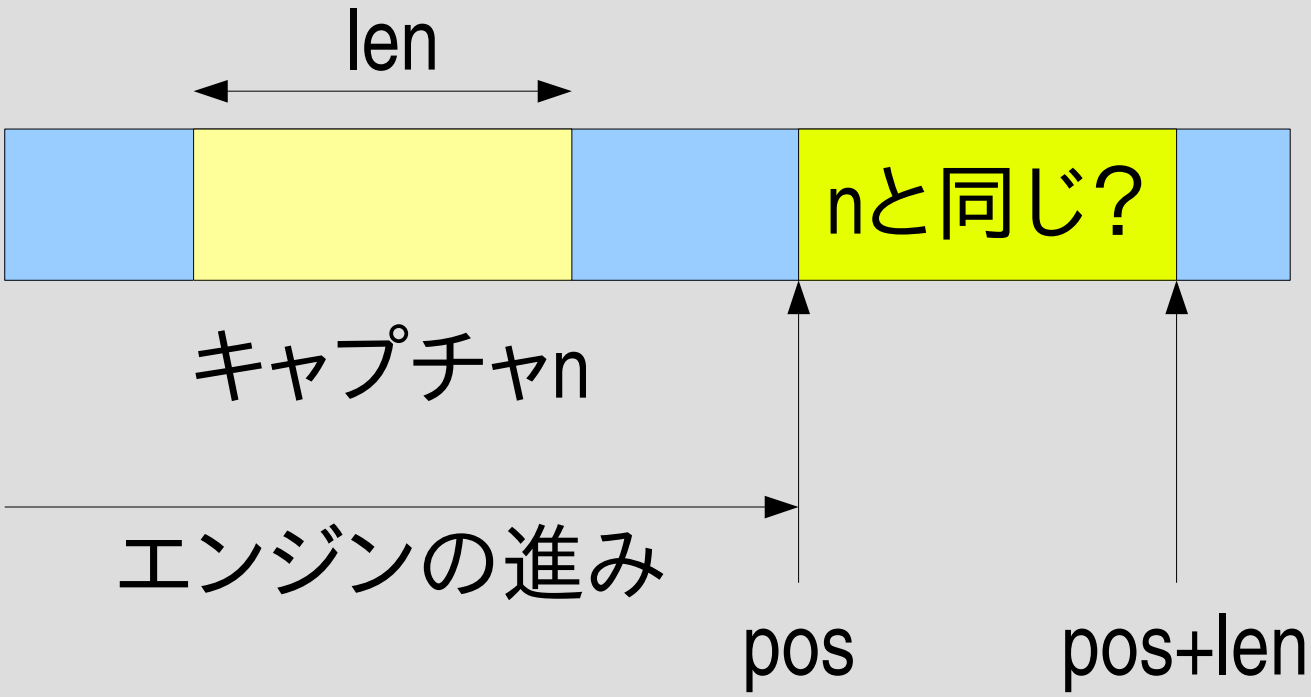
try

- def try(exp, seq, pos, md, &block)
  case exp[0]
  ...
  when :backref
  _, n = exp
  try_backref(n, seq, pos, md, &block)
  ...
  end

try_backref

```
def try_backref(n, seq, pos, md, &block)
  if md[n]
    r = md[n]
    len = r.end - r.begin
    if seq[r] == seq[pos, len]
      yield pos + len, md
    end
  end
end
```

try_backrefの動作



pos からキャプチャと同じ文字列が
続いていれば、
マッチしてその終わりを yield する

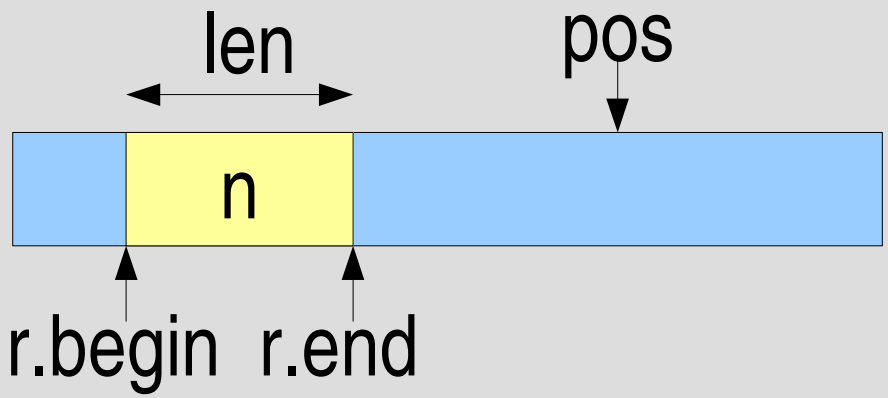
キャプチャされてるかどうか判断

```
def try_backref(n, seq, pos, md, &block)
  if md[n]
    r = md[n]
    len = r.end - r.begin
    if seq[r] == seq[pos, len]
      yield pos + len, md
    end
  end
end
```

キャプチャされてるか?

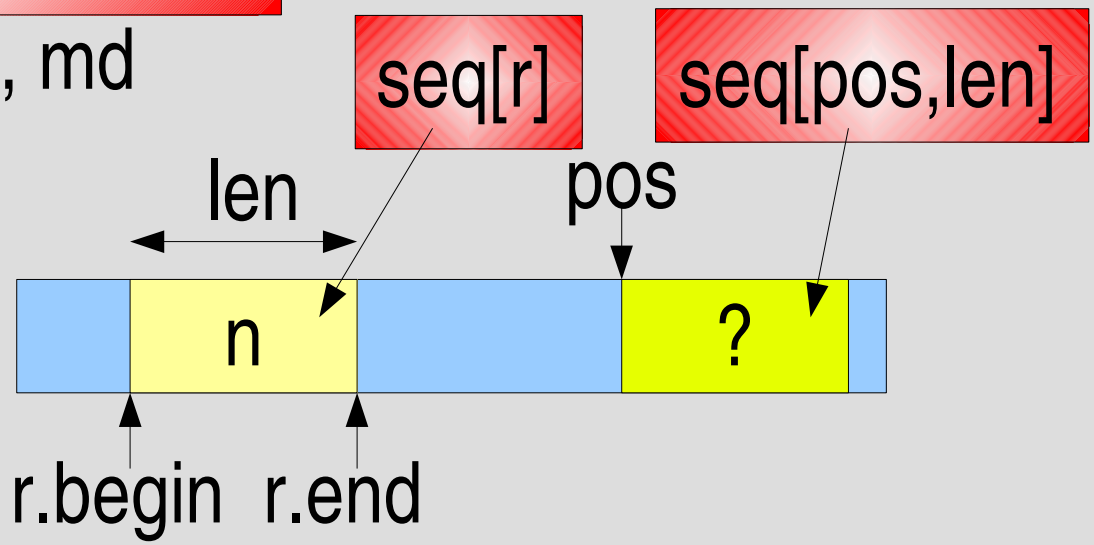
キャプチャの情報を取り出す

```
def try_backref(n, seq, pos, md, &block)
  if md[n]
    r = md[n]
    len = r.end - r.begin
    if seq[r] == seq[pos, len]
      yield pos + len, md
    end
  end
end
```



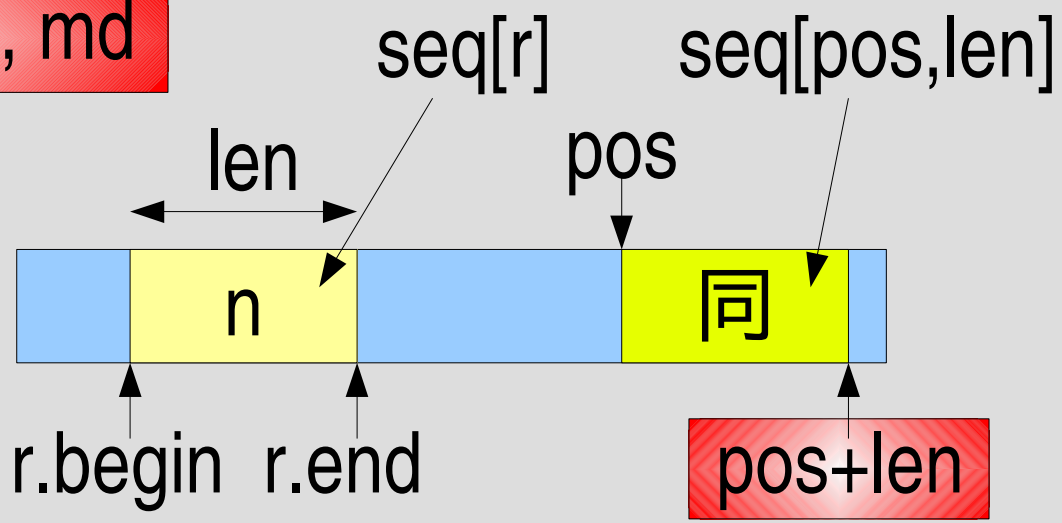
pos 以降がキャプチャと同じか判断

- def try_backref(n, seq, pos, md, &block)
  if md[n]
  r = md[n]
  len = r.end - r.begin
  if seq[r] == seq[pos, len]
  yield pos + len, md
  end
  end
  end



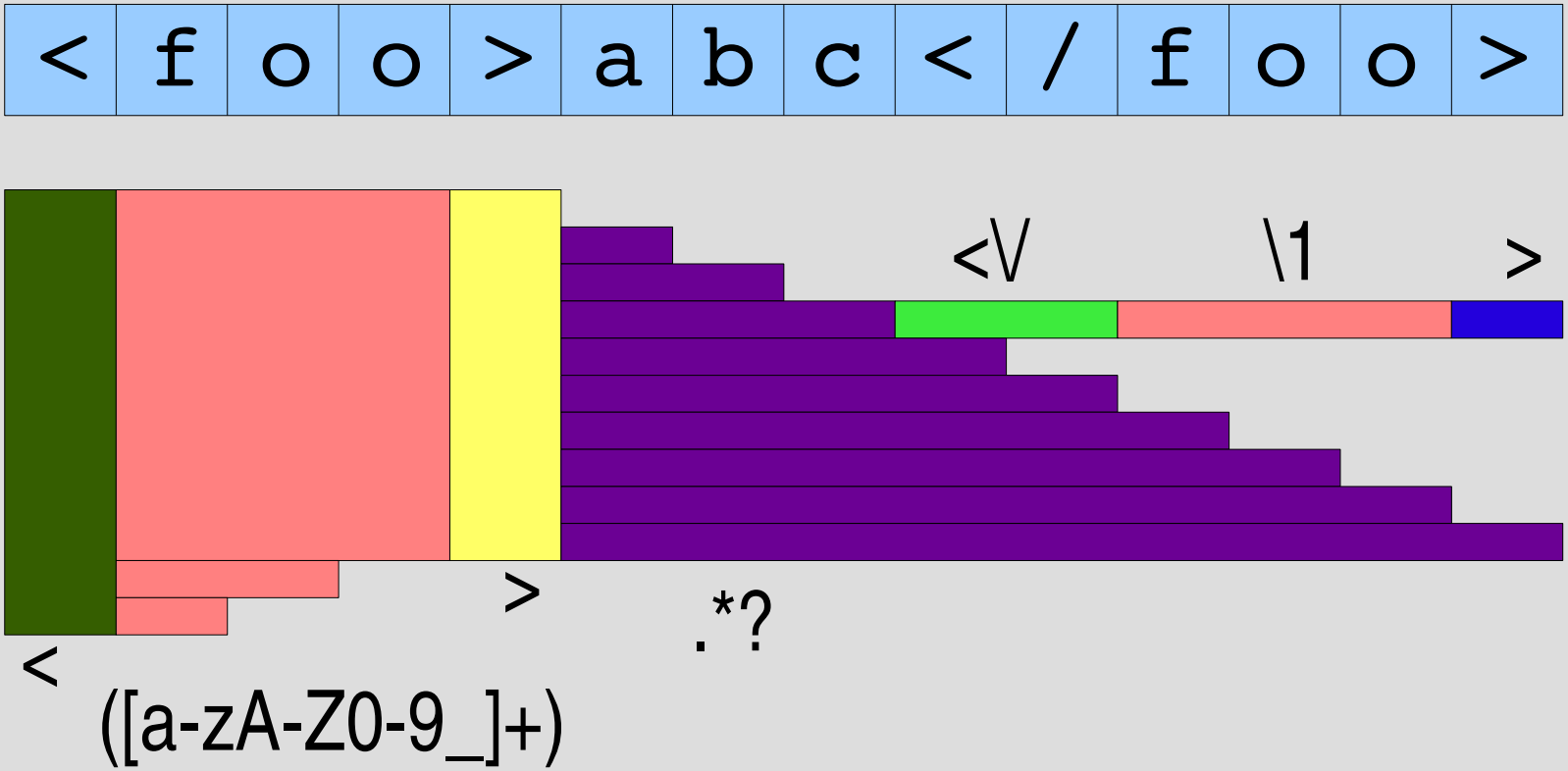
同じだったら終端をyield

```
def try_backref(n, seq, pos, md, &block)
  if md[n]
    r = md[n]
    len = r.end - r.begin
    if seq[r] == seq[pos, len]
      yield pos + len, md
    end
  end
end
```



try_backref の動作例

- /<([a-zA-Z0-9_]+)>.*?<\/\1>/ =~ "<foo>abc</foo>"



空文字列

- rep は空文字列を無視する
- backref を考えると正しくない？
- /()*\1/ =~ "" はマッチするか？
- /(()|()|()|()|())*\2\5/ =~ "" はマッチするか？

try_rep

```
def try_rep(exp, seq, pos, md, &block)
  try(exp, seq, pos, md) {|pos2, md2|
    try_rep(exp, seq, pos2, md2, &block) if pos < pos2 || md != md2
  }
  yield pos, md
end
```

pos==pos2 でも
mdが変わっていたら
続きを試す

倍数にマッチ: 10進数

- 10進数で表現した 2の倍数にマッチ
  /\A[0-9]*[02468]\z/

- 10進数で表現した 5の倍数にマッチ
  /\A[0-9]*[05]\z/

- 10進数で表現した 3の倍数にマッチ: できなくはない
  /\A([0369]|[258][0369]*[147]|([147]|[258][0369]*[258])
  ([0369]|[147][0369]*[258])*([258]|[147][0369]*[147]))*\z/

倍数にマッチ: 文字列の長さ

- 2の倍数の長さの文字列にマッチ
  /\A(00)*\z/
- 5の倍数の長さの文字列にマッチ
  /\A(00000)*\z/
- 3の倍数の長さの文字列にマッチ
  /\A(000)*\z/

- 0 が並んでいる数で数を表す方法を 1進数という流儀がある

合成数にマッチ: 文字列の長さ

- /\A(00+)\1+\z/

合成数

- 二つ以上の素数の積で表せる自然数
- 自然数のうち0でも1でも素数でもない数
- 4=2*2, 6=2*3, 8=2*2*2, 9=3*3, 10=2*5, 12=2*2*3, 14=2*7, 15=3*5, ...

素数

- 1とその数自身以外に正の約数を持たず、1よりも大きな自然数
- 2, 3, 5, 7, 11, 13, ...

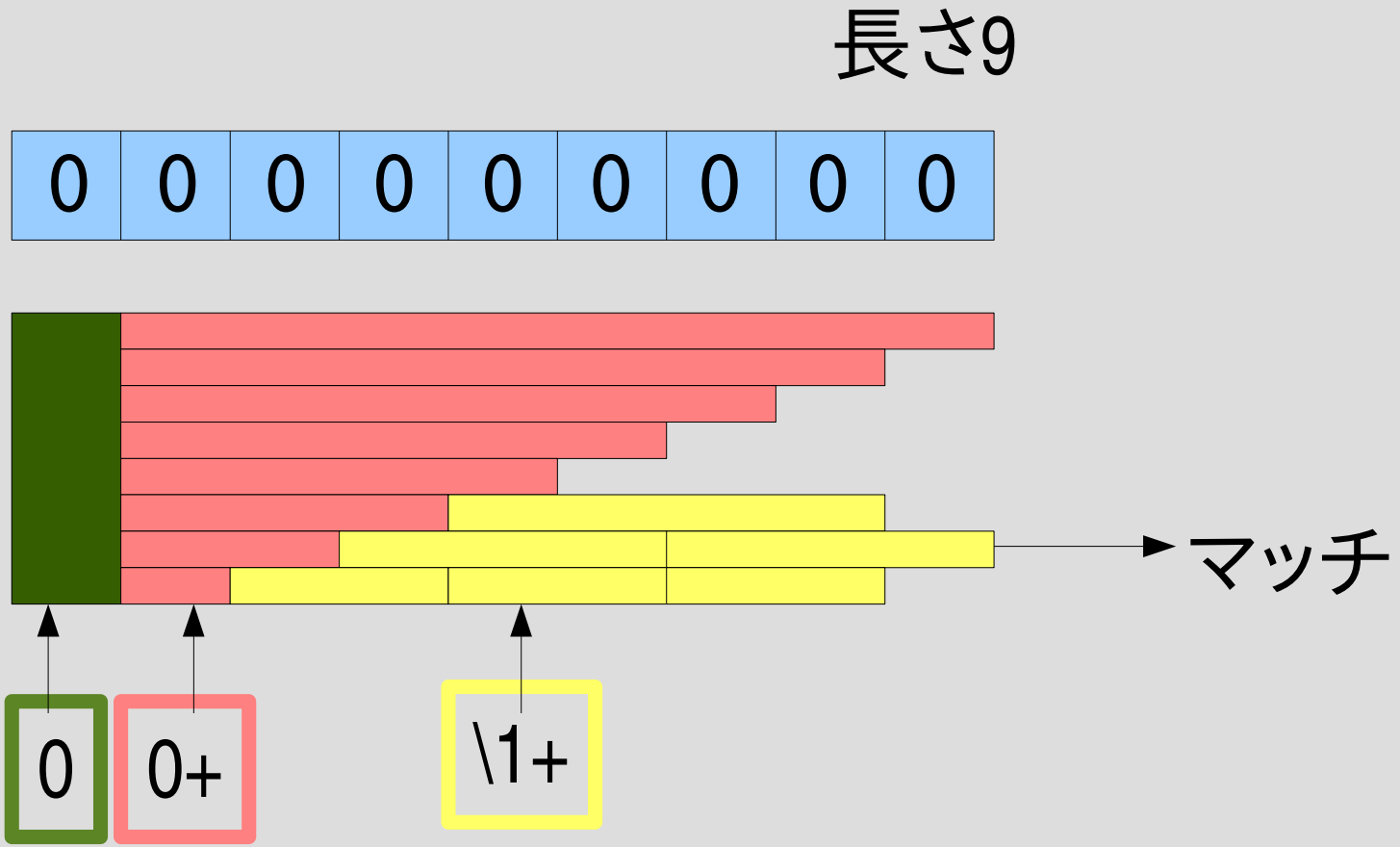
合成数にマッチ: 文字列の長さ

- /\A(00+)\1+\z/

長さ n≥2　　長さ n*l
(l≥1)

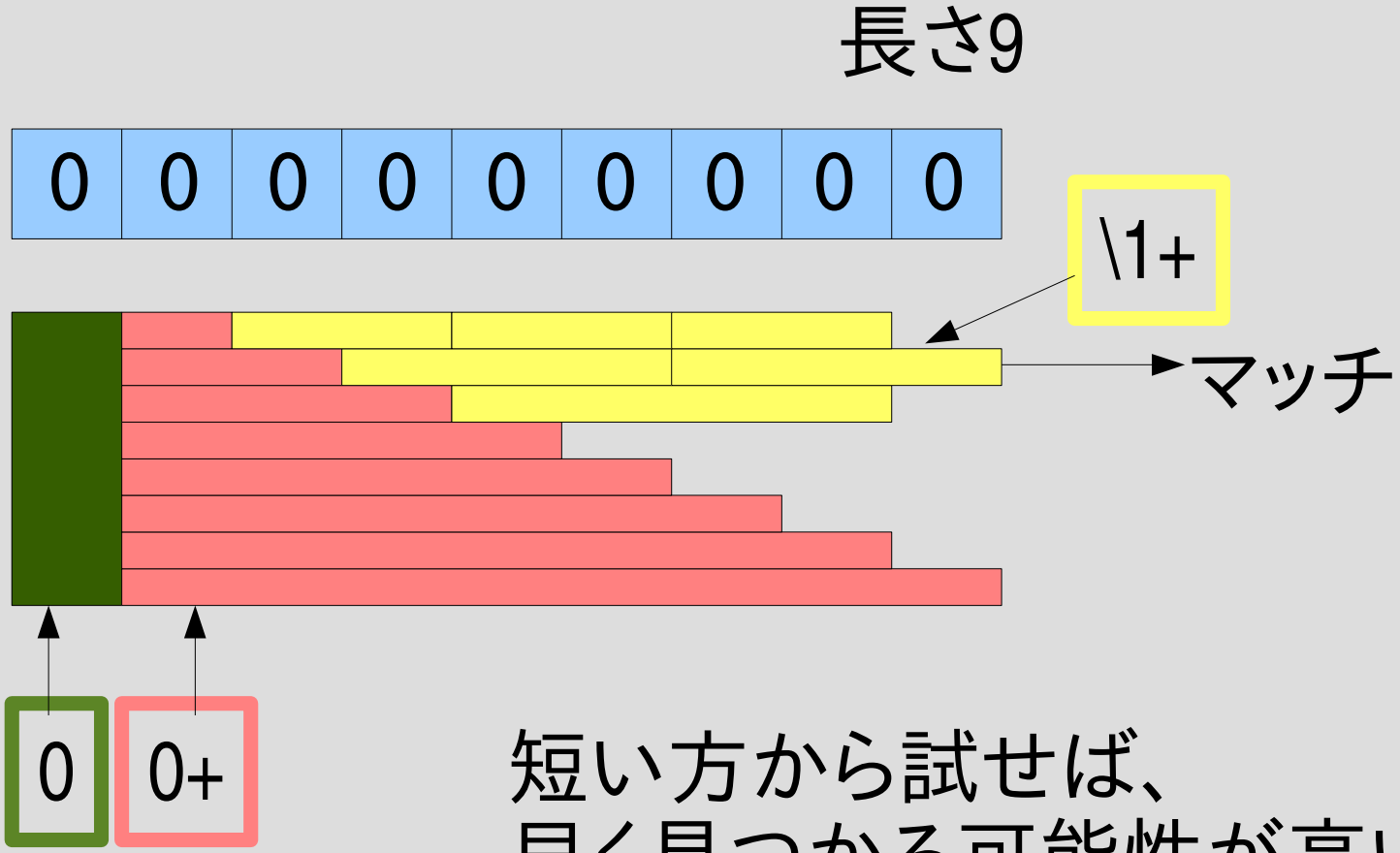
全体の長さ n + n*l = n(1+l)
= nm　　n≥2, m≥2
合成数

/\A(00+)\1+\z/ =~ "000000000"



最初の半分 (上半分) で \1 はマッチしないので無駄

高速化: /\A(00+?)\1+\z/



短い方から試せば、
早く見つかる可能性が高い
とくに 2の倍数なら最初に
見つかる

合成数の列挙

- ```
  0.upto(20) {|n|
    p n if /\A(00+)\1+?\z/ =~ "0" * n
  }
  #=>
  4
  6
  8
  9
  10
  12
  14
  15
  16
  18
  20
  ```
  ```

合成数の列挙

- 0.upto(20) {|n|
  p n if !matchstr(
    [:cat, [:cat, [:capture, :n,
                  [:cat, [:anysym], [:plus_lazy, [:anysym]]]],
                [:plus, [:backref, :n]]],
           [:string_end]],
    "0" * n).empty?
  } #=>
  4 8 10 14 16 20
  6 9 12 15 18

素数の列挙

- 2.upto(20) {|n|
  p n if /\A(00+?)\1+\z/ !~ "0" * n
  }
  #=>
  2
  3
  5
  7
  11
  13
  17
  19

2以上で、かつ、合成数で「ない」

素数にマッチするパターンを
書いたわけではない

素数の列挙

```
2.upto(20) {|n|
  p n if matchstr(
    [:cat, [:cat, [:capture, :n,
                    [:cat, [:anysym], [:plus_lazy, [:anysym]]]],
                  [:plus, [:backref, :n]]],
           [:string_end]],
    "0" * n).empty?
} #=>
2       5       11      17
3       7       13      19
```

試験について

- 2006-07-25 (火) 5時限 60分 (16:30 - 17:30)
- 持込: 一切可
- 学生証を携帯すること

- コンピュータに向かわないでプログラムを書かせたりはしない
- なにかを解説させることになる

- 試験はレポート2回相当程度の予定
- 試験の重みは低いので、レポートをまともに出していない限り単位は難しい

試験問題

1. 正規表現 X を講義で述べた抽象構文木に変換せよ
2. 上で変換した抽象構文木を e として、講義で述べた matchstr を matchstr(e, Y) として実行した場合の結果の値を示せ
3. matchstr(e, Y) 内部の正規表現エンジンの動作を解説せよ

- X, Y は試験および追試で変化する

まとめ

- 前回のレポートの説明
- backreferenceの機能
- backreferenceの実装
- いろんな数に対するマッチ
- 試験について