CPS & TCO

2016/05/19
0x64 Tales
#08 Functional Programming

Livesense Inc.
HORINOUCHI Masato

Continuation-passing style とは

継続渡しスタイル (CPS: Continuation-passing style) とは、プログラムの制御を継続を用いて陽に表すプログラミングスタイルのことである。

継続渡しスタイルで書かれた関数は、通常の直接スタイル (direct style) のように値を「返す」かわりに、「継続」を引数として陽に受け取り、その継続に計算結果を渡す。継続とは、関数の計算結果を受け取るための（一般には元の関数とは別の）関数のことである。

継続渡しスタイル - Wikipedia から引用

factorial (Scheme / direct style)

```
(define (fact n)
  (if (zero? n)
      1
      (* n (fact (- n 1)))))

(fact 10)
```

factorial (Scheme / CPS)

```
(define (fact/cps n cont)  # ← 引数 cont が継続
  (if (zero? n)
      (cont 1)
      (fact/cps (- n 1) (lambda (x) (cont (* n x))))))

(fact/cps 10 (lambda (x) x))
```

factorial (Scheme / direct style / tail recursion)

```
(define (fact_tail n acc)
  (if (zero? n)
      acc
      (fact_tail (- n 1) (* n acc))))

(fact_tail 10 1)
```

benchmark result (1,000times 1000!)

- non tail call
  - 0.767s
- cps
  - 0.854s
- tail call
  - 0.819s
- まさかの非末尾呼出版が最も速い。
- CPS は呼び出し毎に lambda object 生成するから遅いのかな。

Stack Overflow (Scheme)

- 各パターンとも 100000! でも Stack Overflow しない。
- Scheme は末尾呼出最適化を行なうので CPS版と末尾呼出版が overflow しないのはわかる。
- 非末尾呼出版でも overflow しないのは、CPS変換を自動的に行なっているのだろうか?

factorial (Ruby / direct style / recursion)

```ruby
def fact(n)
  if n == 0
    1
  else
    n * fact(n - 1)
  end
end

fact(10)
```

factorial (Ruby / CPS)

```ruby
def fact_cps(n, cont)
  if n == 0
    cont.call 1
  else
    fact_cps(n - 1, -> (x) { cont.call(n * x) })
  end
end

fact_cps(10, -> (x) { x })
```

factorial (Ruby / direct style / tail recursion)

```ruby
def fact_tail(n, acc = 1)
  if n == 0
    acc
  else
    fact_tail(n - 1, n * acc)
  end
end

fact_tail(10)
```

benchmark

```
TIMES = 1000
FACT  = 1000

Benchmark.bm 16 do |r|
  r.report 'non tail call' do
    TIMES.times do
      fact(FACT)
    end
  end

  r.report 'cps' do
    TIMES.times do
      fact_cps(FACT, -> (x) { x })
    end
  end

  r.report 'tail call' do
    TIMES.times do
      fact_tail(FACT)
    end
  end
end
```

benchmark result (1,000times 1000!)

```
                      user     system      total        real
non tail call     0.570000   0.010000   0.580000 (  0.575446)
cps               1.220000   0.060000   1.280000 (  1.280935)
tail call         0.650000   0.060000   0.710000 (  0.705097)
```

- Scheme版と同様の傾向になった。

Stack Overflow (Ruby)

- 5000!
  - 3パターンとも問題なし
- 10000!
  - CPS版が `stack level too deep`
- 11000!
  - 3パターンとも `stack level too deep`

Tail Call Optimization とは

末尾呼出し最適化とは、末尾呼出しのコードを、戻り先を保存しないジャンプに変換することによって、スタックの累積を無くし、効率の向上などを図る手法である。

理論的には、継続渡しによる goto と同等のジャンプ処理では、手続きの呼出し元の情報を保存する必要がないことに帰着する。この場合 return は goto の特殊なケースで、ジャンプ先が呼出し元に等しいケースである。末尾最適化があれば、手続きの再帰を行なう時でも、結果はループと等価な処理手順となり、どれほど深い再帰を行なってもスタックオーバーフローを起こさない。

末尾再帰 - Wikipedia から引用

Tail Call Optimization in Ruby

- YARV だとオプションで TCO 有効にできる。

```
RubyVM::InstructionSequence.compile_option = {
  tailcall_optimization: true,
  trace_instruction: false
}
```

benchmark result (1,000times 1000! / TCO)

```
                    user     system      total        real
non tail call   0.570000   0.010000   0.580000 (  0.575446)
cps             1.220000   0.060000   1.280000 (  1.280935)
tail call       0.650000   0.060000   0.710000 (  0.705097)
```

↓ TCO on

```
                    user     system      total        real
non tail call   0.560000   0.020000   0.580000 (  0.578095)
cps             1.130000   0.050000   1.180000 (  1.183501)
tail call       0.640000   0.040000   0.680000 (  0.688400)
```

- CPS で約 8%改善、末尾呼出で 4%改善

Stack Overflow (Ruby / TCO)

- 10000!
  - 3パターンとも問題なし
- 11000!
  - 非末尾呼出版が `stack level too deep`
- 12000!
  - CPS版が `stack level too deep` ← なぜ?
- 1000000! まで試したが、末尾呼出版では Stack Overflow しない (10分かかった)

TCO Problems

- (1) backtrace: Eliminating method frame means eliminating backtrace.
- (2) set*trace*func(): It is difficult to probe "return" event for tail-call methods.
- (3) semantics: It is difficult to define tail-call in document (half is joking, but half is serious)

Tail call optimization: enable by default? から引用

まとめ

- CPS に置き換えるだけでは速くならない。
- YARV だと TCO 有効化ができる。
- TCO に関係なく `1.upto(FACT).inject(:*)` だと Stack Overflow しないのはヒミツ。

ご清聴ありがとうございました