「アルゴリズムとデータ構造入門」
第4回課題

工学部情報学科
平成25年入学
学籍番号：1029-25-2723
森井 崇斗

November 5, 2013

1 Fibonacci数の再帰型と繰り返し型手続きについて，ファイルを作成せよ

fib.scm内に記述。

- 再帰型：`fib`
- 繰返型：`fib-i`

```
1  (define (fib n)
2   (cond ((= n 0) 0)
3         ((= n 1) 1)
4         (else (+ (fib (- n 1))
5                  (fib (- n 2))))))
6  )
7  (define (fib-i n)
8   (define (iter a b count)
9    (if (= count 0)
10    b
11    (iter (+ a b) a (- count 1))))
12  (iter 1 0 n)
13 )
```

2　2種類のFibonacci数の手続きを実行し，fib(10)，fib(20),fib(30) の出力結果を求めよ

```
> (fib 10)
55
> (fib 20)
6765
> (fib 30)
832040
> (fib-i 10)
55
> (fib-i 20)
6765
> (fib-i 30)
832040
```

3　2種類の Fibonacci 数の手続きを使った fib(n) 実行時に fib が呼ばれる回数を，それぞれ解析的に求めよ．

3.1　fib が呼ばれる回数について

`fib` を呼び出した時、$n = 0 or 1$ の時は 1 回の呼び出しで終了する。
$n > 1$ の時、`fib(n)` 実行時に `fib` が呼ばれる回数を `fibnum(n)` とすると、$fibnum(n) = fibnum(n-1) + fibnum(n-2)$ と表される。

3.2　fib-i 内の iter が呼ばれる回数について

`fib-i` を呼び出した時、内部で定義された手続きである `iter` は `fib-i(n)` に対して、`n+1` 回である。

4　アッカーマン関数に関する練習問題

4.1　(ack 0 2)

- 2

4.2 (ack 1 2)

- (ack 0 (ack 1 1))
- (ack 1 1) +1
- (ack 0 (ack 1 0)) +1
- (ack 1 0) +1 +1
- (ack 0 1) +2
- 2 +2
- 4

4.3 (ack 2 2)

- (ack 1 (ack 2 1))
- (ack 1 (ack 1 (ack 2 0)))
- (ack 1 (ack 1 (ack 1 1)))
- (ack 1 (ack 1 (ack 0 (ack 1 0))))
- (ack 1 (ack 1 (ack 0 (ack 0 1))))
- (ack 1 (ack 1 (ack 0 2)))
- (ack 1 (ack 1 3))
- (ack 1 (ack 0 (ack 1 2)))
- (ack 1 (ack 0 4)) #授業資料より (ack 1 2) の結果は 4 と導出済
- (ack 1 5)
- (ack 0 (ack 1 5))
- (ack 0 (ack 0 (ack 1 4)))
- (ack 0 (ack 0 (ack 0 (ack 1 3))))
- (ack 0 (ack 0 (ack 0 (ack 0 (ack 1 2)))))
- (ack 0 (ack 0 (ack 0 (ack 0 4))))

- (ack 0 (ack 0 (ack 0 5)))
- (ack 0 (ack 0 6))
- (ack 0 7)
- 8

4.4 (ack 3 2)

- (ack 2 (ack 3 1))
- (ack 2 (ack 2 (ack 3 0)))
- (ack 2 (ack 2 (ack 2 1)))
- (ack 2 (ack 2 (ack 1 (ack 2 0))))
- (ack 2 (ack 2 (ack 1 (ack 1 1))))
- (ack 2 (ack 2 (ack 1 (ack 0 (ack 1 0)))))
- (ack 2 (ack 2 (ack 1 (ack 0 (ack 0 1)))))
- (ack 2 (ack 2 (ack 1 (ack 0 2))))
- (ack 2 (ack 2 (ack 1 3)))
- (ack 2 (ack 2 (ack 0 (ack 1 2))))
- (ack 2 (ack 2 (ack 0 4)))
- (ack 2 (ack 2 5))
- (ack 2 (ack 1 (ack 2 4)))
- (ack 2 (ack 1 (ack 1 (ack 2 3))))
- (ack 2 (ack 1 (ack 1 (ack 1 (ack 2 2)))))
- (ack 2 (ack 1 (ack 1 (ack 1 (ack 1 (ack 2 1))))))
- (ack 2 (ack 1 (ack 1 (ack 1 (ack 1 (ack 1 (ack 2 0)))))))
- (ack 2 (ack 1 (ack 1 (ack 1 (ack 1 (ack 1 (ack 1 1)))))))
- (ack 2 (ack 1 (ack 1 (ack 1 (ack 1 (ack 1 (ack 0 (ack 1 0))))))))

- (ack 2 (ack 1 (ack 1 (ack 1 (ack 1 (ack 1 (ack 0 (ack 0, 1))))))))
- (ack 2 (ack 1 (ack 1 (ack 1 (ack 1 (ack 1 (ack 0 2)))))))
- (ack 2 (ack 1 (ack 1 (ack 1 (ack 1 (ack 1 3))))))
- (ack 2 (ack 1 (ack 1 (ack 1 (ack 1 5)))))
- (ack 2 (ack 1 (ack 1 (ack 1 8))))
- (ack 2 (ack 1 (ack 1 (ack 0 (ack 1 7)))))
- (ack 2 (ack 1 (ack 1 (ack 0 (ack 0 (ack 1 6))))))
- (ack 2 (ack 1 (ack 1 (ack 0 (ack 0 (ack 0 (ack 1 5))))))
- (ack 2 (ack 1 (ack 1 (ack 0 (ack 0 (ack 0 8))))))
- (ack 2 (ack 1 (ack 1 (ack 0 (ack 0 9)))))
- (ack 2 (ack 1 (ack 1 (ack 0 10))))
- (ack 2 (ack 1 (ack 1 11)))
- (ack 2 (ack 1 13)) # ここから中略
- (ack 2 15)
- 33

※ただし、これについては (ack 3 2) は 29 を返すべきであるので、途中で計算を誤っている。

5 教科書 1-3-2 1-3-4 を読み、想定質問とそれへの解答と説明を記述せよ

5.1 想定質問

教科書の該当部分では lambda を用いた手続きの構築について記述されていた。
lambda と同等の仕組みとして、例えば Javascript では無名関数 (*anonymous function*) という機能が実装されており、これによって、Scheme の lambda 式を再現することが可能である。
例えば、教科書内で紹介されていた
$f(x, y) = x(1 + xy)^2 + y(1 - y) + (1 + xy)(1 - y)$ lambda 式を用いて実装する Scheme のコードを Javascript の無名関数を用いて実装するとどのように書くことが出来るか。

5.2 想定質問への解答と解説

5.2.1 想定質問への解答

上記の計算式は Javascript で無名関数を利用すると次のように記述できる。

```
1  // define square
2  function square(x){return (x*x)};
3  // rewrite Scheme lambda with Javascript anonymous function
4  function f(x,y){
5      (function(a, b){
6          return ( x*(square(a)) + y*b + a*b );
7      })( (1+x*y), (1-y) )
8  }呼び出し例
9  // x=3,y=4
10 f(3,4)
```

5.2.2 解説

Javascript では無名関数という機能を提供している。
これは Scheme で書くと、
(lambda (<*formal-parameters*> <*body*>)
と同等のものを
(function (<*formal-parameters*>) {<*body*>})
と記述することで再現可能にするものである。
また、Javascript でこの記法を用いることで再現できることには、変数スコープが静的であることに由来している。
これを利用し、上記のように記述することで Scheme で lambda を用いて実装したものを Javascript で再実装することが可能である。