Nyolc királynő négy sorban

Horváth Zoltán
hz@inf.elte.hu

Eötvös Loránd Tudományegyetem, Informatikai Kar

INFO Éra Konferencia, Zamárdi, 2016. nov. 25.

Funkcionális programozási nyelvek

- A deklaratív nyelvekhez tartoznak: a számítási folyamat leírása deklarációk halmaza
- Típus-, osztály-, függvénydefiníciók, kezdeti kifejezés
- A feladat specifikációját adjuk meg függvénykompozícó alakjában
- Nincs (előző értéket felülíró) értékadás, nincs szekvencia, nincs ciklus.
- A program végrehajtása a kezdeti kifejezés (függvénykompozíció) kiértékelése (átírás)
- A matematikai számítási modellje a λ-kalkulus (Church, 1932-33)

Példák függvények deinícióra és kezdeti kifejezésre

```
inc  x       = x + 1
square x     = x * x
squareinc x = square (inc x)
fact n       = prod [1..n]

fact 10
squareinc 7
```

Kiértékelési stratégiák

Szigorú (strict) kiértékelés:

```
squareinc 7 -> square (inc 7)
            -> square (7 + 1)
            -> square 8
            -> 8 * 8
            -> 64
```

Lusta (lazy) kiértékelés:

```
squareinc 7 -> square (inc 7)
            -> (inc 7) * (inc 7)
            -> 8 * (inc 7)
            -> 8 * 8
            -> 64
```

vagy párhuzamos kiértékelés gráfátírással

Egyszerű funkcionális programok

```
module Test
import StdEnv

Start =
  // 5 + 2 * 3
  // sum [1..10]
  // reverse (sort [1, 6, 2, 7])
  // 1 < 2 && 3 < 4
  // 2 < 1 || 3 < 4
  // [0:[1, 2]] ++ [3, 4, 5]
  // and [True, 2 < 1, 6 > 5 ]
  map abs  [7, -4, 3]


[ (x,y) \\ x <- [1..4], y <- [1..x]    | isEven x ]
// [(2,1),(2,2),(4,1),(4,2),(4,3),(4,4)]
```

Másodfokú egyenlet

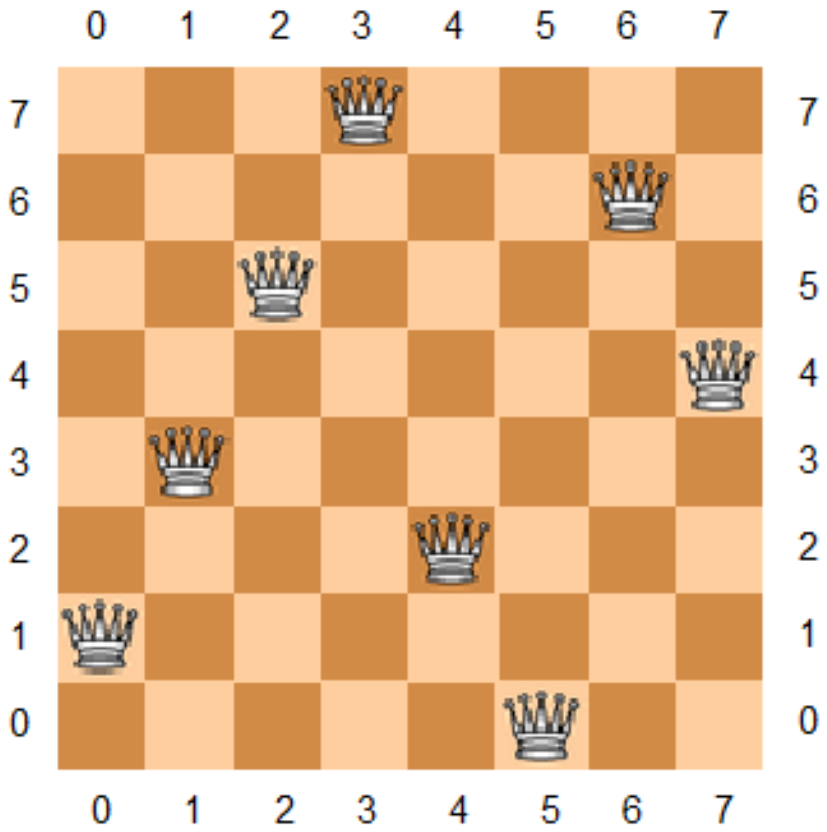
```
module Quadratic
import StdEnv

qeq :: Real Real Real -> (String, [Real])

qeq a b c
 | a == 0.0 = ("nem másodfokú" ,[])
 | d <  0.0 = ("komplex gyökök",[])
 | d == 0.0 = ("egy gyök",  [~b / (2.0 * a)])
 | d >  0.0 = ("két gyök",
               [(~b + r) / (2.0 * a)
               ,(~b - r) / (2.0 * a)])
 where
   d = b * b - 4.0 * a * c
   r = sqrt d

Start = qeq 1.0 (-4.0) 1.0
```

8 királynő



ábra: [1,3,5,7,2,0,6,4] q=2, b=[0,6,4], b !! 1 == 6

8 királynő

```
module Queens
import StdEnv



queens 0 = [[]]
queens n = [ [q:b]  \\ b <- queens (n - 1),
                       q <- [0..7]
                       |  safe q b ]

safe q b = and [not (checks q b i)
                         \\ i <- [0..(length b)-1]]

checks q b i =   q  ==  b !! i
                          ||  abs (q - b !! i)  ==  i + 1

Start  = (length (queens 8), queens 8)
```

Invariánsok

Gyűjtögető rekurzió és fúzió

```
average ls = s / toReal l
  where
    (s, l) = sumlength ls 0.0 0
    sumlength [x:xs] sum l
      = sumlength xs (sum + x) (l + 1)
    sumlength [] sum l = (sum, l)
```

Az invariáns teljesül kezdetben és fennmarad:

$$length(x : xs) + l == length(ls) \wedge \sum(x : xs) + sum == \sum(ls) \Rightarrow$$

$$length(xs) + (l + 1) == length(ls) \wedge \sum(xs) + (sum + x) == \sum(ls)$$

A végén:

$$length([]) + l == length(ls) \wedge \sum([]) + sum == \sum(ls)$$

A funkcionális programozás előnyei

- a feladatra fókuszálunk, magas absztrakciós szintű leírás, magasabbrendű függvények (pl. foldr és and, sum, prod)
- a programsorok száma ötöde, tizede - a hibák száma is ötöde, tizede
- gyors, párhuzamos kiértékelés lehetősége a kód átírása nélkül
- hivatkozási helyfüggetlenség: a helyesség bizonyítás könnyebb, matematikában megszokott módon
- erős típusosság - statikus ellenőrzés, futási hibák kizárása, nagy megbízhatóság
- alkalmazás: telekommunikáció, pénzügyi szektor, workflow rendszerek (haditengerészet), alkalmazási terület specifikus nyelvek

interaktív oktatás középiskolásoknak:
http://lambda.inf.elte.hu/fp/Middle.xml

Kiértékelés

- Kiértékelés = átírási lépések sorozata (redukció)
- Függvénydefiníció – példa: `sqr x = x * x`
  függvény azonosítója, formális paraméterek, függvény törzs (kifejezés)
- Kiszámíthatóság, hatékonyság
- Átírási lépés: függvény alkalmazásában a függvény helyettesítése a függvény törzsével (a normál forma eléréséig)
- Kiértékelési stratégia: redexek (reducible expressions) kiválasztási sorrendje, például lusta (először a függvény), mohó (először az argumentumok) vagy párhuzamos
- Egyértelmű normál forma (konfluens átíró rendszerekben), a lusta kiértékelés mindig megtalálja a normál formát, ha az létezik

Modern funkcionális programozási nyelvek jellemzése

- Nincs előző értéket megsemmisítő értékadás
- Hivatkozási helyfüggetlenség – egyenlőségi érvelés (azonos kifejezés értéke mindig ugyanaz)
- Szigorúan típusos (minden részkifejezésnek fordítási időben meghatározott a típusa), típuslevezetés, polimorfizmus, absztrakt és algebrai adattípusok
- Magasabbrendű függvények (az argumentum vagy érték is függvény)
  `twice f x = f (f x)`
- Curry-féle módszer – minden függvénynek 1 argumentuma van
  `(+) x y` kontra `((+) x) y`
- Rekurzió
- Lusta kiértékelés a mohóság vizsgálatával
  `f x = 0; f (5 + 1); 2 * (5 + 1)`

Modern funkcionális programozási nyelvek jellemzése

- Zermelo-Fraenkel halmazkifejezések

```
{x * x \\ x <- [1..] | odd(x)}
```

- Argumentumok mintaillesztése

```
fac 0 = 1
fac n | n > 0 = n * fac (n - 1)
```

- Margószabály

```
add4 = twice succ   where
  succ x = x + 2
  add    = ... succ ...
```

- I/O modellek: I/O adatfolyam, monádok, egyszeres hivatkozás