

Beck, Calvin   Huang, Jiani   Li, Yishuai

November 29, 2018

# What is this talk about?

What is this talk about?

FuzzChick!

What is this talk about?

# FuzzChick!

What is FuzzChick...?

What is this talk about?

# FuzzChick!

What is FuzzChick...?

FuzzChick is an experiment to improve QuickChick using ideas from fuzzing, such as AFL [3].

# What is this talk about?

## FuzzChick!

What is FuzzChick...?

FuzzChick is an experiment to improve QuickChick using ideas from fuzzing, such as AFL [3].

We'll come back to this...

# QuickChick: A Brief Review

QuickChick is a properties based testing framework for Coq.

- You build (or derive) generators for data types.
- Using those generators you can feed data into test cases.
- These test cases can be any arbitrary predicate.

# QuickChick: Pros and Cons

So what's great about QuickChick?

- Relatively easy to build / derive generators.
- Can generate lots of tests for specific properties automatically.



# QuickChick: Pros and Cons

So what's great about QuickChick?

- Relatively easy to build / derive generators.
- Can generate lots of tests for specific properties automatically.

What's not so great about QuickChick?

# QuickChick: Pros and Cons

So what's great about QuickChick?

- Relatively easy to build / derive generators.
- Can generate lots of tests for specific properties automatically.

What's not so great about QuickChick?

- Getting *good* generators can be hard!

# What makes a good generator?

The basic idea of what makes a generator “good” can vary somewhat based on the context.

# What makes a good generator?

The basic idea of what makes a generator “good” can vary somewhat based on the context.

In general you want good coverage. How can you achieve that with minimal work?

# Finally, FuzzChick!

FuzzChick uses AFL to make the choices between constructors for building data types for tests.

# Finally, FuzzChick!

FuzzChick uses AFL to make the choices between constructors for building data types for tests.

Why is this good?

# FuzzChick Intuition

AFL uses DSE to attempt to get good coverage while fuzzing...

# FuzzChick Intuition

AFL uses DSE to attempt to get good coverage while fuzzing...  
Maybe we can utilize AFL's smarts to achieve better test coverage.



# QuickChick: Now With Coverage!

We instrumented QuickChick using `bisect_ppx` to get coverage estimates!

# QuickChick: Now With Coverage!

We instrumented QuickChick using `bisect_ppx` to get coverage estimates!

This *mostly* went smoothly...

Compiling with absolute paths cause an infinite loop #180



Chobbes opened this issue 2 days ago · 7 comments

# QuickChick: Now With Coverage!

We instrumented QuickChick using `bisect_ppx` to get coverage estimates!

This *mostly* went smoothly...

Compiling with absolute paths cause an infinite loop #180

 **Closed** Chobbes opened this issue 2 days ago · 7 comments

Maintainer fixed this issue promptly, which was *awesome*!

## QuickChick with coverage... Still needs some polish

QuickChick with coverage is cool, but it still does need some polish.

## QuickChick with coverage... Still needs some polish

QuickChick with coverage is cool, but it still does need some polish.

- Includes a lot of excessive extracted Coq code.

# QuickChick with coverage... Still needs some polish

QuickChick with coverage is cool, but it still does need some polish.

- Includes a lot of excessive extracted Coq code.
  - ▶ Like... All of QuickChick :(.
    - ▶ So, the percentages are a little off.

# QuickChick with coverage... Still needs some polish

QuickChick with coverage is cool, but it still does need some polish.

- Includes a lot of excessive extracted Coq code.
  - ▶ Like... All of QuickChick :(.
    - ▶ So, the percentages are a little off.
- QuickChick generates “random” files for each test, and the names aren’t all that useful
  - ▶ Modified QuickChick to include test case in name, but still not ideal.

# QuickChick with coverage... Still needs some polish

QuickChick with coverage is cool, but it still does need some polish.

- Includes a lot of excessive extracted Coq code.
  - ▶ Like... All of QuickChick :(.
    - ▶ So, the percentages are a little off.
- QuickChick generates “random” files for each test, and the names aren’t all that useful
  - ▶ Modified QuickChick to include test case in name, but still not ideal.

But...



# QuickChick with coverage... Still needs some polish

QuickChick with coverage is cool, but it still does need some polish.

- Includes a lot of excessive extracted Coq code.
  - ▶ Like... All of QuickChick :(.
    - ▶ So, the percentages are a little off.
- QuickChick generates “random” files for each test, and the names aren’t all that useful
  - ▶ Modified QuickChick to include test case in name, but still not ideal.













But...

# It works! We can measure stuff!

# QuickChick Coverage: ifc-basic

Coverage with QuickChick in the ifc-basic example:

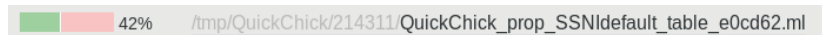
## Coverage report 44.33%

		42%	<a href="#">/tmp/QuickChick/214311/QuickChick_prop_SSNldefault_table_e0cd62.ml</a>
		46%	<a href="#">/tmp/QuickChick/214315/QuickChick_prop_SSNl_deriveddefault_table_65753a.ml</a>
		43%	<a href="#">/tmp/QuickChick/214320/QuickChick_prop_MSNIdefault_table_31e78d.ml</a>
		43%	<a href="#">/tmp/QuickChick/214324/QuickChick_myArgs_71227b.ml</a>
		44%	<a href="#">/tmp/QuickChick/214327/QuickChick_myArgs_81c382.ml</a>
		49%	<a href="#">/tmp/QuickChick/214330/QuickChick_myArgs_100552.ml</a>

Generated on 2018-11-28 21:43:33 by *Bisect\_ppx* 1.3.4

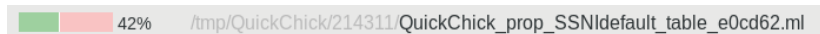
# QuickChick vs FuzzChick: ifc-basic

QuickChick:

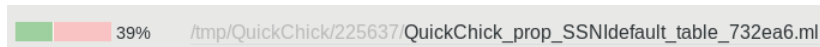


# QuickChick vs FuzzChick: ifc-basic

QuickChick:



FuzzChick:



For some reason it seems that FuzzChick actually gets worse coverage than QuickChick on this test case... At least in the time I let it run (I'm not terribly patient)

# Why Worse Coverage?

- Just need to let it run longer?
  - ▶ AFL needs a while to “warm up”?

# Why Worse Coverage?

- Just need to let it run longer?
  - ▶ AFL needs a while to “warm up”?
- QuickChick test already managed to get good coverage in this instance, so fuzzing doesn't give us much on top of it?
  - ▶ Hard to tell what “good coverage” is due to the extraneous code extracted by QuickChick.

# Why Worse Coverage?

- Just need to let it run longer?
  - ▶ AFL needs a while to “warm up”?
- QuickChick test already managed to get good coverage in this instance, so fuzzing doesn't give us much on top of it?
  - ▶ Hard to tell what “good coverage” is due to the extraneous code extracted by QuickChick.
- Something's not instrumented correctly?

# Why Worse Coverage?

- Just need to let it run longer?
  - ▶ AFL needs a while to “warm up”?
- QuickChick test already managed to get good coverage in this instance, so fuzzing doesn't give us much on top of it?
  - ▶ Hard to tell what “good coverage” is due to the extraneous code extracted by QuickChick.
- Something's not instrumented correctly?
- This test case, for whatever reason, is fuzzer unfriendly?
  - ▶ Maybe extracted Coq could be fuzzer unfriendly? Lots of inefficient data types like `nat` (basically a linked list whose length represents a number).
  - ▶ Could result in excessively long paths and hard to solve predicates for DSE?
  - ▶ Not sure that having pointers everywhere would be AFL's strength...



# Some Further Coverage Testing...

Test case:

```
Extract Constant unlikely_branch =>
" fun i ->
  if (0 < i)
  then if (i mod 100 == 0)
    then if (i mod 1000 == 0)
      then if (i mod 10000 == 0)
        then if (i mod 100000 == 0)
          then if (i mod 1000000 == 0)
            then if (i < 1000001)
              then 42
            else 0
          else 0
        else 0
      else 0
    else 0
  else 0
else 0
".

Definition always_zero := forAll (choose (0%Z, 9999999%Z)) (fun n =>
  unlikely_branch n =? 0).
```

# Some Further Coverage Testing...

Test case:

```
Extract Constant unlikely_branch =>
" fun i ->
  if (0 < i)
  then if (i mod 100 == 0)
    then if (i mod 1000 == 0)
      then if (i mod 10000 == 0)
        then if (i mod 100000 == 0)
          then if (i mod 1000000 == 0)
            then if (i < 1000001)
              then 42
              else 0
            else 0
          else 0
        else 0
      else 0
    else 0
  else 0
else 0
".

Definition always_zero := forAll (choose (0%Z, 9999999%Z)) (fun n =>
  unlikely_branch n =? 0).
```

Trying to give AFL a good chance to find the failing branch...

# Results

In the equivalent C code AFL does quite well...

# Results

In the equivalent C code AFL does quite well...

QuickChick:

```
then if (i mod 1000000 == 0)                then if (i < 1000001)
```

FuzzChick:

```
then if (i mod 10000 == 0)                then if (i mod 100000 == 0)                then if (i mod 1
```

# Results

In the equivalent C code AFL does quite well...

QuickChick:

```
then if (i mod 1000000 == 0)                then if (i < 1000001)
```

FuzzChick:

```
then if (i mod 10000 == 0)                then if (i mod 100000 == 0)                then if (i mod 1
```

Here not so much? FuzzChick doesn't make it as far...

# Results

In the equivalent C code AFL does quite well...

QuickChick:

```
then if (i mod 1000000 == 0)                then if (i < 1000001)
```

FuzzChick:

```
then if (i mod 10000 == 0)                then if (i mod 100000 == 0)                then if (i mod 1
```

Here not so much? FuzzChick doesn't make it as far...

Well, in fairness, it does eventually, but it takes a good 30 minutes. QuickChick was much faster.

# Results

In the equivalent C code AFL does quite well...

QuickChick:

```
then if (i mod 1000000 == 0)                then if (i < 10000001)
```

FuzzChick:

```
then if (i mod 10000 == 0)                then if (i mod 100000 == 0)                then if (i mod 1
```

Here not so much? FuzzChick doesn't make it as far...

Well, in fairness, it does eventually, but it takes a good 30 minutes. QuickChick was much faster.

Suggests maybe the extracted ocaml is harder for AFL to analyze?  
The C branches were discovered very quickly by AFL.

# Honourable Mentions: Some Other Stuff We Did



# Honourable Mentions: Some Other Stuff We Did

- Honggfuzz!

# Honourable Mentions: Some Other Stuff We Did

## ■ Honggfuzz!

- ▶ Finally got this working...
- ▶ Didn't really work out because it takes a long time to find bugs by fuzzing.
- ▶ Decided it wasn't really a great comparison to FuzzChick which is a properties based testing tool anyway.
- ▶ Some useful scripts / documentation to get this running in our git repo. [1]

# Honourable Mentions: Some Other Stuff We Did

## ■ Honggfuzz!

- ▶ Finally got this working...
- ▶ Didn't really work out because it takes a long time to find bugs by fuzzing.
- ▶ Decided it wasn't really a great comparison to FuzzChick which is a properties based testing tool anyway.
- ▶ Some useful scripts / documentation to get this running in our git repo. [1]

## ■ Plain AFL!

# Honourable Mentions: Some Other Stuff We Did

## ■ Honggfuzz!

- ▶ Finally got this working...
- ▶ Didn't really work out because it takes a long time to find bugs by fuzzing.
- ▶ Decided it wasn't really a great comparison to FuzzChick which is a properties based testing tool anyway.
- ▶ Some useful scripts / documentation to get this running in our git repo. [1]




## ■ Plain AFL!

- ▶ Similar story to Honggfuzz.

Conclusion! Questions?

Whew! Questions?

# References

-  Calvin Beck, Jiani Huang, and Yishuai Li. *Quick700*. 2018. URL: <https://github.com/Quick700/Quick700> (visited on 11/29/2018).
-  Leonidas Lampropoulos, Zoe Paraskevopoulou, and Benjamin C Pierce. “Generating Good Generators for Inductive Relations”. In: ().
-  Michal Zalewski. *AFL*. URL: <http://lcamtuf.coredump.cx/afl/> (visited on 11/29/2018).

These are all good resources! You should look at them!