Clojure/conj 2015 test.check Workshop

Gary Fredericks

November 15, 2015

Topics

- Property-based testing
    - an approach to testing software
- `test.check`
    - a clojure library for writing property-based tests

Me

- Gary Fredericks (`@gfredericks_`)
- Using Clojure professionally for ~6 years
  - Currently at Groupon
- User/Maintainer of `test.check`

Property-Based Testing

- Main idea
  - Describe possible "inputs" to your program
  - Describe properties that hold for all inputs
  - Testing framework generates random inputs, checks the properties
- Benefits
  - Get much better test coverage than example-based tests, especially when testing combinations of features & edge-cases
  - Find bugs in tools/libraries
- History
  - Originated in Haskell-land as "QuickCheck"

Skepticism

```clojure
(def gen-record
  ...)

(defspec db-roundtrip 1000
  (prop/for-all [record gen-record]
    (do
      (save-to-db record)
      (= record (read-from-db (:id record))))))
```

Bugs I've Found

- Postgres serializable transactions (`BUG #13667`)
  - Didn't work correctly for some write-before-read uses
- Google Closure integers
  - Base 36 serialization
  - Dividing big-integers larger than `Double/MAX_VALUE`
- Carmine message queues
  - Inadvertently required two connections per worker, allowing deadlock while trying to acquire second connection

`test.check`

- A QuickCheck-inspired testing library in Clojure
- Created by Reid Draper (originally called "simple-check")
- Integrates with `clojure.test`
- Supports ClojureScript

Goals for this Workshop

- How do I use this thing?
  - Writing generators
  - Writing properties
- What tests should I write?

Schedule

- I talk about generators
- You write some generators
- I talk about properties & shrinking
- You write some properties
- I talk about property-based testing in real life
- You go back to your job and do real-life property-based testing

Gentlefolks, start your repls

Visit & clone `http://bit.ly/1MvJH3D`

OR

```
git clone \
https://github.com/gfredericks/test-check-workshop.git
```

AND THEN

```
cd test-check-workshop/part-2
lein repl
```

Generators

Generators In Action

```clojure
(def gen-tweet
  (gen/hash-map :id      gen/large-integer
                :user-id gen/large-integer
                :text    gen/string
                :at      gen/large-integer))

(defspec tweet-roundtrip 100
  (prop/for-all [tweet gen-tweet]
    (do
      (write-tweet-to-db tweet)
      (= tweet (read-tweet-from-db (:id tweet))))))
```

Getting Around

```
(require '[clojure.test.check.generators :as gen])

(gen/generator? gen/nat)
=> true

(gen/sample gen/nat)
=> (0 1 2 2 2 4 5 4 7 6)

(->> (gen/sample gen/nat 100) (reverse) (take 5))
=> (65 56 19 57 13)

(gen/generate gen/nat)
=> 5
```

Some Distinctions

```
(gen/generate gen/string-ascii) => "1|b$lcThVf"

(gen/generate (gen/vector gen/large-integer))
=> [-95 -491 63140429 -3431 0 -311 -5
    -2076 -45628915 -276774]

(def gen-vector-of-integers
  (gen/vector gen/large-integer))

(gen/generate gen-vector-of-integers)
=> [180584 -2 -22045 54270866 -437576 -11216]
```

Generating Simple Values

```
(gen/generate (gen/return 42)) => 42
(gen/generate gen/boolean) => true
(gen/generate gen/uuid)
=> #uuid "bd7b1d9f-7d10-46f0-bf60-e7655bc30013"
(gen/generate (gen/elements [:foo :bar :baz]))
=> :baz
(gen/generate (gen/shuffle [:a :b :c :d]))
=> [:c :a :b :d]
(gen/generate gen/any-printable 5)
=> [3/4 4 "" V2m!j-]
(gen/generate gen/simple-type-printable)
=> 19/4
```

Generating Simple Values — Numbers & Strings

```
(gen/generate gen/nat) => 8
(gen/generate gen/large-integer) => 5908552
(gen/generate gen/double) => -0.05810546875
(gen/generate gen/ratio) => 25/11
(gen/generate gen/byte) => 1

(gen/generate gen/string-ascii) => "L}bOK\"dHF["
(gen/generate gen/string-alphanumeric)
=> "OXXVKNr6T9ZmI5"
(gen/generate gen/keyword)
=> :H?2I!p7+:+1w_*-g-26P+BB!N-:_P!k89327V0y5L5?v6
(gen/generate gen/symbol) => s*
```

Generating Collections

```
(gen/generate (gen/vector gen/ratio))
=> [1/2 8/11 -8/27 -1/5 -25/26 -4/3
    -23/16 -26/11 1/5 1/3 -9/22 17/23
    -2/3 17/6 -10/9 -25/27 2/11]

(gen/generate (gen/map gen/string-alphanumeric
                       gen/large-integer))
=> {"QlfwE28o59osPwD3FYjAAkX7UGoI5n" -3526839,
    "KGYJh06cKIOlMh0" -640463,
    "VYzJ" 2,
    "0" -191059464,
    "OvBETs59ge2Clz29pQj63LZM8fqX5" -6,
    "wr5dAjY23P4qlavDV957UPf4OPmEjh" -12788626,
    "14O211UNiiNE0wg9HG75" -128}
```

Generating Collections — Distinct Elements

```
(gen/generate
 (gen/vector-distinct-by #(last (str %))
                         gen/large-integer
                         {:num-elements 10
                          :max-tries 1000}))
[921893
 -4163099
 -9127473268377199 5
 -2
 -13413836
 5060233600
 -197
 -19709728
 13863511
 44704]
```

Generating Collections — Heterogeneous

```
(gen/generate
 (gen/hash-map :user-id   gen/large-integer
               :parent-id gen/large-integer
               :text      gen/string-ascii))

=> {:user-id 4441
    :parent-id 1155
    :text "R1f^DTs!?-ST0;9q1I-.]0/#L}z"}

(gen/generate
 (gen/tuple gen/boolean gen/double gen/string-ascii))

=> [false -29.0625 ">\""]
```

Combinators — gen/let

```
(gen/sample
 (gen/let [x gen/nat]
   [x (inc x)]))

=> ([0 1] [1 2] [1 2] [1 2] [0 1]
    [0 1] [2 3] [1 2] [8 9] [2 3])

(gen/generate
 (gen/let [s gen/string-ascii
           bounds (gen/vector (gen/large-integer
                                {:min 0, :max (count s)})
                               2)]
   (let [[start end] (sort bounds)]
     [s (subs s start end)])))

=> ["HgSz!u1>nkhyxL|,Q:+/zms=#]2" "L|,Q:+/z"]
```

Combinators — gen/one-of and gen/such-that

```
(gen/sample (gen/one-of [gen/nat
                         gen/boolean
                         (gen/return nil)]))

=> (true nil 2 false 1 true nil true true 4)

(gen/sample
 (gen/such-that #(not= 1 (count %))
                (gen/list gen/nat)))

=> (() () () (2 3 2) (0 3) (1 2 2 4)
    (5 4 1 1 6 5) (1 1) (7 1) (4 6 9 7 5 3 8 2))
```

Generators

Exercises

Exercises – 4clojure

- `http://4clojure.gfredericks.com`
- cheatsheet: `https://github.com/clojure/test.check`

Testing

Testing

Properties, `defspec`

Anatomy of a spec

```clojure
(require
 '[clojure.test.check.clojure-test :refer [defspec]]
 '[clojure.test.check.generators :as gen]
 '[clojure.test.check.properties :as prop])

(defspec numbers-work-pretty-good 1000
  (prop/for-all [x gen/nat
                 y gen/nat]
    (= (+ x y)
       (+ y x))))
```

Running a spec

```
;; clojure.test will run these as normal
;; tests, but they can also be called as
;; functions

(numbers-work-pretty-good)
=> {:num-tests 1000
    :result true
    :seed 1446498477711}
```

Failing tests

```
(defspec numbers-work-pretty-good 1000
  (prop/for-all [x gen/nat
                 y gen/nat]
    (= (- x y)
       (- y x))))

(numbers-work-pretty-good)
=> {:fail [0 3],
    :failing-size 3,
    :num-tests 4,
    :result false,
    :seed 1446498603099,
    :shrunk {:depth 2, :result false,
             :smallest [0 1], :total-nodes-visited 5}}
```

Testing

Shrinking

A Failing Test

```clojure
(def gen-user
  (gen/hash-map :name gen/string-alphanumeric
                :age gen/nat
                :comments (gen/vector
                           gen/string-ascii)))

(defspec users-can't-be-42-years-old
  (prop/for-all [user gen-user]
    (not= 42 (:age user))))
```

Can users be 42 years old?

```
(users-can't-be-42-years-old)
=>
{:fail [{:age 42,
         :comments ["f~Bz;cyd{IYT']][u^g3Zb]bqp^20x'yXbQ+"
                    "cx}:ZiX<hdQ;Dl(tL?>mG#f(K8rkuw`"
                    "M^$"
                    ;; ... 29 more strings omitted ...
                    ],
         :name "Q5JD69vn5ebT8I5Y4PLtS8hw"}],
 :failing-size 42,
 :num-tests 43,
 :result false,
 :seed 1446500137527,
 :shrunk {:depth 56,
          :result false,
          :smallest [{:age 42, :comments [], :name ""}],
          :total-nodes-visited 254}}
```

A More Specific Failing Test

```
(defspec users-with-at-least-two-comments-can't-be-42-years
  (prop/for-all [user gen-user]
    (not (and (= 42 (:age user))
              (<= 2 (count (:comments user)))))))
```

Can users with at least two comments be 42 years old?

```
(users-with-at-least-two-comments-can’t-be-42-years-old)

=>
{:fail [{:age 42,
         :comments ["CE(*bQ>G\\RHwa]t_b_OR3wJi\"9GD_aPOC"
                    "#CSGaoB!{56zzc2{-o\";3Z"
                    ;; ... 38 more strings omitted ...
                    ],
         :name "aRwWqcRV36N97Qy9e8"}],
 :failing-size 60,
 :num-tests 61,
 :result false,
 :seed 1446500367916,
 :shrunk {:depth 90,
          :result false,
          :smallest [{:age 42, :name ""
                      :comments ["" ""]}],
          :total-nodes-visited 598}}
```

The Process of Shrinking — 1

```
(defspec twenty-four-is-the-highest-number
  (prop/for-all [xs (gen/vector gen/int)]
    (every? #(<= % 24) xs)))
```

The Process of Shrinking — 2

```
;; Testing [-18 10 20 12 25 -23]...fail
;; Testing [10 20 12 25 -23]...fail
;; Testing [-18 20 12 25 -23]...fail
;; Testing [20 12 25 -23]...fail
;; Testing [10 12 25 -23]...fail
;; Testing [12 25 -23]...fail
;; Testing [20 25 -23]...fail
;; Testing [25 -23]...fail
;; Testing   [12 -23]...pass
;; Testing   [-23]...pass
;; Testing [25]...fail
;; Testing   [0 -23]...pass
;; Testing   []...pass
;; Testing   [0]...pass
;; Testing   [13]...pass
;; Testing   [19]...pass
;; Testing   [22]...pass
;; Testing   [24]...pass
;; {:fail [[-18 10 20 12 25 -23]], ...
;;  :shrunk {..., :smallest [[25]], :total-nodes-visited 12}}
```

Shrinking Pitfalls — 1

```clojure
(def gen-vector-of-ints
  (gen/let [length gen/nat]
    (gen/vector gen/nat length)))

(defspec lists-don’t-contain-42
  (prop/for-all [xs gen-vector-of-ints]
    (not-any? #{42} xs)))
```

Shrinking Pitfalls — 2

```
(lists-don't-contain-42)

=> {:fail [[29 6 9 33 32 3 40 30 23 42 41 41
            38 30 26 9 27 8 28 3 18 12 3 43 6 6]],
    :failing-size 43,
    :num-tests 44,
    :result false,
    :seed 1447603220629,
    :shrunk {:depth 13,
             :result false,
             :smallest [[0 0 0 0 0 0 0 0 0 42 0]],
             :total-nodes-visited 36}}
```

Testing

What sort of properties should I write?

Easy Wins

- Roundtripping
  - Storage
  - Serialization
- Doing things in different orders (when order isn't supposed to matter): `(= (f (g x)) (g (f x)))`
- Idempotency: `(= (f x) (f (f x)))`
- Run your code to make sure it doesn't crash
  - Especially meaningful if you have runtime assertions
- For some programs verifying that the output is correct is much easier than computing it
- Test complex optimized code using simple unoptimized code (test that they do the same thing)
- When rewriting something, test that the old version and the new version do the same thing

Testing

Exercises

Exercises – Testing some codebases

- Write some tests with some of the given codebases

Applied

Tradeoffs

Property-based testing in general

PRO	CON
More test coverage	Test-writing is slower
Expose assumptions	Tests can be harder to understand
Design to avoid edge-cases	Test-running is slower
Discover bugs in tools/libs	Doesn't work well at very large scales

Additional temporary drawbacks for `test.check` in particular

- Missing more advanced generators
    - time, unicode strings, ...
- Missing some usability features
    - Re-running failures is difficult
    - Can't customize shrinking process
    - No mechanism for storing regression examples
    - No "fast mode" for a quick run of a whole test suite

How do I use this to test my hairy business application?

Testing Difficult Things

- Write Libraries
- Model the External World
    - users, other systems, the clock
- Model Your Application
- Use Schemas & Other Assertions

Applied

Exercises

Exercises

Go home and write property-based tests.

Resources

- test.chuck: github.com/gfredericks/test.chuck
- stateful-check: github.com/czan/stateful-check
- `freenode#test.check`