KATEDRA INFORMATIKY
UNIVERZITA PALACKÉHO V OLOMOUCI
www.inf.upol.cz

Michal Krupka
krupka.inf.upol.cz
michal.krupka@upol.cz
17. listopadu 12, 771 46 Olomouc

Paradigmata programování 1 ◇ poznámky k přednášce

4. Rekurzivní výpočetní proces

verze z 16. října 2024

1 Rekurzivní funkce: opakování

Viz prezentaci.

2 Rekurzivní výpočetní proces

Výpočetní proces

je činnost, kterou vykonává počítač na základě nějakého programu nebo jeho části (funkce).

Výpočetní proces *generovaný* funkcí je výpočetní proces vykonávaný během její aplikace.

Rekurzivní výpočetní proces

Výpočetní proces je *rekurzivní*, když v něm **během** aplikace funkce dochází znovu k aplikaci téže funkce.

(Obecněji: během vykonávání části programu dochází k vykonávání téže části programu.)

- je poznat, když program běží
- některá aplikace funkce by k aplikaci téže funkce vést neměla (*ukončovací podmínka*)

Rekurzivní aplikace funkce

Aplikace funkce, ke které dojde během aplikace téže funkce.

Iterativní výpočetní proces

Výpočetní proces je *iterativní*, když v něm **po** aplikaci funkce dochází opět k aplikaci téže funkce.

(Obecněji: část programu se vykonává opakovaně.)

- bývá generován pomocí **cyklů**

- koncově rekurzivní funkce mohou generovat iterativní výpočetní proces

Překladač programovacího jazyka může koncově rekurzivní funkce optimalizovat tak, že je převede na cyklus. Tak to obvykle pro náš Lisp dělá program *PP Polyglot*. V tomto předmětu se obejdeme bez cyklů a k vytvoření iterativního výpočtu budeme používat koncovou rekurzi.

Minule jsme si ukázali rekurzivní funkci na výpočet obecné mocniny daného čísla:

```
(defun power (a n)
  (if (= n 0)
      1
    (* a (power a (- n 1)))))
```

V první části přednášky jsme podrobně zkoumali, jak u této funkce probíhá výpočet, a to v obecné i iterativní verzi, a dále, jaká prostředí při tom vznikají. Také jsme si ukázali, jak lze výpočet podstatně zrychlit. Podrobnosti můžete vidět na prezentaci.

3 Stromově rekurzivní výpočetní proces

Lineárně rekurzivní výpočetní proces

Výpočetní proces je *lineárně rekurzivní*, když během aplikace funkce po skončení jedné její rekurzivní aplikace nenásleduje další její rekurzivní aplikace.

Stromově rekurzivní výpočetní proces

Výpočetní proces je *stromově rekurzivní*, když alespoň jednou během aplikace funkce po skončení jedné její rekurzivní aplikace další její rekurzivní aplikace následuje.

Jako příklad si ukážeme dvě funkce, které generují stromově rekurzivní výpočetní proces.

Fibonacciho posloupnost

Jde o tuto posloupnost: $0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, \ldots$ Její prvky jsou dány následujícím předpisem:

$$a_0 = 0$$
$$a_1 = 1$$
$$a_n = a_{n-2} + a_{n-1}$$

(Každý prvek kromě prvních dvou je součtem předcházejících dvou prvků.) Následující funkce vrací daný prvek Fibonacciho posloupnosti:

```
(defun fib (n)
  (cond ((= n 0) 0)
        ((= n 1) 1)
        (t (+ (fib (- n 2)) (fib (- n 1))))))
```

Jak je vidět, funkce generuje stromově rekurzivní výpočetní proces.

Rozměňování

Máme k dispozici šest druhů mincí (50, 20, 10, 5, 2 a 1 Kč) a chceme v nich vyplatit danou částku. Řešíme problém, kolika způsoby je možné to udělat. Na přednášce jsem podrobně vysvětloval, jak se napíše funkce `count-change`, která to zjistí. Tady už pouze ukážu výsledek:

```
(defun count-change (amount)
  (cc amount 6))

(defun cc (amount kinds)
  (cond ((= amount 0) 1)
        ((or (< amount 0) (= kinds 0)) 0)
        (t (+ (cc amount (- kinds 1))
              (cc (- amount (first-denom kinds)) kinds)))))

(defun first-denom (kinds)
  (cond ((= kinds 1) 1)
        ((= kinds 2) 2)
        ((= kinds 3) 5)
        ((= kinds 4) 10)
        ((= kinds 5) 20)
        ((= kinds 6) 50)))
```

Otázky a úkoly na cvičení

1. Je možné ve funkci `fast-power` místo funkce `power2` použít funkci `fast-power`? Jinými slovy, je možné místo aplikace

   ```
   (power2 (fast-power a (/ n 2)))
   ```

   napsat

```
(fast-power (fast-power a (/ n 2)) 2)  ?
```

Odhadněte a zdůvodněte, k čemu by to vedlo. Teprve potom to vyzkoušejte.

2. Přepište funkci `fast-power` tak, aby používala koncovou rekurzi.

3. Napište predikát `dividesp`, který zjistí, zda dané celé číslo dělí beze zbytku jiné celé číslo:

```
CL-USER 1 > (dividesp 5 10)
Pravda

CL-USER 2 > (dividesp 1 17)
Pravda

CL-USER 3 > (dividesp 10 17)
NIL
```

Ke zjištění zbytku po dělení můžete použít funkci `rem`, kterou máme napsanou z dřívějšího cvičení. (Připomínám, že v Lispu může sloužit jako logická hodnota *Pravda* libovolná hodnota různá od `NIL`. Proto na prvních dvou řádcích nestanovujeme, jaká konkrétní hodnota to má být.) *(do jazyka)*

4. Prvočíslo je, jak známo, celé číslo větší než 1, které je dělitelné jen sebou samým a jedničkou. Napište predikát, který zjistí, zda dané číslo je prvočíslo:

```
CL-USER 4 > (primep 5)
Pravda

CL-USER 5 > (primep 17)
Pravda

CL-USER 6 > (primep 9)
NIL
```

Je možné predikát napsat bez použití speciálního operátoru `if`? *(do jazyka)*

5. *Dokonalé číslo* (*perfect number*) je číslo, které se rovná součtu všech svých dělitelů kromě sebe sama. Například číslo 6 je dokonalé, protože jeho dělitelé kromě 6 jsou 1, 2 a 3 a $1 + 2 + 3 = 6$. Číslo 12 není dokonalé, protože $1+2+3+4+6 = 16 \neq 12$. Napište predikát `perfectp`, který zjistí, zda dané číslo je dokonalé:

```
CL-USER 7 > (perfectp 6)
Pravda

CL-USER 8 > (perfectp 12)
NIL
```

6. Příklad na stromovou rekurzi. Toto je *Pascalův trojúhelník*:

$$
\begin{array}{ccccccccccc}
 & & & & & 1 & & & & & \\
 & & & & 1 & & 1 & & & & \\
 & & & 1 & & 2 & & 1 & & & \\
 & & 1 & & 3 & & 3 & & 1 & & \\
 & 1 & & 4 & & 6 & & 4 & & 1 & \\
1 & & 5 & & 10 & & 10 & & 5 & & 1 \\
 & 6 & & 15 & & 20 & & 15 & & 6 & \\
 & & & & & \vdots & & & & &
\end{array}
$$

   Každý řádek (kromě prvního) má o jeden prvek víc než předchozí řádek. Řádky vždy začínají a končí jedničkou, ostatní čísla se vypočítají jako součet dvou čísel ležících nad nimi. (Z obrázku by to mělo být vidět; trojúhelník samozřejmě pokračuje směrem dolů donekonečna.)

   Řádky i prvky v nich budeme číslovat od nuly, takže například prvek na řádku číslo 6 (tedy na sedmém řádku), který má na tom řádku pozici 2 (je tedy na řádku třetí), je 15.

   Napište funkci, která vrátí daný prvek Pascalova trojúhelníka vypočítaný uvedeným způsobem:

```
CL-USER 9 > (pascal 0 0)
1

CL-USER 10 > (pascal 3 1)
3

CL-USER 11 > (pascal 3 3)
1

CL-USER 12 > (pascal 4 2)
6

CL-USER 13 > (pascal 6 2)
15
```

7. Napište predikát `sum-of-squares-p`, který zjistí, zda je zadané nezáporné celé číslo součtem druhých mocnin po dvou různých celých čísel („po dvou různých“ znamená, že žádná dvě tato čísla nejsou totožná):

```
CL-USER 14 > (sum-of-squares-p 0)
Pravda

CL-USER 15 > (sum-of-squares-p 1)
Pravda
```

```
CL-USER 16 > (sum-of-squares-p 3)
NIL

CL-USER 17 > (sum-of-squares-p 10)
Pravda

CL-USER 18 > (sum-of-squares-p 14)
Pravda

CL-USER 19 > (sum-of-squares-p 15)
NIL
```

(Nula je součtem nulového počtu čtverců, dále $1 = 1^2$, $10 = 1^2 + 3^2$, $14 = 1^2 + 2^2 + 3^2$. Čísla 3 a 15 takto zapsat nelze.)

8. Všimli jsme si, že funkce `fib` je velice pomalá. Napište její rychlejší verzi. Zrychlení bude spočívat v tom, že funkce bude počítat každé Fibonacciho číslo jen jednou. (Nápověda: zkuste použít koncovou rekurzi.) *(do jazyka)*