Automated Program Verification and Testing
15414/15614 Fall 2016
Lecture 14:
Deductive Verification, Part 2

Matt Fredrikson
mfredrik@cs.cmu.edu

October 18, 2016
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 $wp(c, Q)$

The weakest precondition satisfies the following conditions:

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

For partial correctness, use **weakest liberal precondition** $wlp(c, Q)$
Weakest Liberal Precondition (Review)

\[\text{wlp}(x := a, Q) = Q[a/x]\]
\[\text{wlp}(x[a_1] := a_2, Q) = Q[a\langle a_1 < a_2 \rangle/x]\]
\[\text{wlp}(c_1; c_2, Q) = \text{wlp}(c_1, \text{wlp}(c_2, Q))\]
\[\text{wlp(\text{if } b \text{ then } c_1 \text{ else } c_2, Q)} = (b \rightarrow \text{wlp}(c_1, Q)) \land (\neg b \rightarrow \text{wlp}(c_2, Q))\]
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:

```
while b 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} while b do c \{Q\}
```

is valid, it might not be that:

```
P \Rightarrow wlp(while \{I\} b do c, Q)
```
If we define

\[ \text{wlp}(\text{while } \{I\} b \text{ do } c, Q) = I \]

Then we still need to show that

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

Defining the set of verification conditions,

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

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

1. Each formula in \( \text{vc}(\text{while } \{I\} b \text{ do } c, Q) \) must be valid
2. \( \text{wlp}(\text{while } \{I\} b \text{ do } c, Q) = I \) must be valid
**while** is the only command that introduces new conditions

But other statements might contain loops

Need to define $\text{vc}$ for them as well:

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

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 \( P' = \text{wlp}(c, Q) \)
2. Compute \( \text{vc}(c, Q) \)
3. Check validity of \( P \rightarrow P' \)
4. Check validity of each \( F \in \text{vc}(c, Q) \)

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!
Let’s verify the example from last lecture:

\[
\begin{array}{l}
\{true\} \\
r := x; \; q := 0; \\
\textbf{while} \; y \leq r \; \textbf{do} \\
r := r - y; \; q := q + 1 \\
\{r < y \land x = r + (q \times y)\}
\end{array}
\]

Recall our loop invariant:

\[
\begin{array}{l}
\{true\} \\
r := x; \; q := 0; \\
\textbf{while} \; y \leq r \; \textbf{do} \\
x = r + (q \times y) \\
r := r - y; \; q := q + 1 \\
\{r < y \land x = r + (q \times y)\}
\end{array}
\]
Define the following shorthand:

- \( c_1 : r := x \)
- \( c_2 : q := 0 \)
- \( c_3 : r := r - y \)
- \( c_4 : q := q + 1 \)
- \( c_5 : \textbf{while } y \leq r \textbf{ do } c_3; c_4 \)

We need to show these are valid:

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

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

\[ \text{true} \Rightarrow wlp(c_1; c_2; c_5, r < y \land x = r + (q \times y)) \]

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:
\[ wlp(c_1; c_2; c_5, Q) = wlp(c_1, wlp(c_2, wlp(c_5, Q))) \]

This brings us to \( wlp(c_5, Q) : \)
\[ wlp(\text{while } y \leq r \text{ do } \{I\} c_3; c_4, Q) = I \]

We also have verification conditions:
\[ \text{vc}(c_5, Q) = \{I \land \neg b \Rightarrow Q, I \land b \Rightarrow wlp(c_3; c_4, Q)\} \]
Let’s work out the VC $I \land b \Rightarrow \text{wlp}(c_3; c_4, Q)$

We have that:

$$\text{wlp}(r := r - y; q := q + 1, Q) = \text{wlp}(r := r - y, \text{wlp}(q := q + 1, Q))$$
$$= \text{wlp}(r := r - y, Q[q/q + 1])$$
$$= \text{wlp}(r := r - y, r < y \land x = r + ((q + 1) \times y))$$
$$= (x = (r - y) + ((q + 1) \times y))$$

So, we have:

$$\text{vc}(c_5, Q) = \{ I \land \neg b \Rightarrow Q, I \land b \Rightarrow (x = (r - y) + ((q + 1) \times y)) \}$$
Recalling that \( \text{wlp}(c_5, Q) = I \), we now need \( \text{wlp}(c_2, I) \):
\[
\text{wlp}(q := 0, x = r + (q \times y)) = (x = r + (0 \times y)) = x = r
\]

Moving on, our final step is \( \text{wlp}(c_1, x = r) \):
\[
\text{wlp}(r := x, x = r) = (x = x)
\]

Popping back to our top-level procedure:

1. Compute \( P' = \text{wlp}(c, Q) \)
   \[
P' = (x = x)
\]

2. Compute \( \text{vc}(c, Q) \)
   \[
   \text{vc}(c, Q) = \{ I \land \neg b \Rightarrow Q, \ I \land b \Rightarrow (x = (r - y) + ((q + 1) \times y)) \}
   \]

3. Check validity of \( P \rightarrow P' \)
   Clearly, \( \text{true} \Rightarrow (x = x) \)

4. Check validity of each \( F \in \text{vc}(c, Q) \)
Example

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

$$vc(c, Q) = \begin{cases} 
  x = r + (q \times y) \land \neg(y \leq r) \Rightarrow r < y \land x = r + (q \times y) \\
  x = r + (q \times y) \land y \leq r \Rightarrow (x = (r - y) + ((q + 1) \times y)) 
\end{cases}$$

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

The second we get by algebraic calculation

Therefore, the triple is valid

$$\{\text{true}\}$$

$r := x; q := 0$;

while $y \leq r$ do

$r := r - y; q := q + 1$

$$\{r < y \land x = r + (q \times y)\}$$
Now we’ll add two new features to our language:

▶ Assertion annotations
▶ Procedure

Assertion annotations take the form:

\[ \{ P \} \]

Semantically, treat them like runtime assertions

Execution halts if the expression isn’t true in current environment

Think of assertions as **formal comments**
imp: Procedures

```latex
\begin{align*}
\textbf{proc} \text{ LinearSearch}(& \ a : \text{ array, } l : \text{ int, } u : \text{ int, } e : \text{ int}) \\
\textbf{requires} \ 0 \leq l \land u < |a| \\
\textbf{ensures} \ (rv = 1) \iff (\exists i. l \leq i \leq u \land a[i] = e)
\end{align*}
```

We’ll consider programs with annotated procedures

- **Precondition** annotated with **requires**
- **Postcondition** annotated with **ensures**
- Free variables in the pre- and postconditions can be formal parameters
- Postcondition can also include special variable $rv$
- $rv$ stands for the return value
Proving Partial Correctness

Now, a program is partially correct if for each procedure $P$:
1. Whenever $P$’s preconditions are satisfied on entry
2. $P$’s postconditions are satisfied on exit

We’ll extend the approach we’ve talked about so far
1. Reduce the annotated program to a set of verification conditions
2. If all VCs are valid, then the program is correct

Our approach will be different:
- Use annotations to decompose the program into simpler parts
- Generate VC for each part in isolation, assuming each annotation holds
- Make sure that correctness of the whole follows from correctness of each part
A basic path is a sequence of instructions that:

- Begins at procedure precondition or loop invariant
- Ends at a loop invariant, assertion, or procedure postcondition
- Doesn’t cross loops: invariants only at beginning or end of path

Basic paths correspond to straight-line segments of code

Think of a Hoare triple over a sequence command:

\[ \{P\} c_1; c_2; \ldots; c_n \{Q\} \]

\(P, Q\) are pre-/postconditions, loop invariants, or assertion guards
**Basic Paths: Example**

```plaintext
proc LinearSearch(a : array, l, u, e)
  pre 0 ≤ l ∧ u < |a|
  post (rv = 1) ↔ (∃i. l ≤ i ≤ u ∧ a[i] = e)
  {%
    i := l;
    while (i ≤ u)
      {l ≤ i ∧ ∀j. l ≤ j < i → a[j] ≠ e}
      if (a[i] = e) return 1;
      i := i + 1;
  }
  return 0;
%
```

First basic path:

\[
\{0 ≤ l ∧ u < |a|\}
\]

\[
i := l
\]

\[
\{l ≤ i ∧ ∀j. l ≤ j < i → a[j] ≠ e\}
\]

Second basic path:

\[
\{l ≤ i ∧ ∀j. l ≤ j < i → a[j] ≠ e\}
\]

```plaintext
while (i ≤ u);
if (a[i] = e);
rv := 1;
{rv = 1} ↔ (∃i. l ≤ i ≤ u ∧ a[i] = e)
```
Guarded statements introduce assumptions about environment

We write these in basic paths using an assume statement

\textbf{assume} \ b \ \text{means:}

1. Rest of path executed only if \( b \) is true in current environment
2. When reasoning about rest of path, we can assume \( b \) holds
assume: Path Splitting

Each guarded statement introduces two assumptions

One where \textbf{assume} $b$ holds, one where \textbf{assume} $\neg b$ does

Continuing with our previous basic path, this gives us the next:

$$\{l \leq i \land \forall j.l \leq j < i \rightarrow a[j] \neq e\} \quad \{l \leq i \land \forall j.l \leq j < i \rightarrow a[j] \neq e\}$$

**assume** $i \leq u$;
**assume** $a[i] = e$;
$rv := 1$;
$$\{(rv = 1) \leftrightarrow (\exists i.l \leq i \leq u \land a[i] = e)\} \quad \{(rv = 1) \leftrightarrow (\exists i.l \leq i \leq u \land a[i] = e)\}$$

And one final path:

$$\{l \leq i \land \forall j.l \leq j < i \rightarrow a[j] \neq e\}$$

**assume** $i > u$;
$rv := 0$;
$$\{(rv = 1) \leftrightarrow (\exists i.l \leq i \leq u \land a[i] = e)\}$$
When we enumerate basic paths, we’ll follow a convention

Proceed depth-first through the program

When we encounter a guarded command:
  1. Assume that it holds first, then generate the ensuing paths
  2. Then assume it doesn’t hold, proceed as before
Recall the postcondition summarizes the relationships between:

▶ The procedure’s formal parameters
▶ The return value (special variable \( r_v \))

We replace procedure calls with postcondition assertions

But postcondition only holds when precondition is satisfied on entry

Introduce another basic path to ensure that the precondition holds

Replace formals in pre/postconditions with actuals appearing in call
Example: Procedure Calls

```latex
proc BinarySearch(a : array, l, u, e) 
  pre 0 \leq l \land u < |a| \land \text{sorted}(a, l, u) 
  post (rv = 1) \iff (\exists i. l \leq i \leq u \land a[i] = e) 
  \{ 
  if(l > u) return 0; 
  else \{ 
    m := (l + u)/2; 
    if(a[m] = e) return 1; 
    else if(a[m] < e) \{ 
      return BinarySearch(a, m + 1, u, e); 
    \} else \{ 
      return BinarySearch(a, l, m - 1, e); 
    \} 
  \} 
```

Matt Fredrikson
Deductive Verification
Example: Procedure Calls

First basic path:
\{0 \leq l \land u < |a| \land \text{sorted}(a, l, u)\}
assume \ l > u;
rv := 0;
\{(rv = 1) \iff (\exists i. l \leq i \leq u \land a[i] = e)\}

Second basic path:
\{0 \leq l \land u < |a| \land \text{sorted}(a, l, u)\}
assume \ l \leq u;
m := (l + u)/2;
assume \ a[m] = e;
rv := 1;
\{(rv = 1) \iff (\exists i. l \leq i \leq u \land a[i] = e)\}

Third basic path:
\{0 \leq l \land u < |a| \land \text{sorted}(a, l, u)\}
assume \ l \leq u;
m := (l + u)/2;
assume \ a[m] \neq e;
assume \ a[m] < e;
\{0 \leq m + 1 \land u < |a| \land \text{sorted}(a, m + 1, u)\}
Example: Procedure Calls

Fourth basic path:

\[
\{0 \leq l \land u < |a| \land \text{sorted}(a, l, u)\}
\]

**assume** \(l \leq u\);

\(m := (l + u)/2;\)

**assume** \(a[m] \neq e;\)

**assume** \(a[m] < e;\)

**assume** \((v_1 = 1) \iff (\exists i. m + 1 \leq i \leq u \land a[i] = e);\)

\(rv := v_1;\)

\(\{rv = 1\} \iff (\exists i. l \leq i \leq u \land a[i] = e)\}\)

Fifth basic path:

\[
\{0 \leq l \land u < |a| \land \text{sorted}(a, l, u)\}
\]

**assume** \(l \leq u;\)

\(m := (l + u)/2;\)

**assume** \(a[m] \neq e;\)

**assume** \(a[m] \geq e;\)

\(\{0 \leq l \land m - 1 < |a| \land \text{sorted}(a, l, m - 1)\}\)
Sixth basic path:

\[
\begin{align*}
\{0 \leq l \land u < |a| \land \text{sorted}(a, l, u)\} \\
\textbf{assume } l \leq u; \\
m := (l + u)/2; \\
\textbf{assume } a[m] \neq e; \\
\textbf{assume } a[m] \geq e; \\
\textbf{assume } (v_2 = 1) \iff (\exists i. l \leq i \leq m - 1 \land a[i] = e); \\
rv := v_2; \\
\{(rv = 1) \iff (\exists i. l \leq i \leq u \land a[i] = e)\}
\end{align*}
\]
Summary: Procedure Calls

Given a procedure $f$ with prototype

$$\text{proc } f(x_1, \ldots, x_n)$$
$$\text{pre } P[x_1, \ldots, x_n]$$
$$\text{post } Q[x_1, \ldots, x_n, rv]$$

When $f$ is called in context $w := f(e_1, \ldots, e_n)$;

Augment the calling context with an assertion:

$$\{P[e_1, \ldots, e_n]\};$$
$$w := f(e_1/x_1, \ldots, e_n/x_n);$$

In paths that pass through the call,

1. Create fresh variable $v$ to hold the return value
2. Replace call with an assumption of the postcondition:

$$\text{assume } G[e_1/x_1, \ldots, e_n/x_n, v/rv]$$
As we enumerate basic paths, we generate verification conditions

**Notice:** We only need to generate VCs for three command types

1. **Assignment:** we do this exactly as we did before
2. **Sequence:** same as before
3. **assume.** Recall, with assume we said that only paths satisfying the expression proceed past the command.

\[
\text{wlp(assume } b, Q) = b \rightarrow Q
\]

If \( b \rightarrow Q \) holds before, then satisfying \( b \) ensures that \( Q \) holds afterward

What about the “side conditions” we had for loops?
VC Generation: Loops

\[
\text{proc } f(x_1, \ldots, x_n) \\
\text{pre } P \\
\text{post } Q \\
\begin{aligned}
&\{ \\
&\text{while}(b) \{ I \} \{ \\
&\quad c; \\
&\} \\
&\} \\
\end{aligned} \\
\begin{aligned}
&\{ P \} \\
&\text{skip}; \\
&\{ I \} \\
\end{aligned} \\
\begin{aligned}
&\{ I \} \\
&\text{assume } b; \\
&\{ I \} \\
&c; \\
&\{ I \} \\
\end{aligned} \\
\begin{aligned}
&\{ I \} \\
&\text{assume } \neg b; \\
&\{ Q \} \\
\end{aligned}
VC Generation: Loops

\[
\{P\} \\
\text{skip;} \\
\{I\}
\]

\[P \rightarrow I\]

\[
\{I\} \\
\text{assume } b; \\
c; \\
\{I\}
\]

\[I \rightarrow \text{wl}(\text{assume } b, \text{wl}(c, I))\]
\[\Leftrightarrow\]
\[(I \land b) \rightarrow \text{wl}(c, I)\]

\[
\{I\} \\
\text{assume } b; \\
\{Q\}
\]

\[I \rightarrow \text{wl}(\text{assume } \neg b, Q)\]
\[\Leftrightarrow\]
\[(I \land \neg b) \rightarrow Q\]

These are same conditions as before!
Recall the first basic path:

\[
\begin{align*}
\{0 \leq l \land u < |a|\} \\
i := l \\
\{l \leq i \land \forall j.l \leq j < i \rightarrow a[j] \neq e\}
\end{align*}
\]

The VC for this is:

\[
0 \leq l \land u < |a| \rightarrow \text{wlp}(i := l, l \leq i \land \forall j.l \leq j < i \rightarrow a[j] \neq e)
\]

We have that:

\[
\begin{align*}
\text{wlp}(i := l, l \leq i \land \forall j.l \leq j < i \rightarrow a[j] \neq e) \\
\Leftrightarrow l \leq l \land \forall j.l \leq j < l \rightarrow a[j] \neq e \\
\Leftrightarrow \text{true} \land \forall j.\text{false} \rightarrow a[j] \neq e \\
\Leftrightarrow \text{true}
\end{align*}
\]

Our final condition is valid:

\[
0 \leq l \land u < |a| \Rightarrow \text{true}
\]
Example: Linear Search (2)

Recall the second basic path:

\[
\begin{align*}
\{ P : l \leq i \land \forall j. l \leq j < i \rightarrow a[j] \neq e \} \\
& c_1 : \textbf{assume} \ i \leq u; \\
& c_2 : \textbf{assume} \ a[i] = e; \\
& c_3 : rv := 1; \\
& \{ Q : (rv = 1) \leftrightarrow \exists j. l \leq j \leq u \land a[j] = e \}
\end{align*}
\]

The VC for this is:

\[
P \rightarrow \text{wlp}(c_1; c_2; c_3, Q) \Leftrightarrow P \rightarrow \text{wlp}(c_1, \text{wlp}(c_2, \text{wlp}(c_3, Q)))
\]

We have that:

\[
\begin{align*}
\text{wlp}(rv := 1, (rv = 1) & \leftrightarrow \exists j. l \leq j \leq u \land a[j] = e) \\
\Leftrightarrow (1 = 1) & \leftrightarrow \exists j. l \leq j \leq u \land a[j] = e \\
\Leftrightarrow \exists j. l \leq j \leq u \land a[j] = e
\end{align*}
\]

Our final condition is:

\[
0 \leq l \land u < |a| \Rightarrow \text{true}
\]
Recall the second basic path:

\[
\{ P : l \leq i \land \forall j.l \leq j < i \rightarrow a[j] \neq e \}
\]

\( c_1 : \text{assume } i \leq u ; \)
\( c_2 : \text{assume } a[i] = e ; \)
\( c_3 : \text{rv} := 1 ; \)
\( \{ Q : (\text{rv} = 1) \leftrightarrow \exists j.l \leq j \leq u \land a[j] = e \} \)

The VC for this is:

\[
P \rightarrow \text{wlp}(c_1 ; c_2 ; c_3 , Q) \leftrightarrow P \rightarrow \text{wlp}(c_1 , \text{wlp}(c_2 , \text{wlp}(c_3 , Q)))
\]

Moving on,

\[
\text{wlp}(c_1 , \text{wlp}(\text{assume } a[i] = e , \exists j.l \leq j \leq u \land a[j] = e))
\]
\( \equiv \text{wlp}(c_1 , a[i] = e \rightarrow \exists j.l \leq j \leq u \land a[j] = e) \)
\( \equiv \text{wlp}(\text{assume } i \leq u , a[i] = e \rightarrow \exists j.l \leq j \leq u \land a[j] = e) \)
\( \equiv i \leq u \rightarrow (a[i] = e \rightarrow \exists j.l \leq j \leq u \land a[j] = e) \)

Our final condition is:

\[
0 \leq l \land u < |a| \Rightarrow \text{true}
\]
Our final condition is:

\[(l \leq i \leq u \land \forall j. l \leq j < i \rightarrow a[j] \neq e) \rightarrow (a[i] = e \rightarrow \exists j. l \leq j \leq u \land a[j] = e)\]

\(\iff\) \[(l \leq i \leq u \land a[i] = e \land \forall j. l \leq j < i \rightarrow a[j] \neq e) \rightarrow \exists j. l \leq j \leq u \land a[j] = e\]

Notice that:

\[l \leq i \leq u \land a[i] = e \rightarrow \exists j. l \leq j \leq u \land a[j] = e\]

is valid

So, the condition is valid, and the corresponding triple is as well
Next Class

- Mid-term on Thursday
- Check out mid-term guide posted on Blackboard
- Come to office hours with questions