

Quickcheck – Spezifikationsbasiertes Testen von Haskellprogrammen

(Koen Claessen, John Hughes)

- Testen kann nie das Fehlen von Fehlern zeigen, sondern lediglich vorhandene Fehler aufdecken (Dijkstra)
- Quickcheck
 - erlaubt systematisches Testen von (Haskell-) Programmen
 - liefert keine Korrektheitsnachweise
 - erhöht nur das Vertrauen in Programme



Quickcheck

- definiert eine formale **Spezifikationssprache**
 - um zu testende Eigenschaften direkt im Source-Code auszudrücken
- definiert eine **Testdaten-Generierungssprache**
 - um große Testmengen kompakt zu beschreiben
- umfasst ein **Werkzeug**
 - um die spezifizierten Eigenschaften zu testen und eventuell entdeckte Fehler zu melden
- wird mit **Haskell Plattform** ausgeliefert
(cabal update; cabal install QuickCheck-2.4.0.1)



Einfache Beispiele

Eigenschaften sind monomorphe Boolesche Funktionen.

- **einfach:**

```
prop_PA :: Int -> Int -> Int -> Bool
```

```
prop_PA x y z = (x+y)+z == x+(y+z)
```

Aufrufe: **quickCheck prop_PA** oder
 verboseCheck prop_PA

- **bedingt:**

```
prop_InsertOrdered :: Int -> [Int] -> Property
```

```
prop_InsertOrdered x xs
```

```
    = ordered xs ==> ordered (insert x xs)
```



Fallstudie: Warteschlangen

```
emptyQueue    :: Queue a
enqueue       :: Queue a -> a -> Queue a
dequeue       :: Queue a -> Queue a
firstQueue    :: Queue a -> a
isEmptyQueue  :: Queue a -> Bool
```

Gleichungsspezifikation:

```
dequeue (enqueue emptyQueue x) = emptyQueue
isEmptyQueue q = False
  -> dequeue (enqueue q x) = enqueue (dequeue q) x
```

```
firstQueue (enqueue emptyQueue x) = x
isEmptyQueue q = False
  -> firstQueue (enqueue q x) = firstQueue q
```

```
isEmptyQueue emptyQueue    = True
isEmptyQueue (enqueue q x) = False
```



Testfunktionen zu Gleichungen

```
-- dequeue (enqueue emptyQueue x) = emptyQueue
```

```
prop_1    :: Int -> Bool
```

```
prop_1 x = dequeue (enqueue emptyQueue x) == emptyQueue
```

```
-- isEmptyQueue q = False ->
```

```
--      dequeue (enqueue q x) = enqueue (dequeue q) x
```

```
prop_2    :: Queue Int -> Int -> Property
```

```
prop_2 q x = isEmptyQueue q == False ==>
```

```
    dequeue (enqueue q x) == enqueue (dequeue q) x
```

```
-- firstQueue (enqueue emptyQueue x) = x
```

```
prop_3    :: Int -> Bool
```

```
prop_3 x = firstQueue (enqueue emptyQueue x) == x
```



Testfunktionen zu Gleichungen 2

```
-- isEmptyQueue q = False ->
--           firstQueue (enqueue q x) = firstQueue q
prop_4 :: Queue Int -> Int -> Property
prop_4 q x = isEmptyQueue q == False ==>
           firstQueue (enqueue q x) == firstQueue q

-- isEmptyQueue emptyQueue      = True
-- isEmptyQueue (enqueue q x) = False
prop_5 :: Queue Int -> Int -> Bool
prop_5 q x = (isEmptyQueue emptyQueue == True) &&
           (isEmptyQueue (enqueue q x) == False)
```



Testkontrolle durch Klassifikation von Testfällen

- `classify :: Bool -> String -> Property -> Property`

Beispiel:

```
prop_InsertOrdered x xs
  = ordered xs =>
    (classify (null xs) „empty“
     (classify (length xs == 1) „unit“
              (ordered(insert x xs))))
```

- `collect :: Int -> Property -> Property`

Beispiel:

```
prop_InsertOrdered x xs
  = ordered xs =>
    (collect (length xs)
             (ordered (insert x xs)))
```



Fazit

- leichte Überprüfung von Spezifikationen und Implementierungen durch systematisches Testen
- kein Korrektheitsnachweis, aber viele Fehler oder Inkonsistenzen werden aufgedeckt
- flexibel durch Möglichkeit der Testgenerierung und -kontrolle