How to make Whiley Boogie

Mark Utting
School of Business
University of the Sunshine Coast
Queensland, AUSTRALIA
Email: utting@usc.edu.au

David Pearce, Victoria University of Wellington

SAPLING 21 Nov 2016

Outline

Why Whiley?

Why a WP semantics for Whiley?

The Mapping into Boogie
- Values and Types
- Our Example translated to Boogie

Translation Challenges?

Verification Results

Why Whiley?

Whiley is an attempt to tackle the verifying compiler challenge.
See `http://whiley.org`.

- Designed by David Pearce, VUW, NZ (2009...)
- An open source programming language
- Explicit specifications for functions, methods, data structures
- Verifying compiler statically checks programs against specifications, and prevents runtime errors.
- Aimed at embedded systems (quadcopters etc.), but also general programming.

Whiley Features

These design choices are seen as critical to Whiley's success:

1. Has a clean functional subset, with functions, records, tuples. Distinguishes functions (side-effect free) from methods.
2. Uses unbounded arithmetic.
3. Uses call-by-value. Eliminates aliasing in functions.
4. (Outside scope of this talk) Restricted concurrency model: actor model with no reentrant methods.
5. Computable pre/posts ($\Longrightarrow$ bounded quants, 3-val. logic).

Other features:

- No global variables;
- Method side effects are limited to I/O (library side-effects).

Rich type system:

- Types with constraints: `type pos is (int x) where x > 0`
- And union, intersection, negation types: `int&!pos x`
- Structural subtyping (vs nominal)
- No 'inheritance' between object types
- Flow-sensitive typing. (*a la* dynamic languages)

Whiley Example

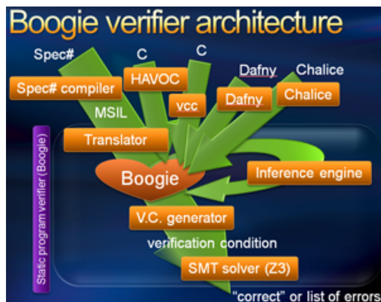
```
import whiley.lang.*
type pos is (int x) where x > 0
type pos5 is (pos[] a) where |a| == 5

function sum(pos5 values) -> (pos result)
ensures result >= 5:
    int i = 0
    int total = 0
    while i < |values| where 0 <= i && i <= total:
        pos val = values[i]
        assert 0 < val      // actually: assume
        total = total + val
        i = i + 1
    return total

method main(System.Console sys):
    sys.out.println(sum([1,3,5,7,9]))
```

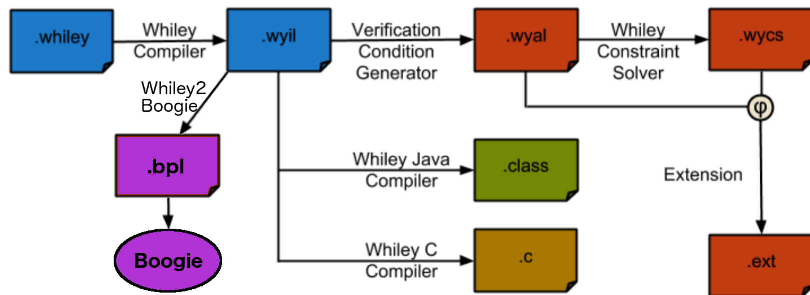
Why a WP semantics for Whiley?

- Every language should have a WP semantics!
- Current semantics is informal (user manual) or implementation-oriented (compilers and interpreters, via a custom bytecode). *We would like a concise and abstract semantics upon which to build tools.*
- **Could be a good basis for translating Whiley to Boogie.** *An alternative verification path would be interesting.*



Boogie is an *intermediate verification language* from Microsoft Research. See `https://research.microsoft.com/boogie`.

Whiley Tools



Adapted from: 'Verifying Whiley Programs using an Off-the-Shelf SMT Solver', by Henry J. Wylde, Eng489 Project Report, 2014, VUW, NZ.

Whiley Values

Here is a simplified data model for Whiley values, *VAL*, specified as a Z free type. *NAME* is used for field names of records and for named functions and methods.

$$[NAME]$$

$$BOOL ::= false | true$$

$$\begin{aligned}
WVal ::= {} & null \mid bool\langle\langle BOOL\rangle\rangle \mid int\langle\langle \mathbb{Z}\rangle\rangle \mid string\langle\langle \mathrm{seq}\,\mathbb{Z}\rangle\rangle \\
& \mid tuple\langle\langle \mathrm{seq}\, WVal\rangle\rangle \\
& \mid array\langle\langle \mathrm{seq}\, WVal\rangle\rangle \\
& \mid record\langle\langle NAME \pfun WVal\rangle\rangle \qquad // \ \textit{Closed-World} \\
& \mid obj\langle\langle NAME \pfun WVal\rangle\rangle \qquad // \ \textit{Open-World} \\
& \mid ref\langle\langle \mathbb{Z}\rangle\rangle \qquad // \ \textit{Not Today...} \\
& \mid func\langle\langle NAME\rangle\rangle \\
& \mid method\langle\langle NAME\rangle\rangle
\end{aligned}$$

$$TYPE == \mathbb{P}\, WVal \qquad // \ \textit{Rich Semantic Types!}$$

Whiley Values in Boogie...

```
type WField;       // field names for records
type WMethodName; // names of methods

// The set of ALL Whiley values.
type WVal;

// For each subtype of WVal, we have:
function isInt(WVal) returns (bool);
function toInt(WVal) returns (int);
function fromInt(int) returns (WVal);
axiom (forall i:int :: isInt(fromInt(i)));
axiom (forall i:int :: toInt(fromInt(i)) == i);
axiom (forall v:WVal :: isInt(v)
       ==> fromInt(toInt(v)) == v);

axiom (forall v:WVal :: isInt(v)
       ==> !isNull(v) && !isBool(v) && ...);
```

Whiley Arrays in Boogie...

```
function isArray(WVal) returns (bool);
function toArray(WVal) returns ([int]WVal);
function arraylen(WVal) returns (int);
function fromArray([int]WVal,int) returns (WVal);

function arrayupdate(a:WVal, i:WVal, v:WVal)
  returns (WVal)
  { fromArray(toArray(a)[toInt(i) := v], arraylen(a)) }

// Whiley array generators [val;len] are written as:
//    fromArray(arrayconst(val), len)
// Array literals are written as:
//    arrayconst(val0)[1 := val1][2 := val2] etc.

function arrayconst(val:WVal) returns ([int]WVal);
axiom (forall val:WVal,i:int :: arrayconst(val)[i]==val);
```

Our Example translated to Boogie, Part 1

```
function is_pos(x:WVal) returns (bool)
{ isInt(x) && toInt(x) > 0 }

function is_pos5(a:WVal) returns (bool)
{ isArray(a) && (forall i:int :: 0 <= i && i < arraylen(a)
  ==> is_pos(toArray(a)[i])) && arraylen(a) == 5 }

function sum__pre(values:WVal) returns (bool)
{ is_pos5(values) && true }

function sum(values:WVal) returns (result:WVal);
axiom (forall values:WVal, result:WVal ::
    sum(values) == (result) && sum__pre(values)
    ==>
      is_pos(result) &&
      toInt(result) >= 5);
```

Our Example translated to Boogie, Part 2

```
procedure sum__impl(values:WVal) returns (result:WVal);
    requires sum__pre(values);
    ensures is_pos(result) && toInt(result) >= 5;
implementation sum__impl(values:WVal) returns(result:WVal)
{ var i : WVal where isInt(i);
  var total : WVal where isInt(total);
  var val : WVal where is_pos(val);
  i := fromInt(0);
  total := fromInt(0);
  while (toInt(i) < arraylen(values))
    invariant 0 <= toInt(i) && toInt(i) <= toInt(total);
  {
    assert 0 <= toInt(i) && toInt(i) < arraylen(values);
    val := toArray(values)[toInt(i)];
    assert 0 < toInt(val);
    total := fromInt(toInt(total) + toInt(val));
    i := fromInt(toInt(i) + 1);
  }
  result := total;  return;
}
```

Verification Results

Whiley to Boogie+Z3

Whiley	NotImpl	Errors	Partly Verified	Fully Verified	Total
NotImpl	18 20.5%	4 4.5%	17 19.3%	49 55.7%	88
Fully Verified	114 28.1%	6 1.5%	8 2.0%	277 68.4%	405
	132 26.8%	10 2.0%	25 5.0%	326 66.1%	493

132 NotImplementedYet?

- indirect invoke (12 tests)
- lambda functions (12 tests)
- references, new (17 tests), and dereferencing (17 tests)
- switch (14 tests) [Whiley semantics]
- functions/methods with multiple return values (4 tests)
- continue statements and named blocks (3 tests)
- bitwise operators (13 tests) [None in Boogie]
- some kinds of complex constants

10 BPL Errors and Translator Exceptions?

- Bugs in my generation of `&&` and `||` operators (equal precedence in Boogie, different in Whiley);
- Whiley variable name is a reserved word in Boogie ('type', 'old');
- Assigning to a function input (immutable in Boogie);
- Undeclared record field name (not used in code, only in implicit typing predicates);
- Same variable used as program variable and quantified variable;
- Proof obligation uses quantified variable;
- functions with no return values!

25 programs that Boogie/Z3 cannot verify?

- Complex_Valid_2.whiley: cannot prove append typing preconditions (array equality issue?)
- Complex_Valid_8.whiley: ibid.
- ConstrainedList_Valid_14.whiley: Cannot prove

$$xs[0] = 1 \implies some\{i \in 0 \ldots |xs| \mid xs[i] > 0\}$$

- DoWhile_Valid_5.whiley: differing semantics for do-while. Should loop invariant hold before first iteration?
- DoWhile_Valid_8.whiley: ibid
- Fail_Valid_1.whiley: test is invalid when input x is null?

Conclusions

- Whiley to Boogie translator is already useful.
- Boogie verifier can already prove 66% of ALL tests.
- Excluding NotImplYet: Boogie can prove 95.2% of the tests that Whiley proves, and 70% of the tests that Whiley cannot prove (with 10 second limit).
- Boogie is a useful verification intermediate language.
- My 'untyped' translation of Whiley to Boogie is working well.
- Future work: indirect invoke (calling unknown functions); then full object-orientation;