Proving Program Termination

Verifying Termination

- “Partial correctness” means that the program is correct, *provided* that it terminates.
- “Total correctness” is partial correctness and termination.

Verifying Termination

- The reason that termination is verified separately is that it requires coming up with a different sort of expression than an invariant.

- Such an expression is a "variant". It describes a program's inexorable movement toward a stopping point.

Variants

- Clearly the only cause for a (non-recursive) program's non-termination could lie in while-loops.
- A **variant** is some expression E such that:
  - $E \geq 0$ is *invariant*
  - The value of E decreases at every iteration.
- If a loop has a variant, then the loop must terminate.

Variant Example

$\{x == x_0 \wedge y == y_0 \wedge x_0 \geq 0\}$

```
while( x > 0 )
   {
   y = y + k;
   x = x-1;
   }
```

$\{y == y_0 + k^*x_0\}$

Here a **variant** for the loop would be x, since:

$x \geq 0$ is invariant, and

x decreases on each iteration.

Variant as a Triple

A sufficient condition for $\mathcal{E}$ to be a variant of

`while`(P) Body;

is that we be able to derive a triple:

$$\{\mathcal{E}_0 = \mathcal{E} \wedge \mathcal{E} > 0 \wedge P\} \text{ Body } \{\mathcal{E}_0 > \mathcal{E} \wedge \mathcal{E} \geq 0\}$$

where $\mathcal{E}_0$ is a free variable.

Variant as a Triple: Example

$\{\mathcal{E}_0 = \mathcal{E} \wedge \mathcal{E} > 0 \wedge P\}$ Block $\{\mathcal{E}_0 > \mathcal{E} \wedge \mathcal{E} \geq 0\}$

Consider the previous while program:

```
while( x > 0 )
   {
   y = y + k;
   x = x-1;
   }
```

$\{x_0 == x \wedge x > 0 \wedge x > 0\}$ `y = y + k; x = x-1;` $\{x_0 > x \wedge x \geq 0\}$
is the triple to be derived.

Variant as a Triple: Example

$\{x_0 == x \wedge x > 0 \wedge x > 0\}$ `y = y + k; x = x-1;` $\{x_0 > x \wedge x \geq 0\}$
is the triple to be derived.

Working backward from the post-condition, we need to derive:

$\{x_0 == x \wedge x > 0 \wedge x > 0\}$ `y = y + k;` $\{x_0 > x-1 \wedge x-1 \geq 0\}$

which follows from the implication rule if we can derive:

$\{x_0 == x \wedge x > 0 \wedge x > 0\} \rightarrow \{x_0 > x-1 \wedge x-1 \geq 0\}$

which follows directly (assuming x integer).

Exercise

- Derive a variant for the gcd program introduced earlier:

$\{x == x_0 \wedge y == y_0 \wedge x > 0 \wedge y > 0 \}$

```
while( x != y )
    if( x > y )
      x = x - y;
    else
      y = y - x;
```

$\{ x == gcd(x_0 , y_0) \}$

Exercise

- Can a variant be derived for the similar triple:

$\{x == x_0 \wedge y == y_0 \wedge x >= 0 \wedge y >= 0\}$

```
while( x != y )
    if( x > y )
      x = x - y;
    else
      y = y - x;
```

$\{ x == gcd(x_0 , y_0) \}$

Weakest Preconditions (WP)

Definition of WP

- For a program statement S and an expectation Q, the **weakest-precondition** is the weakest assumption P satisfying:

  $$\{P\}\ S\ \{Q\}$$

WP Calculus

- wp obeys some fairly obvious rules:
  - wp(x = E;, Q) = Q [E/X]   as in the assignment rule
  - wp(B_1; B_2, Q) =
  - wp(if(P) B_1; else B_2 , Q) =

- wp for a loop is harder, because it generally requires an infinite formula (unwind the loop as an infinite nest of conditions).

Example: WP for a Test

- {??}
  if(x > y) x = x-y; else y = y-x;
  {gcd(x, y) == z}

- wp is

wp's of the assignment statements

$$((x > y) \rightarrow gcd(x\text{-}y, y) == z)$$
$$\wedge\ (\neg(x > y) \rightarrow gcd(x, y\text{-}x) == z)$$

Example 2: WP for a Test

- {??}
  if(x > y) z = x; else z = y;
  {z == max(x, y)}

- wp is

  wp's of the assignment statements

  $((x > y) \rightarrow max(x, y) == x)$

  $\wedge\ (\neg(x > y) \rightarrow max(x, y) == y)$

  which simplifies to *true*.

When the *else* part is missing

- If the *else* part is missing, then T is effectively a "no-op", "skip", or trivial assignment x = x;
- Since wp(x = x; Q) = Q
- the wp for

  if(P) S

  is then

  $(P \rightarrow wp(S, Q))$

  $\wedge (\neg P \rightarrow Q)$

Example: WP for a Test without else

- {??}
  if(x > y) y = x;
  {y = max(x, y)}

- wp is

wp of the assignment statement

$$((x > y) \rightarrow x = max(x, x))$$
$$\wedge \quad (\neg(x > y) \rightarrow \ y = max(x, y))$$

- which simplifies to *true*.

Alternate WP for a Test

- $wp(if(P)\ S\ else\ T, Q) =$

  $(P \wedge wp(S, Q)) \vee$
  $(\neg P \wedge wp(T, Q))$

- To see that this is equivalent to the previous version, let $wp(S, Q)$ be A and $wp(T, Q)$ be B. Then we are asking whether

  $$(P \wedge A) \vee (\neg P \wedge B)$$

  is equivalent to

  $$(P \rightarrow A) \wedge (\neg P \rightarrow B)$$

Alternate WP for a Test

- $(P \wedge A) \vee (\neg P \wedge B) =? (P \rightarrow A) \wedge (\neg P \rightarrow B)$
- For P = true, this becomes A =? A.
- For P = false, this becomes B =? B
- Therefore the two forms are equivalent.

Sorting Example using WP (1)

- Suppose that $S(x, y)$ stands for

  `if( x > y ) {x, y = y, x;}`

  where by the assignment statement we mean ***parallel assignment***: the RHS's are evaluated then the values are *simultaneously* assigned to the LHS variables.

Sorting Example (2)

- S(x, y) stands for

  if(x > y) {x, y = y, x;}

- Using the WP for *if*, we have for any predicate P:

  wp(if(x > y) {x, y = y, x;}, P) =

$$[(x > y) \rightarrow P[x, y \leftarrow y, x]] \ \wedge \ [\neg(x > y) \rightarrow P]$$

Sorting Example (3)

- Suppose that we wish to show:

  {true}

  $S(x, y); S(y, z); S(x, y)$

  $\{x \leq y \wedge y \leq z\}$

- i.e. the sequence shown sorts three numbers.

Sorting Example (4)

- Working backward from the last statement $S(x, y)$:
  $wp(S(x, y), \{x \leq y \wedge y \leq z\}) =$

- $wp(\text{if}(x>y)\ \{x, y = y, x;\}, \{x \leq y \wedge y \leq z\}) =$

- $\quad[\quad (x > y) \rightarrow (y \leq x \wedge x \leq z)]$
  $\wedge[\neg(x > y) \rightarrow (x \leq y \wedge y \leq z)]$

- which simplifies (using reasoning about $\leq$ and $>$) to
  $[(x > y) \rightarrow (x \leq z)] \wedge [(x \leq y) \rightarrow (y \leq z)]$

- which simplifies to
  $(x \leq z) \wedge (y \leq z)$

Sorting Example (5)

- Working backward from the middle statement $S(y, z)$: $wp(S(y, z), (x \le z) \wedge (y \le z)) =$

- $[\quad (y > z) \rightarrow [(x \le y) \wedge (z \le y)]]$
  $\wedge\ [\neg(y > z) \rightarrow [(x \le z) \wedge (y \le z)]]$

- which is equivalent to
  $[(y > z) \rightarrow (x \le y)]$
  $\wedge\ [(y \le z) \rightarrow (x \le z)]$

Sorting Example (6)

- Working backward from the first statement $S(x, y)$:

$$wp(S(x, y), [(y > z) \rightarrow (x \leq y)] \wedge [(y \leq z) \rightarrow (x \leq z)])$$

=

- $$[\ (x > y) \rightarrow [(x > z) \rightarrow (y \leq x)] \wedge [(x \leq z) \rightarrow (y \leq z)]]$$
  $$\wedge[\neg(x > y) \rightarrow [(y > z) \rightarrow (x \leq y)] \wedge [(y \leq z) \rightarrow (x \leq z)]]$$

- which simplifies to
  $$[\ (x > y) \rightarrow [(x \leq z) \rightarrow (y \leq z)]]$$
  $$\wedge[\ (x \leq y) \rightarrow [(y \leq z) \rightarrow (x \leq z)]]$$

- which simplifies to

  true