PPA – BI-WSI-SI-15

Funkcionální programování, funkce vyšších řádů, Lisp: atomy, seznamy, funkce, cons buňky, rekurze, mapovací funkcionály.

Obsah

1 Funkcionální programování

- Zaměřeno na funkce a jejich vyhodnocování.
- Nemá přiřazovací příkaz, nemá vedlejší efekty, typické používání rekurze.
- LISP, Haskell, R, Clojure, Python
- Odpovídající základní model výpočtu lambda kalkulus.
- Má funkce vyšších řádů: parametrem i výsledkem funkce může být funkce. Má typicky uzávěry (closures).

2 Lisp

- Jeden z funkcionálních programovacích jazyků.
- První jazyk, co měl garbage collection.
- Lisp je case insensitive.

Lisp vyhodnocuje nejdřív argumenty (podvýrazy) a potom to zbytek, pokud mu to nezakážeme příkazem `quote` (zkratka apostrof):

`'(+ 1 1)` ⇔ `(quote (+ 1 1))` je ve výsledku `(+ 1 1)`

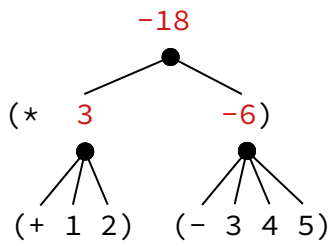

Figure 1: Vyhodnocení výrazu `(* (+ 1 2) (- 3 4 5))`

2.1 Atomy

čísla `235.4 2e10 2/3`

proměnné `foo 2nd-place *foo*`

konstanty `pi t nil`

řetězce, znaky `"Hello!" #\a`

pole `#(1 "foo" A) #1A(1 "foo" A) #2A((A B C) (1 2 3))`

struktury `#s(place FIT Prague)`

bitová pole `#*10110`

hašovací tabulky

2.2 Seznamy

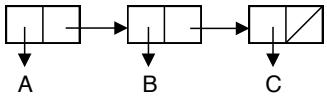

Figure 2: `(A B C)`

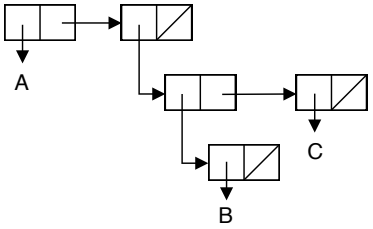

Figure 3: `(A ((B) C))`

Figure 4: Prázdný seznam `() = NIL`

Figure 5: `(A ((B) C))`

Seznam

`(cons 1 (cons 2 (cons 3 nil)))` ⇔ `(list 1 2 3)` je ve výsledku `(1 . (2 . (3 . nil)))`

Strom

`(cons (cons 1 2) (cons 3 4))` je ve výsledku `((1 . 2) . (3 . 4))`

Výběr ze seznamu

```
(list 1 2 3)
car → 1     cdr → (2 3)
cadr → 2    cddr → (3)
caddr → 3   cdddr → NIL
```

Figure 6: Ukázka výběru prvků ze seznamu

2.3 Funkce

```
(defun name (parameters) (body))
```

```
(defun sum (x y) (+ x y))
(defun factorial (n) (if (zerop n) 1 (* n (factorial (- n 1)))))
```

2.4 Podmínky

```
(defun abs (X)
 (cond ((> X 0) X)
       ((= X 0) 0)
       ((< X 0) (- X))
))
```

```
(defun abs (X)
  (if (> X 0) X
   X
   (if (= X 0) 0 (- X))
))
```

2.5 Rekurze

Rekurze používá zasobník pro zachování stavu při vnoření do funkce (aktivační záznam). Doporučuje se využívat koncovou rekurzi, která šetří místo na zásobníku.

Vnořená - rekurze v rekurzi

Stromová - několik rekurzivních volaní

Lineární - jedno rekurzivní volání

Koncová - rekurzivní volání je poslední, co funkce udělá - optimizace překladačů znovupoužití stejného stack framu

```
(defun factorial (N)
  ;;;"Compute the factorial of N."
  (if (= N 0)
    1
    (* N (factorial (- N 1)))
))

(defun fast-factorial (N)
  ;;;"A tail-recursive version of factorial."
  (fast-factorial-aux N 1)
)

(defun fast-factorial-aux (N ACC)
  ;;;"Multiply A by the factorial of N."
  (if (= N 0)
    ACC
    (fast-factorial-aux (- N 1) (* N ACC))
))
```

2.6 Mapovací funkcionály

Funkcionál je funkce, která má funkci jako argument. `(mapcar function list-1 list-2 ... list-n )`

Aplikujeme funkci square na list s položkami 1 2 3 `(mapcar #'square '(1 2 3))`

- `mapcar` prochází všechny seznamy prvek po prvku. Ukončí se jakmile v některém z listů dojdou prvky. Návratovou hodnotou je list s prvky původních listů, na které byla aplikována funkce.
- `mapc` funguje jako mapcar - vrací první list
- `maplist` prochází list prvek po prvku, do další iterace jde vždy cdr ze zpracovaného listu

```
(defun plus (x y) (+ x y))
(defun square (x) (* x x))

(mapcar #'square '(1 2 3)) → (1 4 9)
(mapcar #'plus '(1 2 3) '(3 2 1)) → (4 4 4)
(mapc #'plus '(1 2 3) '(3 2 1)) → (1 2 3)
```