Whiley: a Platform for Research in Verifying Compilers

David J. Pearce

Victoria University of Wellington

@WhileyDave
http://whiley.org
http://github.com/Whiley
Who Am I?

David Pearce
DavePearce

Victoria University of Wellington
New Zealand
david.pearce@ecs.vuw.ac.nz
http://www.ecs.vuw.ac.nz/~djp
Joined on Nov 16, 2010

13 Followers 10 Starred 0 Following

Organizations
Wy

Popular repositories
- jkit
  Java Compiler Kit (JKit)
- Wyscript
  A derivative of the Whiley language which...
- Automark
  A Simple Web Framework used for Hobb...
- TuttePoly
  Tutte Polynomial Computation
- wytone
  A theorem prover generator

Repositories contributed to
- Whiley/WhileyCompiler
  The Whiley Compiler (WyC)
- Whiley/WhileyDocs
  Various documents relating to the Whiley...
- Whiley/Wycleipse
  Eclipse Plugin for the Whiley Programmi...
- Whiley/WyBench
  Benchmark Suite for Whiley
- Whiley/WhileyCompilerCollection
  A compiler framework for managing the ...

Contributions

Year of contributions
1,982 total
Jun 10 2013 - Jun 10 2014

Longest streak
45 days
October 29 - December 12

Current streak
3 days
June 09 - June 11
Background
“A verifying compiler uses automated mathematical and logical reasoning methods to check the correctness of the programs that it compiles”

–Hoare’03
Verification: Who Cares?

Computers rebooted to tackle Dreamliner glitch

5:00 AM Tuesday May 5, 2015

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 / Stuart Dickinson

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?"

Samuel Gibbs
@SamuelGibbs
Thursday 10 April 2014 14:29 BST

Heartbleed is a catastrophic bug that affects thousands of sites and services across the internet, but what is it, and what do you need to do about it to protect yourself from cybercriminals?
Whiley
Overview: What is Whiley?

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
Welcome to the Future...
Example: \texttt{max(int[])}

// Returns index of largest item in array

\texttt{function max(int[] items) → (int r)}
Diagram!
Minesweeper!

Minesweeper (in Whiley)

(use shift-click to flag squares)

Images
Cyclic Buffer!
How does it work?
Verification: How does it work?

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 >= 0
  return x
else
  return -x
```

Assumptions:
- $x \geq 0 \rightarrow x \geq 0$
- $x < 0 \rightarrow -x \geq 0$

Assertion:
Verification: Assertion Language

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

```
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 **automated 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)
Documents (so far)

The Whiley Language Specification
Version 0.3.23

Getting Started with Whiley (Tutorial)
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)
Verification: Flow Typing

```java
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
    return i + 1
  //
  return null
```