A Glympse of Testing Research

Guille Polito — CRIStAL — CNRS — RMoD - Inria Lille
@guillep

The problem of tests

- Do we have too few?
- Do we have too many?
- Do they take long to run?
- Are they redundant?
- Are they good tests? Do they catch bugs?

Automatic Test Generation

When we have too few tests

- How do we automatically generate tests?
- How do we ensure automatically generated tests are *good*?
  - Do they cover different code regions/branches/paths?
  - Do they discover bugs?
- The Oracle Sub-problem:
  - How do we determine what is the expected output of a generated test?

Random Test Generation

Example: Haskell's QuickCheck

- Developers define function *properties*
- Example, we say inserting into a sorted list should keep it sorted

```
InsertIsSorted x xs = ordered xs => ordered (insert x xs)
```

- *Generators* build random input values and validate that properties hold
- Pros: Lightweight and simple solution
- Cons: Expensive to explore border cases

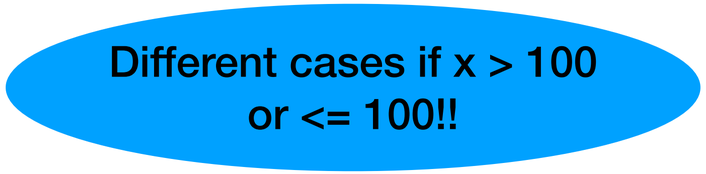
Claessen Koen, Hughes John. QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs. ICFP '00

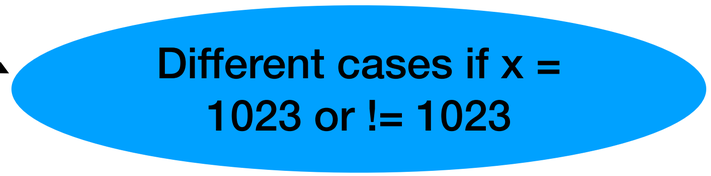
Directed Random Test Generation

Example: DART, CUTE

- Idea: Guide test generation by looking at the implementation

```
int f(int x, int y){
  if (x > 100){
    if (y == 1023){
      segfault(!!)
    }
  }
}
```





Godefroid et al. DART: Directed Automated Random Testing. PLDI' 05
Set et al. CUTE: a concolic unit testing engine for C. FSE'05

Directed Random Test Generation

Implementation: Concolic testing

- **Conc**rete + Symb**olic** execution
- Goal: automatically discover *all* execution *paths*

```
int f(int x, int y){
  if (x > 100){
    if (y == 1023){
      segfault(!!)
    }
  }
}
```

x	y	constraints	next?

Godefroid et al. DART: Directed Automated Random Testing. PLDI' 05
Set et al. CUTE: a concolic unit testing engine for C. FSE'05

Directed Random Test Generation

Implementation: Concolic testing

- **Conc**rete + Symb**olic** execution
- Goal: automatically discover *all* execution *paths*

```
int f(int x, int y){
  if (x > 100){
    if (y == 1023){
      segfault(!!)
    }
  }
}
```

x	y	constraints	next?
0	0	x <= 100	

Godefroid et al. DART: Directed Automated Random Testing. PLDI' 05
Set et al. CUTE: a concolic unit testing engine for C. FSE'05

Directed Random Test Generation

Implementation: Concolic testing

- **Conc**rete + Symb**olic** execution
- Goal: automatically discover *all* execution *paths*

```
int f(int x, int y){
  if (x > 100){
    if (y == 1023){
      segfault(!!)
    }
  }
}
```

x	y	constraints	next?
0	0	x <= 100	x > 100

Godefroid et al. DART: Directed Automated Random Testing. PLDI' 05
Set et al. CUTE: a concolic unit testing engine for C. FSE'05

Directed Random Test Generation

Implementation: Concolic testing

- **Conc**rete + Symb**olic** execution
- Goal: automatically discover *all* execution *paths*

```
int f(int x, int y){
  if (x > 100){
    if (y == 1023){
      segfault(!!)
    }
  }
}
```

x	y	constraints	next?
0	0	x <= 100	x > 100
101	0	x > 100, y != 1023	

Godefroid et al. DART: Directed Automated Random Testing. PLDI' 05
Set et al. CUTE: a concolic unit testing engine for C. FSE'05

Directed Random Test Generation

Implementation: Concolic testing

- **Conc**rete + Symb**olic** execution
- Goal: automatically discover *all* execution *paths*

```
int f(int x, int y){
  if (x > 100){
    if (y == 1023){
      segfault(!!)
    }
  }
}
```

x	y	constraints	next?
0	0	x <= 100	x > 100
101	0	x > 100, y != 1023	x > 100, y == 1023

Godefroid et al. DART: Directed Automated Random Testing. PLDI' 05
Set et al. CUTE: a concolic unit testing engine for C. FSE'05

Directed Random Test Generation

Implementation: Concolic testing

- **Conc**rete + Symb**olic** execution
- Goal: automatically explore *all* execution *paths*

```
int f(int x, int y){
  if (x > 100){
    if (y == 1023){
      segfault(!!)
    }
  }
}
```

x	y	constraints	next?
0	0	x <= 100	x > 100
101	0	x > 100, y != 1023	x > 100, y == 1023
101	1023	x > 100, y != 1023	finished!

Godefroid et al. DART: Directed Automated Random Testing. PLDI' 05
Set et al. CUTE: a concolic unit testing engine for C. FSE'05

Test Case Minimisation and Prioritisation

When we have **too many** tests

- Test Minimisation: How do we discover and eliminate redundant tests?
- Test Selection: How do we choose a subset of relevant tests to run?
- Test Prioritisation: What is the ideal order of running tests?

Yoo Shin, Harman Mark. Regression Testing Minimisation, Selection and Prioritisation: a Survey. STVR'07

Test Case Selection in Industry

- Case Study in WorldLine
- Large test suite that takes hours
- Static approaches: build an application model and find dependencies
- Dynamic approaches: execute the tests and find runtime dependencies

- Dynamic approaches are more accurate than static ones
  - Polymorphism, dynamic binding and reflection harm dependency analysis

Blondeau et al. Test Case Selection in Industry: an Analysis of Issues Related to Static Approaches. Software Qual J **25** 2016

Test Case Validation

Who watches the watchmen?

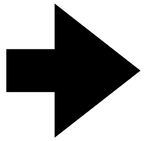
- Tests must detect bugs
  - validate results
  - validate state modifications
  - validate exception cases

- How can we detect *Weak* tests?

Mutation Testing

- Insert code modifications
- Tests should break!
- Otherwise, the modified functionality is not tested

```
if (a && b){
  c = 1;
} else {
  c = 17;
}
```



```
if (a || b){
  c = 1;
} else {
  c = 17;
}
```

Tim A. Budd, Mutation Analysis of Program Test Data. PhD thesis, Yale University New Haven CT, 1980

Rotten Green Tests

- Tests may have assertions, be green
  - And still not execute the assertion!
- Otherwise, the modified functionality is not tested
- Conditional code not executing a branch
- Iterating over an empty collection

```
class RottenTest {
    method testABC {
     if (false) then {self.assert(x)}
    }

}
```

Delplanque et al., Rotten Green Tests. ICSE'19

Conclusion

- Ensuring the validity and conformity of software systems is
  - complex
  - an active area of research
  - an interesting area of research

- Solutions often mix automatic code modification, static and dynamic analyses. They can be used different contexts such as standard industrial setups or compilers