ЛЕКЦИЯ 2
Рекурсия и итерация. Хвостовая рекурсия

Что было раньше

- Подстановочная модель. Правила:

1. Если выражение – литерал, то вернуть само выражение.
2. Если выражение – имя, вернуть его значение в текущем окружении.
3. Если выражение – спец. форма, то вычислить его по правилам этой формы.
4. Если выражение – комбинация, то:
   - 4.1. Вычислить его подвыражения в произвольном порядке.
   - 4.2. Если первое подвыражение – встроенная процедура, то применить её к операндам.
   - 4.3. Если оно – пользовательская функция, подставить в её тело аргументы и вычислить.

Функции и процессы

- Вспомним факториал

```
(define (fact1 n)
        (if (= n 1) 1
            (* n (fact1 (- n 1)))))
```

- Процесс, порождаемый (fact1 3)

```
(if (= 3 1) 1 (* 3 (fact1 (- 3 1))))
(* 3 (fact1 2))
(* 3 (if (= 2 1) 1 (* 2 (fact1 (- 2 1)))))
(* 3 (* 2 (fact1 1)))
(* 3 (* 2 (if (= 1 1) 1 (* 1 (fact1 (- 1 1))))))
(* 3 (* 2 (* 1))
(* 3 2)
6
```

расширение

сжатие

Функции и процессы

(fact1 6)

(* 6 (fact1 5))

(* 6 (* 5 (fact1 4)))

(* 6 (* 5 (* 4 (fact1 3))))

(* 6 (* 5 (* 4 (* 3 (fact1 2))))) …

отложенные операции

- Количество шагов линейно. Память линейна.

Функции и процессы

- Итеративный факториал

```
(define (fact2 n)
    (define (loop i result)
        (if (= i 1) result
            (loop (- i 1) (* i result))))
    (loop n 1))
```

- Процесс порождаемый (fact2 3)

```
(loop 3 1)
(if (= 3 1) 1 (loop (- 3 1) (* 3 1)))
(loop 2 3)
(if (= 2 1) 3 (loop (- 2 1) (* 2 3)))
(loop 1 6)
(if (= 1 1) 6 (loop (- 1 1) (* 1 6)))          ==> 6
```

Функции и процессы

(fact2 6)
(loop 6 1)
(loop 5 6)
(loop 4 30)
(loop 3 120)
(loop 2 360)
(loop 1 720)
720

отложенных операций нет!

- Количество шагов линейно. Память постоянна.

Рекурсивные и итеративные процессы

- Если в ходе процесса возникает цепочка отложенных операций, то процесс рекурсивный.
- Если в ходе процесса отложенных операций нет, то процесс итеративный.
- Рекурсивный процесс, в котором число шагов линейно – линейно рекурсивный.
- Итеративный процесс с линейным количеством шагов – линейно итеративный.

Нелинейный рекурсивный процесс

- Рекурсивное вычисление чисел Фибоначчи

```
(define (fib n)
    (cond ((= n 0) 0)
          ((= n 1) 1)
          (else (+ (fib (- n 1)) (fib (- n 2))))))
```

- Вычисление (fib 5)

(fib 5)

(+ (fib 4) (fib 3))

(+ (fib 4) (+ (fib 2) (fib 1)))

(+ (fib 4) (+ (+ (fib 1) (fib 0)) 1))

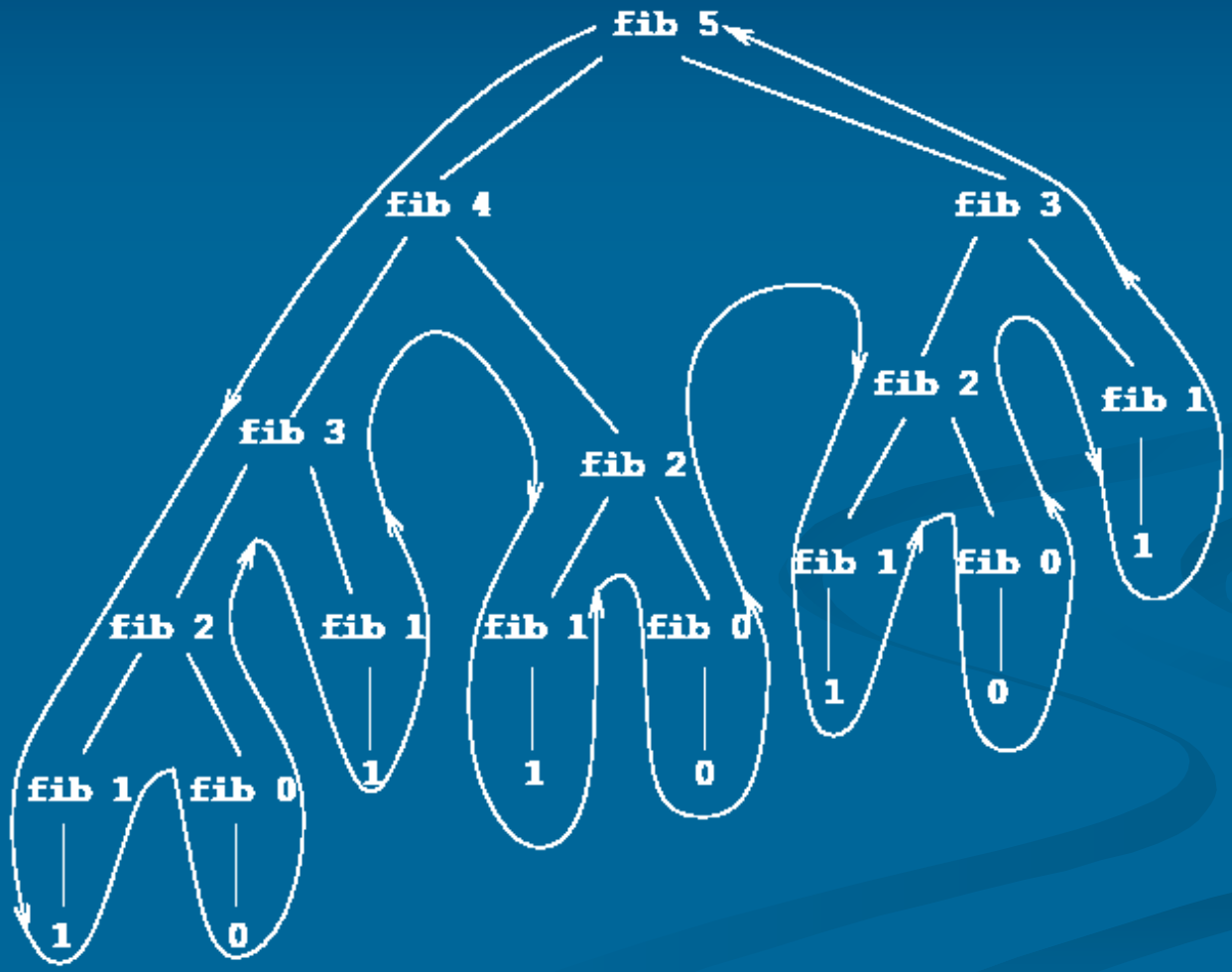
(+ (fib 4) (+ 1 1))

(+ (fib 4) 2)

(+ (+ (fib 3) (fib 2)) 2) … 5

Нелинейный рекурсивный процесс

- Дерево рекурсивного вычисления чисел Фибоначчи



- Количество шагов экспоненциально. Память линейна

Итеративное вычисление чисел Фибоначчи

```
(define (fib2 n)
    (define (loop i fib-n-1 fib-n-2)
        (if (= i 0) fib-n-2
                    (loop (- i 1) (+ fib-n-1 fib-n-2) fib-n-1)))
    (loop n 1 0))
```

- (fib2 5)

  (loop 5 1 0)

  (loop 4 1 1)

  (loop 3 2 1)

  (loop 2 3 2)

  (loop 1 5 3)

  (loop 0 8 5)

- Количество шагов линейно, память константа

Вспомним оборачивание списка

```
(define (my-reverse lst)
    (if  (null? lst)
         '()
         (append (my-reverse (cdr lst)) (list (car lst)))))
(my-reverse '(1 2 3))
    (append (my-reverse '(2 3)) (list 1))
    (append (append (my-reverse '(3)) (list 2)) '(1))
    (append (append (append (my-reverse '()) (list 3)) '(2))
    '(1))
    (append (append (append '() '(3)) '(2)) '(1))
    (append (append '(3) '(2)) '(1))
    (append '(3 2) '(1))     ==> (3 2 1)
```

Отложенные операции! Рекурсивный процесс

Итеративное оборачивание списка

```
(define (reverse2 lst)
    (define (loop lst result)
        (if (null? lst) result
            (loop (cdr lst) (conc (car lst) result))))
    (loop lst '()))
```

- (reverse2 '(1 2 3))

(loop '(1 2 3) '())

(loop '(2 3) '(1))

(loop '(3) '(2 1))

(loop '() '(3 2 1))

Линейно итеративный процесс

Возведение в степень

- $a^n = a \cdot a \cdots a = a \cdot a^{n-1}$

```
(define (my-expt a n)
    (if (= n 0)
        1
        (* a (my-expt a (- n 1)))))
```

- (my-expt 10 4)

```
(if (= 4 0) 1 (* 10 (my-expt 10 (- 4 1))))
```

отложенные операции!

- Количество шагов линейно. Память линейна

Итеративное возведение в степень

- перепишем с накоплением

```
(define (my-expt1 a n)
   (define (loop i result)
     (if (= i 0)
         result
         (loop (- i 1) (* a result))))
   (loop n 1))
```

- (my-expt1 10 4)

(loop 4 1)

(loop (- 4 1) (* 10 1))

(loop (- 3 1) (* 10 10))

(loop (- 2 1) (* 10 100)) ...

отложенные операции отсутствуют!

- Количество шагов линейно. Память постоянна

Нелинейное итеративное возведение в степень

```
(define (my-expt2 a n)
    (define (loop a i result)
      (cond ((= i 0) result)
            ((even? i) (loop (* a a) (/ i 2) result))
            (else (loop a (- i 1) (* a result)))))
    (loop a n 1))
```

- (my-expt2 10 4)

(loop 10 4 1)

(loop (* 10 10) (/ 4 2) 1)

(loop (* 100 100) (/ 2 1) 1)

(loop 10000 1 1)

- Количество шагов O(log n). Память постоянна

Итеративный НОД с логарифмическим ростом

```
(define (my-gcd a b)
   (if (= b 0) a
         (my-gcd b (remainder a b))))
```

(my-gcd 4 12)

(my-gcd 12 4)

(my-gcd 4 0)

4

- Количество шагов O(log n). Память постоянна

Итоги лекции 2

- Рекурсивное описание функции может породить рекурсивный процесс или итеративный процесс.
- Писать программу надо, оценивая сложность по шагам и по памяти.
- Хвостовая рекурсия – способ экономить память. В рекурсивном вызове можно не создавать новое окружение.