テキスト処理 第10回 (2007-07-10)

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-2007/

今日の内容

- 前回のレポートの説明
- 再帰を使う拡張
  - 存在するかもしれない: e?
  - 1回以上の繰り返し: e+
  - 怠惰な繰り返し: e*?
  - 存在しないかもしれない: e??
  - m回以上 n回以下の繰り返し: e{m,n}
- レポート

エンジンの拡張: /e?/

- /e?/ は、e がある場合とない場合にマッチする
- e がある場合を先にためし、ない場合を後に試す
- /e|/ と同じ
- [:opt, e] で表現する (optional の意)

- /behaviou?r/ =~ "behavior" #=> 0
- /behaviou?r/ =~ "behaviour" #=> 0

- matchstr([:opt, [:lit, "a"]], "a") #=> [1,0]
- matchstr([:opt, [:lit, "a"]], "b") #=> [0]

[:opt, e] の実装 (1)

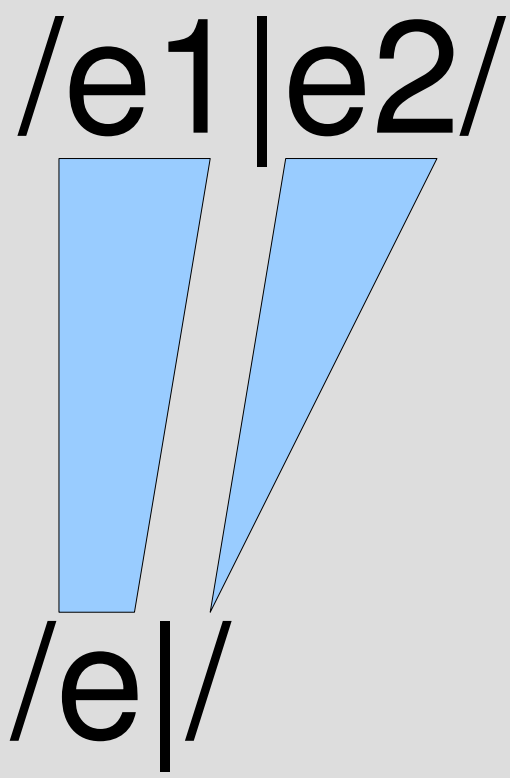
```
def try(exp, seq, pos, &block)
  ...
  when :opt
    _, e = exp
    try_opt(e, seq, pos, &block)
  ...
end
```

[:opt, e] の実装 (2)

```
def try_opt(e, seq, pos, &block)
  try(e, seq, pos, &block)
  yield pos
end
```

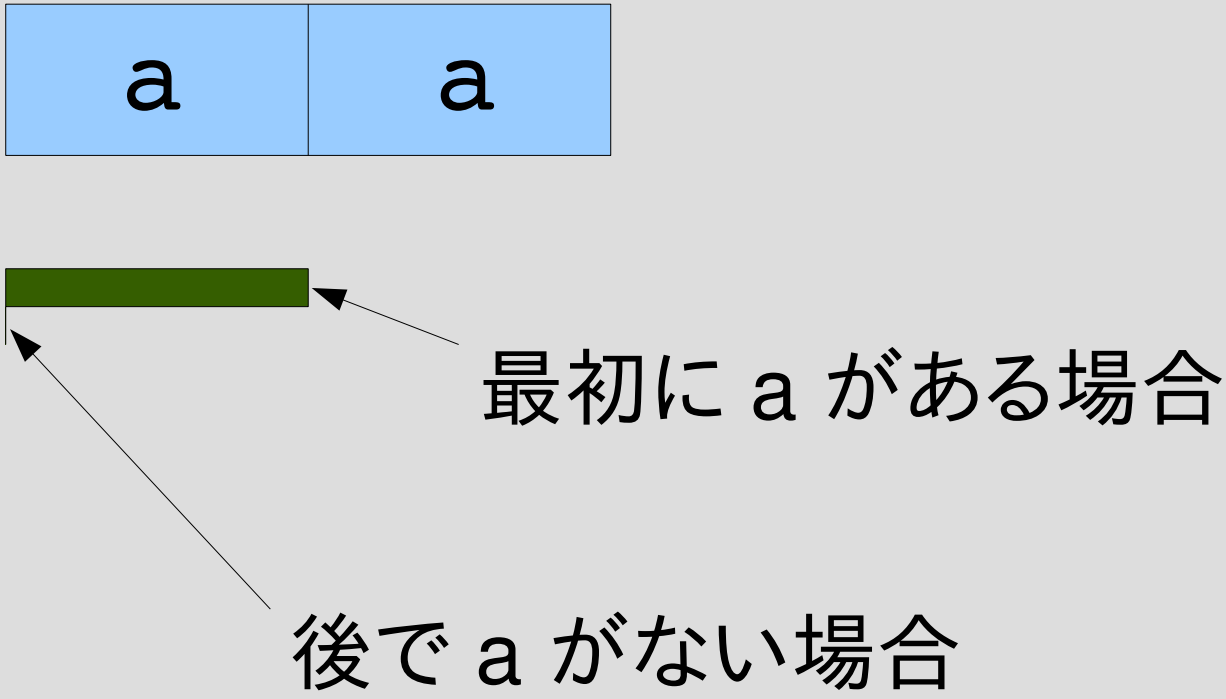
try_alt と try_opt の比較

```
def try_alt(e1, e2, seq, pos, &block)
  try(e1, seq, pos, &block)
  try(e2, seq, pos, &block)
end

def try_opt(e, seq, pos, &block)
  try(e, seq, pos, &block)
  yield pos
end
```

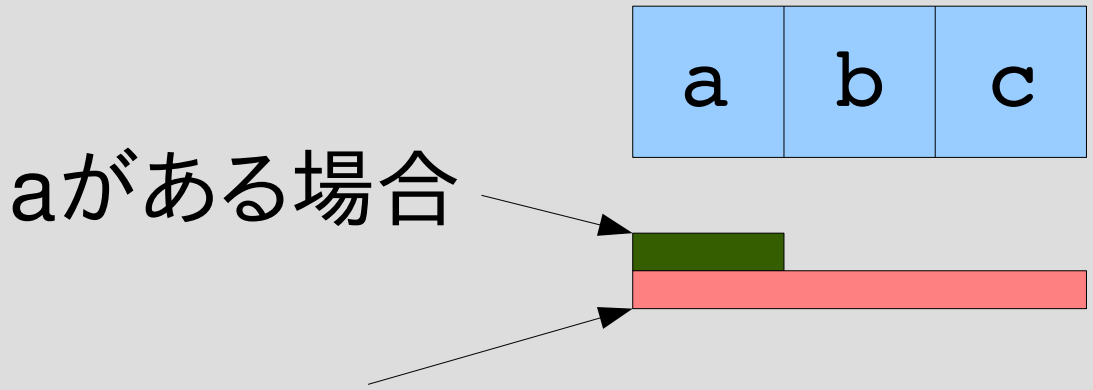


try(//) を展開した形になっている

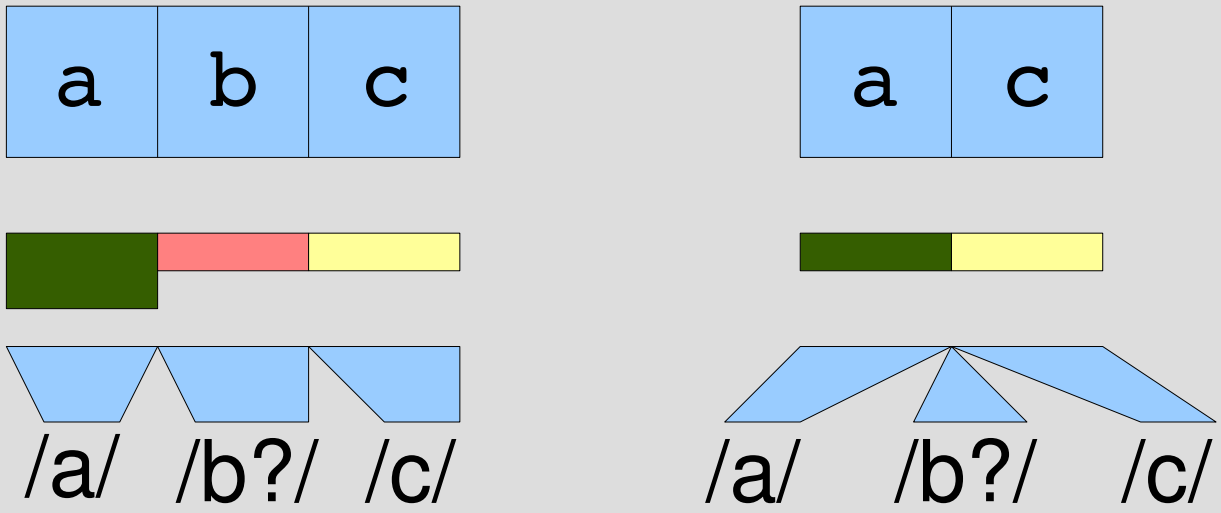
matchstr([:opt, [:lit, "a"]], "aa")



/a?abc/ =~ "abc"



/ab?c/ =~ "abc", /ab?c/ =~ "ac"



エンジンの拡張: /e+/

- /e+/ は、e の 1つ以上の繰り返し
- [:plus, e] で表現する
- /ee*/ と同じ

- /ab+c/ =~ "ac"　　#=> nil
- /ab+c/ =~ "abc"　　#=> 0
- /ab+c/ =~ "abbbc"　　#=> 0

- matchstr([:plus, [:lit, "a"]], "aaa")#=> [3,2,1]
- matchstr([:rep, [:lit, "a"]], "aaa") #=> [3,2,1,0]

[:plus, e] の実装 (1)

```
def try(exp, seq, pos, &block)
  ...
  when :plus
    _, e = exp
    try_plus(e, seq, pos, &block)
  ...
end
```

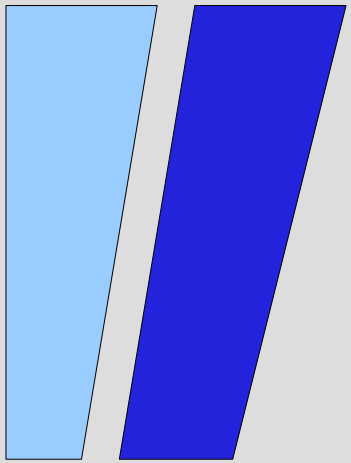
[:plus, e] の実装 (2)

```
def try_plus(e, seq, pos, &block)
  try(e, seq, pos) {|pos2|
    try_rep(e, seq, pos2, &block)
  }
end
```

try_cat と try_plus の比較

```
def try_cat(e1, e2, seq, pos, &block)
  try(e1, seq, pos) {|pos2|
    try(e2, seq, pos, &block)
  }
end
def try_plus(e, seq, pos, &block)
  try(e, seq, pos) {|pos2|
    try_rep(e, seq, pos2, &block)
  }
end
```

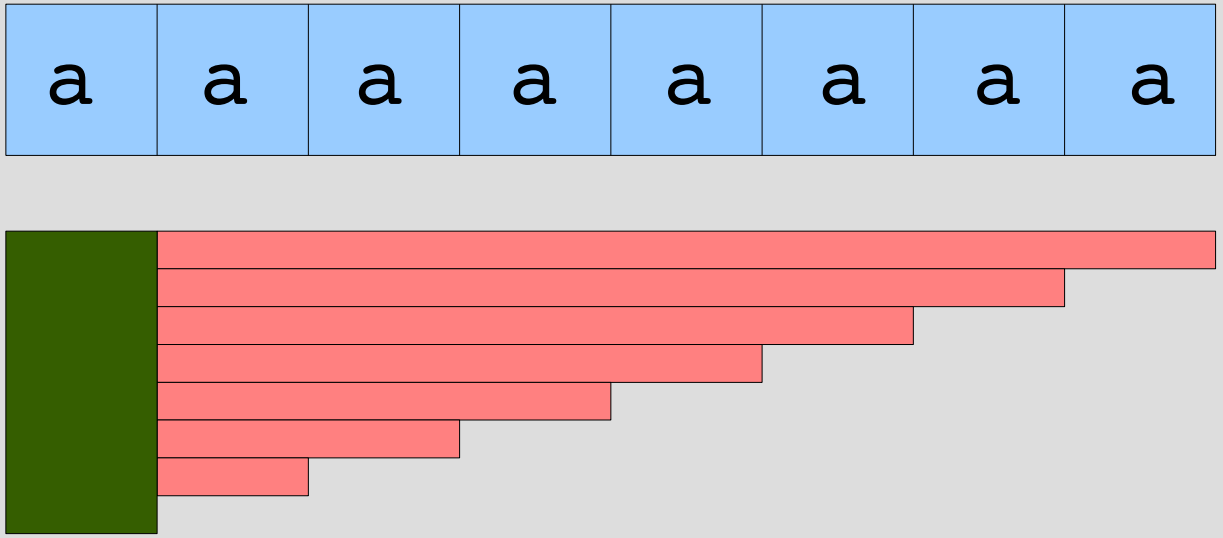
/e1e2/



/ee*/

try(/e*/) を展開した
構造になっている

/a+/ の動作



/a*/

/a/

エンジンの拡張: /e*?/

- /e*?/ は、e の 0個以上の繰り返し
- [:rep_lazy, e] で表現する
- e* とは逆に、少ない繰り返しから試す
- いままでの組合せでは表現できない

- matchstr([:rep_lazy, [:lit, "a"]], "aaa")
  #=> [0,1,2,3]
- matchstr([:rep, [:lit, "a"]], "aaa")
  #=> [3,2,1,0]

e*? と (e*)? の違い

- e*? は [:rep_lazy, e]
- (e*)? は [:opt, [:rep, e]]
- *? はひとつの機能で、* と ? の組合せではない

lazy

- a* は繰り返しが多い場合から続きを試す
  この順序を greedy (貪欲) という
  とりあえずたくさん食べてみる、というイメージ

- a*? は繰り返しが少ない場合から続きを試す
  この順序を lazy (怠惰) もしくは nongreedy (非貪欲) という
  なるべくなら食べないで済ます、というイメージ

- 最終的にはすべて試すのでマッチするかどうかが変わることはない
  (マッチする場所とかは変わるかもしれない)

e*? の用途 (1)

- C のコメントを取り出す
  - /¥/¥*.*?¥*¥// =~ "ab /* ccc */ de /* xxx */"
  - /¥/¥*.*¥*¥// =~ "ab /* ccc */ de /* xxx */"

*? でなく * を使うと
複数のコメントにマッチしてしまう

/¥/¥*[^¥*]*¥*+([^¥/¥*][^¥*]*¥*+)*¥// とすれば
*? を使わなくても書ける (むしろ正しいが、難しい)

e*? の用途 (2)

- HTML のタグの対を取り出すのにも使われる
  - /.*?<¥/b>/ =~
    "aabbbcccdddee"
  - /.*<¥/b>/ =~
    "aabbbcccdddee"
- 残念ながらあまり正しいやりかたではない
  - ネストしていたらうまくいかない
  - /.*?<¥/b>/ =~
    "aabbbcccdddee"
  - 閉じタグがないとうまくいかない
  - /.*?<¥/b>/ =~
    "
bbbccc
dddee
"

[:rep_lazy, e] の実装 (1)

```
def try(exp, seq, pos, &block)
  ...
  when :rep_lazy
    _, e = exp
    try_rep_lazy(e, seq, pos, &block)
  ...
end
```

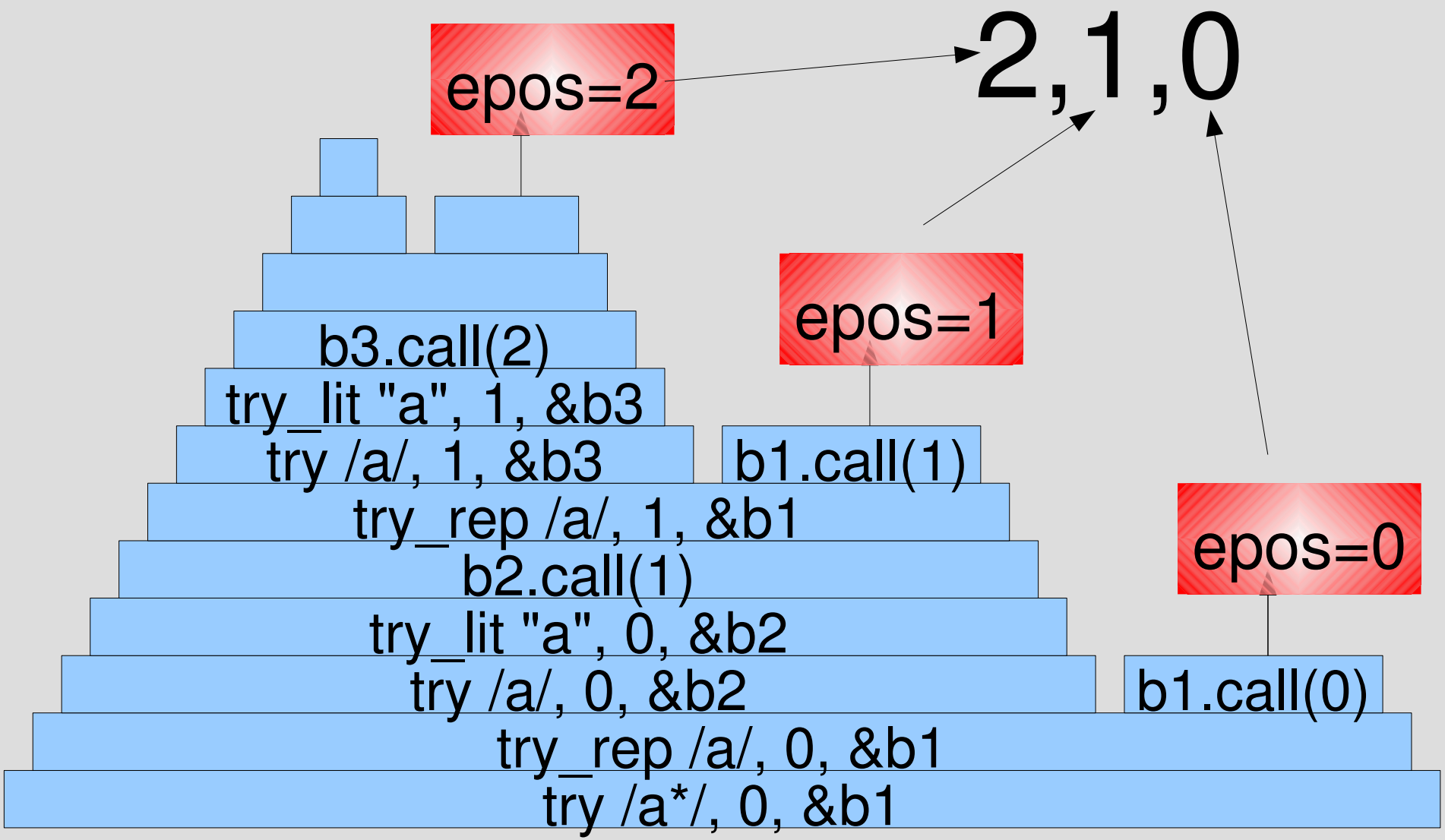
[:rep_lazy, e] の実装 (2)

```
def try_rep_lazy(e, seq, pos, &block)
  yield pos
  try(e, seq, pos) {|pos2|
    try_rep_lazy(e, seq, pos2, &block) if pos < pos2
  }
end
```

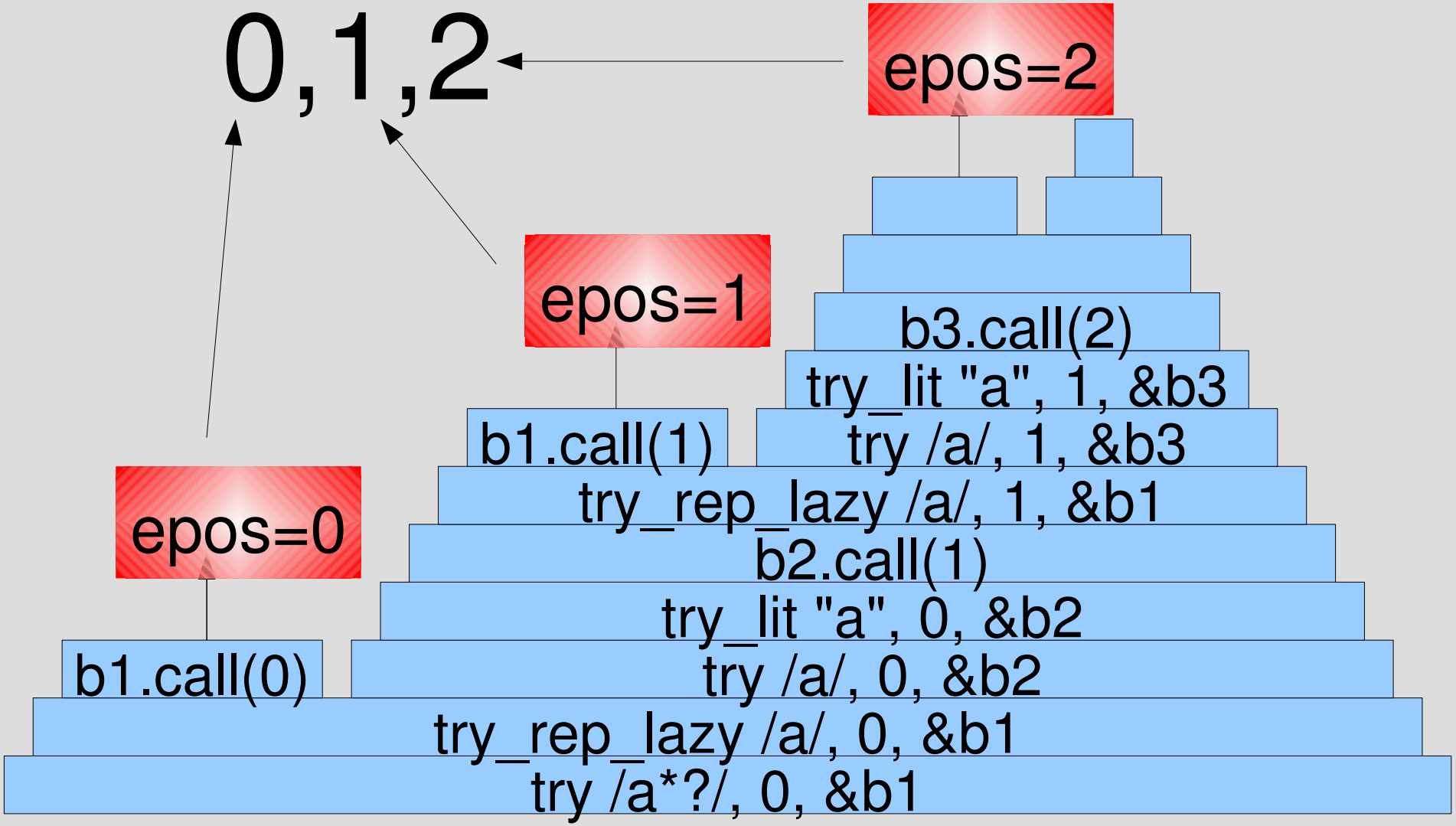
rep と rep_lazy

```
def try_rep(e, seq, pos, &block)
  try(e, seq, pos) {|pos2|
    try_rep(e, seq, pos2, &block) if pos < pos2
  }
  yield pos                                          後に yield
end
def try_rep_lazy(e, seq, pos, &block)
  yield pos                                          先に yield
  try(e, seq, pos) {|pos2|
    try_rep_lazy(e, seq, pos2, &block) if pos < pos2
  }
end
```

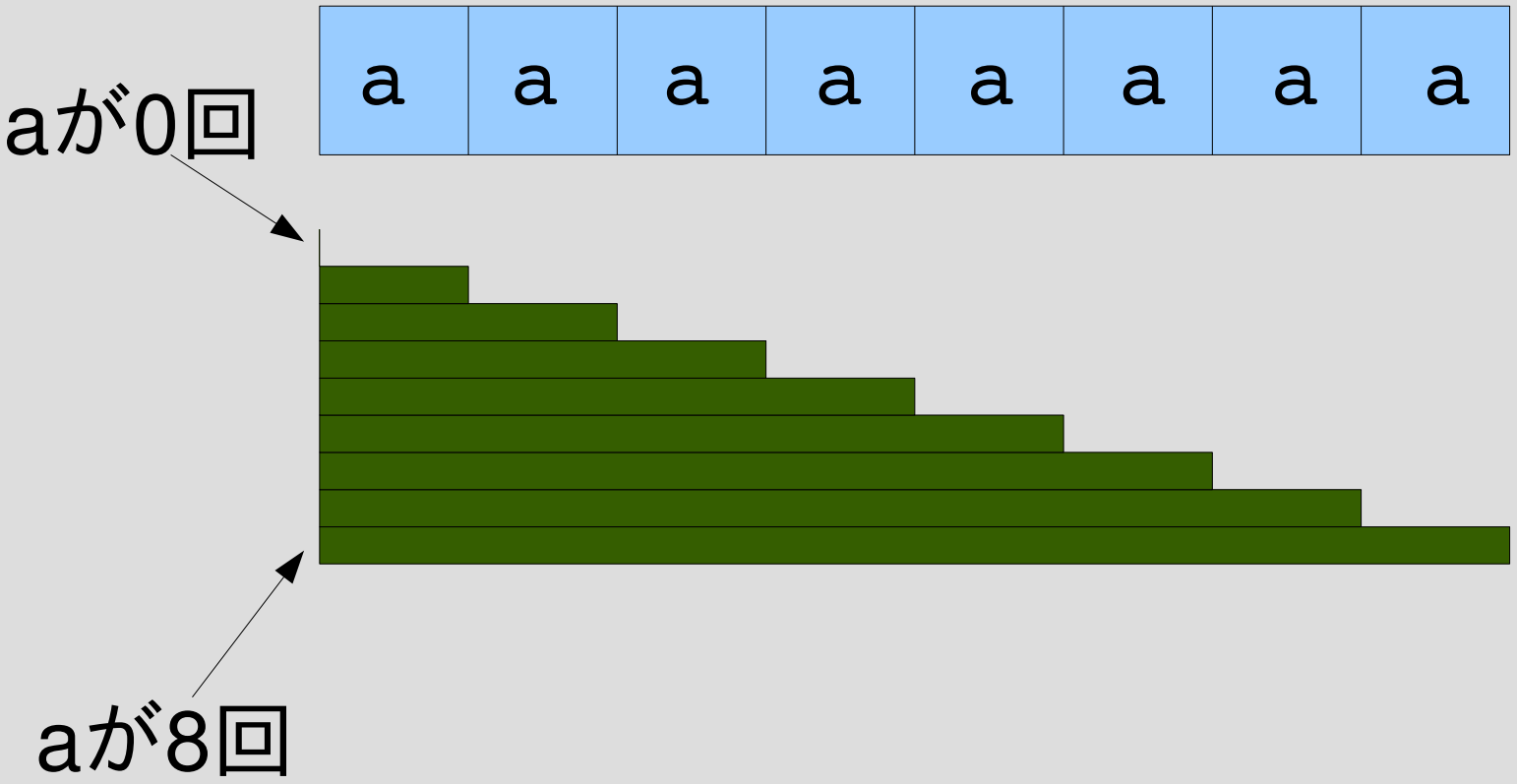
greedy: /a*/ =~ "aa"


epos=2
2,1,0
epos=1
epos=0
b3.call(2)
try_lit "a", 1, &b3
try /a/, 1, &b3
b1.call(1)
try_rep /a/, 1, &b1
b2.call(1)
try_lit "a", 0, &b2
try /a/, 0, &b2
b1.call(0)
try_rep /a/, 0, &b1
try /a*/, 0, &b1

lazy: /a*?/ =~ "aa"


0,1,2
epos=2
epos=1
epos=0
b3.call(2)
try_lit "a", 1, &b3
b1.call(1)
try /a/, 1, &b3
try_rep_lazy /a/, 1, &b1
b2.call(1)
try_lit "a", 0, &b2
b1.call(0)
try /a/, 0, &b2
try_rep_lazy /a/, 0, &b1
try /a*?/, 0, &b1

a*? の動作



エンジンの拡張: /e??/

- /e??/ は、e がない場合とある場合にマッチする
- e がない場合を先に試し、ある場合を後に試す
- /|e/ と同じ
- [:opt_lazy, e] で表現する

- /behaviou??r/ =~ "behavior"　　#=> 0
- /behaviou??r/ =~ "behaviour"　#=> 0

- matchstr([:opt_lazy, [:lit, "a"]], "aa")
  #=> [0,1]
- matchstr([:opt_lazy, [:lit, "a"]], "b")
  #=> [0]

e? と e??

- e? は e がある場合を先に試す: greedy
- e?? は e がない場合を先に試す: lazy

[:opt_lazy, e] の実装 (1)

```
def try(exp, seq, pos, &block)
  ...
  when :opt
    _, e = exp
    try_opt_lazy(e, seq, pos, &block)
  ...
end
```

[:opt_lazy, e] の実装 (2)

```
def try_opt_lazy(e, seq, pos, &block)
  yield pos
  try(e, seq, pos, &block)
end
```

try_opt と try_opt_lazy の比較

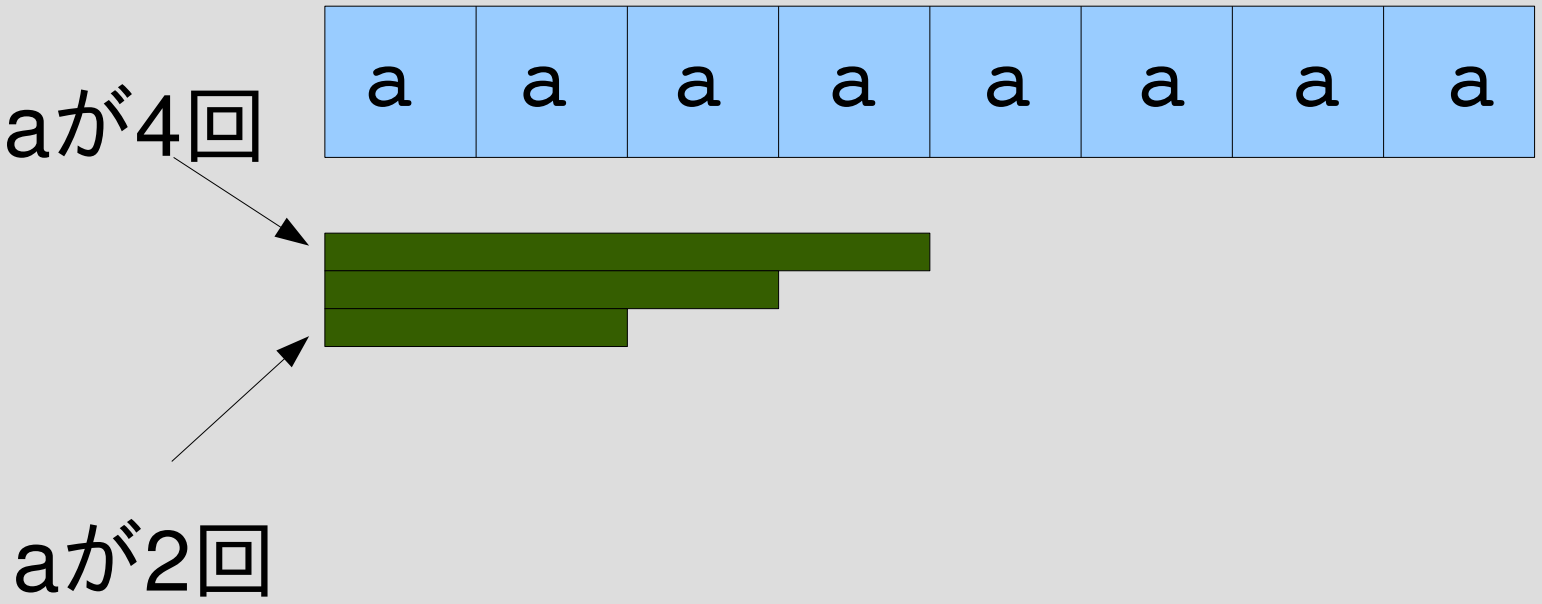
```
def try_opt(e, seq, pos, &block)
  try(e, seq, pos, &block)
  yield pos                                    後に yield
end

def try_opt_lazy(e, seq, pos, &block)
  yield pos                                    先に yield
  try(e, seq, pos, &block)
end
```

エンジンの拡張: e{m,n}

- /e{m,n}/ は e の m回以上 n回以下の繰り返し
- 抽象構文木では [:times, e, m, n] で表現
- たくさん繰り返した方から試す (greedy)

- matchstr([:times, [:lit, "a"], 2, 4], "aaaaa")
  #=> [4,3,2]
- matchstr([:times, [:lit, "a"], 2, 4], "aaa")
  #=> [3,2]
- matchstr([:times, [:lit, "a"], 2, 4], "a")
  #=> []

a{2,4} の動作



[:times, e, m, n] の実装 (1)

```
def try(exp, seq, pos, &block)
  ...
  when :times
    _, e, m, n = exp
    try_times(e, m, n, seq, pos, &block)
  ...
end
```

[:times, e, m, n] の実装 (2)

```
def try_times(exp, m, n, seq, pos, &b)
  if 0 < n
    try(exp, seq, pos) {|pos2|
      try_times(exp, m-1, n-1, seq, pos2, &b)
    }
  end
  yield pos if m <= 0
end
```

各種繰り返し

	0〜∞	0〜1	1〜∞	m〜n
greedy	e*	e?	e+	e{m,n}
lazy	e*?	e??	e+?	e{m,n}?

レポート

- 以下を実装して解説せよ
  - e+?
  - e{m,n}?
- 実装したらユニットテストで確認すること
- ∠切 2007-07-17 16:20
- HIPLUS
- 拡張子が txt なテキストファイル希望

/e+?/

- /e+/ の lazy 版
- 抽象構文木では [:plus_lazy, e]

- matchstr([:plus_lazy, [:lit, "a"]], "aaaaa")
  #=> [1,2,3,4,5]

/e{m,n}?/

- /e{m,n}/ の lazy 版
- 抽象構文木では [:times_lazy, e, m, n]

- matchstr([:times_lazy, [:lit, "a"], 2, 4], "aaaaa")
  #=> [2,3,4]

まとめ

- 前回のレポートの説明
- 再帰を使う拡張
  - 存在するかもしれない: e?
  - 1回以上の繰り返し: e+
  - 怠惰な繰り返し: e*?
  - 存在しないかもしれない: e??
  - m回以上n回以下の繰り返し: e{m,n}
- レポートを出した