What is QuickCheck?

- Write properties:

Test data generator

```
prop_reverse() ->
  ?FORALL({Xs,Ys},{list(int()),list(int())},
     equals(reverse(Xs++Ys),
            reverse(Xs)++reverse(Ys))).
```

Wrong way round

- Get counterexamples:

```
56> eqc:quickcheck(reverse_eqc:prop_reverse()).
.............Failed! After 14 tests.
{[0,-1],[-1]}
[-1,-1,0] /= [-1,0,-1]
Shrinking..(2 times)
{[0],[1]}
[1,0] /= [0,1]
```

Xs and Ys

Shrunk counterexample

Benefits

- Less time spent writing test code
  - One property replaces many tests

- Better testing
  - Lots of combinations you'd never test by hand

- Less time spent on diagnosis
  - Failures minimized automagically

Free QuickCheck for All!

John Hughes

CHALMERS

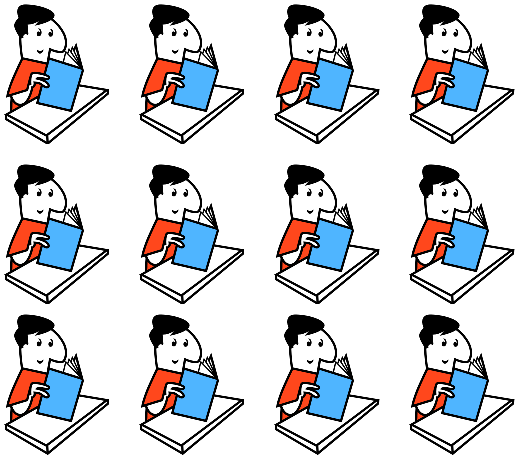
Chalmers University of Technology, Sweden

QuviQ

http://quviq.com/downloads/eqcmini.zip

An Experiment

Unit tests

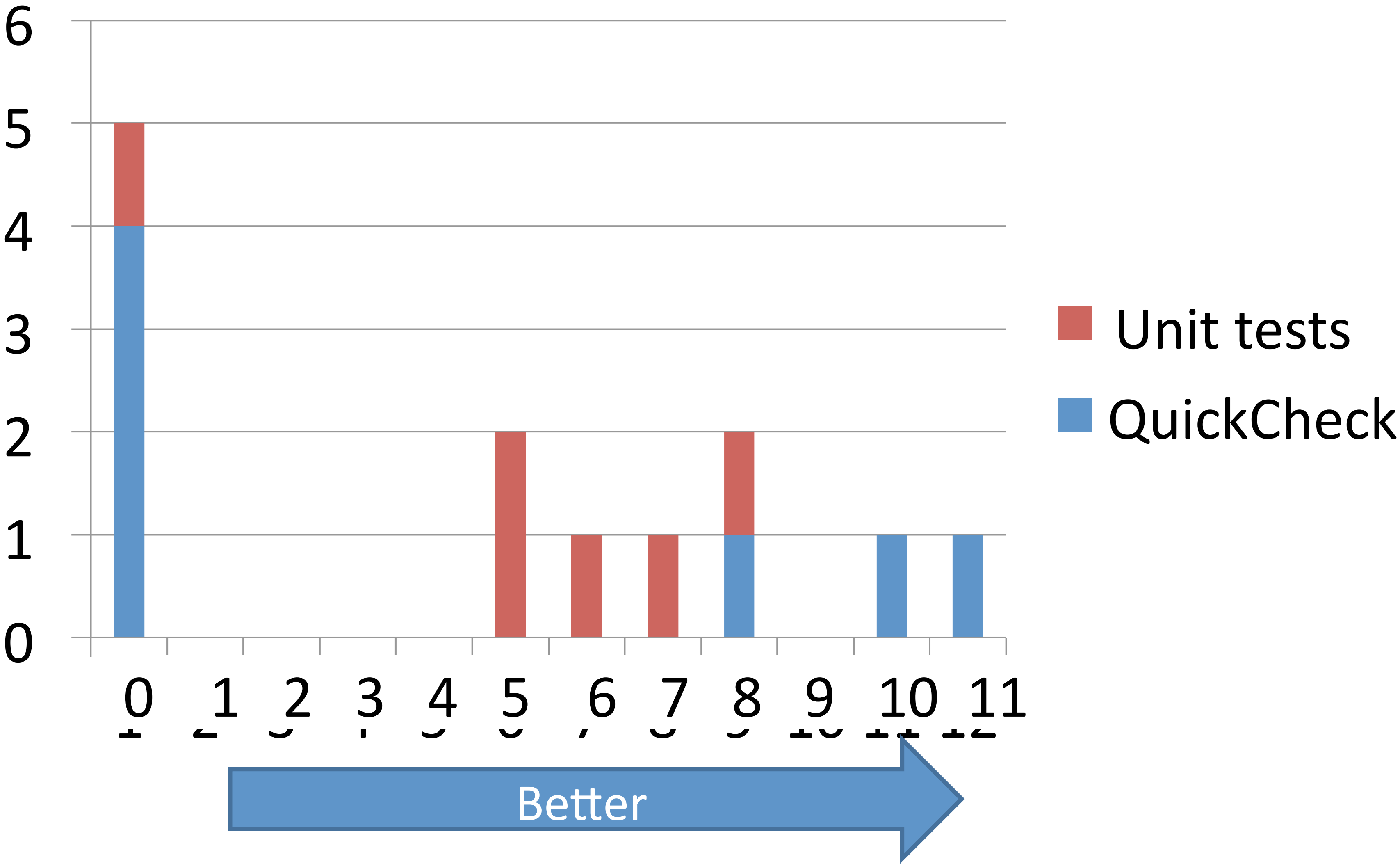


Properties

Comparing Test Suites

	Answer 1	Answer 2	Answer 3	Answer 4
Test Suite 1	✗	✓	✗	✓
Test Suite 2	✓	✗	✗	✗
Test Suite 3	✓	✗	✓	✓
Test Suite 4	✗	✓	✗	✓

Test Suite Quality



Some Unit Tests

Expected results

Test cases

```
base64_encode(Config) when is_
    %% Two pads
    <<"QWxhZGRpbjpvcGVuIHNlc2FtZQ==">> =
         base64:encode("Aladdin:open sesame"),

    %% One pad
    <<"SGVsbG8gV29ybGQ=">> = base64:encode(<<"Hello World">>),

    %% No pad
    "QWxhZGRpbjpvcGVuIHNlc2Ft" =
         base64:encode_to_string("Aladdin:open sesam"),

    "MDEyMzQ1Njc4OSFAIzBeJiooKTs6PD4sLiBbXXt9" =
         base64:encode_to_string(
              <<"0123456789!@#0^&*();:<>,. []{}">>),
    ok.
```

Writing a Property

```
prop_base64() ->
  ?FORALL(Data,list(choose(0,255)),
          equals(base64:encode(Data),
                 ???)).
```

Some Unit Tests

Where did these come from?

```
base64_encode(Config) when is_list(Config) ->
    %% Two pads
    <<"QWxhZGRpbjpvcGVuIHNlc2FtZQ==">> =
        base64:encode("Aladdin:open sesame"),

    %% One pad
    <<"SGVsbG8gV29ybGQ=">> = base64:encode(<<"Hello World">>),

    %% No pad
    "QWxhZGRpbjpvcGVuIHNlc2Ft" =
        base64:encode_to_string("Aladdin:open sesam"),

    "MDEyMzQ1Njc4OSFAIzBeJiooKTs6PD4sLiBbXXt9" =
        base64:encode_to_string(
             <<"0123456789!@#0^&*();:<>,. []{}">>),
    ok.
```

Possibilities

- Someone converted the data
- Another base64 encoder
- The same base64 encoder!
  - Only tests that changes don t that the result is right

Use the other encoder as an oracle

Use an old version (or a simpler version) as an oracle

Round-trip Properties

```
prop_encode_decode() ->
  ?FORALL(L,list(choose(0,255)),
    equals(base64:decode(base64:encode(L)),
           list_to_binary(L))).
```

What does this test?

- **NOT** a complete test—will not find a consistent misunderstanding of base64
- **WILL** find mistakes in encoder or decoder

Simple properties find a lot of bugs!

TAKE HOME MSG

The Student Problem

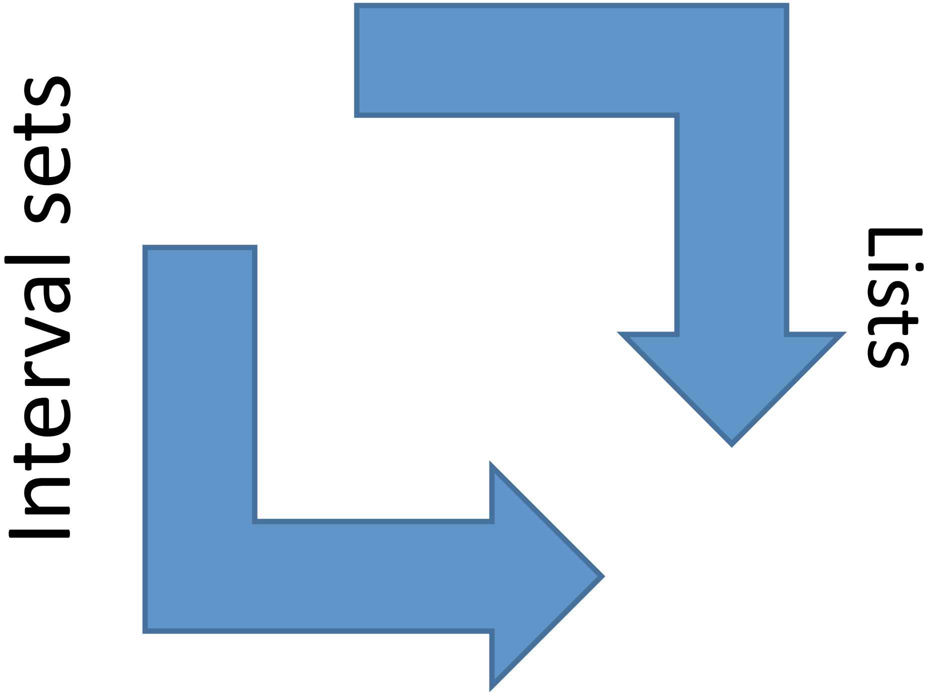
Interval Sets

- Represent sets of integers as interval lists
  - [{1,3},{7,10}] represents [1,2,3,7,8,9,10]

- Implement
  - to_list, member, empty, singleton, union, diff

IDEA!

Use lists as a model!



A Property for union

The property:

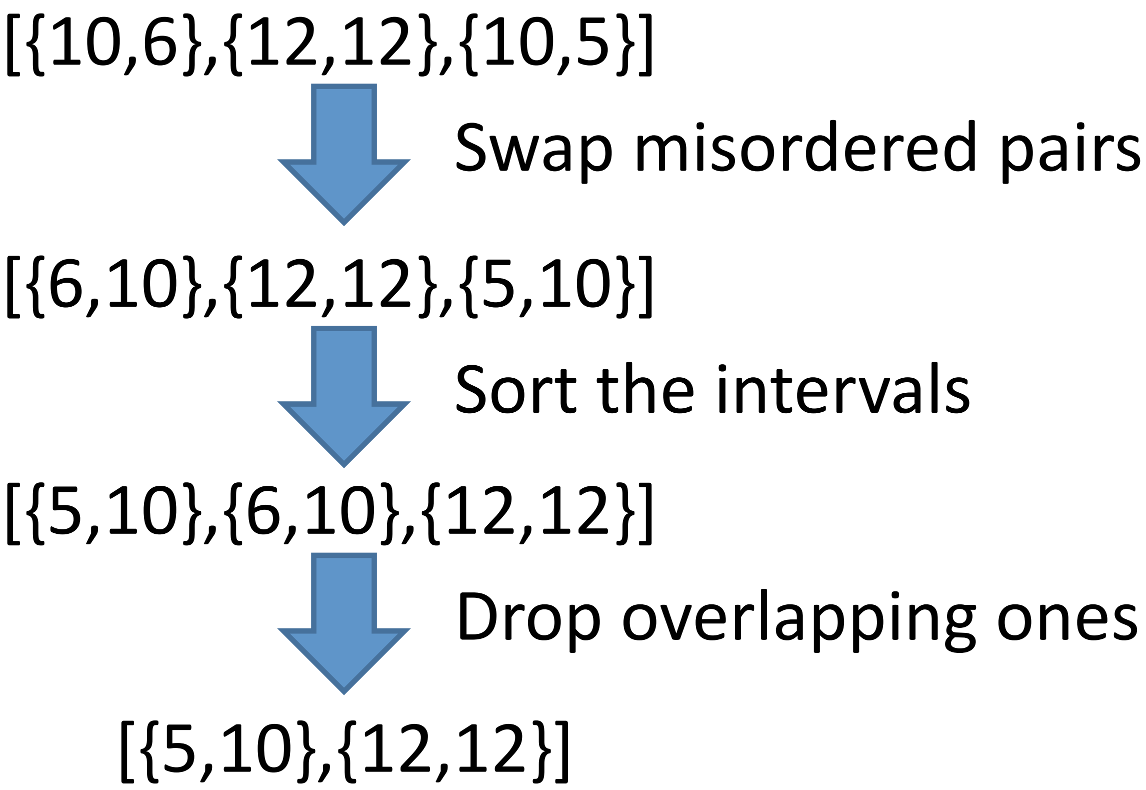
```
prop_union() ->
  ?FORALL({S1,S2},{iset(),iset()},
    equals(
      to_list(union(S1,S2)),
      lists:umerge(to_list(S1),to_list(S2)))).
```

Converting to the model:

```
to_list(S) ->
  lists:merge(
    [lists:seq(Lo,Hi) || {Lo,Hi} <- S]).
```

Generating Interval Sets

- A list of pairs?
  - **`list({nat(),nat()})`** ?



[{10,6},{12,12},{10,5}]

Swap misordered pairs

[{6,10},{12,12},{5,10}]

Sort the intervals

[{5,10},{6,10},{12,12}]

Drop overlapping ones

[{5,10},{12,12}]

The iset() generator

```
iset() ->
  ?LET(L,list({nat(),nat()}),
       drop_overlaps(
         lists:sort(
           [{min(A,B),max(A,B)} || {A,B}<-L]
                   ))).
```

- **?LET** generates values in two steps

Validity

```
valid([{Lo1,Hi1},{Lo2,Hi2}|Rest]) ->
    Lo1 =< Hi1 andalso Hi1 =< Lo2 - 2
        andalso valid([{Lo2,Hi2}|Rest]);
valid([{Lo,Hi}]) -> Lo =< Hi;
valid([])         -> true.
```

```
prop_valid() ->
    ?FORALL(S,iset(),valid(S)).
```

What does this test?

Another nice property

```
prop_union_valid() ->
    ?FORALL({S1,S2},{iset(),iset()},
        valid(union(S1,S2))).
```

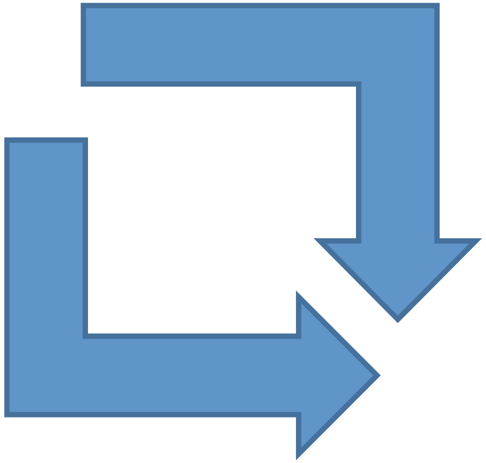
Let’s run some tests!

Lessons

- Simple properties find bugs!

- Use a model to decide test outcomes

- ”Can I compute this another way?”



Property Driven Development

- Property-based development is QUICK!
  - – Effort invested in setting up properties is quickly repaid
- No luxury of leaving code ”half working”
- Resulting code is *very solid*
  - – No going back to fix bugs in last week’s code
- Mistaken design shows up *fast*
  - – Complex properties, complex code

Try doing iset:diff…

http://quviq.com/downloads/eqcmini.zip

It’s free. Use it!