QUICK! Check Your Properties

(Random Testing w/ #fsharp & #fscheck)

twitter.com/pblasucci
#fsharpx

github.com/pblasucci/quickpbt

RANDOM TESTING

"Properties are described as … functions, and can be automatically tested on random input… [or] custom test data generators."

from ICFP'00 – Claessen, Hughes

FROM UNIT TESTING…

```
[<Fact>]
let PlusIgnoresTime () =
  let days  = time.FromDays(7)
  let hours = time.FromHours(7 * 24)
  let civil = date.Now
  Assert.Equal(civil + days, civil + hours)
```

TEST EXECUTION SUMMARY

Tests run: 1, Errors: 0, Failed: 0, Ignored: 0

✔ **PlusIgnoresTime**
OK, Elapsed time: 0.0527666s

TO PROPERTY TESTING!

```
[<Property>]
let ``plus ignores time`` (civil:date) =
  let days  = time.FromDays(7)
  let hours = time.FromHours(7 * 24)

  civil + days = civil + hours
```

TEST EXECUTION SUMMARY

Tests run: 1, Errors: 0, Failed: 0, Ignored: 0

✔ **plus ignores time**
OK, Passed 100 tests

PATTERNS: *Inversion & Idempotence*

```
(* inversion … one action “undoes” the work of another action *)
let ``adding and subtracting are inverses`` (civil :date) (PositiveInt total) =
  let days = time.FromDays(total)
  (civil + days) - days = civil

(* idempotence … an action has a singular effect despite being invoked repeatedly *)
let ``taking a time duration is idempotent`` (value :time) =
  let once = value.Duration()
  let once = value.Duration().Duration()
  once = twice
```

PATTERNS: *Interchange & Invariance*

```
(* interchange … the order of two or more actions does not alter the outcome *)
let ``adding & changing zone can be reordered`` (civil :date) (PositiveInt total) =
  let days = time.FromDays(total)
  let addThenShift = zone.ConvertTimeBySystemTimeZoneId(civil + days, “Pacific Standard Time”)
  let shiftThenAdd = zone.ConvertTimeBySystemTimeZoneId(civil, “Pacific Standard Time”) + days
  addThenShift = shiftThenAdd

(* invariance … someting remains constant, despite actions being taken *)
let ``adding does not change date offset`` (civil :date) (PositiveInt months) =
  let offset = civil.Offset
  civil.AddMonths(months) = offset
```

DIAGNOSTICS: *Labelling Properties*

```
let ``conversion ignores detours`` (civil :date) (zone1 :zone) (zone2 :zone) =
  let viaZone1 = zone.ConvertTime(zone.ConvertTime(civil, zone1), zone2)
  let directly = zone.ConvertTime(civil, zone2)
  (viaZone1 = directly) && (directly.Offset = zone2.BaseUtcOffset)
```

TEST EXECUTION SUMMARY

Tests run: 1, Errors: 0, Failed: 1, Ignored: 0

✘ **conversion ignores detours**
Falsifiable, after 4 tests (5 shrinks), (StdGen (199662269,296213481)):

Original: (1948-04-19 16:18:52 +04:59, (UTC+04:00), (UTC-05:00))
Shrunk: (1948-04-19 00:00:00 +00:00, (UTC+04:00), (UTC-05:00))

DIAGNOSTIC: *Labelling Properties*

```
let ``conversion ignores detours`` (civil :date) (zone1 :zone) (zone2 :zone) =
  let viaZone1 = zone.ConvertTime(zone.ConvertTime(civil, zone1), zone2)
  let directly = zone.ConvertTime(civil, zone2)
  (viaZone1 = directly) |@ sprintf "Not the same date!"
  .&.
  (directly.Offset = zone2.BaseUtcOffset) |@ sprintf "Not the same zone!"
```

TEST EXECUTION SUMMARY

Tests run: 1, Errors: 0, Failed: 1, Ignored: 0

✘ **conversion ignores detours**
Falsifiable, after 4 tests (5 shrinks), (StdGen (199662269,296213481)):

Label of failing property: Not the same zone!
Original: (1948-04-19 16:18:52 +04:59, (UTC+04:00), (UTC-05:00))
Shrunk: (1948-04-19 00:00:00 +00:00, (UTC+04:00), (UTC-05:00))

DIAGNOSTIC: *Labelling Properties*

```
let ``conversion ignores detours`` (civil :date) (zone1 :zone) (zone2 :zone) =
  let viaZone1 = zone.ConvertTime(zone.ConvertTime(civil, zone1), zone2)
  let directly = zone.ConvertTime(civil, zone2)
  (viaZone1 = directly) |@ sprintf “Not the same date!“
  .&.
  (directly.Offset = zone2.GetUtcOffset(directly)) |@ sprintf “Not the same zone!"
```

TEST EXECUTION SUMMARY

Tests run: 1, Errors: 0, Failed: 0, Ignored: 0

✔ **conversion ignores detours**
OK, Passed 100 tests

DIAGNOSTICS:

Gathering Observations

TEST EXECUTION SUMMARY

Tests run: 1, Errors: 0, Failed: 0, Ignored: 0

✔ **timezone supports DST**
OK, Passed 100 tests

53% Odd
47% Even

```
(* a trival observation puts data into two buckets *)
let ``timezone supports DST`` (civil :date)
                              (target :zone)
                              (NonNegativeInt total) =
  let days = time.FromDays(total)
  let addShift = zone.ConvertTime(civil + days, target)
  let shiftAdd = zone.ConvertTime(civil, target) + days

  addShift = shiftAdd
  |> Prop.trivial target.SupportsDaylightSavingsTime
```

DIAGNOSTICS:

Gathering Observations

TEST EXECUTION SUMMARY

Tests run: 1, Errors: 0, Failed: 0, Ignored: 0

✔ **relation to Greenwich**
OK, Passed 100 tests

55% GMT >>
43% << GMT
2% == GMT

```
(* a classification splits data into N named buckets *)
let ``relation to Greenwich`` (civil :date)
                              (target :zone)
                              (NonNegativeInt total) =
  let days = time.FromDays(total)
  let addShift = zone.ConvertTime(civil + days, target)
  let shiftAdd = zone.ConvertTime(civil, target) + days

  addShift = shiftAdd
  |> Prop.classify (civil.Offset < time.Zero) "<< GMT"
  |> Prop.classify (civil.Offset = time.Zero) "== GMT"
  |> Prop.classify (civil.Offset > time.Zero) "GMT >>"
```

DIAGNOSTICS:

Gathering Observations

TEST EXECUTION SUMMARY

Tests run: 1, Errors: 0, Failed: 0, Ignored: 0

✔ **day of the week**
OK, Passed 100 tests

20% Monday
19% Saturday
17% Sunday
14% Tuesday
13% Thursday
9% Friday
8% Wednesday

```
(* not just booleans… collect reports any value *)
let ``day of the week`` (civil :date)
                        (target :zone)
                        (NonNegativeInt total) =
  let days = time.FromDays(total)
  let addShift = zone.ConvertTime(civil + days, target)
  let shiftAdd = zone.ConvertTime(civil, target) + days

  addShift = shiftAdd
  |> Prop.collect (weekdayName civil)
```

DIAGNOSTICS:

Gathering Observations

TEST EXECUTION SUMMARY

Tests run: 1, Errors: 0, Failed: 0, Ignored: 0

✔ **combined observations**
OK, Passed 100 tests

8% Saturday, GMT >>, trivial
8% Monday, << GMT
7% Sunday, GMT >>, trivial
5% Friday, << GMT, trivial
5% Wednesday, GMT >>
5% Tuesday, << GMT
2% Thursday, << GMT
1% Monday, == GMT, trivial
…

```
(* observations may be combined as mush as is desired *)
let ``combined observations`` (civil :date)
                              (target :zone)
                              (NonNegativeInt total) =
  let days = time.FromDays(total)
  let addShift = zone.ConvertTime (civil + days, target)
  let shiftAdd = zone.ConvertTime (civil, target) + days

  addShift = shiftAdd
  |> Prop.trivial  target.SupportsDaylightSavingsTime
  |> Prop.classify (civil.Offset < time.Zero) "<< GMT"
  |> Prop.classify (civil.Offset = time.Zero) "= GMT ="
  |> Prop.classify (civil.Offset > time.Zero) "GMT >>"
  |> Prop.collect  (weekdayName civil)
```

INPUT CONTROL: *Conditional Properties*

```
let ``modern daylight savings test oracle (näive)`` (civil :date) =
  let eastern = zone.FindSystemTimeZoneById("Eastern Standard Time")
  let etDate  = zone.ConvertTime(civil, eastern)

  Zone.inUnitedStatesDaylightTime etDate = eastern.IsDaylightSavingTime etDate
```

TEST EXECUTION SUMMARY

Tests run: 1, Errors: 0, Failed: 1, Ignored: 0

✘ **modern daylight savings test oracle (näive)**
Falsifiable, after 1 test (4 shrinks), (StdGen (2119435949,296213433))

Original: 1908-04-23 23:48:57 -04:02
Shrunk: 1908-04-23 00:00:00 +00:00

INPUT CONTROL: *Conditional Properties*

```
let ``modern daylight savings test oracle`` (civil :date) =
  let eastern = zone.FindSystemTimeZoneById("Eastern Standard Time")
  let etDate  = zone.ConvertTime(civil, eastern)

  ( civil.Year >= 2007 && eastern.IsDaylightSavingTime etDate )
    ==> lazy (Zone.inUnitedStatesDaylightTime etDate = eastern.IsDaylightSavingTime etDate)
```

TEST EXECUTION SUMMARY

Tests run: 1, Errors: 0, Failed: 1, Ignored: 0

✔ **modern daylight savings test oracle**
OK, passed 100 tests

CUSTOM DATA GENERATION

CUSTOM DATA GENERATION

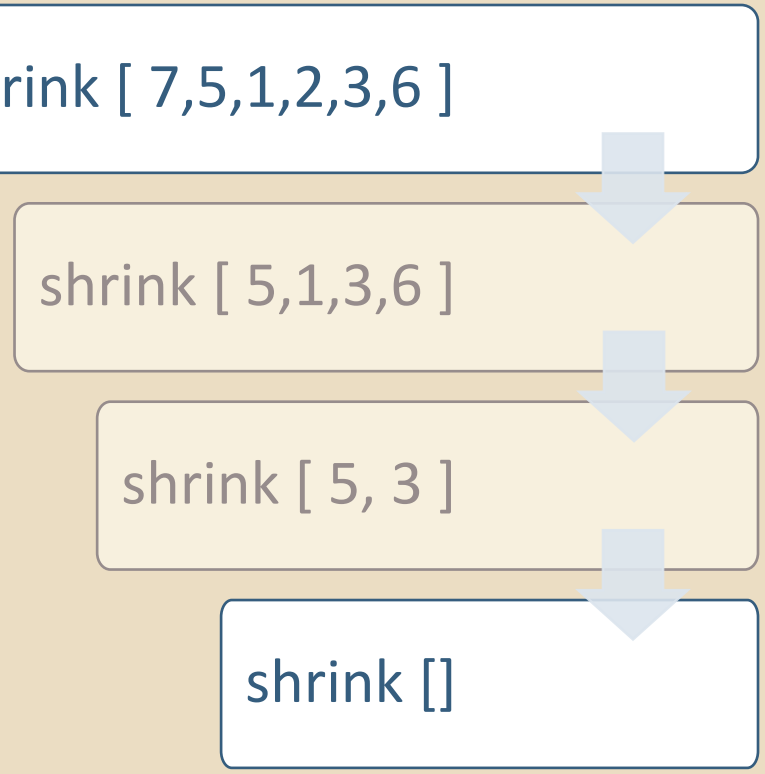
Generators

Gen<int> → int

Gen<Address> → Address

Gen<'T list> → 't list

Shrinkers



INPUT CONTROL:

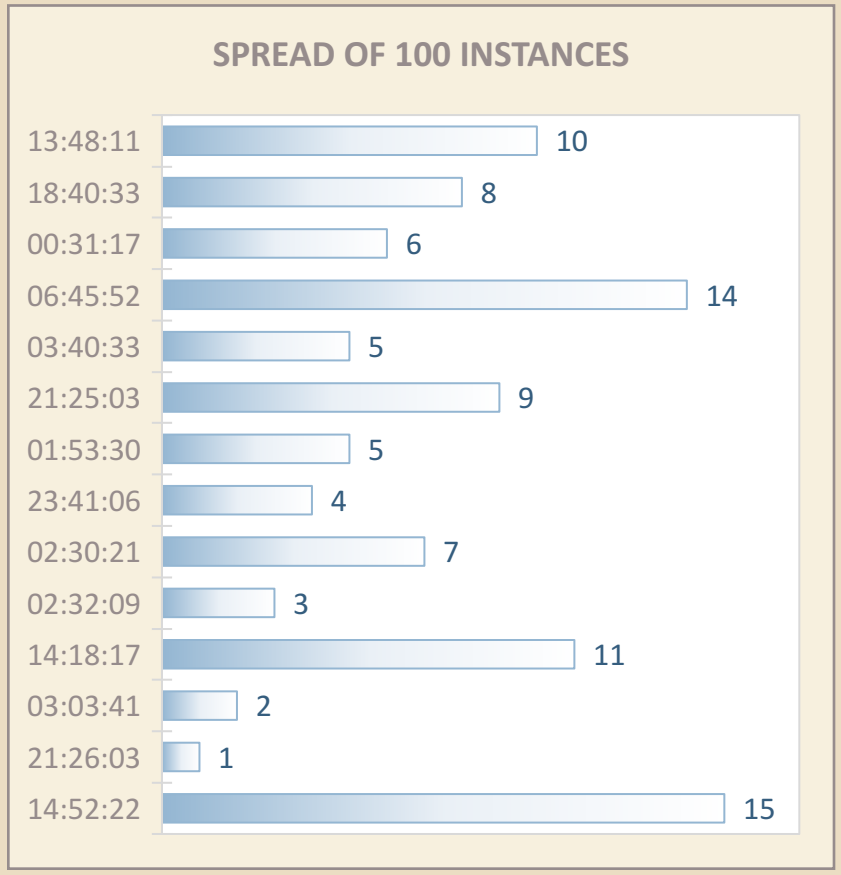
Universal Quantifiers

```
// instead of a conditional property,
// we can use an Arb and universal quantification
let ``serialization is invertable`` () =
  // define arbitrary generator
  let zones =
    zone.GetSystemTimeZones ()
    |> Gen.elements
    |> Arb.fromGen

  // "for all" zones, run a test...
  Prop.forAll zones (fun target ->
    let deflated = target.ToSerializedString ()
    zone.FromSerializedString deflated = target
  )
```

DATA GENERATION:

Arb with Gen & Shrinker




```
/// encapsulates several IArbitrary instances
type Generator =
  (* … other generators elided … *)
  static member PositiveTime =
    let inline isPositive t = t > time.Zero
    Arb.fromGenShrink
      ( // generator
        Arb.generate<time>
        |> Gen.where isPositive
        |> Gen.map positiveTime
      , // shrinker
        (fun (PositiveTime t) ->
          Arb.shrink t
          |> Seq.where isPositive
          |> Seq.map positiveTime) )
```

RANDOM TESTING

"One of the major advantages… is that it encourages us to formulate formal specifications, thus improving our understanding…"

---

from ICFP'00 – Claessen, Hughes

Further Information

This presentation will soon be available on the conference website at...

skillsmatter.com/conferences/8053-f-sharp-exchange-2017#skillscasts

Additional details about FsCheck may be found at...

https://fscheck.github.io/FsCheck