Neprocedurální programování

Úvod do funkcionálního programování

John McCarthy
1927 - 2011

Funkcionální programování

Idea

- program = definice funkce
- výpočet = aplikace funkce na argumenty
  - » skládání funkcí
  - » “matematické” funkce bez vedlejších efektů

Inspirace

- λ - kalkul
- Alonzo Church, 1936

Příklady

```
> 3.14
3.14
> (+ 1 2)
3
> (- (+ 1 2) (* 2 3))
-3
> (sqrt 4)
2
> (quote (1 2 3))    nebo '(1 2 3)
(1 2 3)
```

Funkcionální programování: historie

1930-: Alonzo Church: λ - kalkul

1950-: John McCarthy: LISP

1960-: Peter Landin: ISWIM

1970-: John Backus: FP

1970-: Robin Milner: ML

1980-: David Turner: Miranda

1987: Haskell

2010: Haskell 2010

2000-: Scala, F#, Kotlin

LISP

LISP (John McCarthy, 1958, MIT)

- List Processing
  - » Lots of Irritating Superfluous Parentheses

Scheme

- jeden z dialektů LISPu
- výuka, výzkum
- minimalismus, jednoduchá sémantika
- DrScheme → DrRacket
  - » http://racket-lang.org

Symbolické výrazy

S-výraz

- atom
  - » 3.14
  - » #t
  - » +
  - » fun
- tečka-dvojice
  - » (1 . 2)
- seznam
  - » (1 2 3)
  - » ((1 2) 3)

Vyhodnocení výrazu

Cyklus

- `read-eval-print`
- přečti s-výraz
- vyhodnoť
- vrať výsledek

REPL

- read–eval–print loop
- language shell
- interaktivní rozhraní
- typické pro skriptovací jazyky

Vyhodnocení výrazu

Vyhodnocení

- atomu
  - » číslo → hodnota čísla
  - » operátor → funkce specifikovaná tímto operátorem
  - » ostatní → “objekty” k nim přiřazené
- seznamu
  - » vyhodnoť prvky seznamu
  - » nejlevější → funkce, ostatní → argumenty
  - » aplikuj funkci na její argumenty
- dychtivé vyhodnocení (eager evaluation)

Vyhodnocení výrazu

Výjimky z obecného vyhodnocovacího pravidla

- např. funkce `quote`
  - » potlačí vyhodnocení svého argumentu
- speciální formy
  - » mají vlastní vyhodnocovací pravidla

Obecné pravidlo pro vyhodnocení výrazu

Zvláštní pravidla pro malé množství speciálních forem

Seznamy

```
>(list 1 2 3)
(1 2 3)
>(car '(1 2 3))
1
>(cdr '(1 2 3))
(2 3)
>(cons 1 '(2 3))
(1 2 3)
>(cons 1 2)  ;vytvori tečka-dvojici
(1 . 2)
```

Předdefinované funkce/predikáty

```
>(append '(1 2) '(3 4))
(1 2 3 4)
>(list? 'a)
#f                    ; logická hodnota false
>(null? '())
#t
```

Předdefinované funkce/predikáty

Standardní funkce

- `list cons car cdr append`

Standardní predikáty

- `list? null? equal?`

Logické spojky

- `and or not`
- vše v prefixové notaci

Definice nových funkcí

Speciální forma `define`

```
(define e 2.71828)
> e
2.71828
(define (kvadrat x) (* x x))
> (kvadrat 21)
441
(define ( ⟨jméno⟩ ⟨form. parametry⟩ )
                    ⟨tělo⟩ )
```

Podmíněný výraz: speciální forma `if`

Speciální forma `if`

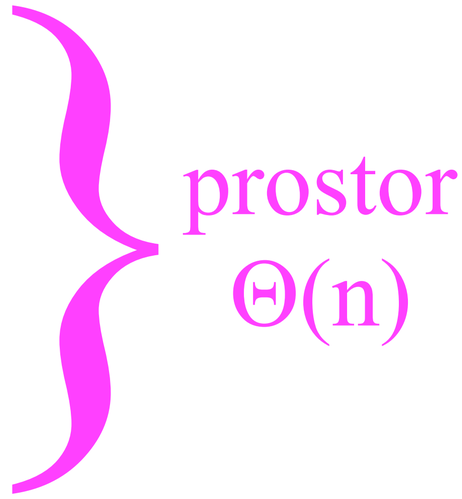
```
(if  ⟨podmínka⟩  ⟨výraz1⟩  ⟨výraz2⟩ )
```

```
(define (delka seznam)
     (if (null? seznam) 0
         (+ 1 (delka (cdr seznam)))
     )
)
```

Jako v Prologu: rekurze

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

```
(factorial 4)
(* 4 (factorial 3)
(* 4 (* 3 (factorial 2)))
(* 4 (* 3 (* 2 (factorial 1))))
(* 4 (* 3 (* 2 1)))
(* 4 (* 3 2))
(* 4 6)
 24
```



Jako v Prologu: akumulátor

```
(define (factorial n)
        (fact-iter 1 1 n))
(define (fact-iter soucin citac max)
   (if (> citac max) soucin
       (fact-iter (* citac soucin)
                  (+ citac 1)
                   max
        )
   )
)
```

fact-iter: příklad

```
(factorial 4)
(fact-iter  1 1 4)
(fact-iter  1 2 4)
(fact-iter  2 3 4)
(fact-iter  6 4 4)
(fact-iter 24 5 4)
 24
```

konstantní prostor

Lambda výrazy

Speciální forma `lambda`

- umožňuje definovat anonymní funkci

```
>((lambda (x y) (+ x y 1)) 4 5)
10
```

```
(define (kvadrat x) (* x x))
```

je syntaktické pozlátko pro

```
(define kvadrat (lambda (x) (* x x)))
```

Příklad: kompozice funkcí

```
(define (comp f g)
    (lambda (x) (f (g x))
))

(define (plus1 x) (+ x 1))
(define f (comp kvadrat plus1))

> (f 2)
9
```

Funkce vyšších řádů: map

```
>(map sqrt '(1 4 9))
(1 2 3)
(define (map fce seznam)
    (if (null? seznam) '()
        (cons (fce (car seznam))
              (map fce (cdr seznam)))))
(define (trans matice)     ; transpozice
 (if (null? (car matice)) '()
     (cons (map car matice)
           (trans (map cdr matice)))))
```

Kombinace – jako v Prologu

```
(define (komb rad seznam)
  (cond ((zero? rad)  '(()))
        ((null? seznam) '())
        (else (append
                (map (lambda (xs)
                       (cons (car seznam) xs))
                     (komb (- rad 1) (cdr seznam)))
                (komb rad (cdr seznam))))))
```

Haskell: historie

Haskell Brooks Curry (1900 - 1982)

1987 FPCA '87

- sestavena komise s cílem sloučit
- existující funkcionální jazyky do jediného
- který bude sloužit jako základ pro další výzkum

1990 Haskell 1.0

1999 Haskell 98 Report

2010 Haskell 2010

Haskell: charakteristika

Čistě funkcionální programovací jazyk

- funkce bez vedlejších efektů

Typový systém: silný & statický

- silně typovaný = argumenty předávané funkci musí být očekávaného typu
- statický = typová kontrola v době překladu
- typová inference Hindley–Milnerova typu
- typové třídy

Líné vyhodnocení

- nedělej dnes co můžeš odložit na zítřek
- Scheme: dychtivé vyhodnocení (eager evaluation)

Haskell: charakteristika

2D syntax (layout rule)

- závislé části programu určeny odsazením
- nahrazuje obvyklé příkazové závorky

Funkce vyšších řádů

Porovnávání se vzorem

- nemá sílu unifikace

Generické seznamy

- syntaktické pozlátko pro vytvoření seznamu na základě seznamů existujících
- imituje matematickou notaci pro konstrukci množin

Kombinace – jako v Prologu

```
{- kombinace bez opakovani -}

-- typova signatura
komb :: Int -> [a] -> [[a]]

-- definice funkce
komb  0    _              = [[]]
komb  _   []              = []
komb  n (x:xs) | n>0 = map (x:) (komb (n-1) xs)
                           ++ komb n xs
```