SOFTWARE FOUNDATIONS

VOLUME 4: QUICKCHICK: PROPERTY-BASED TESTING IN COQ

TABLE OF CONTENTS INDEX

INTRODUCTION

A First Taste of Testing

Consider the following definition of a function `remove`, which takes a natural number `x` and a list of nats `l` and removes `x` from the list.

```
Fixpoint remove (x : nat) (l : list nat) : list nat :=
  match l with
    | [] ⇒ []
    | h::t ⇒ if beq_nat h x then t else h :: remove x t
  end.
```

One possible specification for `remove` might be this property...

```
Conjecture removeP : ∀ x l, ¬ (In x (remove x l)).
```

...which says that `x` never occurs in the result of `remove x l` for any `x` and `l`. (`Conjecture foo...` means the same as `Theorem foo... Admitted`. Formally, `foo` is treated as an axiom.)

Sadly, this property is false, as we would (eventually) discover if we were to try to prove it.

A different — perhaps much more efficient — way to discover the discrepancy between the definition and specification is to *test* it:

```
(* QuickChick removeP. *)
```

(Try uncommenting and evaluating the previous line.)

The `QuickChick` command takes an "executable" property (we'll see later exactly what this means) and attempts to falsify it by running it on many randomly generated inputs, resulting in output like this:

```
0
[0, 0]
```

```
Failed! After 17 tests and 12 shrinks
```

This means that, if we run `remove` with `x` being `0` and `l` being the two-element list containing two zeros, then the property `removeP` fails.

With this example in hand, we can see that the `then` branch of `remove` fails to make a recursive call, which means that only one occurence of `x` will be deleted. The last line of the output records that it took 17 tests to identify some fault-inducing input and 12 "shrinks" to reduce it to a minimal counterexample.

Exercise: 1 star (insertP)

Here is a somewhat mangled definition of a function for inserting a new element into a sorted list of numbers:

```
Fixpoint insert x l :=
  match l with
  | [] ⇒ [x]
  | y::t ⇒ if y <? x then insert x t else y::t
  end.
```

Write a property that says "inserting a number `x` into a list `l` always yields a list containing `x`." Make sure QuickChick finds a counterexample.

```
(* FILL IN HERE *)
```

□

Exercise: 2 stars (insertP2)

Translate the following claim into a `Conjecture` (using `In` for list membership): "For all numbers `x` and `y` and lists `l`, if `y` is in `l` then it is also in the list that results from inserting `x` into `l`" (i.e., `insert` preserves all the elements already in `l`). Make sure QuickChick finds a counterexample.

```
(* FILL IN HERE *)
```

□

Overview

Property-based random testing involves four basic ingredients:

- an *executable property* like `removeP`,
- *generators* for random elements of the types of the inputs to the property (here, numbers and lists of numbers),
- *printers* for converting data structures like numbers and lists to strings when reporting counterexamples, and
- *shrinkers*, which are used to minimize counterexamples.

We will delve into each of these in detail later on, but first we need to make a digression to explain Coq's support for *typeclasses*, which QuickChick uses extensively

both internally and in its programmatic interface to users. This is the Typeclasses chapter.

In the QC chapter we'll cover the core concepts and features of QuickChick itself.

The TImp chapter develops a small case study around a typed variant of the Imp language.

The QuickChickTool chapter presents a command line tool, *quickChick*, that supports larger-scale projects and mutation testing.

The QuickChickInterface chapter is a complete reference manual for QuickChick.

Finally, the Postscript chapter gives some suggestions for further reading.