

Paradigmata programování 1 ♦ poznámky k přednášce

4. Rekurze 2

verze z 23. října 2019

1 Jak probíhá rekurzivní výpočet

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)))))
```

K pochopení, jak funkce pracuje, si můžete představit, že pro každou hodnotu exponentu (parametr *n*) jsme napsali zvláštní funkci. V první části přednášky jsme podrobně zkoumali, jak probíhá výpočet při použití těchto funkcí i funkce **power** (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.

2 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.

- dosud uvedené rekurzivní funkce generovaly lineárně rekurzivní výpočetní proces

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, ... Její prvky jsou dány následujícím předpisem:

$$\begin{aligned}a_0 &= 0 \\a_1 &= 1 \\a_n &= a_{n-2} + a_{n-1}\end{aligned}$$

(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 generovala iterativní výpočetní proces.
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 `remainder`, kterou jsme si ukázali minule. (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.)

4. Mimochodem, kdybyste funkci `remainder` neměli k dispozici, můžete si ji nějak jednoduše napsat sami?
5. 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`?

6. *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
```

7. Jeden příklad na stromovou rekurzi. Toto je *Pascalův trojúhelník*:

```
      1
     1 1
    1 2 1
   1 3 3 1
  1 4 6 4 1
 1 5 10 10 5 1
1 6 15 20 15 6 1
  ⋮
```

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
```