Software Testing with QuickCheck

Lecture 1
Properties and Generators
Testing

• How do we know software works?
  – We test it!

• “lists:delete removes an element from a list”

  4> lists:delete(2,[1,2,3]).
     [1,3]
  5> lists:delete(4,[1,2,3]).
     [1,2,3]

• ... seems to work!
Automated Testing

- Testing accounts for ~50% of software cost!
- Automate it... write once, run often

```erlang
present_present_test() ->
lists:delete(2, [1, 2, 3]) == [1, 3].

present_absent_test() ->
lists:delete(4, [1, 2, 3]) == [1, 2, 3].
```

Lots of code (35%)

Boring

Easy to miss something
Property-based testing

• Generalise test cases

\[ \forall \{I,L\} \in \text{int()} \times \text{list(int())} \]

prop_delete() ->
  ?FORALL({I,L},
    {\text{int()}, \text{list(int())}},
    \text{not lists:member}(I,
      \text{lists:delete}(I,L))
  ).

21> eqc:quickcheck(examples:prop_delete()).

....................
....................
OK, passed 100 tests
Properties

- We test directly against a formal specification

\(?\text{FORALL}(N, \text{int}(), N \times N \geq 0)\)
More tests...

29> eqc:quickcheck(eqc:numtests(1000, examples:prop_delete())).

.........................................................
.........................................................
.........................................................
.........................................................
.........................................................
.........................................................
...Failed! After 346 tests.
{2,[-7,-13,-15,2,2]}
Shrinking (1 times)
{2,[2,2]}
false

c.f. ?FORALL({I,L},...,...)
The fault explained

```erlang
code(
  lists:member(2, [2,2]),
  lists:delete(2, [2,2]),

  not true
)
```

false
Properties with preconditions

- The property holds provided \( L \) contains no duplicates

\[
\text{no_duplicates}(L) \rightarrow \\
\text{lists:usort}(L) \equiv \text{lists:sort}(L).
\]

\[
\text{prop_delete}() \rightarrow \\
?\text{FORALL}({I,L}, \\
\{\text{int()},\text{list(int())}\}, \\
?\text{?IMPLIES} (\text{no_duplicates}(L), \\
\text{not lists:member}(I,\text{lists:delete}(I,L))))).
\]

39> \text{eqc:quickcheck}(\text{examples:prop_delete}()).

.................x..............x.................x.x
......xx..............x..........xx..............x.x
x...........x..
OK, passed 100 tests

Skipped tests
Custom generators

• Why not generate lists without duplicates in the first place?

```prolog
ulist(Elem) ->
  ?LET(L, list(Elem),
    lists:usort(L)).
```

• Use as `?FORALL(L, ulist(Elem), ...)`
• Generators are an abstract data type with `?LET` for sequencing

First: generate a list \( L \)
Then: sort it and remove duplicates
Why was the error hard to find?

- \( I \in \text{int()} \)
- \( L \in \text{list(int())} \)

What is the probability that \( I \) occurs in \( L \)—twice?

```prolog
prop_delete() ->
  ?FORALL({I,L},
    {int(),list(int())},
    collect(lists:member(I,L),
      not lists:member(I,lists:delete(I,L))).
```

34> eqc:quickcheck(examples:prop_delete()).

OK, passed 100 tests
88% false
12% true
true

Usually \( I \) doesn’t even occur once
Generate relevant tests

- Ensure that \( I \) is a member of \( L \)
  - Generate it from \( L \)

```prolog
prop_delete_2() ->
    ?FORALL(L,list(int()),
        ?IMPLIES(L /= [],
            ?FORALL(I,elements(L),
                not lists:member(I,lists:delete(I,L))))).
```

45> eqc:quickcheck(examples:prop_delete_2()).

...x.x.x.xxx...x.x...x....x........xx.....Failed! After 28 tests.

[-8,0,7,0]
0
Shrinking...(3 times)

[0,0]
0
Generate relevant tests

- Ensure that $I$ is a member of $L$
  - Only works if $L$ is non-empty
  - $\exists$SUCHTHAT like $\exists$IMPLIES but for generators
  - $\forall[G \neq []]$, non_empty$(G) \rightarrow \exists[X,G,X \neq []]$

prop_delete_2() ->
    $\forall[L,\text{non}_\text{empty}$(list(int()))],$
    $\forall[I,\text{elements}(L),$
    not lists:member(I,lists:delete(I,L))).
Documenting misconceptions

• Useful to record that an expected property is not true

prop_delete_misconception() ->
  fails(
    ?FORALL(L,non_empty(list(int())),
      ?FORALL(I,elements(L),
        not lists:member(I,lists:delete(I,L))))).

49> eqc:quickcheck(examples:prop_delete_misconception()).
...............OK, failed as expected. After 19 tests.

Good distribution ensures we falsify the property quickly
Remember!

• We test against a formal specification!
  – Often it is the *specification* which is wrong!

• We don’t see the test data!
  – 100 passing tests can give a false sense of security

• Collect statistics!
  – Ensure a good test case distribution
Exercises