SpecCheck: Monadic specification and property based testing of typed communication protocols

Maximilian Algehed

Chalmers University of Technology

algehed@chalmers.se

January 19, 2017
The problem

Building correct distributed applications

**Comic Text**

Alice sends a message to Bob saying to meet her somewhere.

Uh huh.

But Eve sees it, too, and goes to the place. With you so far.

Bob is delayed, and Alice and Eve meet.

Yeah?

I've discovered a way to get computer scientists to listen to any boring story.
The problem, simplified

Building correct client-server applications
Building correct client-server applications

Client
- Specification
- Test-suite
- Update the specification
- Update the test-suite
The problem, simplified

Building correct client-server applications

Client
- Specification
- Test-suite
- Update the specification
- Update the test-suite

Server
- Specification
- Test-suite
- Update the specification
- Update the test-suite
SpecCheck, the "elevator pitch"

- Write *one* specification
- Property based testing
Demo!
The interface

send :: (a <: t) => Predicate a -> Spec t a
get :: (a <: t) => Predicate a -> Spec t a
choose :: (Eq a, a <: t) => [a] -> Spec t a
branch :: (Eq a, a <: t) => [a] -> Spec t a
dual :: Spec t a -> Spec t a
How does it work?

```
data Sop m c where
  Send    :: a :<: c (...) =>
             Predicate a -> (a -> m IO (Sop m c)) -> Sop m c
  Get     :: a :<: c (...) =>
             Predicate a -> (a -> m IO (Sop m c)) -> Sop m c
  End     :: Sop m c

type Predicate a = (Gen a, a -> Bool)

type SpecT m t a = ContT (Sop m t) (m IO) a

type Spec t a = forall m. SpecT m t a

type SpecS st t a = SpecT (StateT st) t a
```
dual (get p) = send p
dual (send p) = get p
dual stop = stop
dual (m >>= f) = dual m >>= (dual . f)
dual (return a) = return a
Duality in action!

game = do
    move <- send validMove
    gameOver <- updateGameState move
    if gameOver then
        stop
    else
        dual game
Shrinking
Shrinking

- Number of messages
- Size of the messages

Sent: -14
Sent: "another"
Sent: -2
Sent: "request"
Got: []

shrinks to

Sent: 0
Sent: "request"
Got: []
Main*> let prop_reverse xs ys =
        reverse (xs ++ ys) == reverse xs ++ reverse ys
Main*> quickCheck prop_reverse
*** Failed! Falsifiable (...):
  [0]
  [1]
data Interaction c = Got c
               | Sent c

type Log c = [Interaction c]

Sent: -14
Sent: "another"
Sent: -2
Sent: "request"
Got: []
How does it work?

- Can’t do anything about Got
- Can shrink Sent values!
- What to do about choose?
How does it work?

- Can’t do anything about `Got` values!
- Can shrink `Send` values!
- What to do about `choose`?

The naïve algorithm

- Try to follow the Log
- Default to random
- If the resulting trace is longer than the current trace, discard it.
- Repeat lots of times...
Revisiting predicates

type Predicate a = (a -> Bool, Gen a, a -> Gen a)
Demo!
Inconsistent specifications
Inconsistent specifications

inconsistent = do
  n1 <- send posNum
  n2 <- send negNum
  get (inRange (n1, n2))

Failed with inability
to generate: inRange (11, -12)
In:
  ---
    Sent: 11
    Sent: -12
How does it work?

In theory:
- Run specification against itself (duality!)
- Detect when a predicate is unsat

In practise:
- Can't detect partiality in Gen...
  (Maybe Idris?)
- Timeout!
- Shrinking is a problem
How does it work?

In theory:
- Run specification against itself (duality!)
- Detect when a predicate is unsat

In practise:
- Can’t detect partiality in Gen... (Maybe Idris?)
How does it work?

In theory:
- Run specification against itself (duality!)
- Detect when a predicate is unsat

In practise:
- Can’t detect partiality in Gen... (Maybe Idris?)
- Timeout!
- Shrinking is a problem
Laziness works well for some things, less well for other things...

Current solution: ad-hoc...

Better solution (?): Logic programming
Some other things

- Automatically create examples of communication
- Specifications parameterized by bugs in the implementation
Summary

- Shrinking is a hard problem
- Duality $\rightarrow$ we only need one specification
- Duality $\rightarrow$ find inconsistencies
Future work

- Asynchronous protocols
- Multiparty communication
- Protocol stacks
- A language for writing generator predicate pairs in Haskell
Questions?