Testing, Debugging, and Verification

Formal Verification, Part I

Srinivas Pinisetty[1]

30 November 2017

[1]Lecture slides based on material from Wolfgang Aherndt,..

Recap: Functions and Predicates

- Method calls are not allowed in specifications.
  - May have side effects - bad for proofs
- Functions and Predicates are allowed in specifications
  - No side effects, cannot manipulate memory.
  - Only allowed in specifications: Not present in compiled code only for verification.
  - `function method` compiled, allowed both in code and specification.

Recap: Loop Invariants and Variants

Loops are difficult to reason about.

- Don't know how many times we go around.
- But Dafny needs to consider all paths! How?

Solution: Loop Invariants

An invariant is an property which is true before entering loop and after each execution of loop body.

But what about termination?

Solution: Loop Variants

An variant is an expression which decrease with each iteration of the loop, and is bounded from below by 0.
Dafny can often guess variants automatically.

Formal Verification

- Three lectures.
- One assignment to hand in.

Todays main topics:

- Dafny behind the scenes: How does it prove programs correct?
- Weakest Precondition Calculus

Formal Software Verification: Motivation

Limitations of Testing

- Testing ALL inputs is usually impossible.
- Even strongest coverage criteria cannot guarantee abcence of further defects.

Goal of Formal Verification

Given a formal specification S of the behaviour of a program P:
Give a mathematically rigorous proof that each run of P conforms to S

P is correct with respect to S

Formal Software Verification: Limitations

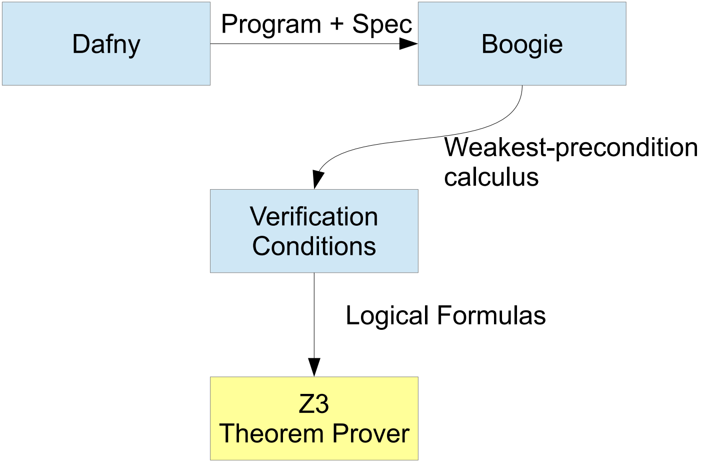
- No absolute notion of program correctness!
  - Correctness always relative to a given specification
- Hard and expensive to develop provable formal specifications
- Some properties may be difficult or impossible to specify.
- Requires lots of expertise and expenses (so far...)
- Even fully specified & verified programs can have runtime failures
  - Defects in the compiler
  - Defects in the runtime environment
  - Defects in the hardware

Possible & desirable:
Exclude defects in source code wrt. a given spec

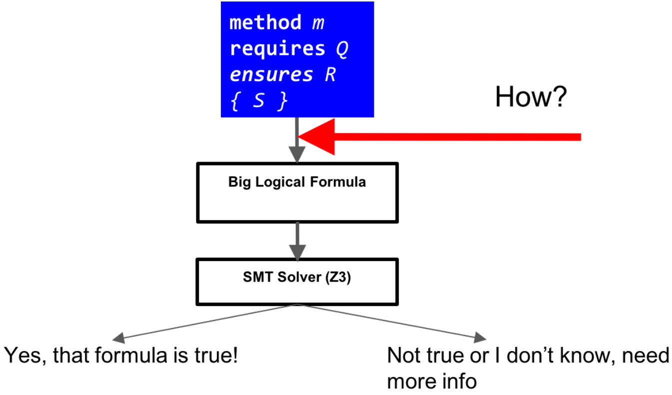
Dafny: Behind the Scenes

What happens when we ask Dafny to compile our program?
How does it prove that it is correct according to its specification?

Dafny: Behind the Scenes



Dafny: Behind the Scenes



- **Our focus:** How do we extract verification conditions (Big Logical Formula)?
- **This module**: Weakest precondition calculus.
- Won't deal with full Dafny/Boogie, but simplified subset involving assignments, if-statements, while loops.

What do we Need to Prove and How?

```
method MyMethod(. . .)
  requires Q
  ensures R
  {
   S: program statements
  }
```

In literature, often expressed as a Hoare Triple: {Q} S {R}

Hoare Triple: {Q} S {R}

If execution of program S starts in a state satisfying pre-condition Q, the is is guaranteed to terminate in a state satisfying the post-condition R.

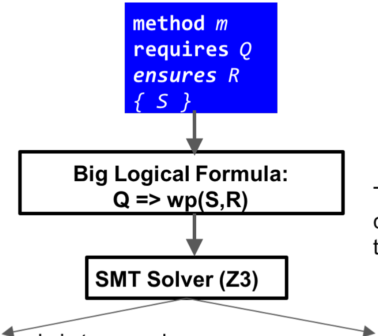
What do we Need to Prove and How?

```
method MyMethod(. . .)
  requires Q
  ensures R
  {
   S: program statements
  }
```

Weakest Precondition:

- Assuming that R holds after executing S,
- What is the least restricted (set of) state we could possibly begin from?
  - Weakest = Fewest restrictions on input state.
  - Formally: $wp(S, R)$
- Does Q satisfy at least these restrictions?
  - i.e. does Q imply the weakest pre-condition?
  - To prove: $Q \rightarrow wp(S, R)$
  - Proving Hoare triple {Q} S {R} amounts to showing that $Q \rightarrow wp(S, R)$.

What do we Need to Prove and How?



This logical formula is also called
the *Verification condition*

Yes, that formula is true, and hence program is correct!

Not true or I don't know, need more info

Weakest Precondition

Weakest Precondition: $wp(S, R)$

The weakest precondition of a program S and post-condition R represents the set of all states such that execution of S started in any of these is guaranteed to terminate in a state satisfying R.

First-Order Formulas and Program States

First-order formulas define sets of program states

What do we mean by $wp(S, R)$ defining a set of program states?

$wp(S, R)$ is a logical predicate F that is true in some states and not true in others.

Example

- `(i>j & j>=0)` is true in exactly those states S where $\texttt{i}^s > \texttt{j}^s$ and $\texttt{j}^s$ is non-negative.
- `exists i :: i == j`

  is true in any state S, because the value of `i` can be chosen to be $\texttt{j}^s$

Example

- Program statement S: `i := i + 1`
- Post-condition R: `i <= 1`

What is the weakest precondition, $wp(S, R)$?

- **Reason backwards**: $wp(i := i + 1, i <= 1) = i <= 0$
- Executing `i := i + 1` in any state satisfying `i <= 0` will end in a state satisfying `i <= 1`.
- **Note:** Taking Q: `i < -5` does also satisfy R. But overly restrictive, excludes initial states where `-5 <= i <=0`. Weakest precondition can help us find a suitable contract.

Mini Quiz: Guess the Weakest Precondition

Write down $wp(S, R)$ for the following S and R:

	S	R
a)	`i := i+1`	`i > 0`
b)	`i := i+2; j := j-2`	`i + j == 0`
c)	`a[i] := 1`	`a[i] == a[j]`
d)	`i := i+1; j := j-1`	`i * j == 0`

Solution:

a)	`i >= 0`
b)	`i + j == 0`
c)	`a[j] == 1`
d)	`i == -1 || j == 1`

Weakest Precondition Calculus

Our Verification Algorithm

- Have a program S, with precondition Q and postcondition R
- Compute $wp(S, R)$
- Prove that $Q \rightarrow wp(S, R)$

The rules of the weakest precondition calculus provide semantics, a logical meaning, for the statements in our programming language.

Weakest Precondition Calculus

We will prove validity of programs written in a slightly simplified subset of Dafny/Boogie featuring:

Assignment: `x := e`

Sequentials: `S1; S2`

Assertions: `assert B`

If-statements: `if B then S1 else S2`

While-loops: `while B S`

Semantics

We will define the weakest precondition for each of these program constructs.

Weakest Precondition Calculus: Assignment

Assignment

$wp(x := e, R) = R[x \mapsto e]$

Note: $R[x \mapsto e]$ means "R with all occurrences of x replaced by e".

Example

Let S:

```
i := i + 1;
```

Let R: $i > 0$

$wp(i := i + 1, i > 0) =$
(By Assignment rule)
$i + 1 > 0$

This program satisfies its postcondition if started in any state where $i \geq 0$.

Weakest Precondition Calculus: Sequential Composition

Sequential Composition

$wp(S1; S2, R) = wp(S1, wp(S2, R))$

Example

Let S:

```
x := i;
i := i + 1;
```

Let R: $x < i$

$wp(x := i; i := i + 1, x < i) =$

(By Sequential rule)

$wp(x := i, wp(i := i + 1, x < i)) =$

(By Assignment rule)

$wp(x := i, x < i + 1) =$

(By Assignment rule)

$i < i + 1$

(trivially true)

This program satisfies its postcondition in any initial state.

Weakest Precondition Calculus: Assertion

Assertion

$wp(assert\ B,\ R) = B \wedge R$

Example

Let S:

```
x := y;
assert x > 0;
```

Let R: $x < 20$

$wp(x := y; assert\ x > 0, x < 20) =$
(By Sequential rule)
$wp(x := y, wp(assert\ x > 0, x < 20)) =$
(By Assertion rule)
$wp(x := y, x > 0 \wedge x < 20) =$
(By Assignment rule)
$y > 0 \wedge y < 20$

This program satisfies its postcondition in those initial states where `y` is a number between 1 and 19 (inclusive).

Weakest Precondition Calculus: Conditional

Conditional

$wp(\textit{if } B \textit{ then } S1 \textit{ else } S2,\ R) =$
$(B \rightarrow wp(S1, R)) \wedge (\neg B \rightarrow wp(S2, R))$

Example

Let S:

```
if (i >= 0) then
  x := i else x := -i
```

Abbreviate:

S1: `x := i`

S2: `x := -i`

Let R: $x \geq 0$

$wp(\textit{if } (i \geq 0) \textit{ then } S1 \textit{ else } S2, x \geq 0) =$

(By Conditional rule)

$i \geq 0 \rightarrow wp(x := i, x \geq 0) \wedge$
$\neg(i \geq 0) \rightarrow wp(x := -i, x \geq 0) =$

(By Assignment rule)

$(i \geq 0 \rightarrow i \geq 0) \wedge (\neg(i \geq 0) \rightarrow -i \geq 0) =$

true

This program satisfies its postcondition in any initial state.

Weakest Precondition Calculus: Conditional

Conditional, empty `else` branch

$wp(\textit{if B then S1, R}) = (B \rightarrow wp(S1, R)) \wedge (\neg B \rightarrow R)$

If `else` is empty, need to show that R follows just from negated guard.

Mini Quiz: Derive the weakest precondition

The Rules

$$wp(x := e, R) = R[x \mapsto e]$$
$$wp(S1; S2, R) = wp(S1, wp(S2, R))$$
$$wp(assert\ B,\ R) = B \wedge R$$
$$wp(if\ B\ then\ S1\ else\ S2,\ R) = (B \rightarrow wp(S1, R)) \wedge (\neg B \rightarrow wp(S2, R))$$

Derive the weakest precondition, stating which rules you use in each step.

	S	R
a)	`i := i+2; j := j-2`	`i + j == 0`
b)	`i := i+1; assert i > 0`	`i <= 0`
c)	`if isEven(x) then y:=x/2 else y:=(x-1)/2`	`isEven(y)`

Mini Quiz: Derive the weakest precondition

Solution:

a) `i + j == 0`
(apply seq. rule followed by assignment rule, simplify)

b) `i+1 > 0 && i+1 <= 0`
(apply seq rule, assert rule, assignment)
Simplifies to `i => 0 && i <= -1` which is false! No initial state can satisfy this postcondition.

c)
```
isEven(x) ==> isEven(x/2) && !isEven(x) ==>
isEven((x-1)/2)
```
(apply cond. rule, followed by assignment.)

Let's Prove ManyReturns Correct!

Recall

To prove correct a program S with precondition Q and postcondition R we need to show that $Q \to wp(S, R)$.

```
method ManyReturns(x:int, y:int) returns (more:int, less:
 int)
 requires 0 < y;
 ensures less < x < more;
 { more := x+y;
   less := x-y;
 }
```

Show that

$$0 < y \to wp(more := x + y; less := x - y,\ less < x < more)$$

Let's Prove ManyReturns Correct!

Show that

$0 < y \to wp(more := x + y; less := x - y,\ less < x < more)$

Seq. rule

$0 < y \to wp(more := x + y, wp(less := x - y,\ less < x < more))$

Assignment rule

$0 < y \to wp(more := x + y, x - y < x < more)$

Assignment rule

$0 < y \to (x - y < x < x + y)$

which follows from the precondition by simple arithmetic.

Hint

This level of detail is expected for your proofs in the lab and exam.

Another Example

```
method f ( x : int) returns (y : int)
requires x > 8
ensures  y > 10
{
y := x + 1;
if (y mod 2 == 0) { y := 100; }
        else { y := y + 2; }
}
```

Exercise: Prove f correct

Show that

$x > 8 \to wp(y := x + 1; if \cdots, y > 10)$.

What Next?

While loops!

Difficulties of While Loops

- Need to "unwind" loop body one by one
- In general, no fixed loop bound known (depends on input)
- How the loop invariants and variants are used in proofs.

Summary

- Testing cannot replace verification
- Formal verification can prove properties for all runs,
  ... but has inherent limitations, too.
- Dafny is compile to intermediate language Boogie.
- Verification conditions (VCs) extracted, using weakest precondition calculus rule.
- VCs are logical formulas, which can be passed to a theorem prover.
- Prove that precondition imply *wp*.

Reading: *The Science of Programming* by David Gries. Chapters 6-10, bearing in mind that the notation and language differ slightly from ours. Available as E-book from Chalmers library.