Rekurencja

- Problem wyznaczenia ciągu *n*-liczb Fibonacci-ego dla danego *n*

- Liczby Fibonacci-ego są nieskończona sekwencją dodatnich liczb całkowitych tzn.:

0, 1, 1, 2, 3, 5, 8, itd.

- Każda kolejna liczba jest sumą poprzednich dwóch liczb w sekwencji

Rekurencja

- Procedura wyznaczająca *n*-elementowy ciąg liczb Fibonacci-ego

```
(define fibonacci
  (lambda (n)
    (let fib ((i n))
      (cond
        ((= i 0) 0)
        ((= i 1) 1)
        (else (+ (fib (- i 1)) (fib (- i 2)))])))))
```

```
(fibonacci 0) ⇒  0
(fibonacci 1) ⇒  1
(fibonacci 2) ⇒  1
(fibonacci 3) ⇒  2
(fibonacci 4) ⇒  3
```

Rekurencja

- Procedura generująca liczby Fibonacci-ego za pomocą dwóch akumulatorów `a1` i `a2` odpowiednio aktualnego i kolejnego elementu

```
(define fibonacci
  (lambda (n)
    (if (= n 0)
        0
        (let fib ((i n) (a1 1) (a2 0))
          (if (= i 1)
              a1
              (fib (- i 1) (+ a1 a2) a1))))))
```

Kontynuacje

- Proces oceny wyrażeń Scheme-a można podzielić na dwa kroki:
  1. „co należy ocenić?”
  2. „co zrobić z wartością?”
- Rozważmy ocenę poniższego wyrażenia

```
(if (null? x) (quote ()) (cdr x))
```

Kontynuacje

- Kontynuacja poprzez ocenę wyrażeń

```
(if (null? x) (quote ()) (cdr x))
```

1. wartość
   `(if (null? x) (quote ()) (cdr x))`
2. wartość `(null? x)`
3. wartość `null?`
4. wartość `x`
5. wartość `cdr`
6. wartość `x` (again)

Kontynuacje

- Scheme wspiera operacje skoków oraz *nielokalnego sterowania*
- Scheme pozwala na kontrolowane skoki do wybranych miejsc programu
- Operator nielokalnego sterowania w Scheme-ie nosi nazwę:

  `call-with-current-continuation`

- Skrótowe oznaczenie: `call/cc`

Kontynuacje

- Scheme pozwala na kontynuacje dowolnego wyrażenia wywołanego przez `call/cc`
- `call/cc` jest procedurą p z pojedynczym argumentem
- `call/cc` tworzy pewną reprezentację aktualnej kontynuacji i przyporządkowuje ją p
- Kontynuacja jest reprezentowana przez siebie za pomocą procedury k

Kontynuacje

- Każdorazowe zastosowanie procedury k do wartości powoduje zwrócenie wartości kontunuacji `call/cc`
- Wartość ta staje się wartością `call/cc`
- Jeżeli p zwraca bez odwołania do k, wówczas zwracana wartość przez procedurę uzyskuje wartość `call/cc`

Kontynuacje

- Rozważmy proste przykłady

```
(call/cc
  (lambda (k)
    (* 5 4))) ⇒ 20

(call/cc
  (lambda (k)
    (* 5 (k 4))))  ⇒  4

(+ 2
   (call/cc
     (lambda (k)
       (* 5 (k 4)))))  ⇒  6
```

Kontynuacje

- Zastosowanie `call/cc` do uzyskania nielokalnego opuszczenia rekursji

```
(define product
  (lambda (ls)
    (call/cc
      (lambda (break)
        (let f ((ls ls))
          (cond
            ((null? ls) 1)
            ((= (car ls) 0) (break 0))
            (else (* (car ls) (f (cdr ls)))))))))))

  (product '(1 2 3 4 5))          ⇒ 120
  (product '(7 3 8 0 1 9 5))      ⇒ 0
```