

QUICK! Check your Properties

(Random Testing on .Net with FsCheck)

fscheck.github.io/FsCheck | github.com/pblasucci/quickpbt

TODAY'S AGENDA

- ✓ *Introduction*
- ✓ Common Patterns
- ✓ Diagnostics
- ✓ Input Control
- ✓ Data Generation*
- ✓ Conclusion

RANDOM TESTING

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

– Claessen, Hughes (*ICFP'00*)

FROM EXAMPLE TESTING...

```
[<Fact>]
let ``days should equal hours`` () =
    let today = Date.Now // NOTE: hard-coded value
    let days   = today + Time.FromDays(daysInAWeek)
    let hours  = today + Time.FromHours(hoursInAWeek)

    Assert.Equal(days, hours)
```

TEST EXECUTION SUMMARY

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

✓ **days should equal hours**

OK, Elapsed time: 0.0527666s

TO PROPERTY TESTING

```
[<Property>]
let ``plus ignores unit of time`` (anyDate :Date) =
    // NOTE: lots of different, random values
    let days   = anyDate + Time.FromDays(daysInAWeek)
    let hours  = anyDate + Time.FromHours(hoursInAWeek)

    days = hours
```

TEST EXECUTION SUMMARY

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

✓ **plus ignores unit of time**

OK, Passed 100 tests

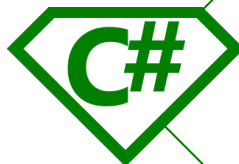
DOMAIN UNDER TEST



```
type Date = System.DateTimeOffset  
type Time = System.TimeSpan  
type Zone = System.TimeZoneInfo
```



```
Imports Dated = System.DateTimeOffset  
Imports Timed = System.TimeSpan  
Imports Zoned = System.TimeZoneInfo
```



```
using Date = System.DateTimeOffset;  
using Time = System.TimeSpan;  
using Zone = System.TimeZoneInfo;
```

PATTERNS: *Interchange & Invariance*

// interchange ... the property by which the order of two or more actions does not affect the outcome

```
public bool adding_and_changing_zone_can_be_reordered(Date anyDate, PositiveInt total) {  
    var days = Time.FromDays((int) total);  
    var addThenShift = Zone.ConvertTimeBySystemTimeZoneId(anyDate + days, CentralEuroTime);  
    var shiftThenAdd = Zone.ConvertTimeBySystemTimeZoneId(anyDate, CentralEuroTime) + days;  
  
    return addThenShift == shiftThenAdd;  
}
```

// invariance ... the property by which something remains constant, despite action being taken

```
public bool adding_does_not_change_the_date_offset(Date anyDate, PositiveInt months) {  
    var offset = anyDate.Offset;  
    var shifted = anyDate.AddMonths((int) months);  
  
    return shifted.Offset == offset;  
}
```

PATTERNS: *Inversion & Idempotence*

' inversion ... the property by which one action “undoes” the work of another action

```
Public Function AddingAndSubtractingDaysAreInverses(anyDate As Dated, total As PositiveInt) As Boolean
```

```
    Dim days = Timed.FromDays(CInt(total))
```

```
    Return (anyDate + days) - days = anyDate
```

```
End Function
```

' idempotence ... the property of an action having the same effect no matter how many times it occurs

```
Public Function TakingTimeDurationIsIdempotent(anyTime As Timed) As Boolean
```

```
    Dim once = anyTime.Duration()
```

```
    Dim twice = anyTime.Duration().Duration()
```

```
    Return once = twice
```

```
End Function
```

DIAGNOSTICS: *Labelling Properties*

```
public bool zone_conversion_is_not_affected_by_detours (Date anyDate, Zone zone1, Zone zone2){  
    var viaZone1 = Zone.ConvertTime(Zone.ConvertTime(anyDate, zone1), zone2);  
    var directly = Zone.ConvertTime(anyDate, zone2);  
    return (viaZone1 == directly)                // same date  
        && (directly.Offset == zone2.BaseUtcOffset); // same shift  
}
```

TEST EXECUTION SUMMARY

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

✗ zone_conversion_is_not_affected_by_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))

DIAGNOSTICS: *Labelling Properties*

```
public bool zone_conversion_is_not_affected_by_detours(Date anyDate, Zone zone1, Zone zone2) {  
    var viaZone1 = Zone.ConvertTime(Zone.ConvertTime(anyDate, zone1), zone2);  
    var directly = Zone.ConvertTime(anyDate, zone2);  
    bool sameDate () => (viaZone1 == directly);  
    bool sameShift() => (directly.Offset == zone2.BaseUtcOffset);  
    return sameDate ().Label($"Same Date? ({viaZone1} = {directly})")  
        .And(sameShift().Label($"Same Shift? ({zone2.BaseUtcOffset} = {directly.Offset})"));  
}
```

TEST EXECUTION SUMMARY

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

✗ zone_conversion_is_not_affected_by_detours

Falsifiable, after 4 tests (5 shrinks), (StdGen (199662269,296213481))

Label of failing property: Same Shift? (03:00:00 = 02:28:00)

...

DIAGNOSTICS: *Labelling Properties*

```
public bool zone_conversion_is_not_affected_by_detours(Date anyDate, Zone zone1, Zone zone2) {  
    var viaZone1 = Zone.ConvertTime(Zone.ConvertTime(anyDate, zone1), zone2);  
    var directly = Zone.ConvertTime(anyDate, zone2);  
    bool sameDate () => (viaZone1 == directly);  
    bool sameShift() => (directly.Offset == zone2.GetUtcOffset(directly));  
    return sameDate ().Label($"Same Date? ({viaZone1} = {directly})")  
        .And(sameShift().Label($"Same Shift? ({zone2.BaseUtcOffset} = {directly.Offset})"));  
}
```

TEST EXECUTION SUMMARY

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

✓ **zone_conversion_is_not_affected_by_detours**

OK, Passed 100 tests

DIAGNOSTICS: *Gathering Observations*

```
// trivial observation partitions data into two buckets
```

```
let ``trivial daylight savings support`` (aDate :Date) (aZone :Zone) (NonNegativeInt total) =
```

```
let days = Time.FromDays(total)
```

```
let addShift = Zone.ConvertTime(aDate + days,aZone)
```

```
let shiftAdd = Zone.ConvertTime(aDate,aZone) + days
```

```
(addThenShift = shiftThenAdd)
```

```
|> Prop.trivial anyZone.SupportsDaylightSavingTime
```

TEST EXECUTION SUMMARY

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

✓ **trivial daylight savings support**

OK, Passed 100 tests

53% true

47% false

DIAGNOSTICS: *Gathering Observations*

' classification partitions into N, labelled buckets

```
Public Function ClassifyMeridianPosition(aDate As Dated, aZone As Zoned, total As NonNegativeInt) As [Property]
```

```
    Dim days = Timed.FromDays(CInt(total))
```

```
    Dim addShift = Zoned.ConvertTime(aDate + days, aZone)
```

```
    Dim shiftAdd = Zoned.ConvertTime(aDate, aZone) + days
```

```
    Return (addShift = shiftAdd)_
```

```
        .Classify(aDate.Offset < Timed.Zero, "< GMT")_
```

```
        .Classify(aDate.Offset = Timed.Zero, "|GMT|")_
```

```
        .Classify(aDate.Offset > Timed.Zero, "GMT <")
```

```
End Function
```

TEST EXECUTION SUMMARY

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

✓ ClassifyMeridianPosition

OK, Passed 100 tests

55% GMT <

43% < GMT

2% |GMT|

DIAGNOSTICS: *Gathering Observations*

```
// instead of a boolean data, collect reports any value
public Property collect_weekday_name(Date aDate, Zone aZone, NonNegativeInt total)
{
    var days = Time.FromDays((int) total);
    var addShift = Zone.ConvertTime(aDate + days, aZone);
    var shiftAdd = Zone.ConvertTime(aDate, aZone) + days;

    return (addShift == shiftAdd)
        .Collect(anyDate.DayOfWeekName());
}
```

TEST EXECUTION SUMMARY

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

✓ collect_weekday_name

OK, Passed 100 tests

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

DIAGNOSTICS: *Gathering Observations*

```
// observations may be combined as much as is desired
```

```
let ``many observations combined`` (aDate :Date) (aZone :Zone) (NonNegativeInt total) =
```

```
    let days = Time.FromDays(total)
```

```
    let addShift = Zone.ConvertTime(aDate + days,aZone)
```

```
    let shiftAdd = Zone.ConvertTime(aDate,aZone) + days
```

```
(addThenShift = shiftThenAdd)
```

```
|> Prop.trivial  anyZone.SupportsDaylightSavingTime
```

```
|> Prop.classify (aDate.Offset < Time.Zero) "< GMT"
```

```
|> Prop.classify (aDate.Offset = Time.Zero) "|GMT|"
```

```
|> Prop.classify (aDate.Offset > Time.Zero) "GMT <"
```

```
|> Prop.collect  (weekdayName anyDate)
```

TEST EXECUTION SUMMARY

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

✓ many observations combined

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

...

Input Control: *Conditional Properties*

' naive test fails (because the range of inputs is too broad)

```
Public Function DaylightSavingsTestOracle(anyDate As Dated) As Boolean
```

' NOTE: this test also demonstrates the common pattern of the "test oracle" pattern

```
Dim eastern = Zoned.FindSystemTimeZoneById(ZoneName)
```

```
Dim eastDate = Zoned.ConvertTime(anyDate, eastern)
```

```
Return Zone.InUnitedStatesDaylightTime(eastDate) = eastern.IsDaylightSavingTime(eastDate)
```

```
End Function
```

TEST EXECUTION SUMMARY

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

✗ DaylightSavingsTestOracle

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*

' naive test fails (because the range of inputs is too broad)

```
Public Function DaylightSavingsTestOracle(anyDate As Dated) As Boolean
```

```
    Dim eastern    = Zoned.FindSystemTimeZoneById(ZoneName)
```

```
    Dim eastDate   = Zoned.ConvertTime(anyDate, eastern)
```

```
    Dim check = Function() Zone.InUnitedStatesDaylightTime(eastDate)
```

```
    Return check().When(anyDate.Year >= 2007 AndAlso eastern.IsDaylightSavingTime(eastDate))
```

```
End Function
```

TEST EXECUTION SUMMARY

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

✓ **DaylightSavingsTestOracle**

OK, Passed 100 tests

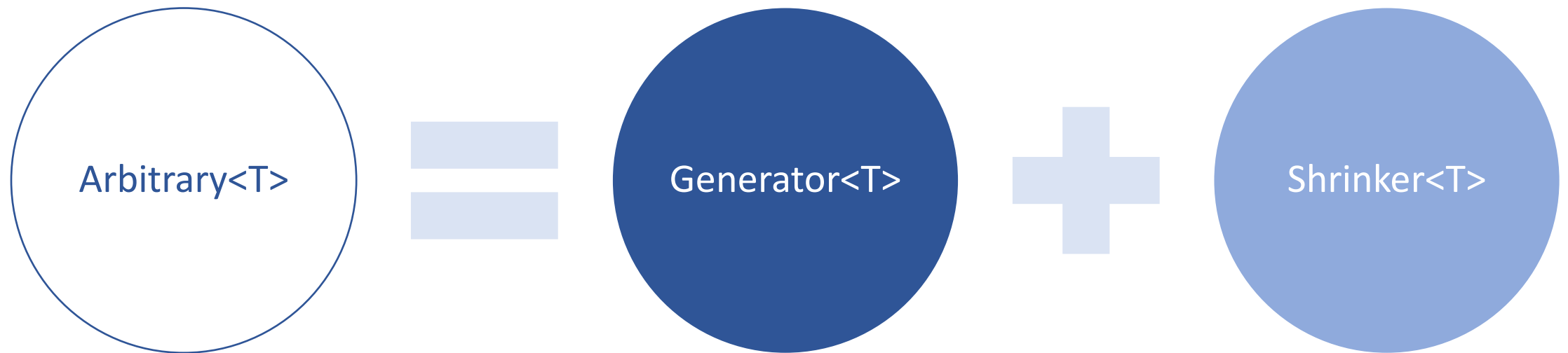
Input Control: *Universal Quantifiers*

```
// instead of a conditional property, here we use a IArbitrary with a "universal quantifier"
let ``zone is unchanged through round-trip serialization`` () =
    // define a simple test
    bool check (Zone anyZone)
    {
        var deflated = anyZone.ToSerializedString();
        var inflated = Zone.FromSerializedString(deflated);
        return inflated.Equals(anyZone);
    }

    // arbitrary generators can be easily defined
    var zones = Gen.Elements(from z in Zone.GetSystemTimeZones() select z).ToArbitrary();

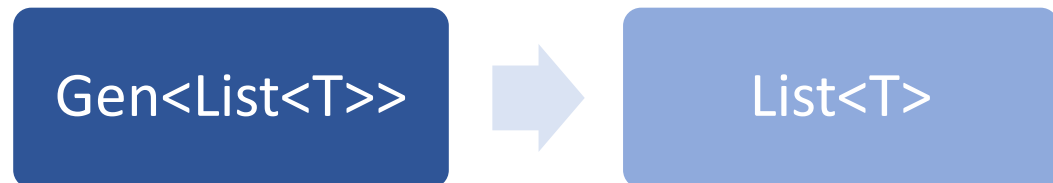
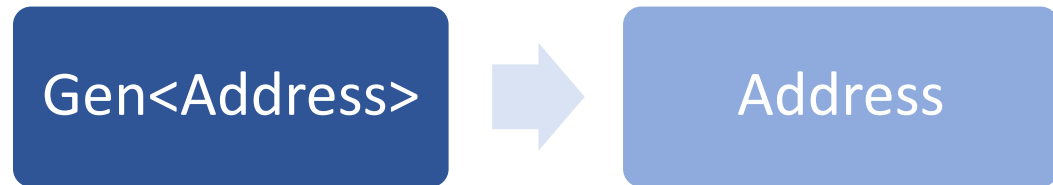
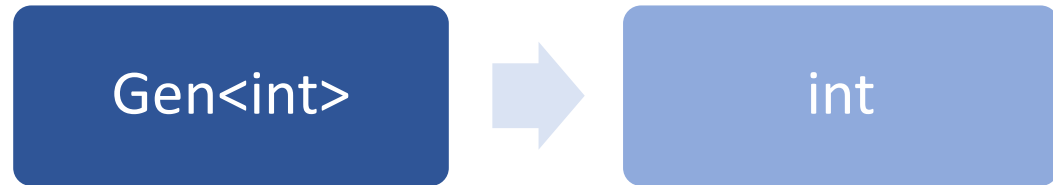
    // "for all" zones, run a test...
    return Prop.ForAll(zones, check);
```

CUSTOM DATA GENERATION

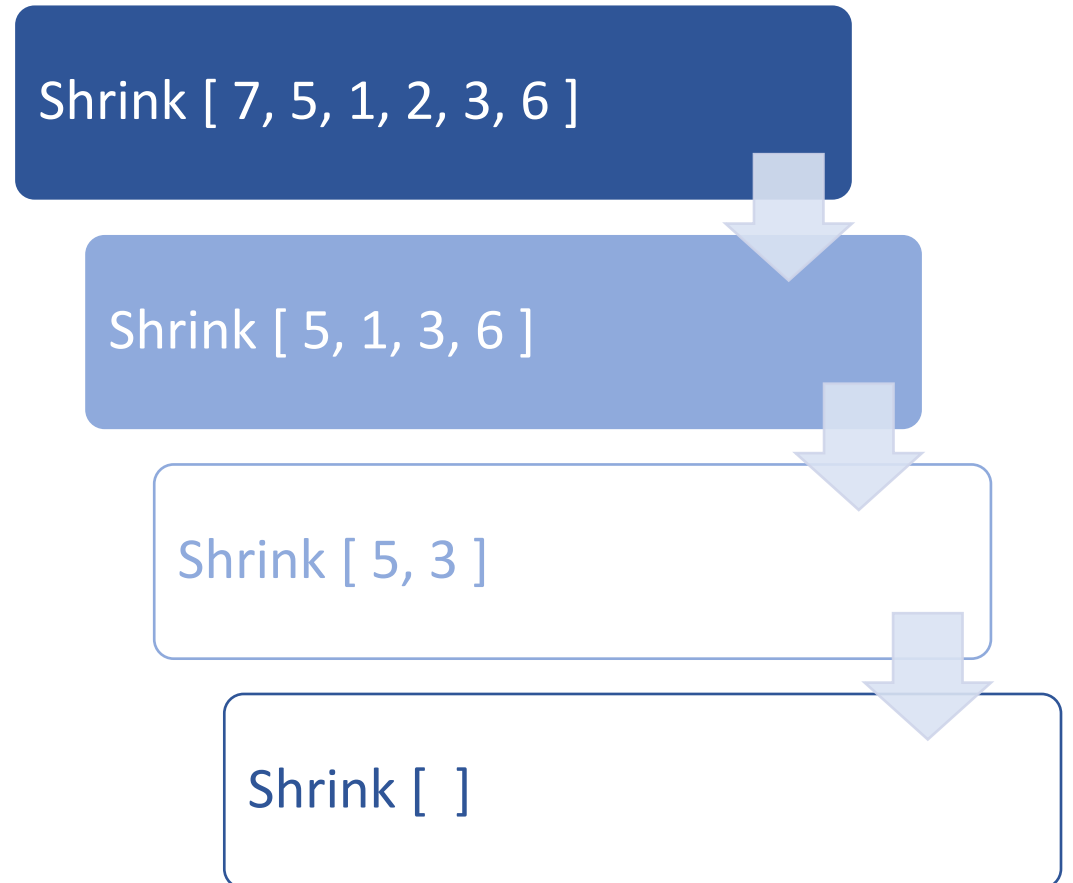


CUSTOM DATA GENERATION

Generators



Shrinkers



DATA GENERATION: *Arb with Gen & Shrinker*

```
/// encapsulates several IArbitrary instances
```

```
type Generator =
```

```
/// generates PositiveTime instances
```

```
static member PositiveTime =
```

```
let inline isPositive t = Time.Zero < t
```

```
Arb.fromGenShrink
```

```
( // generator
```

```
Arb.generate<Time>
```

```
|> Gen.where isPositive
```

```
|> Gen.map positiveTime
```

```
, // shrinker
```

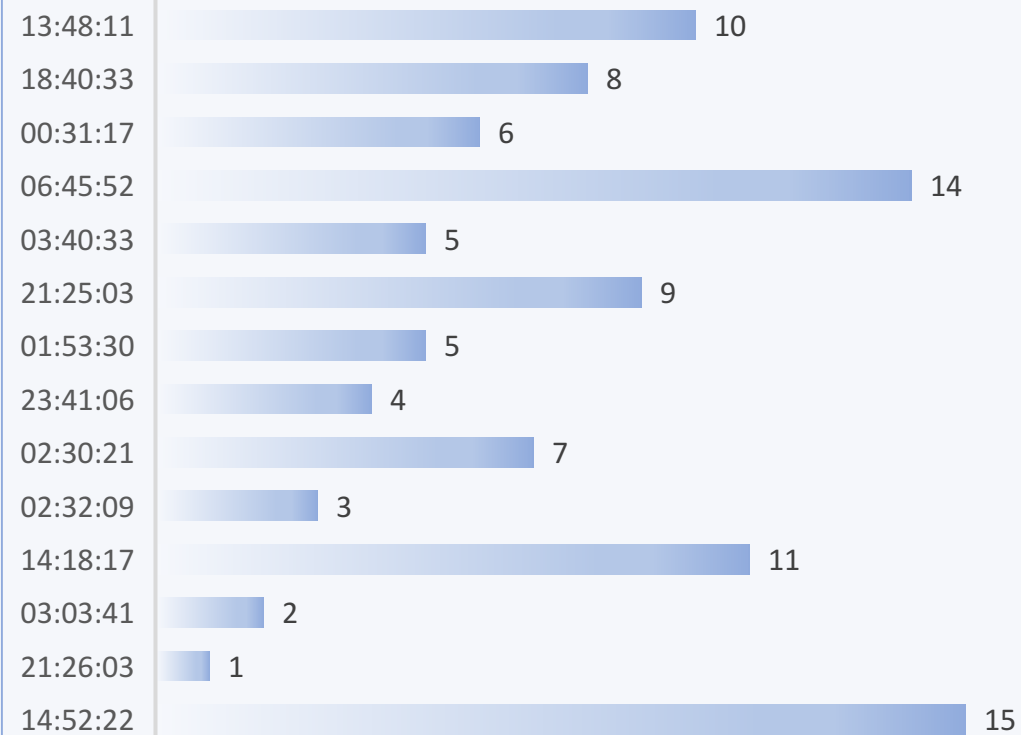
```
(fun (PositiveTime t) ->
```

```
Arb.shrink t
```

```
|> Seq.where isPositive
```

```
|> Seq.map positiveTime) )
```

SPREAD OF 100 INSTANCES

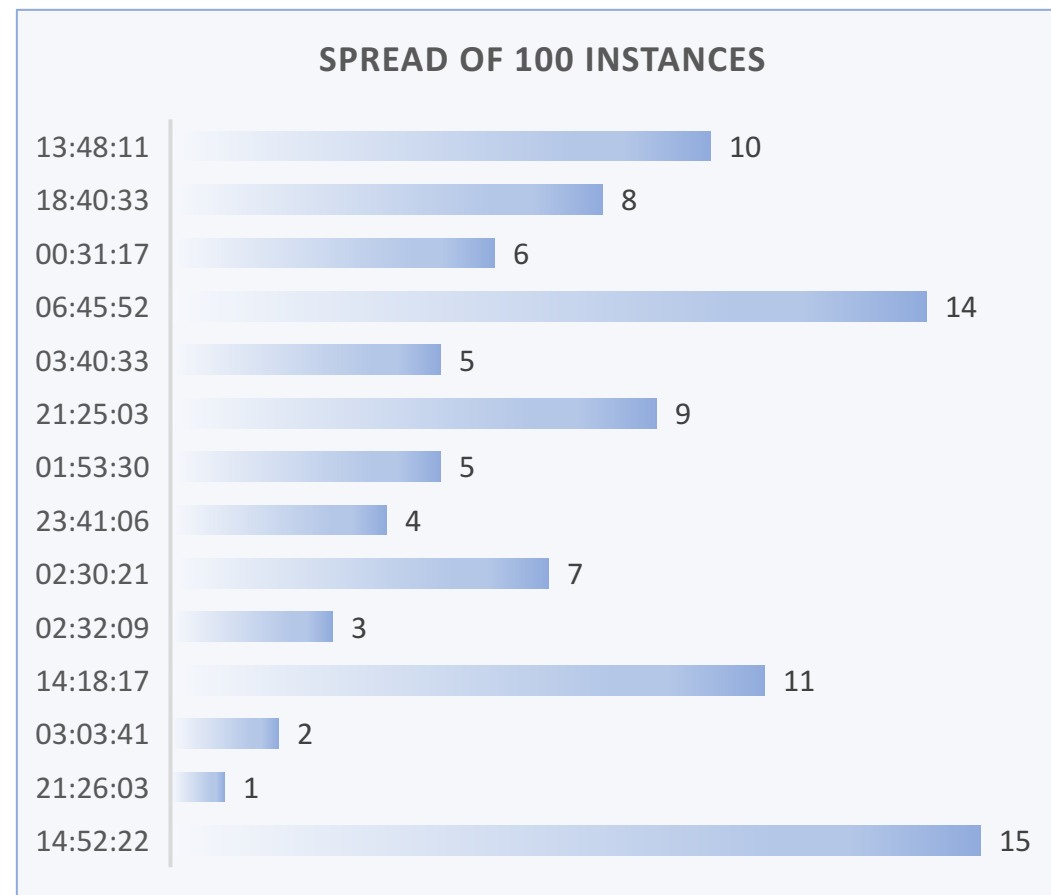


DATA GENERATION: *Arb with Gen & Shrinker*

```
/// a time value which is always greater then zero
/// (note: only meant for use with FsCheck)
type PositiveTime = private PosTime of Time

/// returns a new PositiveTime instance,
/// throwing an exception on values less than zero
let positiveTime value =
    if value <= Time.Zero then
        invalidArg "value" "value must be greater than 0"
    PosTime value

/// extracts the TimeSpan from a PositiveTime instance
let (|PositiveTime|) (PosTime value) = value
```



RANDOM TESTING

“One of the major advantages... is that it **encourages** us to formulate **formal specifications**, thus **improving** our **understanding...**”

– Claessen, Hughes (*ICFP'00*)