

Testing with QuickCheck

John Hughes

CHALMERS

QuviQ
●●●



50%



State of the Art

1,5MLOC Erlang, 2MLOC

C++



Report of
test case
failures

Test
Server

Automate
d test
cases

700KLOC Erlang

- Nightly runs provide rapid feedback
- New test cases added for each error found



10/KLOC



\$60 billion



15%



Can we do
better?

Lower costs+better quality

Test Cases



Properties

- Cover *one* case
- Specify correct behaviour only for that case

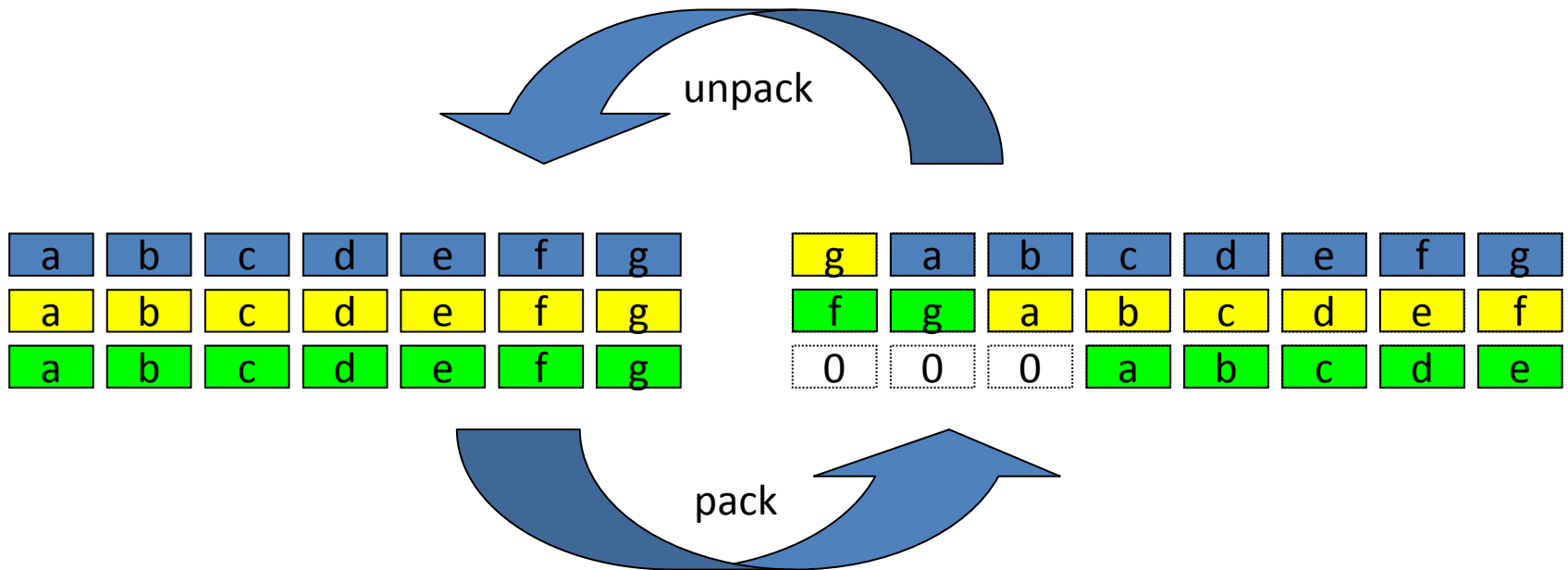
- Cover *many* cases at once
- Specify correct behaviour in *any* of the cases covered

More general
More concise test code
More test cases

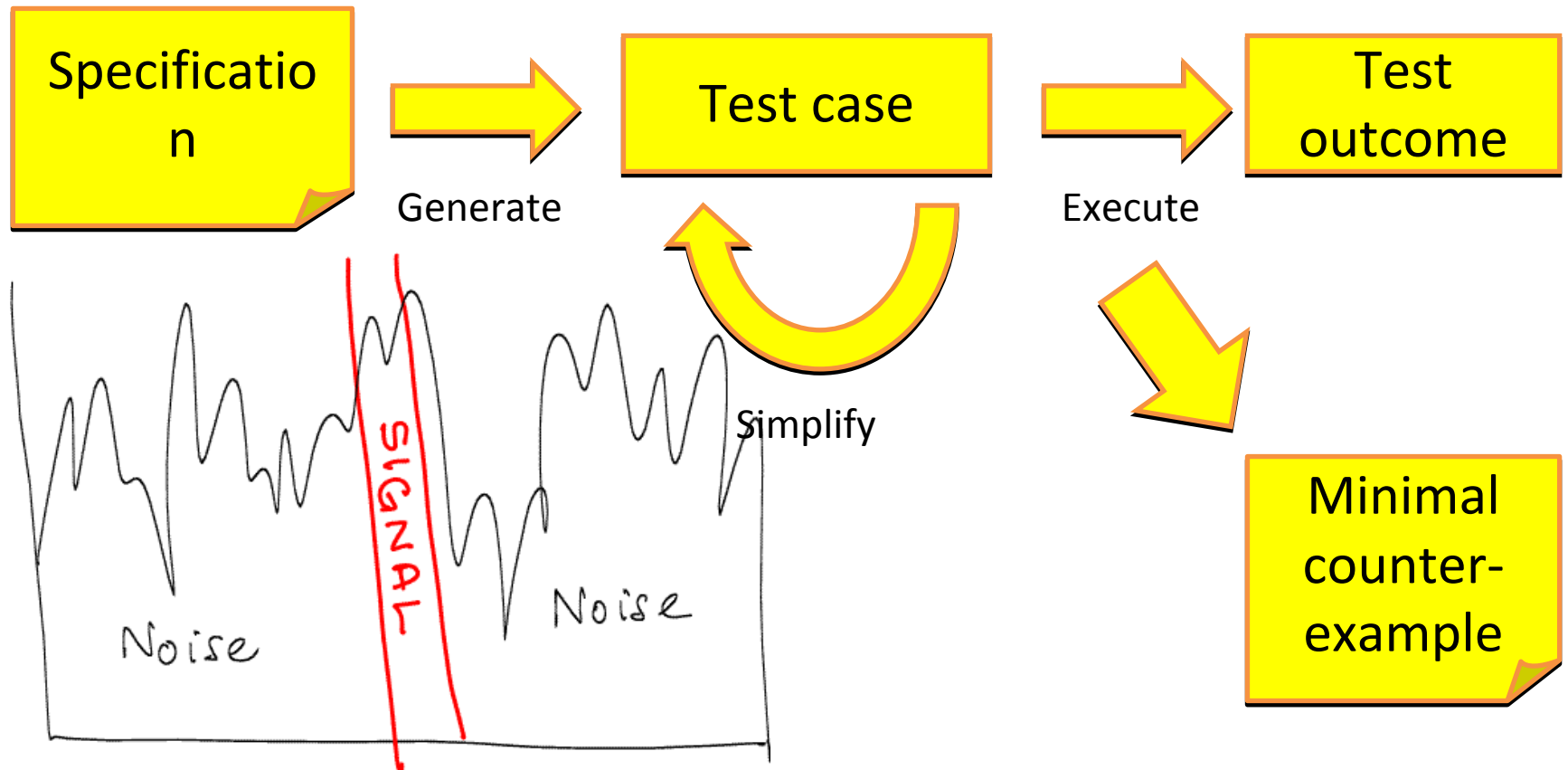


Example: SMS Encoding

- 7-bit characters packed into 8-bit bytes



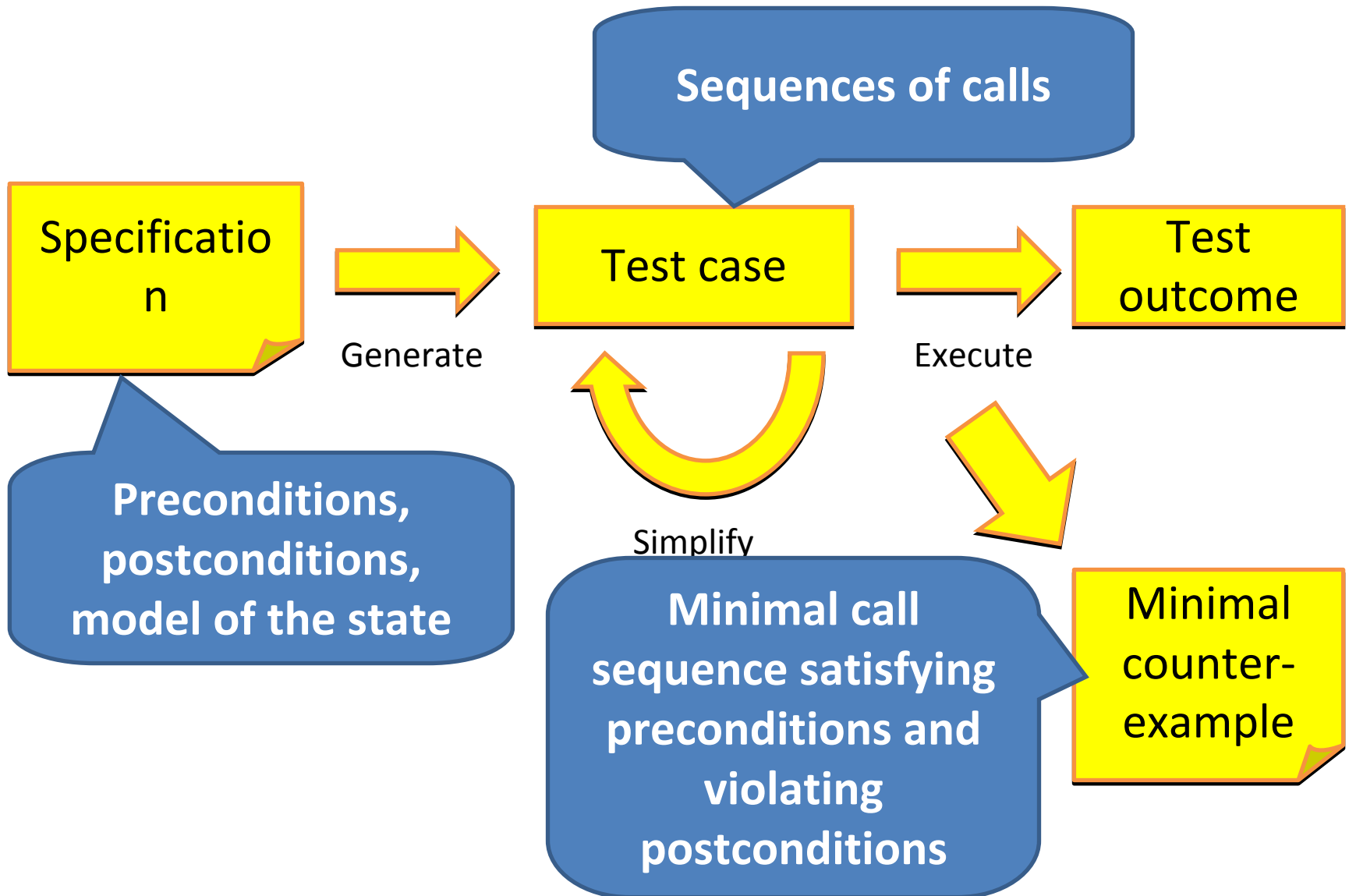
QuickCheck in Brief



Example: File I/O

- `fread(buffer,size,stream)`
 - Read size bytes from a file into a buffer
- `fwrite(buffer,size,stream)`
 - Write size bytes from a buffer into a file
- `fseek(stream,pos)`
 - Set the file pointer to a given position

QuickCheck in Brief



Modelling File Contents

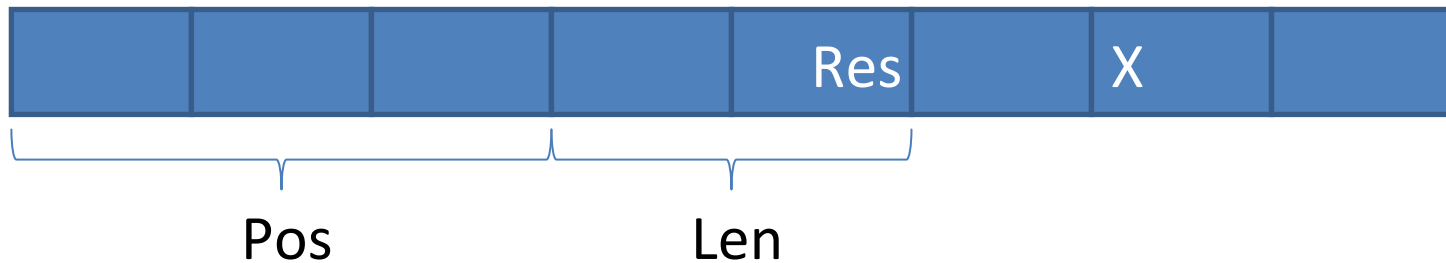
- A *list* of bytes is a natural model



- List processing functions used to specify behaviour
 - E.g. `split(N,L) → {Prefix,Suffix}`

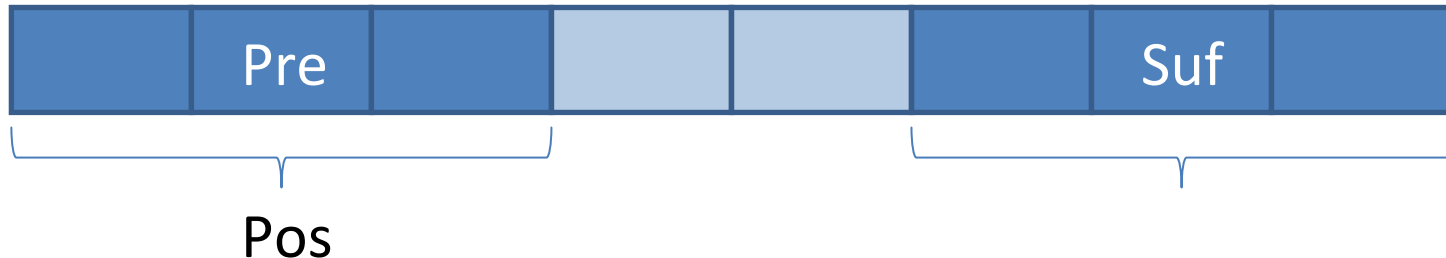
Modelling a Read


```
extract(L, Pos, Len) ->  
  {_, X} = split(Pos, L),  
  {Res, _} = split(Len, X),  
  Res.
```



Modelling a Write

```
insert(L,Pos,Data) ->  
  {Pre,X} = split(Pos,L),  
  {_,Suf} = split(length(Data),X),  
  extend(Pre,Pos,0)++Data++Suf.
```



- Only tricky point  *ending* the file if need be

QuickCheck Code Fragment: Generating Commands

```
command(_) ->
  oneof([ {call,c,fread,[size()]},
          {call,c,fwrite,
            [list(choose(-128,127))]},
          {call,c,fseek,[pos()]} ]).
```

- We generate the parameters we want to vary in tests
- Test cases are *sequences* of these commands

QuickCheck Code Fragment: Modelling Effects

- State of a file is modelled by a record with *contents* and *position* fields
- Effect of a command is modelled by a *state transition* function

```
next_state(S,_,{call,c,fwrite,[Data]}) ->  
  #c_file{contents=L,position=Pos} = S,  
  S#c_file{contents=insert(L,Pos,Data),  
           position=Pos+length(Data)};
```

QuickCheck Code Fragment: Translating to C

- Translating to C is just a matter of
 - Generating source code for the call
 - Specifying result to return to QuickCheck

```
compile(C,{call,c,fread,[Size]}) ->
io:format(C,
    "TUPLE(INT(n=fread(buffer,1,~w,stream));"++
    "LIST(for(i=0;i<n;i++) INT(buffer[i])));",
    [Size]);
```

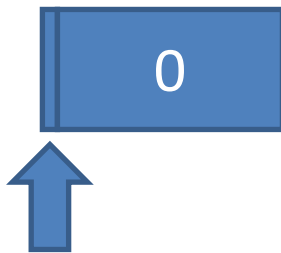
QuickCheck Code Fragment: Postcondition

- Check that the actual result corresponds to the functional model

```
postcondition(s, {call, c, fread, [Size]}, v) ->  
  {N, Data} = v,  
  #c_file{contents=L, position=Pos} = s,  
  N == length(Data) andalso  
  Data == extract(L, Pos, Size);
```

First Problem

```
[ {set, {var, 9}, {call, c, fseek, [1]}},  
  {set, {var, 11}, {call, c, fwrite, [[]]}},  
  {set, {var, 12}, {call, c, fseek, [0]}},  
  {set, {var, 16}, {call, c, fread, [1]}} ]  
[0, 0, 0, {0, []}]
```



We expect t
[0]

```
V9 = fseek(1);  
V11 = fwrite("");  
V12 = fseek(0);  
V16 = fread(1);
```

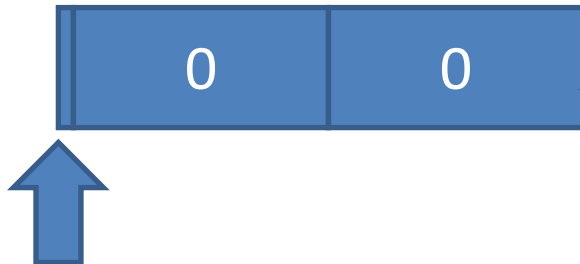
The Fix

- Writing *zero* bytes does not cause null-padding (undocumented)

```
next_state(S,_,{call,c,fwrite,[Data]}) ->
  case Data of
    [] -> S;
    _ ->
      #c_file{contents=L,position=Pos} = S,
      S#c_file{contents=insert(L,Pos,Data),
              position=Pos+length(Data)}
  end;
```

Second Problem

```
[ {set, {var, 20}, {call, c, fwrite, [[0, 0]]}},  
  {set, {var, 25}, {call, c, fseek, [0]}},  
  {set, {var, 26}, {call, c, fread, [1]}},  
  {set, {var, 27}, {call, c, fwrite, [[0]]}},  
  {set, {var, 28}, {call, c, fseek, [0]}},  
  {set, {var, 30}, {call, c, fread, [3]}}]  
[2, 0, {1, [0]}, 1, 0, {3, [0, 0, 0]}]
```



We expect to read *two* zeroes, but we get *three*!

The "Fix"

- Disallow
 - read after write
 - write after read, except at the end of the file
 - *Unless* there is an intervening seek
- It's in the C standard



Media Proxy

- Multimedia IP-telephony (IMS)
- Connects calls across a firewall
- Test adding and removing callers from a call

