Secure Your Programming Future!

David J. Pearce

School of Engineering and Computer Science
Victoria University of Wellington

@WhileyDave
http://whiley.org
http://github.com/Whiley
Background
Verification: Who Cares?

Heartbleed bug: what do you actually need to do to stay secure?

Catastrophic bug leaves thousands of sites vulnerable, but what exactly is Heartbleed and how does it affect me?

Rebooting computers will overcome glitch that could cut all power.

Air New Zealand says it will comply with the directive to undertake a “power cycle” on its Dreamliners every three months. Photo / Evert Dirkx
“A verifying compiler uses automated mathematical and logical reasoning methods to check the correctness of the programs that it compiles”

—Hoare’03
Whiley
Overview: What is Whiley?

function max(int x, int y) → (int z)
// result must be one of the arguments
ensures x == z || y == z
// result must be greater-or-equal than arguments
ensures x <= z && y <= z:
    ...

- A language designed specifically to simplify verifying software
- Several trade offs e.g. performance for verifiability
  - Unbounded Arithmetic, value semantics, etc
- Goal: to statically verify functions meet their specifications
History of Whiley

- 2009 — **Initial** version of Whiley released (GPL Licence)
- 2010 — **GitHub** repository and [http://whiley.org](http://whiley.org) go live
- 2010 — **Version 0.3.0** released (BSD Licence)
- 2016 — **Version 0.4.0** released
- 2017 — **Version 0.4.1** released
Demo Time...
Example: \( \text{max}(\text{int}[]) \)

// Returns index of largest item in array

function max(int[] items) \( \rightarrow \) (int r)
Diagram!

Minesweeper (in Whiley)
How does it work?
Verification: How does it work?

```plaintext
function abs(int x) => (int r)
// return value cannot be negative
ensures r >= 0:
    //
    if x >= 0:
        return x
    else:
        return -x
```

- To verify above function, compiler generates verification conditions
- Verification conditions are (roughly) first-order logic formulas
Verification: Verification Condition Generation

If $x \geq 0$

- Return $x$
- Return $-x$

Assumptions

$x \geq 0 \rightarrow x \geq 0$

Assertion

$x < 0 \rightarrow -x \geq 0$
Verification: Assertion Language

- Whiley compiler emits verification conditions in **assertion language**

```plaintext
assert:
   forall (int x):
      x >= 0 ==> x >= 0

assert:
   forall (int x):
      x < 0 ==> -x >= 0
```

- Verification conditions from `abs()` example shown above

- In principle, can hook up different **automatic theorem provers**
People (so far)

Art
(built C backend, 2012)

Melby
(built GPGPU backend, 2013)

Daniel
(helping with WhileyWeb)

Matt
(compiling for a QuadCopter, 2014)

Henry
(improving verification, 2014)

Sam
(started PhD on Parallelisation, 2014)

Lindsay
(A/Prof, Victoria University)

Mark
(A/Prof, University of Waikato)
http://whiley.org

@WhileyDave
http://github.com/Whiley
Verification: Constrained Types

```
type N is (T x) where e
```

- Above defines **constrained type**

- **Invariant:** for any variable of type \( N \), follows that \( e \) always holds

- Constrained types can **simplify** specifications / invariants

- **Example:** *natural numbers*

```
type nat is (int n) where n >= 0
```
Verification: Structural Typing

```whiley

type nat is (int n) where n >= 0

function cut(int x) → (nat y):
  if x >= 0:
    return x
  else:
    return 0
```

- Variable types in Whiley are **ephemeral** ...

  ... and determined by what is **known** (not what was declared)
function indexOf(int[] items, int item) → (int|null r)
// If integer value returned, must be index of item
ensures r is int ==> items[r] == item
// No element before integer r matches item
ensures r is int ==> all { k in 0..r | items[k] != item }
// If null returned, no matching item
ensures r is null ==> all { k in 0..|items| | items[k] != item }:
  //
  int i = 0
  //
  while i < |items|
  where i >= 0 && i <= |items|
  where all { j in 0..i | items[j] != item }:
    if items[i] == item:
      return i
      i = i + 1
  //
  return null