StrPack.jl Documentation

Release development

Patrick O'Leary

February 05, 2014

This package is a Julia implementation of QuickCheck by Koen Claessen and John Hughes. QuickCheck is a technique for testing software by specifying properties of a function, then randomly searching for inputs for which the property fails to hold.

CHAPTER 1

Example

Let's look at a property of `zip`. Zipping two iterables should produce a new iterable whose length is the same as the length of the shorter of the two iterables:

```
julia> using QuickCheck

julia> property((x::Vector{Int}, y::Vector{Int}) -> length(zip(x,y)) == min(length(x), length(y)))
OK, passed 100 tests.
```

QuickCheck supplied 100 random pairs of arguments to this function, all of which met the specification. What happens when the function does not meet the specification?:

```
julia> property((x::Vector{Int}, y::Vector{Int}) -> length(zip(x,y)) == max(length(x), length(y)))
ERROR: Falsifiable, after 1 tests:
{[2, 1], [2]}
```

QuickCheck not only points out that the specification is not met, but provides the test inputs that failed.

CHAPTER 2

Methods

property (*prop[, typs][, ntests]*)

Test the property `prop`, with argument types `typs`, running `ntests` tests. If `prop` is an anonymous function with typed arguments, and those types are defined in Base, the types can be inferred, otherwise, `typs` should be a `Vector{Type}`. By default, QuickCheck will run 100 tests.

condproperty (*prop*[, *typs*], *ntests*, *maxtests*, *argconds...*)

Test the property `prop`, with argument types `typs`, attempting to run `ntests` tests (but no more than `maxtests` tests), with arguments subject to the predicates `argconds...`. This can be used to discard test cases where the property is not expected to hold. For instance, the following test will have random failures due to division by zero:

```
julia> property((x::Int, y::Int)->(x/y)>=floor(x/y))
OK, passed 100 tests.

julia> property((x::Int, y::Int)->(x/y)>=floor(x/y))
ERROR: Falsifiable, after 1 tests:
{0, 0}
```

That probem can be solved by adding a condition that requires `y` to be nonzero:

```
julia> condproperty((x::Int, y::Int)->(x/y)>=floor(x/y), 100, 1000, (x,y) -> y!=0)
OK, passed 100 tests.
```

quantproperty (*prop*[, *typs*], *ntests*, *arggens...*)

Test the property `prop`, with argument tpyes `typs`, running `ntests` tests, and using the argument generators arggens. Each argument generator must be a function in a single positive integer which returns an item of the appropriate type which has "size" of the integer. This integer is used to scale the tests. This is a more efficient and flexible approach than conditional properties, but is sometimes harder to express:

```
julia> quantproperty((x::Int, y::Int)->(x/y)>=floor(x/y), 100, (size)->QuickCheck.generator(Int,
OK, passed 100 tests.
```

CHAPTER 3

Generators

QuickCheck specifies properties for most of Julia's built in numeric types and strings, as well as arrays and composite types that are composed of those types. However, you might need to extend the specifications to new types for which a generator does not exist and cannot be automatically constructed, or for which the default generator might provide poor coverage. To do this, `import QuickCheck.generator` and extend the `generator` method, which takes a type and a "size" argument (a positive integer) and should return a new instance of that type scaled appropriately by the size. As a contrived example, consider the type:

```
type A
    b::Int
end
```

We can create a custom generator as follows:

```
import QuickCheck.generator
generator(::Type{A}, size) = A(rand(10000:(10000*size)))
```

Then we can test a property using the new type:

```
julia> property((x)->x.b > sqrt(x.b), [A])
OK, passed 100 tests.
```