Quick Check

Illés János
AAIT

Quick Check tömören

- Automatikus tesztelő eszköz
- Véleletlen adatokat generál és teszteseteket futtat velük
- Ha hibát talál megpróbálja minimálisra zsugorítani

Haskell

- Funkcionális programozás
  - függvény is adat
  - szereti a rekurziót
- Lusta (lazy evaluation)
- (Nagyon) erősen típusos
- Tiszta (pure) - mellékhatásmentes
  - ugyanarra a bemenetre ugyanaz a kimenet
  - egy függvény csak a bemenettől függ!
  - nincs globális állapot
- A tiszta kód csak melegíti a gépet?!
  - IO és monad -ok (kategória elmélet)
- "avoid success at all costs"

Haskell

- Előnyök
  - új szemlélet
  - tömör, átgondolt kód
  - könyebb érvelni a kód mellett
  - gyors
  - a típusrendszer
- Hátrányok
  - új szemlélet
  - a típusrendszer
  - átszokni nehéz
  - nem minden problémára illeszkedik jól (pl GUI)

Haskell példa - quicksort

```
quicksort :: Ord a => [a] -> [a]
quicksort []     = []
quicksort (p:xs) = (quicksort lesser) ++ [p] ++ (quicksort greater)
    where
        lesser  = filter (< p) xs
        greater = filter (>= p) xs
```

Haskell példa - quicksort

```
quicksort :: Ord a => [a] -> [a]
quicksort []     = []
quicksort (p:xs) = (quicksort lesser) ++ [p] ++ (quicksort greater)
    where
        lesser  = filter (< p) xs
        greater = filter (>= p) xs
```

C nyelven

```
// To sort array a[] of size n: qsort(a,0,n-1)

void qsort(int a[], int lo, int hi)
{
  int h, l, p, t;

  if (lo < hi) {
    l = lo;
    h = hi;
    p = a[hi];

    do {
      while ((l < h) && (a[l] <= p))
          l = l+1;
      while ((h > l) && (a[h] >= p))
          h = h-1;
      if (l < h) {
          t = a[l];
          a[l] = a[h];
          a[h] = t;
      }
    } while (l < h);

    a[hi] = a[l];
    a[l] = p;

    qsort( a, lo, l-1 );
    qsort( a, l+1, hi );
  }
}
```

Miért Quick Check?

Hogyan tesztelhetünk?

- Formális specifikáció és bizonyítás
  - nehéz, sok idő és drága
- Kézzel írt tesztesetek
  - sok idő és erőforrás (mock/stub gyártás)
  - nem biztos hogy mindent lefedünk
  - átsiklunk cornercasek fölött
- Létezik középút?

Hogyan?

- Nem írunk teszteseteket
- Nem is írunk teljes formális specifikációt

Helyette:

- Tulajdonságokat (property) keresünk
  - Függvény mint tesztelendő egység

Rövid példa

Daraboljunk karaktersorozatot!

```
split c [] = []
split c xs = xs' : if null xs'' then [] else split c (tail xs'')
     where xs' = takeWhile (/=c) xs
           xs''= dropWhile (/=c) xs
```

```
split '@' "pbv@dcc.fc.up.pt" = ["pbv","dcc.fc.up.pt"]
split '/' "/usr/include" = ["", "usr", "include"]
```

Rövid példa

1. lépés: Keressünk tulajdonságot

```
prop_join_split c xs = join c (split c xs) == xs
```

Rövid példa

1. lépés: Keressünk tulajdonságot

```
prop_join_split c xs = join c (split c xs) == xs
```

2. lépés: kész vagyunk.

Ellenpélda

Véletlen adatokkal eteti a függvényt

- Alapból 100 -al, de tetszőlegesen sokkal

Ha ellenpéldát talál jelzi, pl:

- **`Falsifiable, after 48 tests:`**
  `'a' "a"`

Miért is?

```
split 'a' "a" = [""]
join 'a' [""] = ""
```

Zsugorítás

```
[1, 2, 4, 5, 6, 7, -1]  -> [-1]
```

Hova dugtam a típusokat?

A Haskell nagyon erősen típusrendszere megoldja helyettünk.

- Hindley–Milner
- Type inference
- Ahol csak teheti polimorfikus

QuickCheck buktatók

- Nem látjuk a tesztadatot
- Fontos a tesztadat eloszlása
- A lefedettség vizsgálata a felhasználóra van bízva
- A propertyket meg is kell találni

Nem csak Haskell

- Használhatjuk teszt-eset generálásra
  - Ehhez modelleznünk kell Haskellben az adatstruktúráinakat
- Használhatjuk nativan
  - implementálták sok nyelvre:
  - **C, C++,** Chicken Scheme, Clojure, Common Lisp, **D**, Erlang, F#, Factor, Io, **Java, JavaScript, Node.js, Objective-C,** OCaml, **Perl, Python, Ruby,** Scala, Scheme, **Smalltalk**, Standard ML.

Köszönöm a figyelmet