COMP4075/G54RFP: Lecture 15

Property-based Testing

Henrik Nilsson

University of Nottingham, UK

QuickCheck: What is it? (1)

- Framework for property-based testing
- Flexible language for stating properties
- Random test cases generated automatically based on type of argument(s) to properties.
- Highly configurable:
  - Number, size of test cases can easily be specified
  - Additional types for more fine-grained control of test case generation
  - Customised test case generators

QuickCheck: What is it? (2)

- Support for checking test coverage
- Counterexample produced when test case fails
- Counterexamples automatically shrunk in attempt to find minimal counterexample

Basic Example

$$\textbf{import}\ Test.QuickCheck$$

$$prop\_RevRev :: [Int] \to Bool$$

$$prop\_RevRev\ xs =$$

$$\quad reverse\ (reverse\ xs) \equiv xs$$

$$prop\_RevApp :: [Int] \to [Int] \to Bool$$

$$prop\_RevApp\ xs\ ys =$$

$$\quad reverse\ (xs +\!\!+\ ys) \equiv reverse\ ys +\!\!+\ reverse\ xs$$

$$quickCheck\ (prop\_RevRev\ .\&\&.\ prop\_RevApp)$$

Basic Example

$$\textbf{import}\ \mathit{Test.QuickCheck}$$

$$prop\_RevRev :: [\,Int\,] \rightarrow Bool$$
$$prop\_RevRev\ xs =$$
$$\quad reverse\ (reverse\ xs) \equiv xs$$

$$prop\_RevApp :: [\,Int\,] \rightarrow [\,Int\,] \rightarrow Bool$$
$$prop\_RevApp\ xs\ ys =$$
$$\quad reverse\ (xs +\!\!+\ ys) \equiv reverse\ ys +\!\!+\ reverse\ xs$$

$$quickCheck\ (prop\_RevRev\ .\&\&.\ prop\_RevApp)$$

Result: `+++ OK, passed 100 tests`

Class *Testable*

Type of quickCheck:

$$quickCheck :: Testable\ prop \Rightarrow prop \rightarrow IO\ ()$$

Class *Testable*

Type of quickCheck:

$$quickCheck :: Testable\ prop \Rightarrow prop \rightarrow IO\ ()$$

Testable and some instances:

$$\begin{array}{l}
\textbf{class}\ Testable\ prop\ \textbf{where} \\
\quad property \quad :: prop \rightarrow Property \\
\quad exhaustive :: prop \rightarrow Bool \\
\textbf{instance}\ Testable\ Bool \\
\textbf{instance}\ Testable\ Property \\
\textbf{instance}\ (Arbitrary\ a, Show\ a, Testable\ prop) \Rightarrow \\
\qquad\qquad Testable\ (a \rightarrow prop)
\end{array}$$

Class *Arbitrary*

$$
\begin{array}{l}
\textbf{class}\ \mathit{Arbitrary}\ a\ \textbf{where} \\
\quad \mathit{arbitrary} :: \mathit{Gen}\ a \\
\quad \mathit{shrink} :: a \to [a]
\end{array}
$$

$$\mathit{generate} :: \mathit{Gen}\ a \to \mathit{IO}\ a$$

Arbitrary instance for all basic types provided. Easy to define additional ones.

Class *Arbitrary*

$$
\begin{array}{l}
\textbf{class}\ \mathit{Arbitrary}\ a\ \textbf{where} \\
\quad \mathit{arbitrary} :: \mathit{Gen}\ a \\
\quad \mathit{shrink} :: a \to [a]
\end{array}
$$

$$\mathit{generate} :: \mathit{Gen}\ a \to \mathit{IO}\ a$$

Arbitrary instance for all basic types provided. Easy to define additional ones.

Gen is a *Monad*, *Applicative*, *Functor* (and more).

Class *Arbitrary*

$$\textbf{class}\ Arbitrary\ a\ \textbf{where}$$
$$arbitrary :: Gen\ a$$
$$shrink :: a \to [a]$$

$$generate :: Gen\ a \to IO\ a$$

Arbitrary instance for all basic types provided. Easy to define additional ones.

Gen is a *Monad*, *Applicative*, *Functor* (and more).

Example:

$$generate\ (arbitrary :: Gen\ [Int])$$

Result: `[28,-2,-26,6,8,8,1]`

Stating Properties (1)

Implication:

$$(==>) :: Testable\ prop \Rightarrow Bool \rightarrow prop \rightarrow Property$$

Stating Properties (1)

Implication:

$$(==>) :: \mathit{Testable\ prop} \Rightarrow \mathit{Bool} \to \mathit{prop} \to \mathit{Property}$$

Universal quantification:

$$\begin{aligned} \mathit{forAll} :: (&\mathit{Show}\ a, \mathit{Testable\ prop}) \Rightarrow \\ &\mathit{Gen}\ a \to (a \to \mathit{prop}) \to \mathit{Property} \end{aligned}$$

Stating Properties (1)

Implication:

$$(==>) :: \mathit{Testable}\ \mathit{prop} \Rightarrow \mathit{Bool} \rightarrow \mathit{prop} \rightarrow \mathit{Property}$$

Universal quantification:

$$\begin{aligned} \mathit{forAll} :: (&\mathit{Show}\ a, \mathit{Testable}\ \mathit{prop}) \Rightarrow \\ &\mathit{Gen}\ a \rightarrow (a \rightarrow \mathit{prop}) \rightarrow \mathit{Property} \end{aligned}$$

Conjunction and disjunction:

$$\begin{aligned} (.\&\&.) :: &(\mathit{Testable}\ \mathit{prop1}, \mathit{Testable}\ \mathit{prop2}) \\ &\Rightarrow \mathit{prop1} \rightarrow \mathit{prop2} \rightarrow \mathit{Property} \\ (.||.) :: &(\mathit{Testable}\ \mathit{prop1}, \mathit{Testable}\ \mathit{prop2}) \\ &\Rightarrow \mathit{prop1} \rightarrow \mathit{prop2} \rightarrow \mathit{Property} \end{aligned}$$

Stating Properties (2)

$$
\begin{aligned}
&\mathit{prop\_Index} :: \mathit{Eq}\ a \Rightarrow [\,a\,] \rightarrow \mathit{Property}\\
&\mathit{prop\_Index}\ \mathit{xs} =\\
&\quad \mathit{length}\ \mathit{xs} > 0 ==>\\
&\qquad \mathit{forAll}\ (\mathit{choose}\ (0, \mathit{length}\ \mathit{xs} - 1))\ \$\ \lambda i \rightarrow\\
&\qquad\quad \mathit{xs}\ !!\ i \equiv \mathit{head}\ (\mathit{drop}\ i\ \mathit{xs})
\end{aligned}
$$

Modifiers (1)

A number of newtypes with $Arbitrary$ instances. E.g. $NonEmptyList\ a$, $SortedList\ a$, $NonNegative\ a$

Modifiers (1)

A number of newtypes with $Arbitrary$ instances. E.g. $NonEmptyList\ a$, $SortedList\ a$, $NonNegative\ a$

Typical definitions:

$$\begin{aligned}&\mathbf{newtype}\ NonEmptyList\ a = \\ &\qquad NonEmpty\ \{\,getNonEmpty :: [\,a\,]\,\}\\ &\mathbf{newtype}\ NonNegative\ a = \\ &\qquad NonNegative\ \{\,getNonNegative :: a\,\}\end{aligned}$$

Modifiers (2)

Example:

$$
\begin{array}{l}
prop\_Index :: \\
\quad Eq\ a \Rightarrow NonEmptyList\ [a] \rightarrow Property \\
prop\_Index\ (NonEmpty\ xs) = \\
\quad forAll\ (choose\ (0, length\ xs - 1))\ \$\ \lambda i \rightarrow \\
\qquad xs\ !!\ i \equiv head\ (drop\ i\ xs)
\end{array}
$$

Runnnig Tests

Basic function to run tests:

$$quickCheck :: Testable\ prop \Rightarrow prop \to IO\ ()$$

Runnnig Tests

Basic function to run tests:

$$quickCheck :: Testable\ prop \Rightarrow prop \rightarrow IO\ ()$$

Printing of all test cases:

$$verboseCheck :: Testable\ prop \Rightarrow prop \rightarrow IO\ ()$$

Runnnig Tests

Basic function to run tests:

$$quickCheck :: Testable\ prop \Rightarrow prop \rightarrow IO\ ()$$

Printing of all test cases:

$$verboseCheck :: Testable\ prop \Rightarrow prop \rightarrow IO\ ()$$

Controlling e.g. number and size of test cases:

$$quickCheckWith ::$$
$$\quad Testable\ prop \Rightarrow Args \rightarrow prop \rightarrow IO\ ()$$

$$quickCheckWith$$
$$\quad (stdArgs\ \{ maxSize = 10, maxSuccess = 1000 \})$$
$$\quad prop\_XXX$$

Labelling and Coverage (1)

$label$ attaches a label to a test case:

$$label :: Testable\ prop \Rightarrow String \to prop \to Property$$

Example:

$$\begin{array}{l} prop\_RevRev :: [Int] \to Property \\ prop\_RevRev\ xs = \\ \quad label\ (\texttt{"length is "} +\!\!+\ show\ (length\ xs))\ \$ \\ \qquad reverse\ (reverse\ xs) === xs \end{array}$$

Labelling and Coverage (2)

Result:

```
+++ OK, passed 100 tests:
7% length is 7
6% length is 3
5% length is 4
4% length is 6
```

There are also $cover$ and $checkCover$ for checking/enforcingig specific coverage requirements.

A Cautionary Tale (1)

$$
\begin{array}{l}
prop\_Sqrt :: Double \to Bool \\
prop\_Sqrt\ x \\
\quad \begin{array}{ll}
\mid x < 0 & = isNaN\ sqrtX \\
\mid x \equiv 0 \lor x \equiv 1 & = sqrtX \equiv x \\
\mid x < 1 & = sqrtX > x \\
\mid x > 1 & = sqrtX > 0 \land sqrtX < x
\end{array} \\
\quad \textbf{where} \\
\qquad sqrtX = sqrt\ x
\end{array}
$$

$$main = quickCheck\ propSqrt$$

A Cautionary Tale (1)

$$
\begin{array}{l}
\mathit{prop\_Sqrt} :: \mathit{Double} \to \mathit{Bool} \\
\mathit{prop\_Sqrt}\ x \\
\quad \begin{array}{l|ll}
 & x < 0 & = \mathit{isNaN}\ \mathit{sqrtX} \\
 & x \equiv 0 \lor x \equiv 1 & = \mathit{sqrtX} \equiv x \\
 & x < 1 & = \mathit{sqrtX} > x \\
 & x > 1 & = \mathit{sqrtX} > 0 \land \mathit{sqrtX} < x
\end{array} \\
\quad \textbf{where} \\
\qquad \mathit{sqrtX} = \mathit{sqrt}\ x \\
\mathit{main} = \mathit{quickCheck}\ \mathit{propSqrt}
\end{array}
$$

Result: `+++ OK, passed 100 tests`

A Cautionary Tale (2)

$$prop\_Sqrt :: Double \rightarrow Bool$$
$$prop\_Sqrt\ x$$
$$\ldots$$
$$\textbf{where}$$
$$sqrtX = flawedSqrt\ x$$
$$\begin{aligned} flawedSqrt\ x \mid x \equiv 1 &= 0 \\ \mid otherwise &= sqrt\ x \end{aligned}$$
$$main = quickCheck\ propSqrt$$

A Cautionary Tale (2)

$$
\begin{array}{l}
prop\_Sqrt :: Double \rightarrow Bool \\
prop\_Sqrt\ x \\
\quad \quad \ldots \\
\quad \mathbf{where} \\
\quad \quad sqrtX = \mathit{flawedSqrt}\ x \\
\quad \quad \mathit{flawedSqrt}\ x \mid x \equiv 1 \quad \quad = 0 \\
\quad \quad \quad \quad \quad \quad \quad \quad \mid otherwise = sqrt\ x \\
main = quickCheck\ propSqrt
\end{array}
$$

Result: `+++ OK, passed 100 tests`

A Cautionary Tale (2)

$$
\begin{array}{l}
prop\_Sqrt :: Double \rightarrow Bool \\
prop\_Sqrt\ x \\
\quad\quad \ldots \\
\quad \mathbf{where} \\
\quad\quad sqrtX = flawedSqrt\ x \\
\quad\quad flawedSqrt\ x \mid x \equiv 1 \quad\quad = 0 \\
\quad\quad\quad\quad\quad\quad\quad\quad \mid otherwise = sqrt\ x \\
main = quickCheck\ propSqrt
\end{array}
$$

Result: `+++ OK, passed 100 tests`

Errr ...

A Cautionary Tale (3)

$$
\begin{aligned}
&prop\_Sqrt :: Double \rightarrow Bool \\
&prop\_Sqrt\ x \\
&\qquad \ldots \\
&\quad \textbf{where} \\
&\qquad sqrtX = flawedSqrt\ x \\
&\qquad \ldots \\
&main = quickCheckWith \\
&\qquad\qquad (stdArgs\ \{\ maxSuccess = 1000000\ \}) \\
&\qquad\qquad propSqrt
\end{aligned}
$$

A Cautionary Tale (3)

$$
\begin{array}{l}
prop\_Sqrt :: Double \rightarrow Bool \\
prop\_Sqrt\ x \\
\qquad \ldots \\
\quad \mathbf{where} \\
\qquad sqrtX = flawedSqrt\ x \\
\qquad \ldots \\
main = quickCheckWith \\
\qquad\qquad (stdArgs\ \{maxSuccess = 1000000\}) \\
\qquad\qquad propSqrt
\end{array}
$$

Result: `+++ OK, passed 1000000 tests`

A Cautionary Tale (3)

$$
\begin{array}{l}
prop\_Sqrt :: Double \to Bool \\
prop\_Sqrt\ x \\
\qquad \ldots \\
\quad \textbf{where} \\
\qquad sqrtX = flawedSqrt\ x \\
\qquad \ldots \\
main = quickCheckWith \\
\qquad\qquad (stdArgs\ \{ maxSuccess = 1000000 \}) \\
\qquad\qquad propSqrt
\end{array}
$$

Result: `+++ OK, passed 1000000 tests`

Oops.

A Cautionary Tale (4)

Simply test specific cases when needed:

$$prop\_Sqrt0 :: Bool$$
$$prop\_Sqrt0 = mySqrt\ 0 \equiv 0$$

$$prop\_Sqrt1 :: Bool$$
$$prop\_Sqrt1 = mySqrt\ 1 \equiv 1$$

A Cautionary Tale (5)

$$
\begin{aligned}
&prop\_SqrtX :: Double \to Bool\\
&prop\_SqrtX\ x\\
&\quad \mid x < 0 \ = isNaN\ \ sqrtX\\
&\quad \mid x \leqslant 1 = sqrtX \geqslant x\\
&\quad \mid x > 1 \ = sqrtX > 0 \wedge sqrtX < x\\
&\quad \mathbf{where}\\
&\qquad sqrtX = mySqrt\ x
\end{aligned}
$$

A Cautionary Tale (6)

$$
\begin{aligned}
&\mathit{prop\_Sqrt} :: \mathit{Property}\\
&\mathit{prop\_Sqrt} = \mathit{counterexample}\\
&\qquad\qquad\qquad \texttt{"sqrt 0 failed"}\\
&\qquad\qquad\qquad \mathit{prop\_Sqrt0}\\
&\qquad .\&\&. \quad\;\; \mathit{counterexample}\\
&\qquad\qquad\qquad \texttt{"sqrt 1 failed"}\\
&\qquad\qquad\qquad \mathit{prop\_Sqrt1}\\
&\qquad .\&\&. \quad\;\; \mathit{prop\_SqrtX}
\end{aligned}
$$

Testing Interval Arithmetic (1)

Lifting a unary operator $\ominus$ to an operator $\hat{\ominus}$ working on intervals is defined as follows, assuming $\ominus$ is defined on the entire interval:

$$\hat{\ominus} i = [\min_{\forall x \in i} \ominus x,\ \max_{\forall x \in i} \ominus x]$$

Testing Interval Arithmetic (1)

Lifting a unary operator $\ominus$ to an operator $\hat{\ominus}$ working on intervals is defined as follows, assuming $\ominus$ is defined on the entire interval:

$$\hat{\ominus} i = [\min_{\forall x \in i} \ominus x, \ \max_{\forall x \in i} \ominus x]$$

And for binary operators:

$$i_1 \mathbin{\hat{\otimes}} i_2 = [\min_{\forall x \in i_1, y \in i_2} x \otimes y, \ \max_{\forall x \in i_1, y \in i_2} x \otimes y]$$

Testing Interval Arithmetic (2)

But how can we test that? In general, very difficult to find the global minimum/maximum of a function over an interval without further information e.g. about its derivatives.

Testing Interval Arithmetic (2)

But how can we test that? In general, very difficult to find the global minimum/maximum of a function over an interval without further information e.g. about its derivatives.

However, for a given interval i, it follows that:

$$\forall x \in i.\, \ominus\, x \in \hat{\ominus} i$$

Testing Interval Arithmetic (3)

Unfortunately, $\hat{\ominus} i = [-\infty, +\infty]$ satisfies

$$\forall x \in i.\ \ominus\, x \in \hat{\ominus} i$$

Testing Interval Arithmetic (3)

Unfortunately, $\hat{\ominus} i = [-\infty,\ +\infty]$ satisfies

$$\forall x \in i.\ \ominus\ x \in \hat{\ominus} i$$

We should ideally test that the result interval is not larger than necessary. But that is hard too.

Testing Interval Arithmetic (3)

Unfortunately, $\hat{\ominus} i = [-\infty,\ +\infty]$ satisfies

$$\forall x \in i.\ \ominus x \in \hat{\ominus} i$$

We should ideally test that the result interval is not larger than necessary. But that is hard too.

However, the definition does imply that a 1-point interval must be mapped to a 1-point interval:

$$\hat{\ominus}[x, x] = [\ominus x,\ \ominus x]$$

While not perfect, does rule out trivial implementations and it is easy to test.

Testing Interval Arithmetic (4)

For binary operators:

- For given intervals i_1 and i_2:

$$\forall x \in i_1, y \in i_2.\ x \otimes y \in i_1 \hat{\otimes} i_2$$

- For given x and y:

$$[x, x] \mathbin{\hat{\otimes}} [y, y] = [x \otimes y,\ x \otimes y]$$

Let us turn the above into QuickCheck test cases interactively.