Paradigmas de Programação

Fabrício Olivetti de França

26 de Junho de 2018

Quickcheck

Testando propriedades do algoritmo

Uma parte importante da Engenharia de Software é o teste de seu produto final. Dado que o programa compilou sem erros, ele faz o que é esperado?

O Haskell permite, em algumas situações, provar matematicamente que seu programa está correto (usando indução).

Outra forma de verificar a corretude é fazer testes de entrada e saída das funções criadas e verificar se elas apresentam as propriedades esperadas.

Testando propriedades do algoritmo

Se você criou um novo algoritmo de ordenação, que propriedades são esperadas?

- A lista de saída está ordenada
- A lista de saída tem o mesmo tamanho da lista de entrada
- A lista de saída contém os mesmos elementos da lista de entrada

Criando um projeto

Vamos criar nosso primeiro projeto completo com o *stack*:

```
> stack new quickcheck simple
```

Criando um projeto

Edite o arquivo *quickcheck.cabal* e acrescente o seguinte ao final da linha *build-depends*:

build-depends: base >= 4.7 && <5, ***QuickCheck***

Criando um projeto

Digite:

```
> stack setup
> stack build
```

Testando QuickSort

Uma implementação famosa de QuickSort funcional é, copie os códigos seguintes no arquivo *Main.hs*:

```
import Test.QuickCheck

qsort :: Ord a => [a] -> [a]
qsort []     = []
qsort (x:xs) = qsort lhs ++ [x] ++ qsort rhs
  where
    lhs = filter (< x)  xs
    rhs = filter (> x) xs
```

Esse código contém um erro!

Funções de propriedades

Vamos testar uma primeira propriedade de algoritmos de ordenação: **idempotência**.

Queremos mostrar que `qsort (qsort xs) == qsort xs`:

```
prop_idempotencia :: Ord a => [a] -> Bool
prop_idempotencia xs = qsort (qsort xs) == qsort xs
```

Idempotência

Vamos testar essa função no ghci (use *stack ghci src/Main.hs*):

```
> :l quickcheck.hs
> prop_idempotencia [1]
True
> prop_idempotencia [1,2,3,4]
True
> prop_idempotencia [3,2,4,1]
True
> prop_idempotencia [4,3,2,1]
True
> prop_idempotencia []
True
```

Tamanho da lista

Outra propriedade é que o tamanho da lista seja o mesmo após a execução do algoritmo:

```
prop_length :: Ord a => [a] -> Bool
prop_length xs = length (qsort xs) == length xs
```

Tamanho da lista

```
> :l quickcheck.hs
> prop_length [1]
True
> prop_length [1,2,3,4]
True
> prop_length [3,2,4,1]
True
> prop_length [4,3,2,1]
True
> prop_length []
True
```

Casos de teste

Os casos de teste utilizado são representativos?

A biblioteca `QuickCheck` automatiza a geração de dados para testes (e faz outras coisas úteis também).

Verificando nossas propriedades

```
> quickCheck prop_idempotencia
+++ OK, passed 100 tests.
> quickCheck prop_length
*** Failed! Falsifiable (after 4 tests):
[(),()]
```

Oops!

QuickCheck

A biblioteca QuickCheck gera casos de testes progressivos, começando de casos simples até casos mais complexos em busca de erros.

Ao encontrar um erro, ele retorna a instância mais simples que deu errado.

QuickCheck

Para entender melhor vamos executar essa função para listas de inteiros:

```
> quickCheck (prop_length :: [Int] -> Bool)
*** Failed! Falsifiable (after 5 tests and 1 shrink):
[1,1]
```

O que houve?

QuickCheck

```
> qsort [1,1]
[1]
```

Corrigindo o QuickSort

Basta alterar para não descartar os elementos iguais a x:

```
import Test.QuickCheck

qsort :: Ord a => [a] -> [a]
qsort []     = []
qsort (x:xs) = qsort lhs ++ [x] ++ qsort rhs
  where
    lhs = filter (<= x)  xs
    rhs = filter (> x) xs
```

QuickCheck

Para entender melhor vamos executar essa função para listas de inteiros:

```
> quickCheck (prop_length :: [Int] -> Bool)
+++ OK, passed 100 tests.
```

Mínimo == head

Outra propriedade é que primeiro elemento da lista é igual ao mínimo:

```
prop_minimum :: Ord a => [a] -> Bool
prop_minimum xs = head (qsort xs) == minimum xs
```

Mínimo == head

Vamos testar essa função no ghci (use *stack ghci src/Main.hs*):

```
> quickCheck prop_minimum
*** Failed! Exception:
'Prelude.head: empty list' (after 1 test):
[]
```

Mínimo == head

Tanto a função `minimum` quanto a função `head` retornam erro em listas vazias, podemos especificar que não queremos testar instâncias nulas com o operador ==>:

```
prop_minimum :: Ord a => [a] -> Property
prop_minimum xs = not (null xs)
                  ==> head (qsort xs) == minimum xs
```

Esse operador retorna uma propriedade interpretável pelo quickCheck.

Mínimo == head

Vamos testar essa função no ghci (use *stack ghci src/Main.hs*):

```
> quickCheck prop_minimum
+++ OK, passed 100 tests.
```

Testando com um modelo

Finalmente, se temos um algoritmo que cumpre a mesma tarefa e temos certeza de que está correto, podemos usá-lo na comparação:

```haskell
import Data.List  -- sort

prop_model :: Ord a => [a] -> Bool
prop_model xs = qsort xs == sort xs
```