Exercise (Wednesday) – Writing and Tuning Generators

Load the file **`Coin.hs`** into ghci. It defines a **`Coin`** type with a maximum value, inspired by a cryptocurrency application, along with a function to check validity of coins and to add them together (provided the result is valid).

```
newtype Coin = …
maxCoinValue :: Int
valid        :: Coin -> Bool
add          :: Coin -> Coin -> Maybe Coin
```

There is a generator and shrinker for the **`Coin`** type provided in the file, together with properties to check that all generated coins are valid, and that **`add`** works as it should:

```
prop_Valid :: Coin -> Bool
prop_Add :: Coin -> Coin -> Property
```

1. Begin by testing **`prop_Valid`**. You will find that it fails, because the given generator can generate invalid **`Coin`**s. Correct the generator and ensure that **`prop_Valid`** passes a large number of tests. You may find the function **`abs`** useful[1].
2. Now test **`prop_Add`**. You should find that it passes. But are the tests effective? Instrument the property using **`cover`**[2] (from the lecture) to measure the proportion of test cases that cover each branch of the **if-then-else**. (You will need to decide what proportion of test cases—at a minimum—should cover each branch. Perhaps 40%?). Rerun the tests to measure the coverage achieved. Use **`checkCoverage`** to see whether your coverage requirements are met:
   ```
   quickCheck . checkCoverage $ prop_Add
   ```
3. Assuming that they are not, adjust the generator so that the requirements are met. *Hint:* my solution uses **`choose`**[3]**.**
4. Think about the unit tests you would write for add. Wouldn't you include some *boundary cases* where the sum falls close to the maximum possible value, on either side? Let's call a test a 'boundary case' if the sum is within 3 of the boundary. Instrument the property (using **`cover`** again) to report the proportion of boundary cases tested. Only a minority of tests can be expected to be boundary cases; choose a coverage requirement that ensures that most test runs will contain a boundary case.
5. Is your new coverage requirement met? If not, refine the generator again so that it is. *Hint:* my solution uses **`NonNegative`**[4]**,** **`choose`** and **`oneof`**[5].

[1] https://hackage.haskell.org/package/base-4.18.0.0/docs/Prelude.html#v:abs
[2] https://hackage.haskell.org/package/QuickCheck-2.14.3/docs/Test-QuickCheck.html#v:cover
[3] https://hackage.haskell.org/package/QuickCheck-2.14.3/docs/Test-QuickCheck.html#v:choose
[4] https://hackage.haskell.org/package/QuickCheck-2.14.3/docs/Test-QuickCheck.html#t:NonNegative
[5] https://hackage.haskell.org/package/QuickCheck-2.14.3/docs/Test-QuickCheck.html#v:oneof