Funkcionální programování úvod

- Imperativní jazyky založeny na von Neumann architektuře
  - primárním kriteriem je efektivita
  - Modelem je Turingův stroj
  - Základní konstrukcí je příkaz
  - Příkazy mění stavový prostor programu
- Funkcionální jazyky založeny na matematických funkcích
  - Program definuje funkci
  - Výsledkem je fční hodnota
  - Modelem je lambda kalkul (Church 30tá léta)
  - Základní konstrukcí je výraz (definuje algoritmus i vstup)
  - Výrazy
    - Čisté = nemění stavový prostor programu
    - S vedlejším efektem = mění stavový prostor

Funkcionální programování úvod

Vlastnosti čistých výrazů:

- Hodnota výsledku nezávisí na pořadí vyhodnocování (tzv.Church-Roserova vlastnost)
- Výraz lze vyhodnocovat paralelně
- nahrazení podvýrazu jeho hodnotou je nezávislé na výrazu, ve kterém je uskutečněno (tzv. referenční transparentnost)
- -vyhodnocení nezpůsobuje vedlejší efekty
- -operandy operace jsou zřejmé ze zápisu výrazu
- -výsledky operace jsou zřejmé ze zápisu výrazu

Funkcionální programování- úvod

Př. nalezení největšího čísla

a) Ze dvou

max2(X, Y) def jestliže X > Y pak X jinak Y

b) Ze čtyř

max4(U, V, X, Y) def max2(max2(U, V), max2(X,Y))

c) Z n čísel

```
max(N)  def  jestliže délka(N) = 2
             pak max2(prvý-z(N), duhý-z(N))
             jinak max2(prvý-z(N), max(zbytek(N)))
```

Prostředky funkc. progr.:

- **Kompozice složitějších funkcí z jednodušších**
- **rekurze**

Funkcionální programování- úvod

Def.: Matematická fce je zobrazení prvků jedné množiny, nazývané definiční obor fce do druhé množiny, zvané obor hodnot

- Funkcionální program je tvořen výrazem E.
- Výraz E je redukován pomocí přepisovacích pravidel
- Proces redukce se opakuje až nelze dále redukovat
- Tím získaný výraz se nazývá normální formou výrazu E a je výstupem funkcionálního programu

Př. Aritmetický výraz E = (4 +7+ 10) * (5 – 2)

s přepis. pravidly určenými tabulkami pro +, *, …

Smysl výrazu je redukcemi zachován = vlastnost referenční transparentnosti je vlastností vyhodnocování. funkcionálního programu

Funkcionální programování- úvod

- **Church-Roserova věta:** Získání normální formy je nezávislé na pořadí vyhodnocování subvýrazů

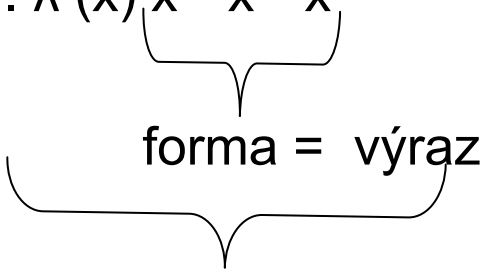
- Funkcionální program sestává z definic funkcí (algoritmu) a aplikace funkcí na argumenty (vstupní data).
- Aparát pro popis funkcí = **lambda kalkul**

  používá operaci aplikace fce F na argumenty A, psáno FA

  „ „ abstrakce ve tvaru λ (x) M [x]

  definuje fci (zobrazení) x → M [x]

Př. λ (x) x * x * x



definuje fci kubicka(x) = x * x * x lambda výrazem

Funkcionální programování- úvod

- Lambda výrazy popisují bezejmenné fce
- „ „ „ jsou aplikovány na parametry např.

  (λ (x) x * x * x) 5 = 5 * 5 * 5 = 125

  popis funkce argumenty

  aplikace (vyvolání) funkce
- Ve funkcionálním zápisu je zvykem používat prefixovou notaci ~ funkční notaci

  př. ((lambda (x) (* x x)) 5)

  ((lambda (y) ((lambda (x) (+ (* x x) y)) 2)) 3)

  → 7 přesvědčíme se?

Funkcionální programování- úvod

V LISPu lze zapisovat i

((lambda (y x) (+ (* x x) y)) 2 3) dá to také 7 ?

((lambda (y x) (+ (* x x) y)) 3 2)

Pořadí vyhodnocování argumentů:

- Všechny se vyhodnotí před aplikací fce = eager (dychtivé) evaluation
- Vyhodnotí se těsně před jeho použitím v aplikaci fce = lazy evaluation

Pochopit význam pojmu

- volná proměnná
- Vázaná proměnná

Funkcionální programování- LISP

- Vývoj
- Maclisp, Franclisp, Scheme, Commonlisp, Autolisp
- Použití:
  - UI (exp.sys., symb.manipulace, robotika, stroj.vidění, přiroz.jazyk)
  - Návrh VLSI
  - CAD
- Základní vlastnost: **Vše je seznamem**

  (program i data)

Funkcionální programování- LISP

Datové typy:

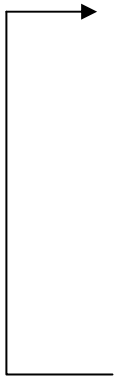
S - výrazy
- -atomy:
  - -čísla (celá i reálná)
  - -znaky
  - -řetězce "to je řetězec"
  - -symboly (T, NIL, ostatní)
    - k označování proměnných
    - a funkcí
- -seznamy (el1 el2 ...elN), NIL

- Seznamy mohou být jednoduché a vnořované
  např (sqrt (+ 3 8.1)) je seznam použitelný k vyvolání fce
- Velká a malá písmena se nerozlišují v CommonLispu

Funkcionální programování- LISP

Postup vyhodnocování (často interpretační):

Cyklus prováděný funkcí eval:



1. **Výpis promptu**
2. **Uživatel zadá lispovský výraz (zápis fce)**
3. **Provede se vyhodnocení argumentů**
4. **Aplikuje funkci na vyhodnocené argumenty**
5. **Vypíše se výsledek (fční hodnota)**

- **Pár příkladů s aritmet. funkcemi**
- **Při chybě se přejde do nové úrovně interpretu**
- **V chybovém stavu lze napsat help ¬**
- **Abort = návrat o úroveň výše**
- **(exit) ukončí Lisp**

Funkcionální programování- LISP

- Funkce pracují s určitými typy hodnot
- Typování je prostředkem zvyšujícím spolehlivost programů
- Typová kontrola:
  - -statická
  - -dynamická (Lisp, Prolog) – nevyžadují deklaraci typů argumentů a funkčních hodnot

Zajímavosti -komplexní čísla (* #c(2 2) #c(1.1 2.1))
-nekonečná aritmetika

Funkcionální programování- LISP

Elementární funkce

- **CAR FIRST selektor**
- **CDR REST selektor**
- **CONS konstruktor**
- **ATOM test**
- **EQUAL test**

Ostatní fce lze odvodit z elementárních

např. test prázdného seznamu

(NULL X) je (EQUAL X NIL)

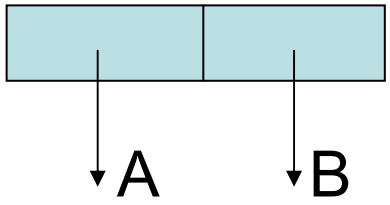
Jak zabránit vyhodnocování - QUOTE

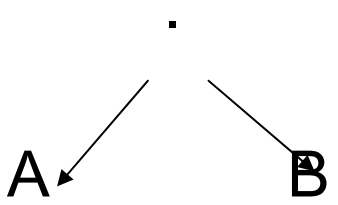
Př. (FIRST ‘(REST (1 2 3))) (FIRST (REST ‘(1 2 3)))

Funkcionální programování- LISP

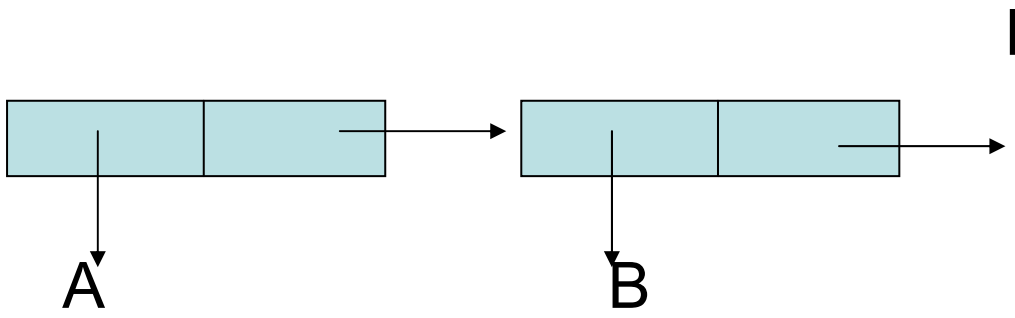
(CONS ‘a ‘b) $\rightarrow$ (A . B) tečka dvojice

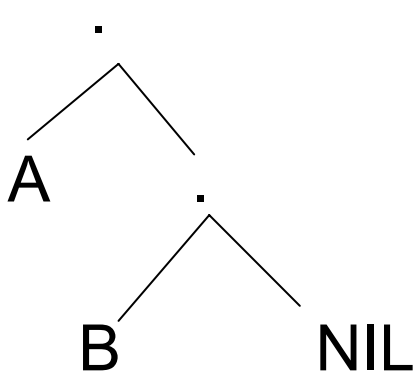
Lispovská buňka





(CONS ‘a ‘(b))





Funkcionální programování- LISP

Převod tečka notace do seznamové notace:

- Vyskytuje-li se „. „ před „(„ lze vynechat „. „ i „(„ i odpovídající „)„
- Při výskytu „. NIL„ lze „. NIL„ vynechat

Př.

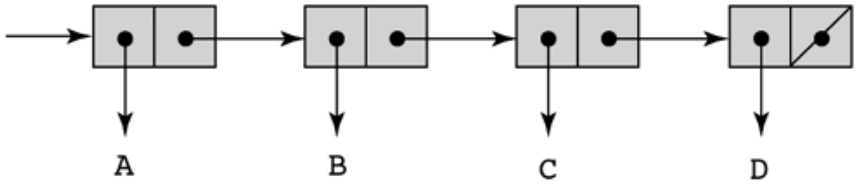
Seznam je S-výraz, který má na konci NIL

Forma je vyhodnotitelný výraz

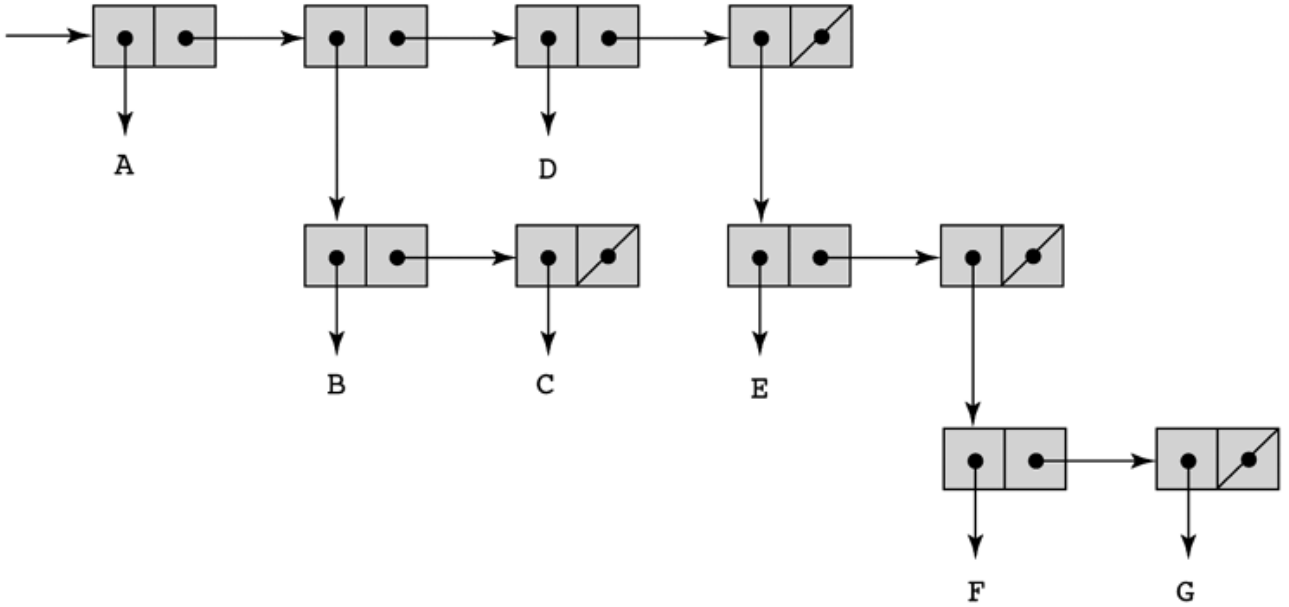
Funkcionální programování- LISP

Interní reprezentace

seznamů



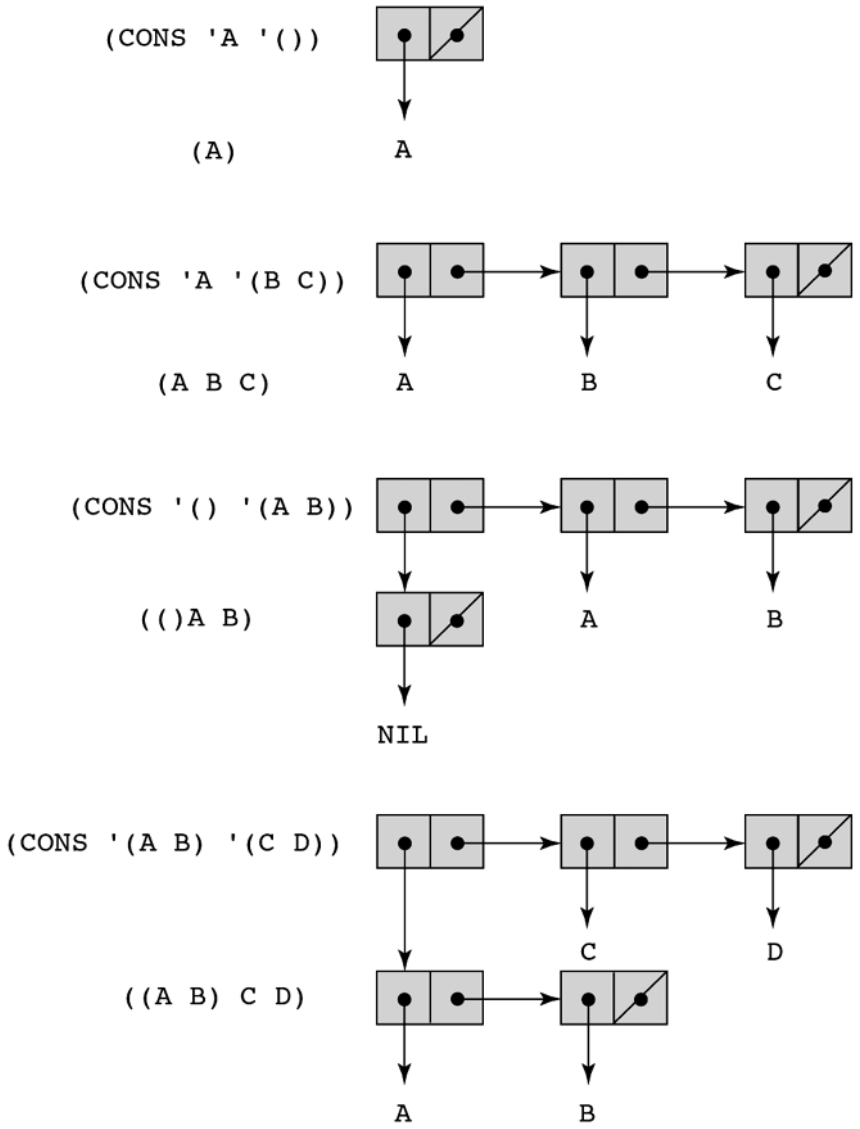
(A B C D)

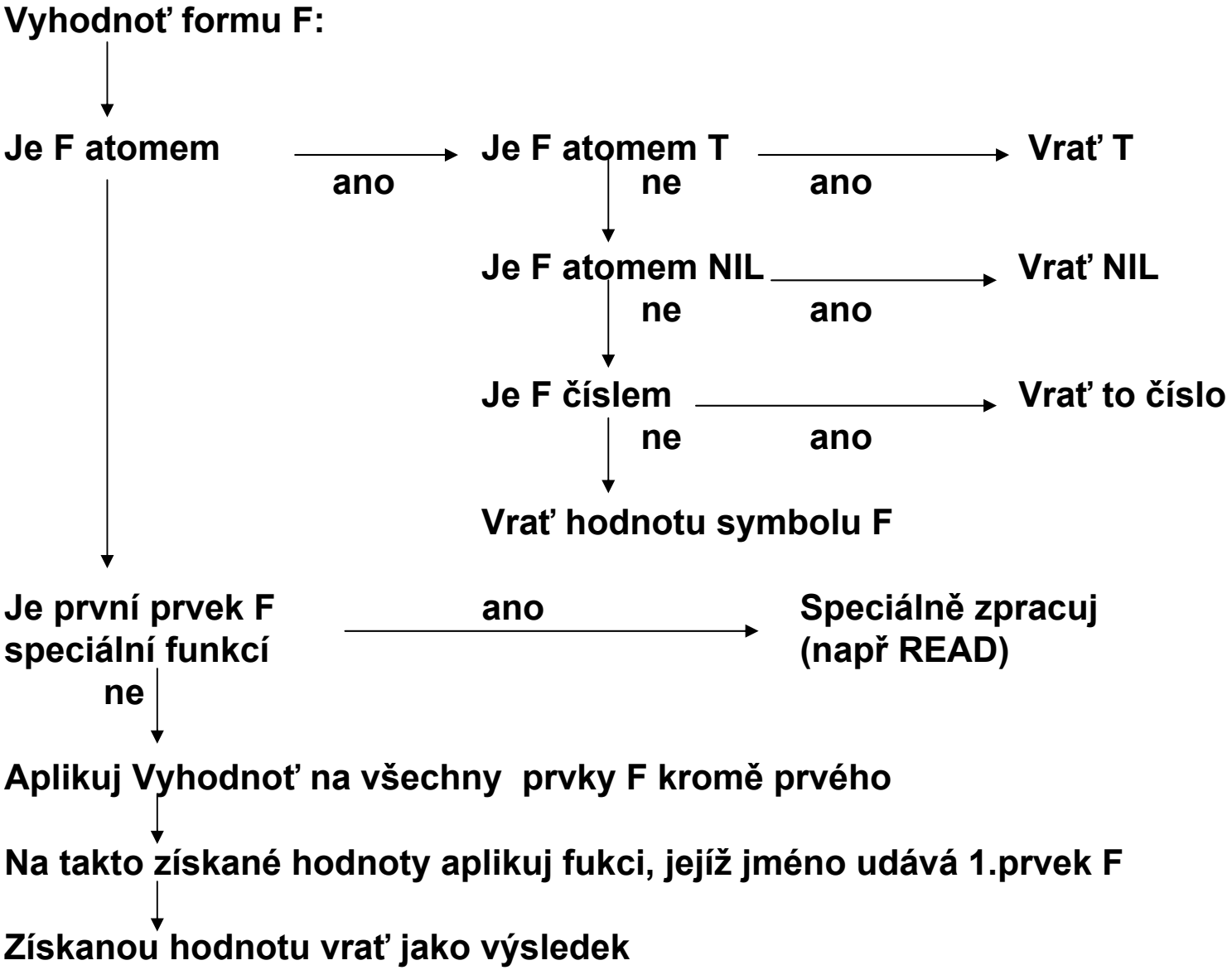


(A (B C) D (E (F G)))

Funkcionální programování- LISP

Výsledky fce CONS




Vyhodnoť formu F:
Je F atomem
ano
Je F atomem T
ano
Vrať T
ne
Je F atomem NIL
ano
Vrať NIL
ne
Je F číslem
ano
Vrať to číslo
ne
Vrať hodnotu symbolu F
Je první prvek F speciální funkcí
ano
Speciálně zpracuj (např READ)
ne
Aplikuj Vyhodnoť na všechny prvky F kromě prvého
Na takto získané hodnoty aplikuj fukci, jejíž jméno udává 1.prvek F
Získanou hodnotu vrať jako výsledek

Konstruktory APPEND a LIST

APPEND:: (seznam x seznam) → seznam

Vytvoří seznam z argumentů – vynechá jejich vnější závorky

Př. (APPEND ‘(A B) ‘(B A))

(APPEND ‘(A) ‘(B A))

Common Lisp připouští libovolně argumentů Append

(APPEND NIL ‘(A) NIL)

(APPEND () ‘(A) ())

(APPEND)

(APPEND ‘(A))

(APPEND ‘((B A)) ‘(A) ‘(B A))

výsledky (A B B A) (A B A)

výsledky (A) (A) nil (A) ((B A) A (B A)

Konstruktory APPEND a LIST

LIST:: (seznam x seznam x … x seznam) → seznam

Vytvoří seznam ze zadaných argumentů

Př.(LIST ‘A ‘(A B) ‘C) →

(A (A B) C)

(LIST ‘A) →

(A)

(LIST) →

nil

(LIST ‘(X (Y Z)) ‘(X Y)) →

((X (Y Z)) (X Y))

(APPEND ‘(X (Y Z)) ‘(X Y)) →

(X (Y Z) X Y)

(CONS ‘(X (Y Z)) ‘(X Y)) →

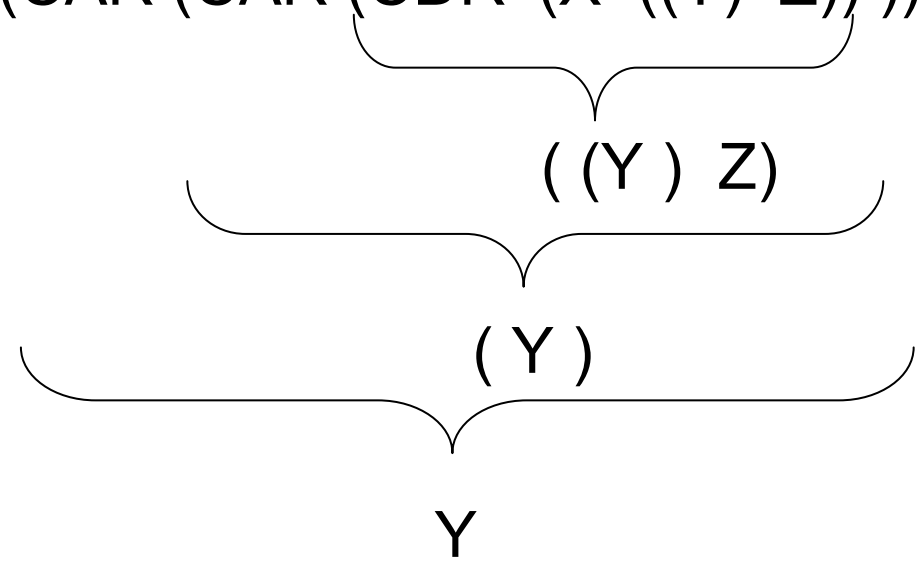
((X (Y Z)) X Y)

Selektory CAR a CDR lze vnořovat zkráceně

CxxR **CAAR, CADR, CDAR, CDDR**

CxxxR **CAAAR, …**

Př. (CAADR ‘(X (Y Z)) = (CAR (CAR (CDR ‘(X ((Y) Z)))))



(CADAR ‘ ((1 2 3) (4 5)))

2

LISP - Testy a větvení

ATOM

NULL

NUMBERP

SYMBOLP

LISTP

Př. (NULL NIL) → (NULL ( )) →

(LISTP ( )) → (LISTP NIL) →

(SYMBOLP NIL) → (SYMBOLP ( )) →

vše je T

(NUMBERP ‘(22)) (NUMBERP ’22) (NUMBERP 22)

NIL T T

LISP - Testy a větvení

=				jsou hodnoty argumentů stejná čísla?
EQ	„	„	„	stejné atomy?
EQUAL	„	„	„	stejné svýrazy? =hodnota
>	„	„	„	v sestupném pořadí?
>=	„	„	„	v nestoupajícím pořadí?
<	<=	obdobné		

Př. (EQ 3.0 3) (= 3.0 3)

(EQ ‘(A) ‘(A)) (EQ ‘A ‘A)

NIL T

(EQUAL ‘(A) ‘(A)) (EQUAL (+ 2 3 3) (CAR ‘(8))

T vypisují stejnou hodnotu T

(< 2 3 5 (* 4 4) 20) →

T

LISP - Testy a větvení

AND, OR mají libovolně argumentů, NOT má jen jeden

Všechny hodnoty různé od NIL považují za pravdivé

Argumenty vyhodnocují zleva doprava. Hodnotou fce je hodnota naposledy vyhodnoceného argumentu.

Používá zkrácený výpočet argumentů

Př. (AND (NULL NIL) (ATOM 5) (+ 4 3))

7

(OR NIL (= 2 (CAR '(2)) (+ 1 1) (- 3.0 1.0) 0) 9)

9

LISP - Testy a větvení

COND je spec. fcí s proměnným počtem argumentů

(COND (podm1 forma11 forma12 … forma1n)
(podm2 forma21 forma22 … forma2m)
. . .
(podmk formak1 formak2 … formako))

Postupně vyhodnocuje podmínky, dokud nenarazí na prvou, která je pravdivá. Pak vyhodnotí formy patřící k pravdivé podmínce. Hodnotou COND je hodnota poslední z vyhodnocených forem. Při nesplnění žádné z podm, není hodnota COND definována (u Common Lispu).

Pseudozápis pomocí IF:

LISP - Testy a větvení

COND if **podm1 then**

{ forma11 forma12 … forma1n}

else

if **podm2 then**

{ forma21 forma22 … forma2m}

else

. . .

if **podmk then**

{ formak1 formak2 … formako}

else NIL

LISP - Testy a větvení

IF forma

(IF podm then-část else-část)

Větvení se potřebuje k vytváření uživatelových fcí

Př.

((lambda (x) (if (< x 0) (* x x (- x)) (* x x x))) -5)

125

LISP - Přiřazování

Přiřazení je operace, která pojmenuje hodnotu a uloží ji do paměti

- Je ústupkem od čistě funkcionálního stylu
- Může zefektivnit a zpřehlednit i funkcionální program
- Mění vnitřní stav výpočtu (vedlejší efekt přiřazení)

Zahrnuje funkce pro: -I/O,

-pojmenování uživ. fcí

-pojm. hodnot symbolů (SET, SETQ)

(SETQ jméno-symbolu argument)

vrátí hodnotu argumentu a naváže hodnotu argumentu na

nevyhodnocené jméno symbolu

SET je obdobná, ale vyhodnotí i jméno symbolu

LISP - Přiřazování

(SETQ X 1) → 1

(SETQ X (+ 1 X) → 2

X → 2

>(SETQ LETADLO ‘BOING)

BOING

>LETADLO

BOING

>(SETQ BOING ‘JUMBO)

JUMBO

>(SETQ LETADLO BOING)

JUMBO

>LETADLO

JUMBO

LISP - Přiřazování

>(SET LETADLO ‘AEROBUS)

AEROBUS

>LETADLO

JUMBO

>JUMBO

AEROBUS

LISP - Přiřazování

```
(LET (
      (var-name-1 expression-1)
      (var-name-2 expression-2)
      ...
      (var-name-n expression-n))
      body
   )
```

- Vyhodnotí všechny výrazy, provede vazbu hodnot na jména a pak vyhodnotí tělo (v prostředí LET).
- Proměnné jsou lokální v LET příkazu, mimo neexistují
- Hodnotou LET je hodnota poslední formy těla LET.

Př.
```
(LET ((a 3) (b 4))
        (SQRT (+ (* a a) (* b b)))
)
5
```