QuickCheck Evolution

John Hughes

CHALMERS
QuviQ
Why is testing hard?

- \( n \) features \( \Rightarrow O(n) \) test cases
- Pairs of features \( \Rightarrow O(n^2) \) test cases
- Triples of features \( \Rightarrow O(n^3) \) test cases

3—4 tests per feature
Don’t write tests!

Generate them
QuickCheck

1999—invented by Koen Claessen and myself, for Haskell

2006—Quviq founded marketing Erlang version

Many extensions

Finding deep bugs for Ericsson, Volvo Cars, Basho, etc...
Example—binary trees

```
to_list(leaf) -> [];  
to_list({node,L,X,R}) ->  
    to_list(L) ++ [X] ++ to_list(R).

member(_,leaf) -> false;  
member(X,{node,L,Y,R}) ->  
    if X==Y -> true;  
        X<Y  -> member(X,R);  
        X>Y  -> member(X,L)  
    end.
```
A property of member

prop_member() ->

?FORALL({X,T}, {nat()}, {tree()},
member(X,T) == lists:member(X, to_list(T))).

For all X and T...

...generated like this...

...the member function behaves like lists:member
Let’s run some tests...
But... what was that example again?

• We may want to *preserve* examples that failed before, as a regression suite

• In reality, a failing case may take a long time to find... we don’t want to throw it away!
Enter... QuickCheck CI

DEMO
QuickCheck CI

- Builds a regression test suite automatically
  - See progress in terms of tests which now pass
  - Save *rare* tests which were hard to find

- Presents coverage information in depth
  - See at a glance what has been tested
  - See the effects of test case *distribution*
  - Helps localize bugs!
State machine testing—example

• Let’s test the process registry
  • register(Name,Pid)
  • unregister(Name)
  • spawn()—to create pids for test data

• What’s different now?
  • These functions change the state of the registry

• Not looking for bugs!
  • We’ll reverse engineer preconditions instead
State Machine Models

API Calls

postconditions

Model state

Model state

Model state

Model state
State Machine Models

API Calls → API Calls → API Calls → API Calls

Model state → Model state → Model state → Model state

postconditions
Modelling the registry state

```bash
#state{
  pids  = [...<0.32.0>...],
  regs  = [{a,<0.32.0>},...]
}
```

Added by spawn

Added by register, removed by unregister
Specification of register

register_pre(S) ->
    S#state.pids /= [].

register_args(S) ->
    [name(), elements(S#state.pids)].

register(Name, Pid) ->
    erlang:register(Name, Pid).

register_next(S, _, [Name, Pid]) ->
    S#state{regs=S#state.regseqs++ [{'Name', Pid}]}.
DEMO
State machine models

- Conveniently specify the intended behaviour of stateful systems

- QuickCheck CI reports a variety of interesting test cases, and groups them sensibly

- Testing in practice involves
  - Reverse engineering of specifications (yes, really!)
  - Finding and correcting bugs in the code
Doing it for real...
3,000 pages of specifications
20,000 lines of QuickCheck
1,000,000 LOC, 6 suppliers
200 problems
100 problems in the standard
10x shorter test code
Want to try it out?

• Go to https://github.com/hanssv/example_proj
A small example project

- 24 commits
- 1 branch
- 0 releases
- 1 contributor

branch: master

example_proj/ +

Another update to README.md
example_proj

QuickCheck passed

A small example project. The only purpose of this project is to serve as a demonstrator for QuickCheck (http://quickcheck-ci.com/). That means that the interesting parts of this project is not the code, nor the properties. Instead, the interesting bits are the configuration file (./eqc_ci), the licence file (./EQC_CI_LICENCE.txt), and this readme file (./README.md).

/Hans
hanssv/example_proj build #3

Build info

<table>
<thead>
<tr>
<th>Module</th>
<th>Result</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>locker</td>
<td>{0,0,0,0}</td>
<td>0.00s</td>
</tr>
<tr>
<td>locker_eqc</td>
<td>{1,0,0,0}</td>
<td>1.25s</td>
</tr>
<tr>
<td>myqueue</td>
<td>{0,0,0,0}</td>
<td>0.00s</td>
</tr>
</tbody>
</table>