QuickChick Quick Start

Will Thomas

February 7, 2023

1 Introduction

I want to start off by giving an idea on how to install QuickChick. The best way I have found that minimizes the installation and maintenance is using *opam*. More info can be found at QuickChick Github.

Rather than providing explicit proofs for theorems in Coq, we can test them using QuickChick and then assume that they hold if QuickChick cannot disprove them. This functionality is offered in the form of "Conjectures": A **Conjecture** is the QuickChick equivalent of a normal Coq axiom.

```
Conjecture <NAME> : <PROP>.
```

This is essentially syntactic sugar for:

```
Theorem <NAME> : <PROP>. Admitted.
```

We would then use QuickChick to test our Conjecture via the following command (in Coq):

```
QuickChick <NAME>.
```

There are 4 main ingredients to Property-Based Testing:

1. Executable Properties: We need our properties to be actually testable/executable
2. Generators: We want well defined generators to create random tests
3. Printers: For showing failing examples
4. Shrinkers: To help find the smallest failing examples for easier debugging

2 TypeClasses

TypeClasses are one of the coolest features I have learned about in Coq. They are roughly equivalent to interfaces where we can know certain things about a type based on its implementing a TypeClass. This will be a brief introduction to TypeClasses as they have so many features they cannot all be covered here.

There are 2 fundamental parts to TypeClasses:

1. **Class Definitions**: Used to create an interface that Types must instantiate

```
Class <CLASS_NAME> (...ARGS) : <RETURN_TYPE> :=
{
    <FIELD_NAME> : <TYPE_SIGNATURE> ;
}.
```

- Typically the RETURN_TYPE can be omitted (it will typically be inferred properly by Coq)
- You can include any amount of fields in the class definitions

2. **Instantiations**: The actual instantiaion of a Class with arguments and obligations fulfilled

```
<LOCALITY> Instance <INSTANCE_NAME> : <CLASS_NAME> (...ARGS) :=
{
    <FIELD_NAME> : <DEFINITION> ;
}.
```

- The DEFINITION must fulfill the TYPE_SIGNATURE of the corresponding field
- Every field must be instantiation (you cannot leave any fields out)
- It is sometimes helpful to create instances in the *prover* to leverage automation

Example 2.1 The "Show" TypeClass

```
Class Show A : Type :=
{
    show    : A -> string
}.

Local Instance showBool : Show bool :=
{
    show    := fun b : bool => if b then "true" else "false"
}.
```

or equivalently

```
Local Instance showBool' : Show bool.
constructor; intros b; destruct b eqn:B.
- (* b = true *) apply "true".
- (* b = false *) apply "false".
Defined.
```

We can then use the "show" function on any type where the *TypeClass Show* has a valid *Instantiation* in the Environment

Let us assume for a moment we have instantiated Show for Nats, bools, and strings (and assume all the obvious implementations for the corresponding "show" functions). We can then make the following calls

```
Compute (show 21) (* "21" *)
Compute (show true) (* "true" *)
Compute (show "hello") (* "hello" *)
```

3 QuickChick

4 Case Study

5 Summary and Use