



Software Testing with QuickCheck

Lecture 2
Symbolic Test Cases

Testing Data Structure Libraries

- **dict**—purely functional key-value store
 - `new()`—create empty dict
 - `store(Key,Val,Dict)`
 - `fetch(Key,Dict)`
 - ...
- Complex representation... just test the API
 - “black box testing”
 - In contrast to testing e.g. dict invariants

A Simple Property



- Perhaps the keys are unique...

```
prop_unique_keys() ->
  ?FORALL(D, dict(),
    no_duplicates(dict:fetch_keys(D))).
```

- Cannot test this without a *generator for dicts*

Generating dicts



- Black box testing → *use the API to create dicts*

```
dict() ->
  ?LAZY (
    oneof ([dict:new(),
           ?LET ({K,V,D}, {key(), value(), dict()},
                dict:store(K,V,D))])
  ).
  oneof ([int(), real(), atom()]).
```

Ac

Constant
time

recursion!

- We simulate lazy evaluation where we need it

Let's test it!



```

Erlang
File Edit Options View Help
-----
9> eqc:quickcheck(eqc:numtests(10000,examples2:prop_unique_keys())).
.....
.....Failed! After 451 tests.
{dict,2,16,16,8,80,48,
 {[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]},
 {[[[0.0|0.0],[0|0.0]],[],[],[],[],[],[],[],[],[],[],[],[],[],[]]}},
false
10>
  
```



- **Black box testing:** we need to know how this was constructed!

Symbolic Test Cases



```

dict () ->
  ?LAZY (oneof ( [
    {call,dict,new, []},
    {call,dict,store, [key (), value (), dict () ] }
  ] ) ) .
  
```

- {call,M,F,Args} represents a function call
M:F(...Args...) symbolically.

```

prop_unique_keys () ->
  ?FORALL (D, dict (),
    no_duplicates (dict:fetch_keys (eval (D) ) ) ) .
  
```

Test case is now *symbolic*

- eval(X) evaluates symbolic calls in a term X

dict() with Shrinking



```
dict() ->
?LAZY(oneof(
  [{call,dict,new,[]},
   ?LETSHRINK([D],[dict()],
              {call,dict,store,[key(),value(),D]})])).
```

- ?LETSHRINK makes shrinking recursive types very easy

Let's test and shrink!



```
Erlang
File Edit Options View Help
20> eqc:quickcheck(eqc:numtests(10000,examples2:prop_unique_keys())).
...Failed! After 4 tests.
{call,dict,store,
 {call,dict,store,
  [-1.0,b,
   {call,dict,store,
    [-1,-1.0,
     {call,dict,store,
      [c,c,
       {call,dict,store,
        [0,-1,
         {call,dict,store,
          [1,0,{call,dict,new,[]}]}}]}]}]}]}
Shrinking....(5 times)
{call,dict,store,[-1.0,a,{call,dict,store,[-1,0.0,{call,dict,new,[]}]}}
false
```

Nice shrinking result

-1 and -1.0, eh?

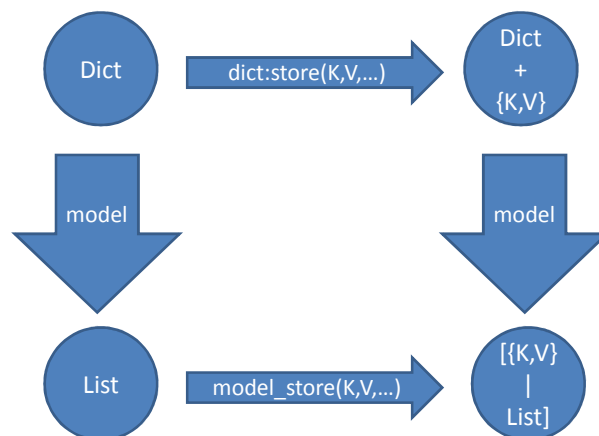
Testing vs. an Abstract Model

- How *should* **dict** operations behave?
 - The “same” as a list of key-value pairs
 - Use the **proplists** library as a reference
- Make comparisons in the “model” world

```
model (Dict) ->
  dict:to_list (Dict) .
```

Returns list of key-value pairs

Commuting Diagrams



Hoare: *Proof of Correctness of Data Representations*, 1972

Testing store



```
prop_store() ->
  ?FORALL({K,V,D},
    {key(),value(),dict()}),
  begin
    Dict = eval(D),
    model(dict:store(K,V,Dict)) ==
      model_store(K,V,model(Dict))
```

```
Erlang
File Edit Options View Help
29> eqc:quickcheck(eqc:numtests(10000,examples2:prop_store())).
..Failed! After 3 tests.
{0,0.0,{call,dict,store,[0,0.0,{call,dict,new,[]}]}}
false
30> █
```

Next Steps...



- Write similar properties for *all* the dict operations
- Extend the dict generator to use *all* the dict-returning operations
 - Each property tests *many* operations
- ...and, of course, correct the specification!

Debugging properties



- *Why* is a property false?
 - We need more information!

```

Erlang
File Edit Options View Help
31> eqc:quickcheck(eqc:numtests(10000,examples2:prop_store())).
Failed? After 1 tests.
{a,0,
 {call,dict,store,
  [a,undefined,
   {call,dict,store,
    [0,b,{call,dict,store,[undefined,0,{call,dict,new,[]}]}}}]}
 [{0,b},{a,0},{undefined,0}] /= [{a,0},{0,b},{a,undefined},{undefined,0}]
Shrinking..(3 times)
{a,0,{call,dict,store,[a,a,{call,dict,new,[]}]}}
[{a,0}] /= [{a,0},{a,a}]
false
32>

```



Exercises