テキスト処理 第13回 (2008-07-15)

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-2008/

今日の内容

- レポートの解説
- アトミックなグループ
- 強欲な繰り返し
- レポート

評価について

- レポートで行う
- 今日も出す

アトミックなグループ

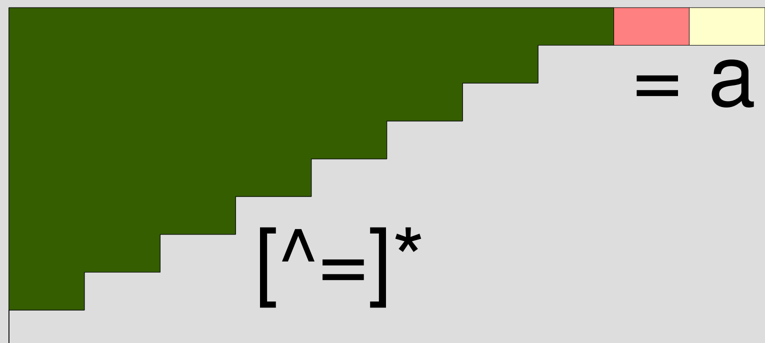
- Ruby では (?>r)
- r が一回マッチに成功したら、ほかのマッチの可能性は無視する
  (r 内でのバックトラックを抑止する)
- 処理を少なくできるので、高速化に利用できる

- 例
  - /¥A(?>[^=]*)=apple¥z/ =~ "favorite=avocado"
    #=> nil

  - /¥A[^=]*=apple¥z/ =~ "favorite=avocado"
    #=> nil

/¥A[^=]*=apple¥z/

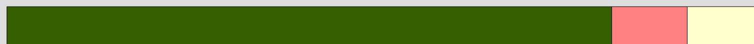
favorite=avocado



[^=]* は最初のマッチ以外は時間の無駄

/¥A(?>[^=]*)=apple¥z/

f	a	v	o	r	i	t	e	=	a	v	o	c	a	d	o



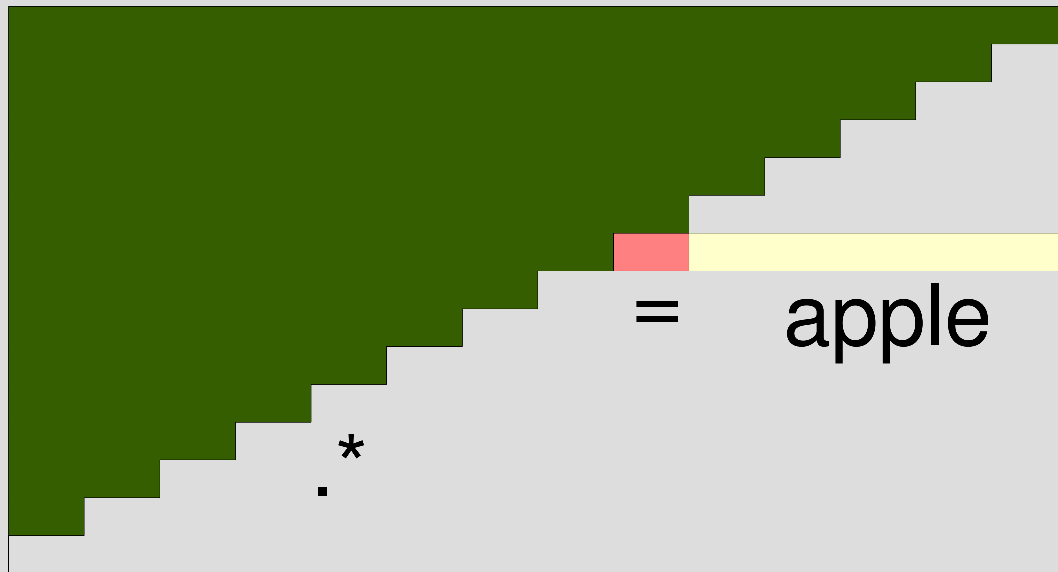
(?>[^=]*) = a

アトミックなグループにより、
最初のマッチ以外は試さない

マッチの結果が変わることもあるので注意

- /¥A.*=apple¥z/ =~ "favorite=apple" #=> 0
- /¥A(?>.*)=apple¥z/ =~ "favorite=apple" #=> nil

/¥A.*=apple¥z/



.* の最初のマッチは文字列の最後まで

/¥A(?>.*)=apple¥z/

favorite=apple

(?>.*)

.* の最初のマッチの後では

=apple はマッチしない

挙動が変わる単純な例

- /a+a/ =~ "aaa" #=> 0
- /(?>a+)a/ =~ "aaa" #=> nil

a+ は aaa, aa, a のどれかにマッチする

(?>a+) は aaa にしかマッチしない

アトミックなグループを try に実装

- 配列表現は [:atomic, r]

- 例
  - p rx_ends([:atomic, [:rep, "a"]], "aaa", 0) #=> [3]
  - p rx_ends([:rep, "a"], "aaa", 0) #=> [3, 2, 1, 0]

  - p rx_ends([:cat, [:atomic, [:rep, [:anychar]]],
                                        "=", "a", "p", "p", "l", "e"],
                          "favorite=apple", 0) #=> []
  - p rx_ends([:cat, [:rep, [:anychar]],
                                        "=", "a", "p", "p", "l", "e"],
                          "favorite=apple", 0) #=> [14]

try に :atomic の選択肢を追加

```
def try(re, str, pos, md, &b)
  ...
    when :atomic
      try_atomic(re, str, pos, md, &b)
  ...
end
```

try_atomic の実装

```
def try_atomic(re, str, pos, md, &b)
  try(re[1], str, pos, md) {|pos2, md2|
    yield pos2, md2
    return
  }
end
```

- 最初のマッチを呼出元に yield した後、有無をいわさず return する
- つまり2つめ以降の可能性をスキップする

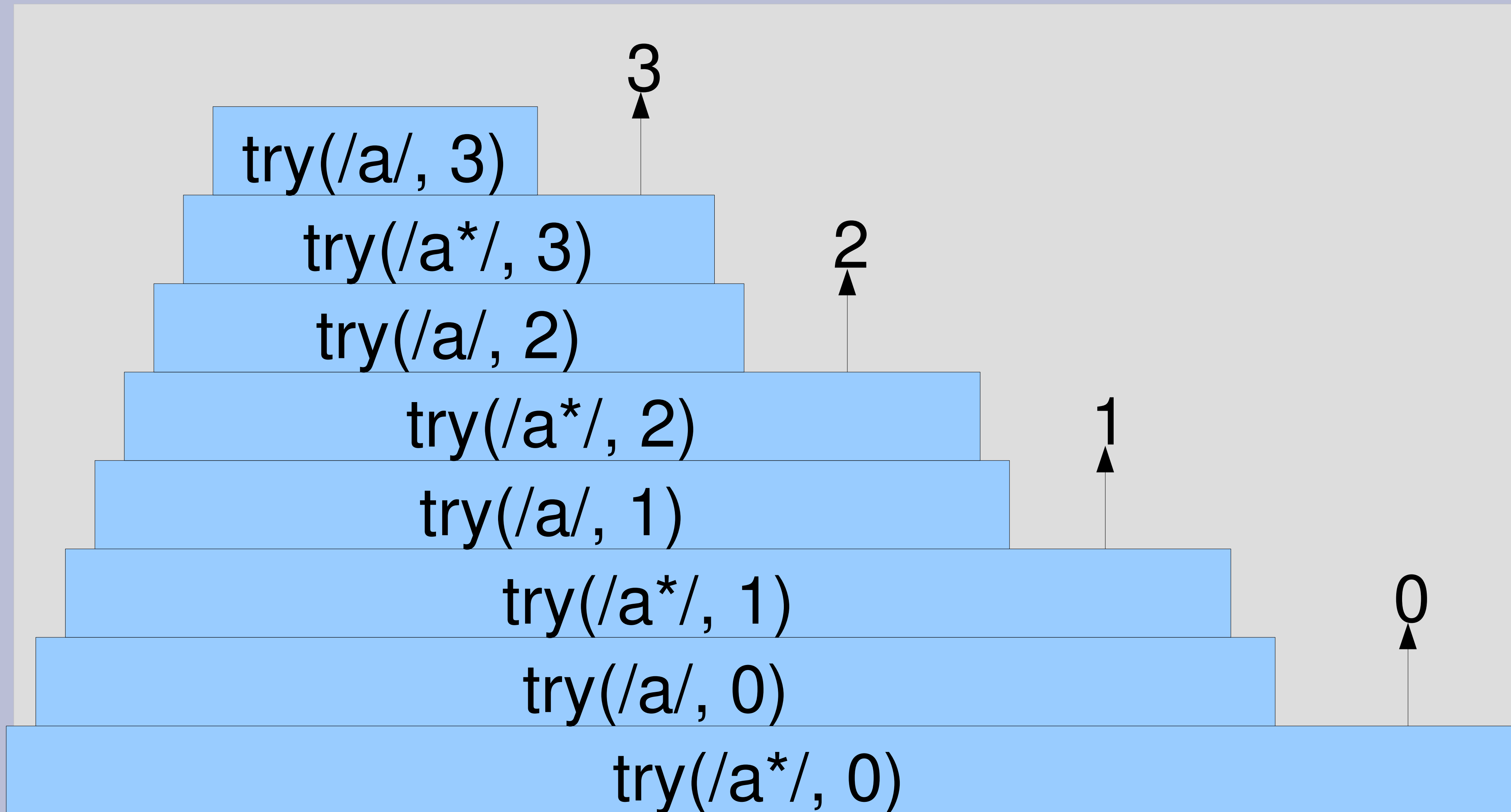
count_try で /(?>a*)/ と /a*/

- p rx_ends([:atomic, [:rep, "a"]], "aaa", 0)
  #=> [3]
- p count_try([:atomic, [:rep, "a"]], "aaa")
  #=> 9

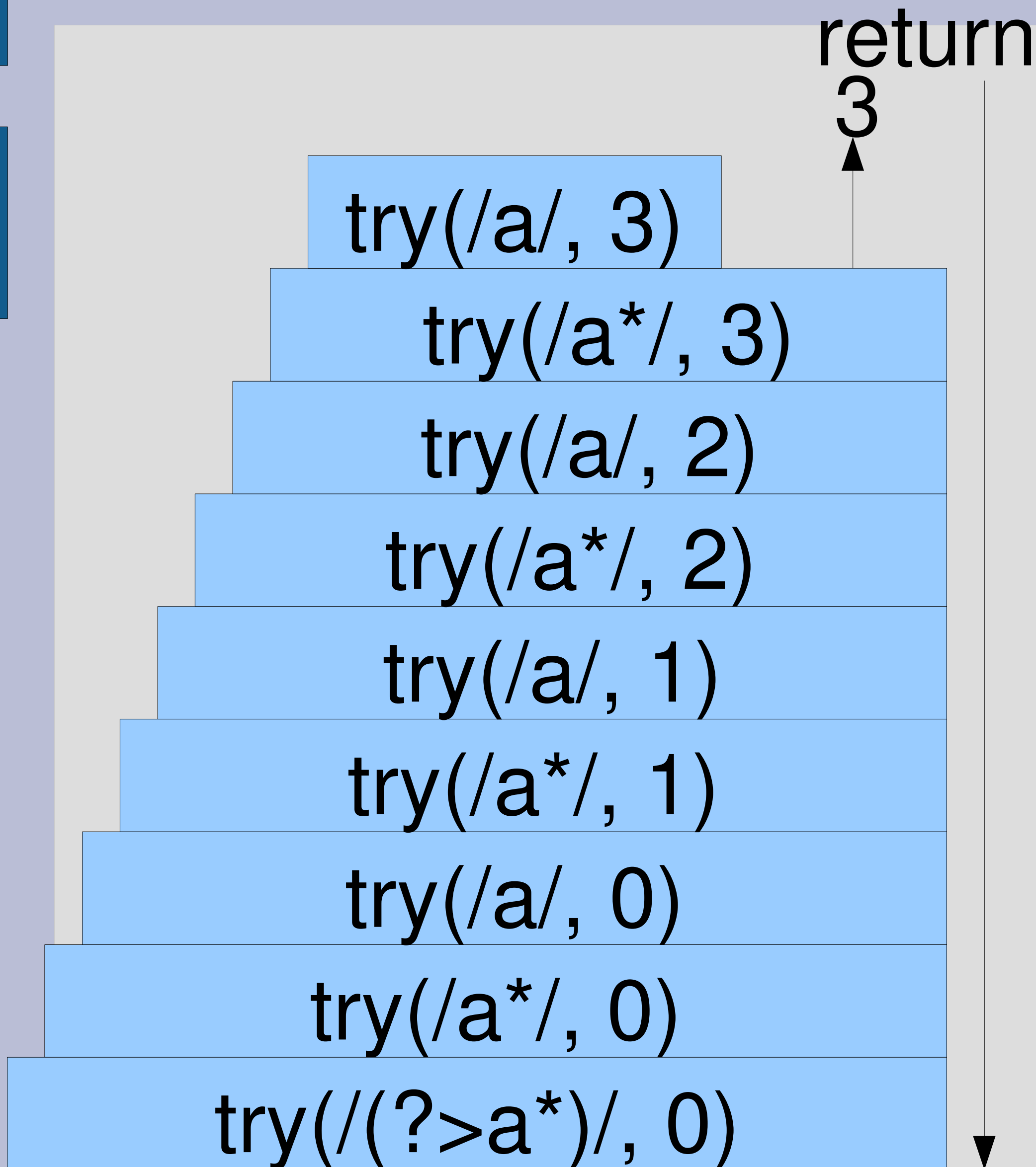
- p rx_ends([:rep, "a"], "aaa", 0)
  #=> [3, 2, 1, 0]
- p count_try([:rep, "a"], "aaa")
  #=> 8

この場合、try 呼出し回数はあまり変わらない

/a*/ =~ "aaa"

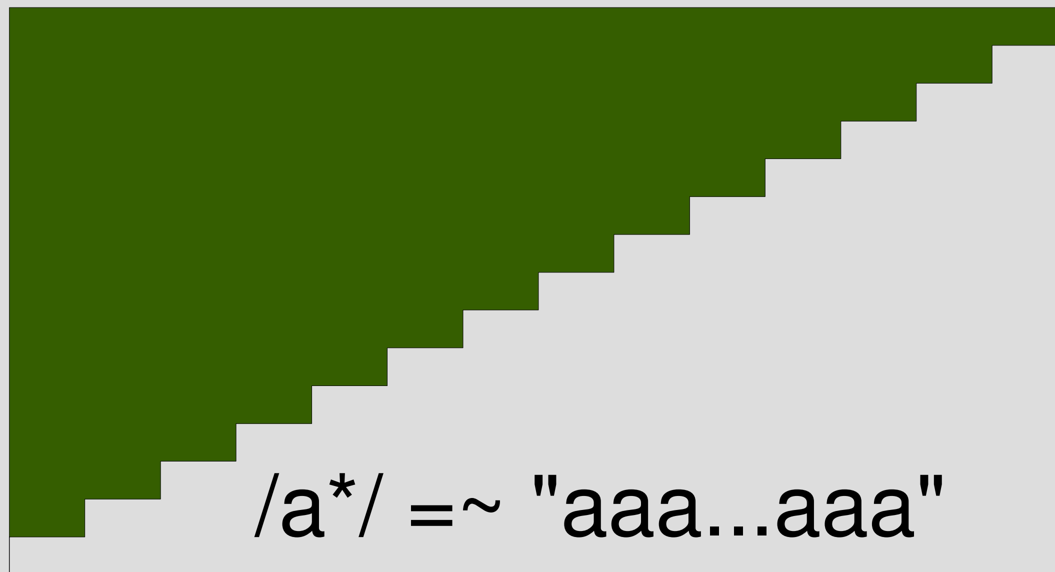


/(?>a*)/ =~ "aaa"



- 3 が yield されると return で try_atomic が終わる
- 2, 1, 0 は yield されない
- try が呼ばれる回数は変わらない (try(/?(>a*)/) を入れるとひとつ増える)

/a*/ と /(?>a*)/



- このふたつのマッチでの try 呼出し回数はどちらもだいたい同じ
- マッチした最大長に比例する

/(?>a*a*)/ と /a*a*/

- p rx_ends([:atomic,
  [:cat, [:rep, "a"], [:rep, "a"]]],
  "aaa", 0) #=> [3]
- p count_try([:atomic,
  [:cat, [:rep, "a"], [:rep, "a"]]],
  "aaa") #=> 12

- p rx_ends([:cat, [:rep, "a"], [:rep, "a"]],
  "aaa", 0)
  #=> [3, 3, 2, 3, 2, 1, 3, 2, 1, 0]
- p count_try([:cat, [:rep, "a"], [:rep, "a"]],
  "aaa") #=> 29

/(?>a*a*)/ と /a*a*/ (2)

- p count_try([:atomic,
  [:cat, [:rep, "a"], [:rep, "a"]]],
  "a"*20)   #=> 46
- p count_try([:atomic,
  [:cat, [:rep, "a"], [:rep, "a"]]],
  "a"*100) #=> 206

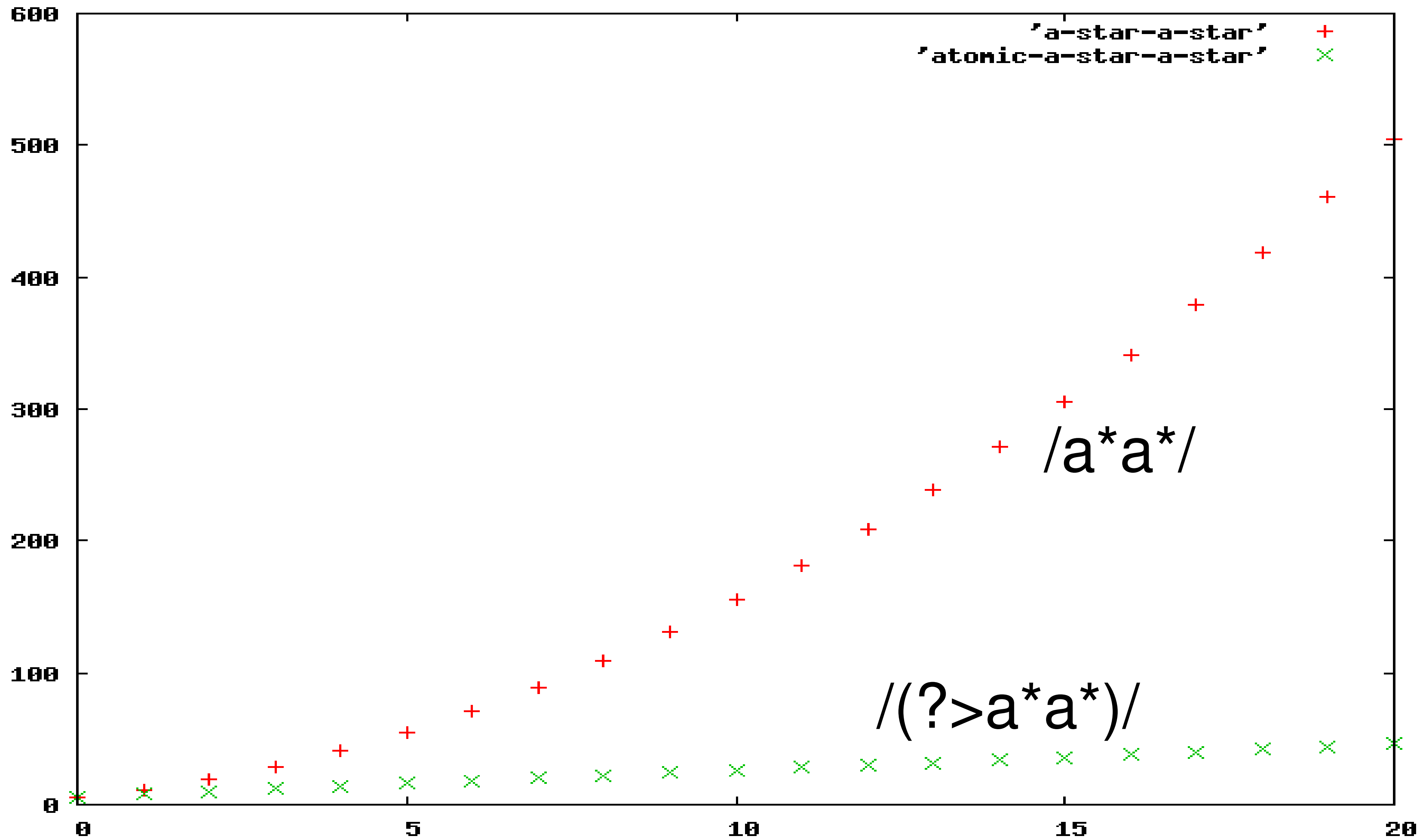
- p count_try([:cat, [:rep, "a"], [:rep, "a"]],
  "a"*20)   #=> 505
- p count_try([:cat, [:rep, "a"], [:rep, "a"]],
  "a"*100) #=> 10505

atomic 無しでの呼出し回数

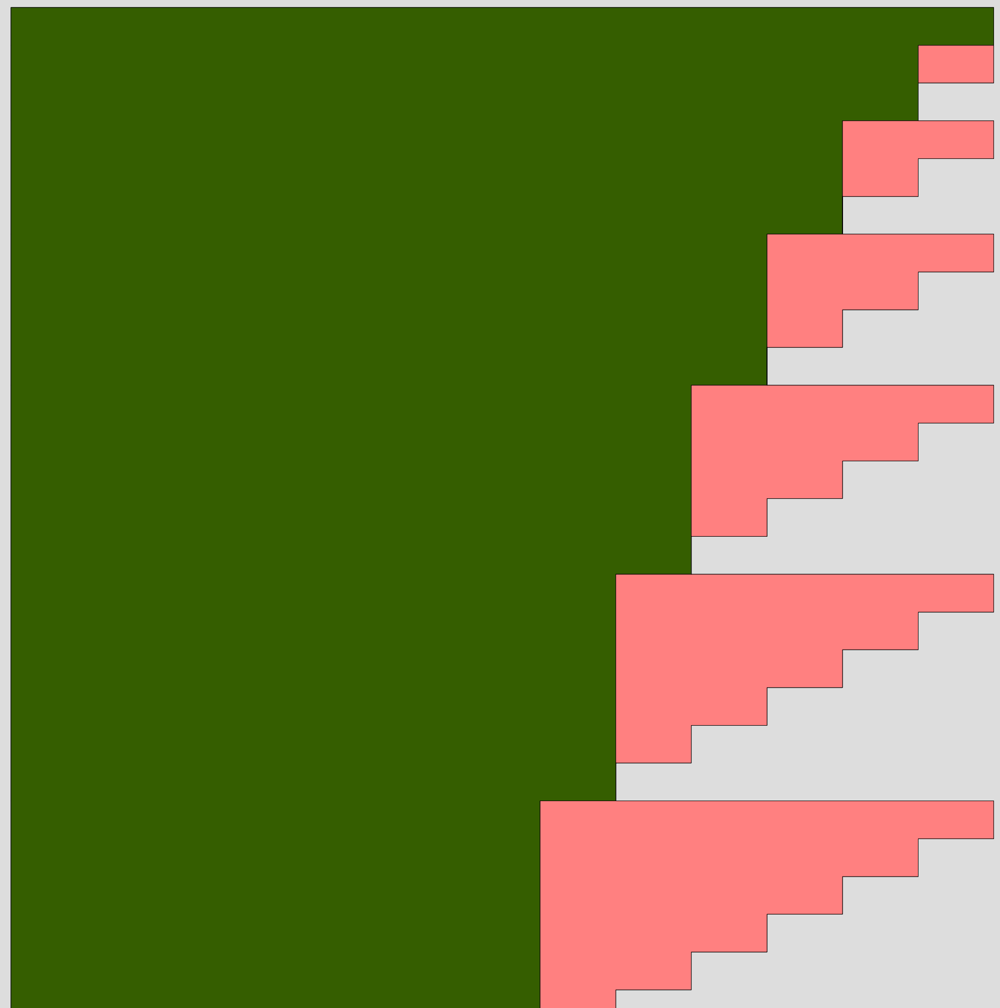
- ```
  0.upto(20) {|n|
    puts "#{n} #{count_try([:cat, [:rep, "a"],
                           [:rep, "a"]], "a"*n)}"
  }
  #=>
  ```

0 5	5 55	10 155	15 305	20 505
1 11	6 71	11 181	16 341	
2 19	7 89	12 209	17 379	
3 29	8 109	13 239	18 419	
4 41	9 131	14 271	19 461	
  ```

atomic 有りでの呼出し回数

- ```
  0.upto(20) {|n|
    puts "#{n} #{count_try([:atomic,
              [:cat, [:rep, "a"], [:rep, "a"]]], "a"*n)}"
  }
  #=>
  ```

0 6	5 16	10 26	15 36	20 46
1 8	6 18	11 28	16 38	
2 10	7 20	12 30	17 40	
3 12	8 22	13 32	18 42	
4 14	9 24	14 34	19 44	
  ```

/a*a*/ と /(?>a*a*)/ のグラフ



/a*a*/ =~ "aaa...aaa"



/(?>a*a*)/ =~ "aaa...aaa"

a	a	a	a	a	a	a	a	a	a	a	a	a

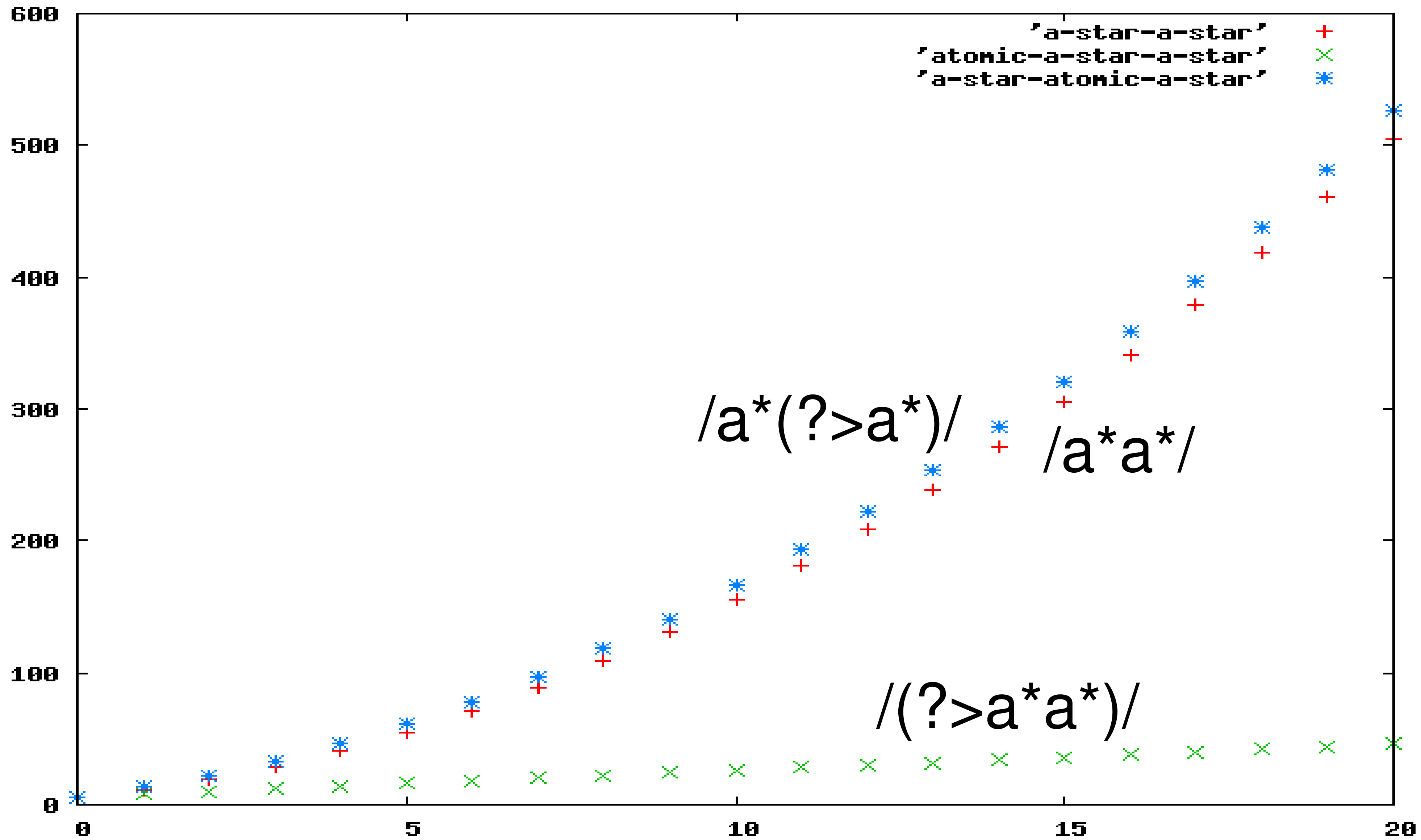
最初のマッチだけで処理が終わる

/(?>a*)a*/ は /(?>a*a*)/ と同じ回数

- ```
  0.upto(20) {|n|
    puts "#{n} #{count_try(
      [:cat, [:atomic, [:rep, "a"]], [:rep, "a"]],
      "a"*n)}"
  } #=>
  ```

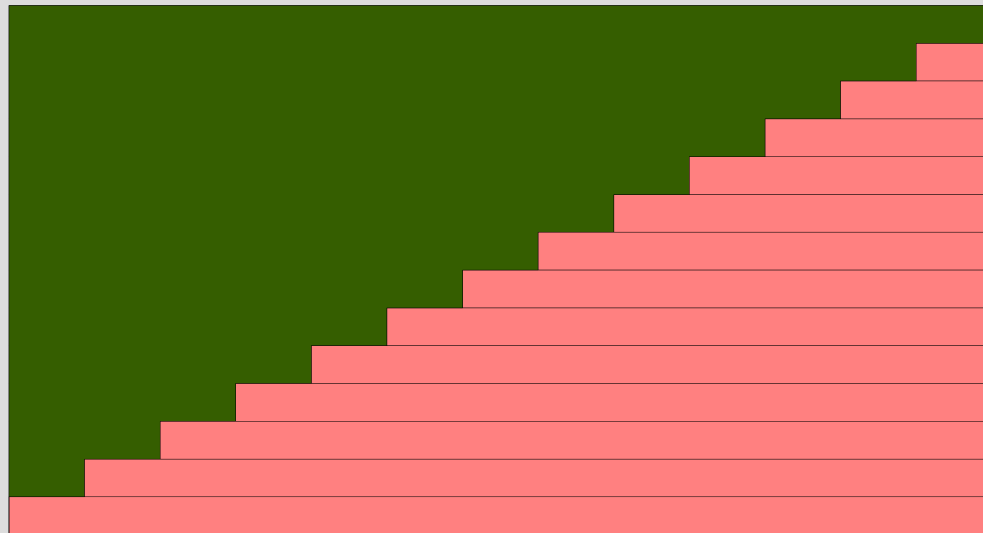
0 6	5 16	10 26	15 36	20 46
1 8	6 18	11 28	16 38	
2 10	7 20	12 30	17 40	
3 12	8 22	13 32	18 42	
4 14	9 24	14 34	19 44	
  ```

/a*(?>a*)/ は /a*a*/ より多くなる

- ```
  0.upto(20) {|n|
    puts "#{n} #{count_try(
      [:cat, [:rep, "a"], [:atomic, [:rep, "a"]]],
      "a"*n)}"
  } #=>
  ```

0 6	5 61	10 166	15 321	20 526
1 13	6 78	11 193	16 358	
2 22	7 97	12 222	17 397	
3 33	8 118	13 253	18 438	
4 46	9 141	14 286	19 481	
  ```

/a*a*/, /(?>a*a*)/, /a*(?>a*)/



/a*(?>a*)/ =~ "aaa...aaa"

a	a	a	a	a	a	a	a	a	a	a	a	a



強欲な繰り返し (possessive)

- 繰り返しの回数が最大の場合「だけ」が成功する
- Java 由来
- Ruby では Ruby 1.9 から使える
- 繰り返し記号の後に + をつける
  - r*+
  - r?+
  - r++
- 普通の繰り返しとアトミックなグループの組合せに等しい
  - r*+ は (?>r*) と同じ

各種繰り返し

	0〜∞	0〜1	1〜∞
greedy	r*	r?	r+
lazy	r*?	r??	r+?
possessive	r*+	r?+	r++

	m〜n	m	m〜∞
greedy	r{m,n}	r{m}	r{m,}
lazy	r{m,n}?	r{m}?	r{m,}?
possessive	r{m,n}+	r{m}+	r{m,}+

問題なく定義できるが Ruby はサポートしていない

強欲な繰り返しを try に拡張

- 配列表現
  - [:rep_possessive, r]
  - [:opt_possessive, r]
  - [:plus_possessive, r]

try に追加

```
def try(re, str, pos, md, &b)
  ...
    when :rep_possessive
      try([:atomic, [:rep, re[1]]], str, pos, md, &b)
    when :opt_possessive
      try([:atomic, [:opt, re[1]]], str, pos, md, &b)
    when :plus_possessive
      try([:atomic, [:plus, re[1]]], str, pos, md, &b)
  ...
end
```

強欲な繰り返しの例

- p count_try([:cat, [:rep_possessive, "a"], "b"],
  "aaaaaaaaaab")
  #=> 26
- p count_try([:cat, [:rep, "a"], "b"],
  "aaaaaaaaaab")
  #=> 34
- p count_try([:cat, [:rep_possessive, "a"], "b"],
  "a"*100 + "b")
  #=> 206
- p count_try([:cat, [:rep, "a"], "b"],
  "a"*100 + "b")
  #=> 304

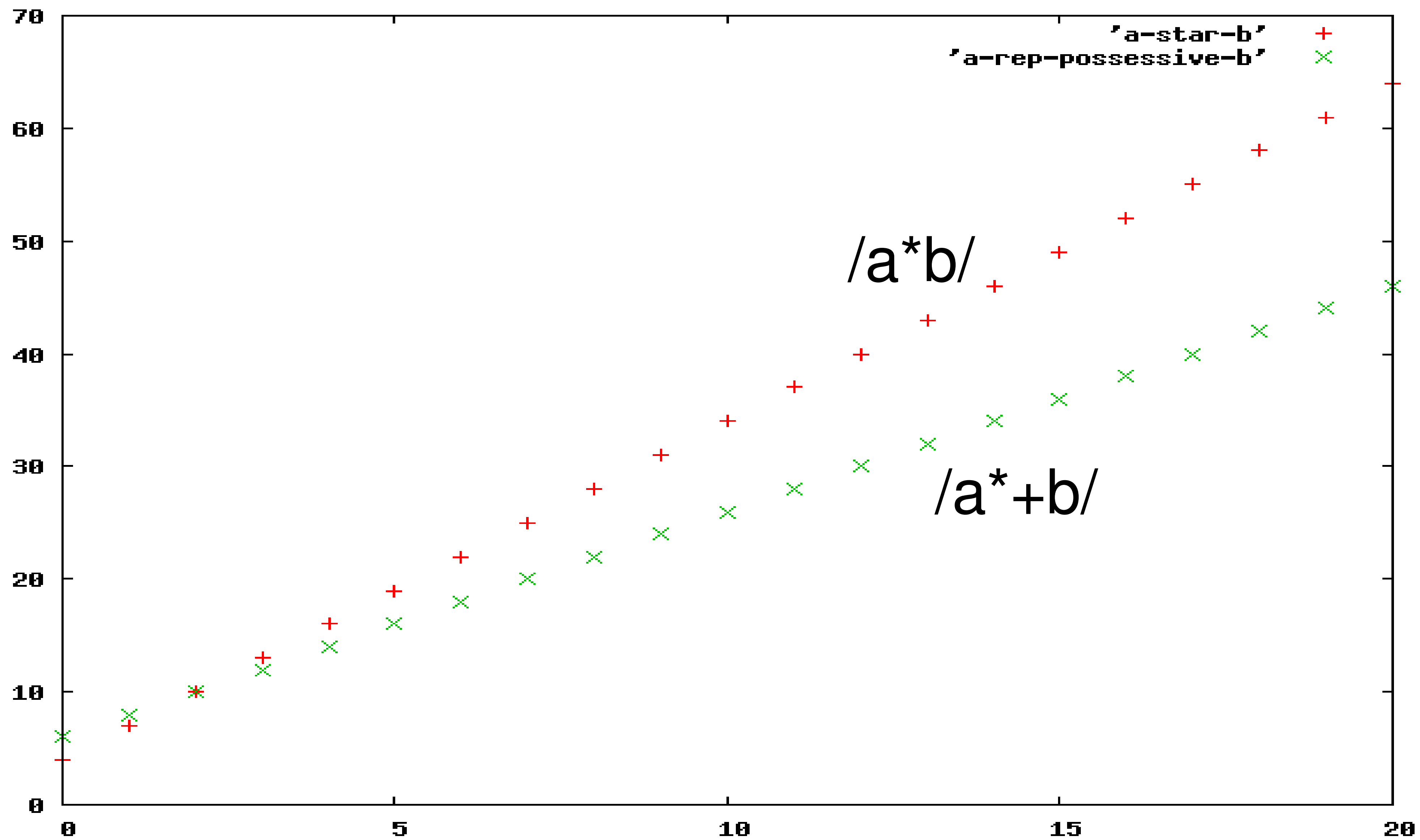
/a*b/ =~ "a...ab"

```
0.upto(20) {|n|
  s = "a"*n + "b"
  puts "#{n} #{count_try([:cat, [:rep, "a"], "b"], s)}"
}
#=> 0 4      5 19     10 34     15 49     20 64
    1 7      6 22     11 37     16 52
    2 10     7 25     12 40     17 55
    3 13     8 28     13 43     18 58
    4 16     9 31     14 46     19 61
```

/a*+b/ =~ "a...ab"

```
0.upto(20) {|n|
  s = "a"*n + "b"
  puts "#{n} #{count_try([:cat, [:rep_possessive, "a"],
                                         "b"], s)}"
}
#=>  0 6     5 16    10 26    15 36    20 46
     1 8     6 18    11 28    16 38
     2 10    7 20    12 30    17 40
     3 12    8 22    13 32    18 42
     4 14    9 24    14 34    19 44
```

/a*b/ と /a*+b/



/a*b/ と /a*+b/ の比較

- /a*+b/ のほうが try の呼出しが少ない (aが3つ以上あれば)
- マッチするものは同じ

レポート

- /a+a*/ を "a"*n にマッチさせたときの try 呼出し回数を求めよ
- /a++a*/ を "a"*n にマッチさせたときの try 呼出し回数を求めよ

- 〆切 2008-07-22 12:00
- RENANDI

count_try での求めかた

- /a+a*/ での回数は
  count_try(
    [:cat, [:plus, "a"], [:rep, "a"]],
    "a" * n)
  で求められる

- /a++a*/ での回数は
  count_try(
    [:cat, [:plus_possessive, "a"], [:rep, "a"]],
    "a" * n)
  で求められる

まとめ

- 前回のレポートの解説
- アトミックなグループ
- 強欲な繰り返し
- レポートを出した