Automated Program Verification and Testing 15414/15614 Fall 2016 Lecture 13: Deductive Verification

Matt Fredrikson
mfredrik@cs.cmu.edu

October 13, 2016

Partial Correctness (Review)

Partial correctness refers to a program's terminating behavior

We specify partial correctness using **Hoare triples**

$$\{P\}\ c\ \{Q\}$$

- c is a program
- P and Q are **assertions** in a first-order theory
- Free variables in P, Q can range over program variables
- P is the **precondition** and Q is the **postcondition**

Hoare Triples: Meaning (Review)

The meaning of $\{P\}\ c\ \{Q\}$ is as follows:

- If we begin executing c in an **environment satisfying** P,
- and **if** c **terminates**,
- then its final environment will satisfy Q

The specification says nothing about:

- Executions that do not terminate (i.e., **diverge**)
- Executions that do not begin in P

Notice: $\{P\}\ c\ \{Q\}$ is a predicate

Goal of verification: prove that it holds, i.e., is a **valid** Hoare triple

Hoare Triples: Total Correctness (Review)

Partial correctness doesn't require termination

Total correctness is a stronger statement, written:

$$[P]\ c\ [Q]$$

The meaning of $[P]\ c\ [Q]$ is:

- If we begin executing c in an **environment satisfying** P,
- **then** c **terminates**,
- **and** its final environment will satisfy Q

Total correctness introduces another obligation for verification

Hoare Logic Inference Rules

$$\text{Skip}\ \frac{}{\{P\}\ \textbf{skip}\ \{P\}} \qquad \text{Asgn}\ \frac{}{\{Q[a/x]\}\ x := a\{Q\}}$$

$$\text{Conseq}\ \frac{P \Rightarrow P' \qquad \vdash \{P'\}\ c\ \{Q'\} \qquad Q' \Rightarrow Q}{\{P\}\ c\ \{Q\}}$$

$$\text{Seq}\ \frac{\{P\}\ c_1\ \{P'\} \qquad \{P'\}\ c_2\ \{Q\}}{\{P\}\ c_1; c_2\{Q\}}$$

$$\text{If}\ \frac{\{P \wedge b\}\ c_1\ \{Q\} \qquad \{P \wedge \neg b\}\ c_2\ \{Q\}}{\{P\}\ \textbf{if}\ b\ \textbf{then}\ c_1\ \textbf{else}\ c_2\{Q\}}$$

$$\text{While}\ \frac{\{P \wedge b\}\ c\ \{P\}}{\{P\}\ \textbf{while}\ b\ \textbf{do}\ c\ \{P \wedge \neg b\}}$$

Soundness of Hoare Logic

The proof rules we've just covered are sound for partial correctness:

$$\text{If } \vdash \{P\}\, c\, \{Q\}, \text{ then } \models \{P\}\, c\, \{Q\}$$

If we can derive a triple using the rules, then it is valid

To prove this, we use the operational semantics

Show equivalence between proof rules and reductions

Need to use induction on derivations

Completeness of Hoare Logic

Completness of Hoare logic is stated as:

$$\text{If } \models \{P\}\ c\ \{Q\}, \text{ then } \vdash \{P\}\ c\ \{Q\}$$

If $\{P\}\ c\ \{Q\}$ is valid, then we can derive it using the rules

Is this true?

For strengthening, we need to prove statements of the form:

$$P \Rightarrow Q$$

This requires proving a universal implication in Peano arithmetic

Recall that T_{PA} is undecidable!

Relative Completeness of Hoare Logic

So, we know there can't be a proof system that derives all valid triples

The Hoare logic has **relative completeness**

If we assume an **oracle** for deciding $P \Rightarrow Q$

Then we can derive any valid Hoare triple for Imp

However, for more complex languages, this isn't always the case

Mechanics of Verification

Working in Hoare logic is nicer than working directly with semantics

But still isn't "fun", and not quite trivial

- How to decompose the program?
- When to apply the rule of consequence, and how?
- Lots of tedious details, e.g., discharging
  $(x = r + (q \times y) \wedge y \leq r) \Rightarrow x = (r - y) + ((q + 1) \times y)$
- Even for short programs, the proofs can be long (and boring)

Now we'll take steps towards mechanizing this process

Basic Approach

Given a program c annotated with a specification:

$$\{P\}\ c\ \{Q\}$$

To prove the triple, we'll generate a set of **verification conditions**

- VC's are a function of the code, specification, and loop invariants
- Each VC is a first-order formula in some theory
- If all VC's are **valid**, then so is $\{P\}\ c\ \{Q\}$

Program verifiers consist of two main components:

1. **Verification condition generator** producing T-formulas
2. **Decision procedure** for first-order theory T

Intuitively, VC gen. "compiles" a verification problem into a math problem

Automation

This is the approach taken by tools like Dafny

As you know, it doesn't prove things automatically

You often need to provide:

1. Loop invariants
2. Termination metrics
3. Extra assertions
4. Different (but possibly equivalent) contracts
5. Moral support and encouragement

These are all the subject of active research (except 5)

- (1), (2) addressed by **static analysis**, but unsolvable in general
- (3), (4) are improved by more powerful decision procedures

VC Generation

Given an assertion Q and program c, we'll describe a function:

- That is a **predicate transformer**: produces another assertion
- Assertion for the corresponding precondition P for c
- P guaranteed to be the **weakest** such assertion

This is the **weakest precondition** predicate transformer $\mathsf{wp}(c, Q)$

The weakest precondition satisfies the following conditions:

1. The triple $[\mathsf{wp}(c, Q)]\ c\ [Q]$ is valid
2. For any P where $[P]\ c\ [Q]$ is valid, $P \Rightarrow \mathsf{wp}(c, Q)$

For partial correctness, use **weakest liberal precondition** $\mathsf{wlp}(c, Q)$

“Backwards” VC Generation

Intuitively, wlp allows us to:

1. Start with a desired postcondition
2. Work backwards to a precondition that must hold
3. Verify that $P \Rightarrow \mathsf{wlp}(c, Q)$
4. Or just use $\mathsf{wlp}(c, Q)$ to write a contract

There is also a “forward” transformer: **strongest postcondition**

1. Written $\mathsf{sp}(c, P)$
2. The triple $[P]\ c\ [\mathsf{sp}(c, P)]$ is valid
3. For any Q where $[P]\ c\ [Q]$ is valid, $\mathsf{sp}(c, P) \Rightarrow Q$

What does $\mathsf{sp}(c, \textit{true})$ characterize?

For the rest of today (and most of the semester), we’ll focus on wlp

Language Extension: Arrays

But first, let's add arrays to Imp

$$
\begin{array}{rcl}
a \in \textbf{AExp} & ::= & n \in \mathbb{Z} \mid x \in \mathsf{Var} \mid a_1 + a_2 \mid a_1 \times a_2 \mid x[a] \\
b \in \textbf{BExp} & ::= & \mathsf{true} \mid \mathsf{false} \mid a_1 = a_2 \mid a_1 \leq a_2 \mid \neg b \mid b_1 \wedge b_2 \\
c \in \textbf{Com} & ::= & \textbf{skip} \mid x := a \mid c_1 ; c_2 \mid x[a_1] := a_2 \\
& & \mid \textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2 \\
& & \mid \textbf{while } b \textbf{ do } c
\end{array}
$$

Allow array lookup, assignment to array indices

Array Semantics

The environment can map to array values (i.e., from T_A)

$$\text{ALookup}\ \frac{\langle a, \sigma\rangle \Downarrow n \qquad \sigma(x) \text{ is an array}}{\langle x[a], \sigma\rangle \Downarrow \sigma(x)[n]}$$

$$\text{AWrite}\ \frac{\langle a_1, \sigma\rangle \Downarrow n_1 \qquad \langle a_2, \sigma\rangle \Downarrow n_2}{\langle x[a_1] := a_2, \sigma\rangle \Downarrow \sigma[x \mapsto \sigma(x)\langle n_1 \triangleleft n_2\rangle]}$$

Assignment produces a new array using the array update term

Defining wlp(c, Q): Assignment

We'll use Hoare triples to define wlp

Recall the rule for assignment:

$$\text{Asgn}\ \frac{}{\{Q[a/x]\}\ x := a\{Q\}}$$

The corresponding transformer is:

$$\text{wlp}(x := a, Q) = Q[a/x]$$

This will produce valid preconditions. Is it weak enough?

If $P \not\Rightarrow Q[a/x]$, then $\{P\}\ c\ \{Q\}$ won't hold

Example

What is $\mathsf{wlp}(y := x + 1, (\forall x. x < z \to x < y) \to x + 1 \leq y)$?

Applying the definition of wlp for assignment,

$$(\forall x. x < z \to x < x + 1) \to x + 1 \leq x + 1$$

Is this right?

No. When we substituted y with $x + 1$ in

$$(\forall x. x < z \to x < y)$$

the variable x was **captured**

Recall: x is bound in $\forall x. x < a \to x < y$

Outside the scope of this $\forall$, we're referring to a different x

Capture-avoiding substitution

When performing substitution, we need to be careful about scoping

When we expand out $P[a/x]$, we need to:

- Only substitute **free occurrences** of x
- Rename bound variables appearing in a to avoid capture

Renaming bound variables is called α**-substitution**

For a substitution $P[F/G]$, let

$$V_{\text{free}} = \bigcup_i \text{free}(F) \cup \text{free}(G)$$

To expand $P[F/G]$:

- For each quantified variable x in P such that $x \in V_{\text{free}}$, rename x to a fresh variable to produce P'
- Apply the substitution directly to P'

Example

Consider the formula:

$$F : (\forall x.p(x,y)) \to q(f(y),x)$$

Suppose we want $F[y/f(x), q(f(y),x)/(\exists x.h(x,y))]$

First, we find all the free variables:

$$\begin{array}{rl} & \text{free}(y) \cup \text{free}(f(x)) \cup \text{free}(q(f(y),x)) \cup \ \text{free}(\exists x.h(x,y)) \\ = & \{x\} \cup \{x\} \cup \{y\} \cup \{x\} \cup \{x,y\} \cup \{y\} \\ = & \{x,y\} \end{array}$$

F has one quantified variable x, which is in this set. So:

$$F : (\forall x'.p(x',y)) \to q(f(y),x)$$

Applying the substitution:

$$(\forall x'.p(x',f(x))) \to \exists x.h(x,y)$$

Example

$$F' : (\forall x'.p(x', y)) \rightarrow q(f(y), x)$$

Applying the first substitution $q(f(y), x)/(\exists x.h(x, y))$:

$$F' : (\forall x'.p(x', y)) \rightarrow \exists x.h(x, y)$$

Now we apply the second substitution: $y/f(x)$

We need to rename bound variables that occur in V_{free}:

$$F'' : (\forall x'.p(x', y)) \rightarrow \exists x'.h(x', y)$$

Which brings us to:

$$F'' : (\forall x'.p(x', f(x))) \rightarrow \exists x'.h(x', f(x))$$

Defining wlp(c, Q): Array Assignment

Array assignment gives the rule

$$\text{AsgnArr}\ \frac{}{\{Q[x\langle a_1 \triangleleft a_2\rangle/x]\}\ x[a_1] := a_2\ \{Q\}}$$

The corresponding transformer is:

$$\text{wlp}(x[a_1] := a_2, Q) = Q[a\langle a_1 \triangleleft a_2\rangle/x]$$

This is no different than normal assignment

In this case, we're working with array values rather than integers

Example

Let's compute:

$$\mathsf{wlp}(b[i] := 5, b[i] = 5)$$

Proceeding with the substitution,

$$\begin{aligned} \mathsf{wlp}(b[i] := 5, b[i] = 5) &= (b\langle i \triangleleft 5\rangle[i] = 5) \\ &= (5 = 5) \\ &= \textit{true} \end{aligned}$$

Example

Let's compute:

$$\mathsf{wlp}(b[n] := x, \forall i. 1 \leq i < n \rightarrow b[i] \leq b[i+1])$$

Proceeding with the substitution,

$$\begin{aligned}
&\mathsf{wlp}(b[n] := x, \forall i. 1 \leq i < n \rightarrow b[i] \leq b[i+1]) \\
&\quad = \forall i. 1 \leq i < n \rightarrow (b\langle n \triangleleft x \rangle)[i] \leq (b\langle n \triangleleft x \rangle)[i+1] \\
&\quad = (b\langle n \triangleleft x \rangle)[n-1] \leq (b\langle n \triangleleft x \rangle)[n] \\
&\qquad\quad \wedge \forall i. 1 \leq i < n \rightarrow (b\langle n \triangleleft x \rangle)[i] \leq (b\langle n \triangleleft x \rangle)[i+1] \\
&\quad = b[n-1] \leq x \wedge \forall i. 1 \leq i < n-1 \rightarrow b[i] \leq b[i+1]
\end{aligned}$$

Defining $\mathsf{wlp}(c, Q)$: Sequencing

$$\text{Seq}\ \frac{\{P\}\ c_1\ \{P'\} \qquad \{P'\}\ c_2\ \{Q\}}{\{P\}\ c_1; c_2\{Q\}}$$

The corresponding transformer is:

$$\mathsf{wlp}(c_1; c_2, Q) = \mathsf{wlp}(c_1, \mathsf{wlp}(c_2, Q))$$

Compose the transformer, working backwards from c_2

The precondition for c_2 becomes the postcondition for c_1

Note: we don't need to use consequence to "glue" these together

Defining $\mathsf{wlp}(c, Q)$: Conditional

$$\text{If } \frac{\{P \land b\}\ c_1\ \{Q\} \qquad \{P \land \neg b\}\ c_2\ \{Q\}}{\{P\}\ \textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2 \{Q\}}$$

$$\mathsf{wlp}(\textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2, Q) = (b \rightarrow \mathsf{wlp}(c_1, Q)) \land (\neg b \rightarrow \mathsf{wlp}(c_2, Q))$$

If b holds, then wlp of c_1 branch must hold

Otherwise if $\neg b$, then wlp of c_2 branch must hold

Why isn't the formula a disjunction?

Defining wlp(c, Q): While Loop

$$\text{While}\ \frac{\{P \land b\}\ c\ \{P\}}{\{P\}\ \textbf{while}\ b\ \textbf{do}\ c\ \{P \land \neg b\}}$$

Recall the equivalence:

$$\textbf{while}\ b\ \textbf{do}\ c\ \equiv \textbf{if}\ b\ \textbf{then}\ c; \textbf{while}\ b\ \textbf{do}\ c\ \textbf{else skip}$$

Let's apply wlp for **if**:

$$\begin{aligned}
&\mathsf{wlp}(\textbf{if}\ b\ \textbf{then}\ c; \textbf{while}\ b\ \textbf{do}\ c\ \textbf{else skip}, Q) \\
&\quad = (b \to \mathsf{wlp}(c; \textbf{while}\ b\ \textbf{do}\ c, Q)) \land (\neg b \to Q) \\
&\quad = (b \to \mathsf{wlp}(c, \mathsf{wlp}(\textbf{while}\ b\ \textbf{do}\ c, Q))) \land (\neg b \to Q) \\
&\quad = (b \to \mathsf{wlp}(c, \mathsf{wlp}(\textbf{if}\ b\ \textbf{then}\ c; \textbf{while}\ b\ \textbf{do}\ c\ \textbf{else skip}, Q))) \land (\neg b \to Q)
\end{aligned}$$

We haven't accomplished anything

Approximate Weakest Precondition

In general, we can't always compute wlp for loops

Instead, we'll **approximate** it with help from annotations

Now we'll assume loops have the syntax:

$$\textbf{while } b \textbf{ do } \{I\}\ c$$

I is a loop invariant provided by the programmer

The approximate wlp for **while** will still be a valid precondition

But it may not be the weakest precondition: even if

$$\{P\}\ \textbf{while } b \textbf{ do } c\ \{Q\}$$

is valid, it might not be that:

$$P \Rightarrow \mathsf{wlp}(\textbf{while } \{I\}\ b \textbf{ do } c, Q)$$

Approximate wlp: While Loop

Now, suppose we define:

$$\text{wlp}(\textbf{while } \{I\}\ b \textbf{ do } c, Q) = I$$

A couple of things are missing:

- We haven't checked that $I \wedge \neg b$ gives us Q
- We don't know that I is actually a loop invariant

We need to track additional verification conditions: $\text{vc}(c, Q)$

Verification Conditions: While Loop

If we define

$$\text{wlp}(\textbf{while } \{I\}\ b \textbf{ do } c, Q) = I$$

Then we still need to show that

- $I \land \neg b$ establishes Q
- I is a loop invariant

Defining the set of verification conditions,

$$\text{vc}(\textbf{while } \{I\}\ b \textbf{ do } c, Q) = \left\{ \begin{array}{l} I \land \neg b \Rightarrow Q \\ I \land b \Rightarrow \text{wlp}(c, Q) \end{array} \right\}$$

To summarize, for Q to hold after executing a loop:

1. Each formula in $\text{vc}(\textbf{while } \{I\}\ b \textbf{ do } c, Q)$ must be valid
2. $\text{wlp}(\textbf{while } \{I\}\ b \textbf{ do } c, Q) = I$ must be valid

Propagating Verification Conditions

while is the only command that introduces new conditions

But other statements might contain loops

Need to define vc for them as well:

- $\mathsf{vc}(x := a, Q) = \varnothing$
- $\mathsf{vc}(c_1; c_2, Q) = \mathsf{vc}(c_1, \mathsf{wlp}(c_2, Q)) \cup \mathsf{vc}(c_2, Q)$
- $\mathsf{vc}(\textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2, Q) = \mathsf{vc}(c_1, Q) \cup \mathsf{vc}(c_2, Q)$

In short, compound statements collect conditions from constituents

Verification Using wlp

Bringing all of this together, we can verify

$$\{P\}\ c\ \{Q\}$$

for an annotated program c

1. Compute P' = wlp(c, Q)
2. Compute vc(c, Q)
3. Check validity of $P \to P'$
4. Check validity of each $F \in$ vc(c, Q)

If (3) and (4) pass, then $\{P\}\ c\ \{Q\}$ is valid

If $\{P\}\ c\ \{Q\}$ is valid, then will (3) and (4) pass?
No. Loop invariants might be too weak!

Example

Let's verify the example from last lecture:

$$
\begin{array}{l}
\{\textit{true}\} \\
r := x; q := 0; \\
\textbf{while } y \leq r \textbf{ do} \\
\quad r := r - y; q := q + 1 \\
\{r < y \wedge x = r + (q \times y)\}
\end{array}
$$

Recall our loop invariant:

$$
\begin{array}{l}
\{\textit{true}\} \\
r := x; q := 0; \\
\textbf{while } y \leq r \textbf{ do} \\
\{x = r + (q \times y)\} \\
\quad r := r - y; q := q + 1 \\
\{r < y \wedge x = r + (q \times y)\}
\end{array}
$$

Example

Define the following shorthand:

- $c_1 : r := x$
- $c_2 : q := 0$
- $c_3 : r := r - y$
- $c_4 : q := q + 1$
- $c_5 :$ **while** $y \leq r$ **do** $c_3; c_4$

$\{\textit{true}\}$
$c_1; c_2;$
while $y \leq r$ **do**
$\{x = r + (q \times y)\}$
$c_3; c_4$
$\{r < y \wedge x = r + (q \times y)\}$

We need to show these are valid:

$$\textit{true} \Rightarrow \text{wlp}(c_1; c_2; c_5, r < y \wedge x = r + (q \times y))$$
$$\text{vc}(c_1; c_2; c_5, r < y \wedge x = r + (q \times y))$$

We'll start with *true* $\Rightarrow \text{wlp}(c_1; c_2; c_5, r < y \wedge x = r + (q \times y))$

Example

$$\textit{true} \Rightarrow \mathsf{wlp}(c_1; c_2; c_5, r < y \land x = r + (q \times y))$$

Let's use $Q : r < y \land x = r + (q \times y)$, $I : x = r + (q \times y)$

We begin by applying the rule for composition twice:

$$\mathsf{wlp}(c_1; c_2; c_5, Q) = \mathsf{wlp}(c_1, \mathsf{wlp}(c_2, \mathsf{wlp}(c_5, Q)))$$

This brings us to $\mathsf{wlp}(c_5, Q)$:

$$\mathsf{wlp}(\textbf{while } y \le r \textbf{ do } \{I\}\ c_3; c_4, Q) = I$$

We also have verification conditions:

$$\mathsf{vc}(c_5, Q) = \{I \land \neg b \Rightarrow Q, I \land b \Rightarrow \mathsf{wlp}(c_3; c_4, Q)\}$$

Example

Let's work out the VC $I \wedge b \Rightarrow \mathsf{wlp}(c_3; c_4, Q)$

We have that:

$$\begin{aligned}
\mathsf{wlp}(r := r - y; q := q + 1, Q) &= \mathsf{wlp}(r := r - y, \mathsf{wlp}(q := q + 1, Q)) \\
&= \mathsf{wlp}(r := r - y, Q[q/q + 1]) \\
&= \mathsf{wlp}(r := r - y, r < y \wedge x = r + ((q + 1) \times y)) \\
&= (x = (r - y) + ((q + 1) \times y))
\end{aligned}$$

So, we have:

$$\mathsf{vc}(c_5, Q) = \{I \wedge \neg b \Rightarrow Q, I \wedge b \Rightarrow (x = (r - y) + ((q + 1) \times y))\}$$

Example

Recalling that $\mathsf{wlp}(c_5, Q) = I$, we now need $\mathsf{wlp}(c_2, I)$:

$$\begin{aligned}\mathsf{wlp}(q := 0, x = r + (q \times y)) &= (x = r + (0 \times y)) \\ &= x = r\end{aligned}$$

Moving on, our final step is $\mathsf{wlp}(c_1, x = r)$:

$$\mathsf{wlp}(r := x, x = r) = (x = x)$$

Popping back to our top-level procedure:

1. Compute $P' = \mathsf{wlp}(c, Q)$
   $P' = (x = x)$
2. Compute $\mathsf{vc}(c, Q)$
   $\mathsf{vc}(c, Q) = \{I \wedge \neg b \Rightarrow Q, I \wedge b \Rightarrow (x = (r - y) + ((q + 1) \times y))\}$
3. Check validity of $P \rightarrow P'$
   Clearly, *true* $\Rightarrow (x = x)$
4. Check validity of each $F \in \mathsf{vc}(c, Q)$

Example

Check validity of each $F \in \mathsf{vc}(c, Q)$:

$$\mathsf{vc}(c, Q) = \left\{ \begin{array}{l} x = r + (q \times y) \wedge \neg(y \leq r) \Rightarrow r < y \wedge x = r + (q \times y) \\ x = r + (q \times y) \wedge y \leq r \Rightarrow (x = (r - y) + ((q + 1) \times y)) \end{array} \right\}$$

The first is true because $\neg(y \leq r) \Leftrightarrow r < y$

The second we get by algebraic calculation

Therefore, the triple is valid

$$\begin{array}{l} \{\mathit{true}\} \\ r := x; q := 0; \\ \textbf{while } y \leq r \textbf{ do} \\ \quad r := r - y; q := q + 1 \\ \{r < y \wedge x = r + (q \times y)\} \end{array}$$

Next Lecture

- We'll add procedures to the language
- Start on total correctness
- **Mid-term 1 week from now**
- We'll post a study guide by the weekend
- **Assignment 3 due date pushed to October 25**