# Reflecting on the Design of Whiley

David J. Pearce

*School of Engineering and Computer Science*
*Victoria University of Wellington*

@WhileyDave

http://whiley.org

http://github.com/Whiley

# Background

# **Verification:** A Challenge for Computer Science

*“A* **verifying compiler** *uses automated mathematical and logical reasoning methods to check the correctness of the programs that it compiles”*

–Hoare’03

# **Verification**: An Idea for the Future?

```
function indexOf([int] items, int item) => (int|null i)
// If return is an int r, then items[r] == item
ensures i is int ==> items[i] == item
// If return is null, then no element x in items where x == item
ensures i is null ==> no { x in items | x == item }
// If return is an int i, then no index j where j < i and items[j] == item
ensures i is int ==> no { j in 0..i | items[j] == item }:
    //
    ...
```

- Can we turn documentation into **code**?
- Can we **statically check** that it is correct and clients adhere to it?
- And, if we can do these things ... is it **useful**?

# Overview

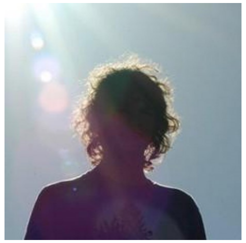
# **People** (so far)

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

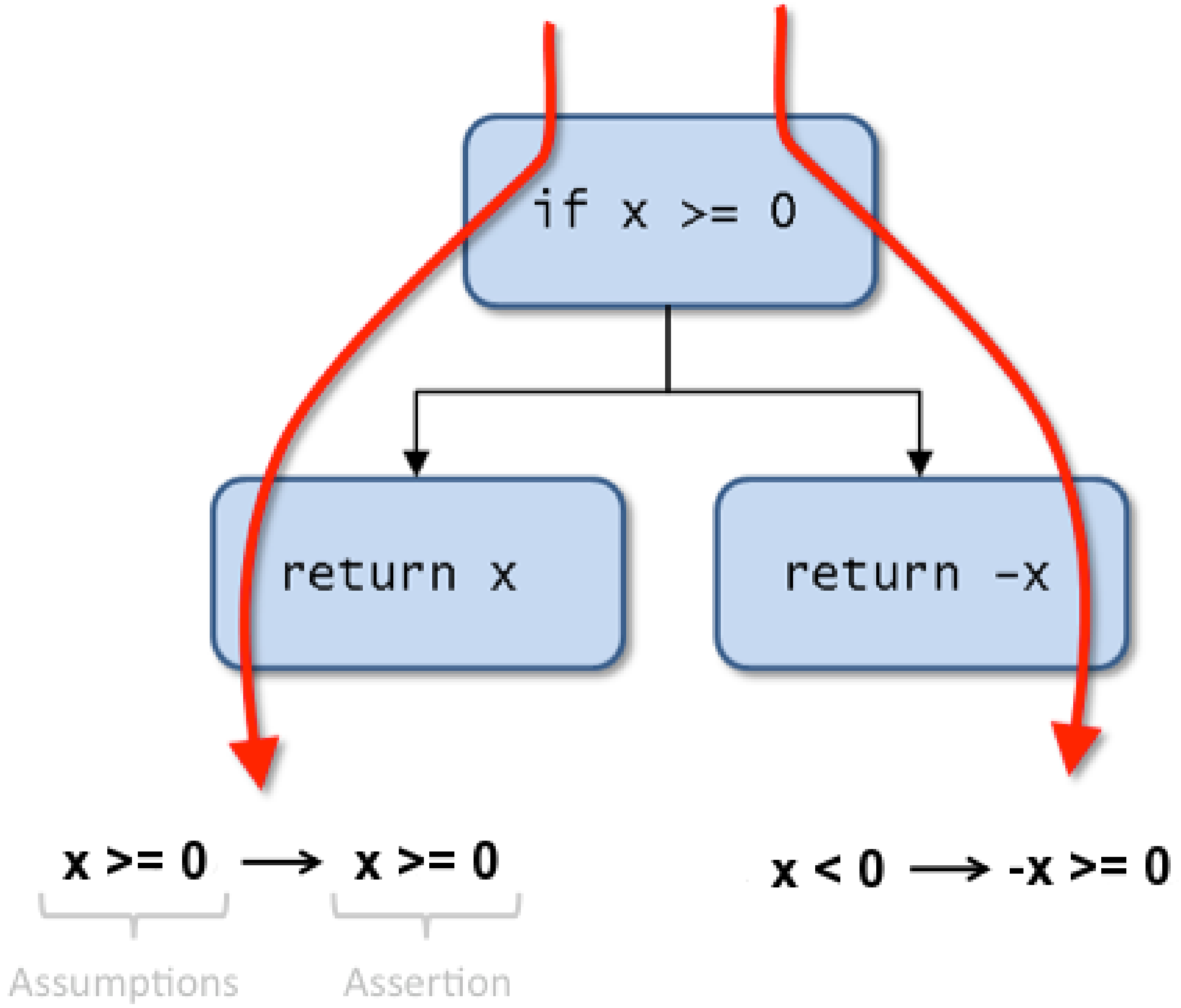
(A/Prof, University of Waikato)

# **Verification**: Overview

```
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**: Example



# Demo

# **Teaching** Whiley ...

**In 2014, Trimester 2 ...**

- Whiley used in **SWEN224** “Formal Foundations of Software”
- SWEN224 covers **reasoning** about programs using **Hoare Logic**
- About **120 students** enrolled in SWEN224
- Whiley helped with teaching **pre-/post-conditions** and **loop invariants**

# **Observations:** Simple Examples went Well

**Needed lots of simple examples like this ...**

```
type Change is { int twentyCents, int fiftyCents }

function getChange(int fiveDollars) => (Change r):
// REQUIRES: one or more fiveDollar Notes to turn into change
// ENSURES: Total return should match the amount given
    ...
```

- **Part 1)** Translate pre-/post-conditions into Whiley.
- **Part 2)** Give an appropriate implementation.

# The **Water Jugs** Example

```
type State is { nat small, nat medium }
where small <= SMALL_SIZE && medium <= MEDIUM_SIZE

function pourSmall2Medium(State jugs) => (State r):
    int amount = MEDIUM_SIZE - jugs.medium
    //
    if amount > jugs.small: // emptying small jug
        jugs.medium = jugs.medium + jugs.small
        jugs.small = amount
    else:                   // filling up medium jug
        jugs.medium = MEDIUM_SIZE
        jugs.small = jugs.small - amount
    return jugs
```

- **Question):** Provide post-condition to ensure water isn't lost.
  *Does the implementation meet this specification?*

# **Observations:** Loop Invariants are Hard

**Here's a "simple" loop invariant example ...**

```
function add([int] xs, int x) => ([int] ret)
// Return value is same size as parameter
ensures |ret| == |xs|:
    //
    int i = 0
    int ghostVar = |xs|
    while i < |xs|:
        xs[i] = xs[i] + x
        i=i + 1
    return xs
```

- **Question)** Add loop invariant so above will verify ...

# **Observations:** Error Messages are Important!

## In particular, Students need to Debug their Code

```
function max(int x, int y) => (int r)
ensures r >= x && r >= y && (r == x || r == y):
    //
    if x > y:
        return x
    else:
        return 0
```

- The **error message** “Postcondition not satisfied” isn’t helpful!
- Students need to **narrow down** which part isn’t satisfied...

# Whiley on **Embedded Systems ...**

**BitCraze Crazyfly QuadCopter:**

- ARM Cortex STM32F103CB @ 72 MHz (128kb flash, 20kb RAM)
- 3-axis MEMs gyros and 3-axis accelerometer
- Operating System is FreeRTOS, with Applications on Top

**The Project:**

- Construct Whiley-to-C Translator
- Port several modules (e.g. for stabilisation) to Whiley
- Go “full circle” by generating C from Whiley code and integrating

# **Observations:** Memory is Tight!

**With only 20Kb of RAM ...**

- Need to **stack allocate** as much as possible
- In CrazyFlie code, structures on **stack** & pointers passed down
- Whiley does support **references**, but no “address of” operator

```
function f():
    [int] aList = [1,2,3]
    int x = g(&aList)
    ...

function g(&[int] list):
    ...
```

- Whiley’s value semantics **does not help** here.
- Need **flow analysis** to determine list size (i.e. because of append)

# **Observations:** Integration is Tricky!

**Whiley does include an FFI...**

```
native function cFun(int x, int y) => int

export function whileyFun(int x) => int:
    return x + 1
```

- FFI consists of **two keywords**: **`export`** and **`native`**
- With **`native`**, can **prototype** C functions in Whiley
- With **`export`**, can make Whiley functions accessible to C
- **Marshalling** data across boundaries is then the problem

# **Observations:** Global Variables!

**Existing Crazyflie code uses global variables...**

- ... to **communicate** between concurrent tasks (one writer, many readers)
- Whiley **does not support** global variables!
- Then, how to integrate with **existing RTOS** ... ?
- **Answer:** Easy, write globals in C and access via FFI!

# Conclusions

- Need **better support** for stack allocated objects...

  ... e.g. some kind of *object lifetime* system (like Rust)

- Need **flow analyses** to bound width of integer variables ...

  ... and compound variables (e.g. lists)

- Need support for **global variables** ... really ??

- Need **much better** error messages...

# http://whiley.org

@WhileyDave
http://github.com/DavePearce/Whiley