Jazyky pro umělou inteligenci

Miroslav Dobšíček - m.dobsicek@sh.cvut.cz

Zápočty: docházka 70%, účast na třech písemkách (6, 9, 12 týden), semestrální práce

Náplň: Základy funkcionálního a logického programování
Jazyky: Lisp a Prolog

Základy:
- praktické aplikace v oblasti umělé inteligence
- pro nás spíše nový pohled na řešení problémů
- založeno na rekurzi a matematické indukci
- proměnné se nemodifikují, pouze vznikají a zanikají
- typická je práce se seznamy
- i funkce je seznam, který interpret vyhodnocuje a redukuje
- dynamické typování
- automatická správa paměti
- zápis v prefixovém notaci

Rekurzivní funkce:
1. rekurzivní definice funkce (např. v Lispu)

```
sude n  = true              //pro n == 0
        = liche (pred n)    //jinak

liche n = false             //pro n == 0
        = sude (pred n)     //jinak
```

2. definice vzorem (např. v Prologu)

```
sude 0        = true
sude (succ n) = liche n

liche 0        = false
liche (succ n) = sude n
```

Druhy rekurze:
Funkce f je definována **vnořenou rekurzí** jestliže se v její definici objeví výraz tvaru (f...(f ...)...).
Ackermanova funkce ***ack x y = ack (x-1) (ack x (y-1))*** pro x,y různo od nuly.

O **kaskádní rekurzi** hovoříme když se několik volání funkce f objeví ve stejném výrazu bez vzájemného vnoření.
Fibonnačiho funkce ***fib n = fib (n-2) + fib (n-1)*** pro n větší jedné.

Lineární rekurze znamená, že v každé alternativě definice funkce je nejvýše jedno její volání.
Faktoriál ***fact n = n * fact (n-1)*** pro n různo od nuly.

Koncově rekurzivní funkce je speciální případ lineární rekurze, kdy poslední operací rekurze je daná alternativa funkce. Např sudé-liché.

LISP 1-1

Základní jednoduchý typ dat - atomy
1) číselné např. 434, -3.14
2) symbolické např. PEPIK, defun, *
Základní strukturovaný datový typ - seznam
() prázdný seznam
(a b c) tříprvkový seznam
(a (b (c d))) prvky seznamu jsou opět seznamy

seznam + atomy = symbolické výrazy (S-výrazy)
program stejně jako data mají tvar S-výrazů

Definice funkce:
(defun název (parametry) tělo)
návratová hodnota = výsledek posledního výrazu

Větvení programu:
(cond (podmínka1) (akce1) (podmínka2) (akce2) .....)

Predikáty:
t - vždy pravdivá hodnota
nil resp. () - vždy nepravdivá hodnota resp. prázdný seznam
zerop - test na nulovost argumentu
(zerop 0) -> t
(zerop 20) -> nil

Matematické funkce:
+, - , *, /, 1-, 1+
(+ 1 2) -> 3
(+ (* 2 3) (1+ 4)) -> 11

Faktoriál:
```
(defun fac (n)
  (cond ( (zerop n) 1 )
        ( t (* n (fac (- n 1))))
) )
```
příklad: (fac 3) -> 6

Součet čísel od 1 do n:
```
(defun sumall(n)
  (cond ( (zerop n) 0)
        ( t (+ n (sumall (1- n))))
) )
```
příklad: (sumall 5) -> 15

Mocnina a^n:
```
(defun mocnina(a n)
  (cond ( (zerop n) 1)
        ( t (* a (mocnina a (- n 1))))
) )
```
příklad: (mocnina 3 4) -> 81

Lisp 2-1

Funkce pro práci se seznamy:

cons - přidá prvek do seznamu
(cons 'a ()) -> (a)
(cons 'a (cons 'b ())) -> (a b)

car - vrátí první prvek ze seznamu
(car (cons 'a (cons 'b ()))) -> a

cdr - vrátí zbytek seznamu bez prvního prvku
(cdr (cons 'a (cons 'b (cons 'c ())))) -> (b c)

Predikáty:

null - test na prázdný seznam
(null ()) -> t
atom - test zda je operand atom
(atom '(a b)) -> nil
<, > - menší, větší
(> 3 5) -> nil

Největší společný dělitel Euklidovým algoritmem:

```
(defun nsd(a b)
  (cond ((zerop a) b)
        ((zerop b) a)
        ((> a b) (nsd (- a b) b))
        (t (nsd a (- b a)))
)  )
```

příklad: (nsd 91 35) -> 7

Počet prvků v nejvyšší úrovni seznamu:

```
(defun delka(s)
  (cond ((null s) 0)
        (t (+ 1 (delka (cdr s)))))) ;sestupujeme pouze po cdr větvy
```

příklad: (delka '(a b c d)) -> 4
příklad: (delka '(a (b c) d)) -> 3

Počet atomů ve víceúrovňovém seznamu:

```
(defun atomcount(s)
  (cond ((null s) 0)
        ((atom s) 1)
        (t (+ (atomcount (car s)) (atomcount (cdr s)))))) ;sestupujeme po car i cdr větvy
```

příklad: (atomcount '(a (b c) d)) -> 4

Poslední prvek v seznamu:

```
(defun posledni(s)
  (cond ((null s) nil)
        ((null (cdr s)) (car s)) ;poslední prvek v seznamu? tak ho vypíšeme
        (t (posledni (cdr s))))) ;sestupujeme dál po cdr větvy
```

příklad: (posledni '(a b c d)) -> d

Lisp 2-2

Funkce pro práci se seznamy:

append - z operandů, které jsou seznam vytvoří nový seznam
(append '(a)) -> (a)
(append '(a) ()) -> (a)
(append '(a) '(b) '(c d)) -> (a b c d)

list - z operandů vytvoří nový seznam
(list 'a '(b c)) -> (a (b c))
(list '(a b) '(c d)) -> ((a b) (c d))

Spojení dvou seznamů:

```
(defun spoj(x y)
  (cond ((null x) y) ;k prvkům z x spojených pomocí cons již zbývá připojit pouze seznam y
        (t (cons (car x) (spoj (cdr x) y))))) ;z x odebíráme prvky a řetězíme je pomocí cons
```

příklad: (spoj '(a b) '(c d)) -> (a b c d)

Spojení dvou setříděných seznamů:

```
(defun merger(a b)
  (cond ((null a) b)
        ((null b) a)
        ((> (car a) (car b)) (cons (car a) (merger (cdr a) b)))
        (t (cons (car b) (merger a (cdr b))))
) )
```

příklad: (merger '(5 2 1) '(4 3)) -> (5 4 3 2 1)

Obrácení pořadí prvků v seznamu:

```
(defun obrat(s)
  (cond ((null s) nil)
        (t (append (obrat (cdr s)) (list (car s))))))
```

příklad: (obrat '(a b c d)) -> (d c b a)

podrobněji:

(obrat '(a b)) ->
(append (obrat '(b)) (list 'a))
(append (append (obrat ()) (list 'b)) (list 'a))
(append (append nil (list 'b)) (list 'a))
(append '(b) '(a))
'(b a)