Proving Programs Correct

Floyd-Hoare Logic:

- A formal system for proving correctness
- A program operates on state - moving through code changes state
- Hoare logic follows state changes through triples:

  $$\{P\}\ C\ \{Q\}$$

  where P is an assertion about the state before the execution of line C and Q is an assertion about the state after execution of C.

- **Examples:**

  $\{P\}$ nop $\{P\}$

  $\{$`a == 1, a != b`$\}$ `b := 2` $\{$`a == 1, a != b`$\}$

  $\{$`??`$\}$ `a := b+1` $\{$`a > 1`$\}$

Proving Programs Correct

Floyd-Hoare Logic:

- A formal system for proving correctness
- A program operates on state - moving through code changes state
- Hoare logic follows state changes through triples:

  $$\{P\}\ C\ \{Q\}$$

  where P is an assertion about the state before the execution of line C and Q is an assertion about the state after execution of C.

- **Examples:**

  $\{P\}$ nop $\{P\}$

  $\{$`a == 1, a != b`$\}$ `b := 2` $\{$`a == 1, a != b`$\}$

  $\{$`??`$\}$ `a := b+1` $\{$`a > 1`$\}$

  $\{$`b == 20`$\}$ `a := b+1` $\{$`a > 1`$\}$

  $\{$`b > 10`$\}$ `a := b+1` $\{$`a > 1`$\}$

  $\{$`b > 0`$\}$ `a := b+1` $\{$`a > 1`$\}$ Weakest precondition

Proving Programs Correct

Floyd-Hoare Logic:

- A formal system for proving correctness
- A program operates on state - moving through code changes state
- Hoare logic follows state changes through triples:

  $$\{P\}\ C\ \{Q\}$$

  where P is an assertion about the state before the execution of line C and Q is an assertion about the state after execution of C.

- **Examples:**

  $\{P\}$ nop $\{P\}$

  $\{$`a == 1, a != b`$\}$ `b := 2` $\{$`a == 1, a != b`$\}$

  $\{$`b > 10`$\}$ `a := b+1` $\{$`??`$\}$

Proving Programs Correct

Floyd-Hoare Logic:

- A formal system for proving correctness
- A program operates on state - moving through code changes state
- Hoare logic follows state changes through triples:

  $\{P\}\ C\ \{Q\}$

  where P is an assertion about the state before the execution of line C and Q is an assertion about the state after execution of C.
- **Examples:**

  $\{P\}$ nop $\{P\}$

  `{a == 1, a != b} b := 2 {a == 1, a != b}`

  `{b > 10} a := b+1 {??}`

  `{b > 10} a := b+1 {a > 0}`

  `{b > 10} a := b+1 {a > 11}` Strongest postcondition

Proving Programs Correct

Floyd-Hoare Logic:

- **A loop invariant:**

  Used to prove loop properties. Assertions on state entering the loop and guaranteed to be true at every iteration of the loop.

  The invariant will be the postcondition for the loop on exit.

- **Example:**

```
while (x < 10) x = x+1
```

Proving Programs Correct

Floyd-Hoare Logic:

- **A loop invariant:**

  Used to prove loop properties. Assertions on state entering the loop and guaranteed to be true at every iteration of the loop.

  The invariant will be the postcondition for the loop on exit.

- **Example:**

  ```
  while (x < 10) x = x+1
  ```

  Start with this:

  ```
  {x <= 10} while (x < 10) x = x+1 {??}
  ```

Proving Programs Correct

Floyd-Hoare Logic:

- **A loop invariant:**

  Used to prove loop properties. Assertions on state entering the loop and guaranteed to be true at every iteration of the loop.

  The invariant will be the postcondition for the loop on exit.

- **Example:**

  ```
  while (x < 10) x = x+1
  ```

  Start with this:

  {`x <= 10`} `while (x < 10) x = x+1` {??}

  Move inside the test:

  {`x < 10` $\wedge$ `x <= 10`} `x = x+1` {`x <= 10`}

Proving Programs Correct

Floyd-Hoare Logic:

- **A loop invariant:**

  Used to prove loop properties. Assertions on state entering the loop and guaranteed to be true at every iteration of the loop.

  The invariant will be the postcondition for the loop on exit.

- **Example:**

  ```
  while (x < 10) x = x+1
  ```

  Start with this:

  ```
  {x <= 10} while (x < 10) x = x+1 {??}
  ```

  Move inside the test:

  ```
  {x < 10 ∧ x <= 10} x = x+1 {x <= 10}
  ```

  Backing out:

  ```
  {x < 10 ∧ x <= 10} while (x<10) x=x+1 {¬(x < 10) ∧ x <= 10}
  ```

Proving Programs Correct

Floyd-Hoare Logic Rules:

- **Assignment:**

  Let $P[E/x]$ mean that in predicate P, expression E is substituted for symbol x where x is a free variable.

$$\frac{}{\{P[E/x]\}\ x := E\ \{P\}}$$

- **Example:**

  In $\{$`y == 10`$\}$ `x := y+1` $\{$`x == 11`$\}$

  $P = \{\texttt{x == 11}\}$

  $E/\texttt{x} = (\texttt{y} + \texttt{1})/\texttt{x} = \{\texttt{y+1}\}$

  $P[E/\texttt{x}] = \{\texttt{y+1 == 11}\} = \{\texttt{y == 10}\}$

Proving Programs Correct

Floyd-Hoare Logic Rules:

- **Composition:**

$$\frac{\{P\}\ C_1\ \{Q\},\quad \{Q\}\ C_2\ \{R\}}{\{P\}\ C_1; C_2\ \{R\}}$$

- **Example:**

  In $\{$`y == 10`$\}$ `x := y+1` $\{$`x == 11`$\}$ and

  $\{$`x == 11`$\}$ `z := x+1` $\{$`z == 12`$\}$

  We can substitute

  $\{$`y == 10`$\}$ `x := y+1; z := x+1` $\{$`z == 12`$\}$

Proving Programs Correct

Floyd-Hoare Logic Rules:

- **Condition:**

$$\frac{\{A \ \wedge \ B\} \ C_1 \ \{Q\}, \quad \{\neg A \ \wedge \ B\} \ C_2 \ \{Q\}}{\{B\} \text{ if } A \text{ then } C_1 \text{ else } C_2 \ \{Q\}}$$

- **Example:**

In $\{\mathtt{a} == \mathtt{T} \ \wedge \ \mathtt{b} == \mathtt{6} \ \wedge \ \mathtt{c} == \mathtt{10}\} \ \ \mathtt{x} := \mathtt{b}$

$\{(\mathtt{x} == \mathtt{6} \ \wedge \ \mathtt{a} == \mathtt{T}) \ \vee \ (\mathtt{x} == \mathtt{10} \ \wedge \ \mathtt{a} == \mathtt{F})\}$

and $\{\mathtt{a} == \mathtt{F} \ \wedge \ \mathtt{b} == \mathtt{6} \ \wedge \ \mathtt{c} == \mathtt{10}\} \ \ \mathtt{x} := \mathtt{c}$

$\{(\mathtt{x} == \mathtt{6} \ \wedge \ \mathtt{a} == \mathtt{T}) \ \vee \ (\mathtt{x} == \mathtt{10} \ \wedge \ \mathtt{a} == \mathtt{F})\}$

After applying the condition rule:

$\{(\mathtt{b} == \mathtt{6} \ \wedge \ \mathtt{c} == \mathtt{10})\}$

```
If a == T then x := b; else x := c
```

$\{(\mathtt{x} == \mathtt{6} \ \wedge \ \mathtt{a} == \mathtt{T}) \ \vee \ (\mathtt{x} == \mathtt{10} \ \wedge \ \mathtt{a} == \mathtt{F})\}$

Proving Programs Correct

Floyd-Hoare Logic Rules:

- **Consequence:**

$$\frac{P' \rightarrow P, \quad \{P\}\ C\ \{Q\}, \quad Q' \rightarrow Q}{\{P'\}\ C\ \{Q'\}}$$

- **Example:**

  Starting with:

  $$\{P\} = \{\mathtt{x} + \mathtt{1} == \mathtt{10}\} \ \ \mathtt{y} := \mathtt{x} + \mathtt{1} \ \ \{\mathtt{y} == \mathtt{10}\} = \{Q\}$$

  and

  $$P' = (\mathtt{x} == \mathtt{9} \ \rightarrow \ \mathtt{x} + \mathtt{1} == \mathtt{10}),$$

  $$Q' = (\mathtt{x} == \mathtt{9} \ \wedge \ \mathtt{y} == \mathtt{10} \ \rightarrow \ \mathtt{y} == \mathtt{10}),$$

  application of the consequence rule gives:

  $$\{\mathtt{x} == \mathtt{9}\} \ \ \mathtt{y} := \mathtt{x} + \mathtt{1} \ \ \{\mathtt{x} == \mathtt{9} \ \wedge \ \mathtt{y} == \mathtt{10}\}$$

Proving Programs Correct

Floyd-Hoare Logic Rules:

- **While:**

$$\frac{\{P \ \wedge \ A\}\ C\ \{P\}}{\{P\} \ \text{ while } A \text{ do } C\ \{\neg A \ \wedge \ P\}}$$

- **Example:**

Starting with:

$$\{\texttt{x} < \texttt{10} \ \wedge \ \texttt{x} <= \texttt{10}\} \ \ \texttt{x} = \texttt{x} + \texttt{1}; \ \ \{\texttt{x} <= \texttt{10}\}$$

After application of the while rule:

$$\{\texttt{x} <= \texttt{10}\} \ \ \texttt{while}(\texttt{x} < \texttt{10})\ \texttt{x} = \texttt{x} + \texttt{1}; \ \ \{\neg(\texttt{x} < \texttt{10}) \ \wedge \ \texttt{x} <= \texttt{10}\}$$

After application of the consequence rule:

$$\{\texttt{x} <= \texttt{10}\} \ \ \texttt{while}(\texttt{x} < \texttt{10})\ \texttt{x} = \texttt{x} + \texttt{1}; \ \ \{\texttt{x} == \texttt{10}\}$$

Proving Programs Correct

Weakest Precondition:

If Q is a predicate on states and C is a code fragment, then the weakest precondition for C with respect to Q is a predicate that is true for precisely those initial states in which C *must terminate* and *must produce a state satisfying* Q.

Notation:

$\mathtt{wp}(C, Q)$ - weakest precondition wrt Q, given code C.

Example:

$\{P\}\ \ \mathtt{a} := \mathtt{b} + \mathtt{1}\ \ \{\mathtt{a} > \mathtt{11}\}$

$P = \mathtt{wp}(\mathtt{a} := \mathtt{b} + \mathtt{1},\ \mathtt{a} > \mathtt{11}) = \mathtt{b} > \mathtt{10}$

Nomenclature:

`wp` is called a *predicate transformer*

Proving Programs Correct

Weakest Precondition:

Example:

$\{Q(\mathtt{y} + \mathtt{3} * \mathtt{z} - \mathtt{5})\}\ \ \mathtt{x} := \mathtt{y} + \mathtt{3} * \mathtt{z} - \mathtt{5}\ \ \{Q(\mathtt{x})\}$

Let v be the value assigned to `x`.

If $Q(\mathtt{x})$ is true after assignment so is $Q(v)$ (before and after)

Then $Q(\mathtt{y} + \mathtt{3} * \mathtt{z} - \mathtt{5})$ is true initially

So, $Q(\mathtt{x})$ holds after assignment iff $Q(\mathtt{y} + \mathtt{3} * \mathtt{z} - \mathtt{5})$ held

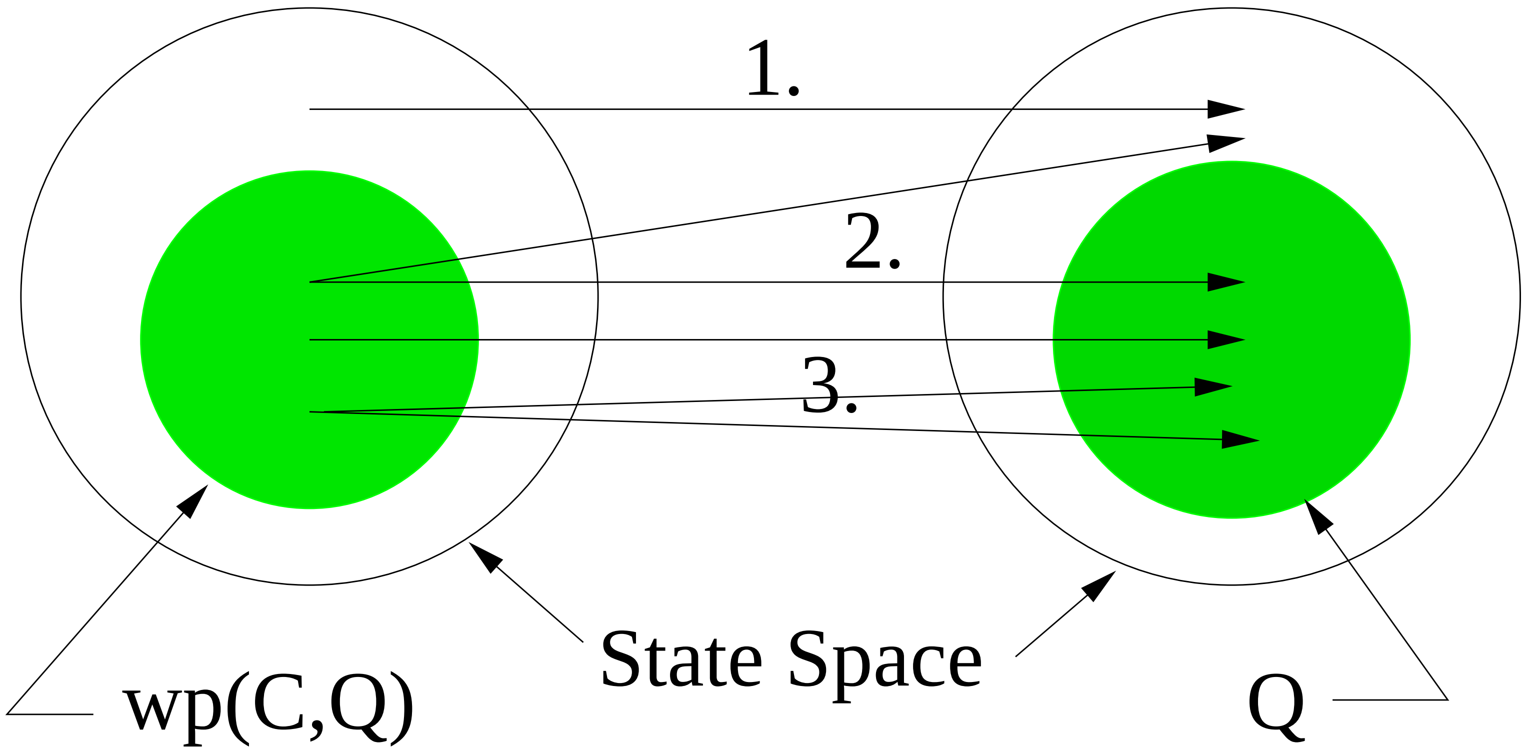
Usefulness:

Any predicate that implies $\mathtt{wp}(C,Q)$ also implies Q

Hence, if C is an entire program and Q specifies a property of C and if $\mathtt{wp}(C,Q)$ is known to be the weakest precondition from the initial state, ACL2 can be used to determine input values that cause Q to be true.

Proving Programs Correct

Weakest Precondition:



1. from initial state to state not satisfying Q
2. from initial state to some state statisfying Q
3. from initial state to states statisfying Q

Proving Programs Correct

Strongest Postcondition:

If P is a predicate on states and C is a code fragment, then the strongest postcondition for C with respect to P is a predicate that is implied by P when acted upon by C

Notation:

$\mathtt{sp}(C, P)$ - strongest postcondition wrt P, given code C.

Example:

$\{\mathtt{b} > \mathtt{10}\}\ \ \mathtt{a} := \mathtt{b} + \mathtt{1}\ \ \{Q\}$

$Q = \mathtt{sp}(\mathtt{a} := \mathtt{b} + \mathtt{1},\ \mathtt{b} > \mathtt{10}) = \mathtt{a} > \mathtt{11}$

Nomenclature:

sp is also called a *predicate transformer*

Proving Programs Correct

Predicate Transformers:

- PT semantics are a reformulation of Floyd-Hoare logic
- Used to implement an effective process for transforming F-H logic to predicate logic (so ACL2 can operate on it).

WP - nop:

$\texttt{wp}(nop, Q) = Q$

WP - abort:

$\texttt{wp}(abort, Q) = false$

WP - assignment:

$\texttt{wp}(x := E, Q) = \forall z.z == E \rightarrow Q[z/x]$

$\texttt{wp}(x := E, Q) = Q[E/x]$

ex.: $\texttt{wp}(x := x + 1, x > 10) = x + 1 > 10 \Leftrightarrow x > 9$

Proving Programs Correct

WP - composition:

$$\mathtt{wp}(C_1; C_2, Q) = \mathtt{wp}(C_1, \mathtt{wp}(C_2, Q))$$

ex.: $\mathtt{wp}(\mathtt{x} := \mathtt{x} + 2; \mathtt{y} := \mathtt{y} - 2, (\mathtt{x} + \mathtt{y} == 0))$

$$= \mathtt{wp}(\mathtt{x} := \mathtt{x} + 2, \mathtt{wp}(\mathtt{y} := \mathtt{y} - 2, (\mathtt{x} + \mathtt{y} == 0)))$$

$$= \mathtt{wp}(\mathtt{x} := \mathtt{x} + 2, (\mathtt{x} + (\mathtt{y} - 2) == 0))$$

$$= ((\mathtt{x} + 2) + \mathtt{y} - 2) == 0$$

$$= \mathtt{x} + \mathtt{y} == 0$$

WP - condition:

$$\mathtt{wp}(\mathtt{if}\ E\ \mathtt{then}\ C_1\ \mathtt{else}\ C_2, Q) = (E \rightarrow wp(C_1, Q))\ \wedge\ (\neg E \rightarrow wp(C_2, Q)))$$

ex.: $\mathtt{wp}(\mathtt{if}\ \mathtt{x}\ >\ 2\ \mathtt{then}\ \mathtt{y} := 1\ \mathtt{else}\ \mathtt{y} := -1,\ (\mathtt{y}\ >\ 0))$

$$= ((\mathtt{x}\ >\ 2) \rightarrow \mathtt{wp}(\mathtt{y} := 1, (\mathtt{y} > 0)))\ \wedge$$

$$(\neg(\mathtt{x}\ >\ 2) \rightarrow \mathtt{wp}(\mathtt{y} := -1, (\mathtt{y} > 0)))$$

$$= ((\mathtt{x}\ >\ 2) \rightarrow (1 > 0))\ \wedge\ (\neg(\mathtt{x}\ >\ 2) \rightarrow (-1 > 0))$$

$$= ((\mathtt{x}\ >\ 2) \rightarrow \mathtt{T})\ \wedge\ (\neg(\mathtt{x}\ >\ 2) \rightarrow \mathtt{F}))$$

$$= ((\mathtt{x}\ >\ 2) \rightarrow \mathtt{T})\ \wedge\ ((\mathtt{x}\ >\ 2) \vee \mathtt{F}))$$

$$= (\mathtt{x}\ >\ 2)$$

Proving Programs Correct

WP - while:

$\texttt{wp}(\texttt{while}\ E\ \texttt{do}\ C,\ Q) =$

$I\ \wedge$ — invariant true before execution

$\forall y.((E \wedge I) \rightarrow \texttt{wp}(C, I \wedge (x < y)))[y/x]\ \wedge$ — invariant and variant preserved

$\forall y.((\neg E \wedge I) \rightarrow Q)[y/x]$ — Q holds on exit

where $<$ is well-founded, y represents the state before loop execution

Alternatively,

$\texttt{wp}(\texttt{while}\ E\ \texttt{do}\ C,\ Q) = \exists k.\ (k \geq 0)\ \wedge\ P_k$ where

$P_0 = \neg E\ \wedge\ Q$ and

$P_{k+1} = E\ \wedge\ \texttt{wp}(C,\ P_k)$

Example:

$\texttt{wp}(\texttt{while n} > \texttt{0 do n} := \texttt{n} - \texttt{1},\ (\texttt{n} == \texttt{0}))$

$P_0 = \neg(\texttt{n}\ >\ \texttt{0})\ \wedge\ (\texttt{n} == \texttt{0}) = (\texttt{n} == \texttt{0})$

$P_1 = (\texttt{n}\ >\ \texttt{0})\ \wedge\ \texttt{wp}(\texttt{n} := \texttt{n} - \texttt{1}, (\texttt{n} == \texttt{0})) = (\texttt{n} == \texttt{1})$

$P_2 = (\texttt{n}\ >\ \texttt{0})\ \wedge\ \texttt{wp}(\texttt{n} := \texttt{n} - \texttt{1}, (\texttt{n} == \texttt{1})) = (\texttt{n} == \texttt{2}) \ldots$

$\exists k.\ (k \geq 0) \wedge P_k = (\texttt{n} >= \texttt{0}).$

Proving Programs Correct

Example:

$\texttt{wp}(\texttt{while n != 0 do n} := \texttt{n} - 1,\ (\texttt{n} == 0))$

$P_0 = \neg(\texttt{n != 0}) \ \wedge \ (\texttt{n} == 0) = (\texttt{n} == 0)$

$P_1 = (\texttt{n != 0}) \ \wedge \ \texttt{wp}(\texttt{n} := \texttt{n} - 1, (\texttt{n} == 0)) = (\texttt{n} == 1)$

$P_2 = (\texttt{n != 0}) \ \wedge \ \texttt{wp}(\texttt{n} := \texttt{n} - 1, (\texttt{n} == 1)) = (\texttt{n} == 2) \ldots$

$\exists k.\ (k \geq 0) \wedge P_k = (\texttt{n} >= 0).$

Proving Programs Correct

Example:

$1 + 3 + 5 \ldots + n = ((n + 1)/2)^2$

$\forall n.\ (n \geq 1) \rightarrow \sum_{i=1}^{n}(2 * i - 1) = n^2$

A program for implementing this:

```
{(n >= 0)}
i := 0; s := 0;
while i != n do
   i := i+1; s := s + (2*i - 1);
{(s == n*n)}
```

$P_0 = ((\mathtt{i} == \mathtt{n})\ \wedge\ (\mathtt{s} == \mathtt{n} * \mathtt{n}))$

$P_1 = ((\mathtt{i} == \mathtt{n} - 1)\ \wedge\ (\mathtt{s} == \mathtt{i} * \mathtt{i}))$

$P_2 = ((\mathtt{i} == \mathtt{n} - 2)\ \wedge\ (\mathtt{s} == \mathtt{i} * \mathtt{i}))\ldots$

$P_j = ((\mathtt{i} == \mathtt{n} - \mathtt{j})\ \wedge\ (\mathtt{s} == \mathtt{i} * \mathtt{i}))$

$\mathtt{wp}(\mathtt{while...},\ (\mathtt{s}\ ==\ \mathtt{n} * \mathtt{n})) = ((\mathtt{i} <= \mathtt{n}) \wedge (\mathtt{s} == \mathtt{i} * \mathtt{i}))$

$\mathtt{wp}(\mathtt{i} := 0; \mathtt{s} := 0,\ ((\mathtt{i} <= \mathtt{n}) \wedge (\mathtt{s} == \mathtt{i} * \mathtt{i})) = ((0 <= \mathtt{n}) \wedge (\mathtt{s} == 0))$

Proving Programs Correct

Computing P_1:

$$P_1 = (\mathtt{i} \,!= \ \mathtt{n}) \ \wedge \ \mathtt{wp}(\mathtt{i} := \mathtt{i} + 1;\ \mathtt{s} := \mathtt{s} + 2 * 1 - 1, ((\mathtt{i} == \mathtt{n}) \wedge (\mathtt{s} == \mathtt{n} * \mathtt{n}))$$

$$= (\mathtt{i} \,!= \ \mathtt{n}) \ \wedge \ \mathtt{wp}(\mathtt{i} := \mathtt{i} + 1, ((\mathtt{i} == \mathtt{n}) \wedge (\mathtt{s} + 2 * \mathtt{i} - 1 == \mathtt{n} * \mathtt{n}))$$

$$= (\mathtt{i} \,!= \ \mathtt{n}) \ \wedge \ \mathtt{wp}(\mathtt{i} := \mathtt{i} + 1, ((\mathtt{i} == \mathtt{n}) \wedge (\mathtt{s} == (\mathtt{i} - 1) * (\mathtt{i} - 1))$$

$$= (\mathtt{i} \,!= \ \mathtt{n}) \ \wedge \ (\mathtt{i} + 1 == \mathtt{n}) \wedge (\mathtt{s} == (\mathtt{i} + 1 - 1) * (\mathtt{i} + 1 - 1))$$

Proving Programs Correct

Useful special cases:

$\texttt{wp}(C, Q) = T$ C terminates with Q from any initial state

$\texttt{wp}(C, T) = T$ C terminates from any initial state

$\texttt{wp}(C, Q) = F$ C produces a state where Q is false from any initial state that results in C terminating

$\texttt{wp}(C, T) = F$ C does not terminate from any initial state

Other properties:

$\{\texttt{wp}(C, Q)\}\ C\ \{Q\}$

$\texttt{wp}(C, F) = F$

$\texttt{wp}(C, Q \ \wedge \ R) = (\texttt{wp}(C, Q) \ \wedge \ \texttt{wp}(C, R))$

$\texttt{if}\ Q \ \rightarrow \ R\ \texttt{then}\ \texttt{wp}(C, Q) \ \rightarrow \ \texttt{wp}(C, R)$

Proving Programs Correct

A multiply program:

{ F1=F1SAVE & F1<2**bits & F2<2**bits & LOW<2**bits }

```
        LDX  #bits   load the X register immediate with number bits
        LDA  #0      load the A register immediate with the value 0
LOOP    ROR  F1      rotate F1 right circular through the carry flag
        BCC  ZCOEF   branch on carry flag clear to ZCOEF
        CLC          clear the carry flag
        ADC  F2      add with carry F2 to the contents of A
ZCOEF   ROR  A       rotate A right circular through the carry flag
        ROR  LOW     rotate LOW right circular through the carry flag
        DEX          decrement the X register by 1
        BNE  LOOP    branch if X is non-zero to LOOP
```

{ LOW + 2**bits*A = F1SAVE*F2 }

- `A` is an 8 bit accumulator - for the high bits of the multiply
- `LOW` is an 8 bit accumulator - for the low bits of the multiply
- Right rotation of `A` and then `LOW` is a 16 bit right rotation through both
- There are `Z` and `C` bits, and registers `X`, `F1`, `F2`
- `ADC` adds `F2` and the `C` bit to `A`

Proving Programs Correct

Setup weakest preconditions:

{ F1=F1SAVE & F1<256 & F2<256 & LOW<256 }

	LDX	#bits	$S_1 \equiv [8/\mathtt{X}]S_2()$
	LDA	#0	$S_2 \equiv [0/\mathtt{A}]S_3()$
LOOP	ROR	F1	$S_3 \equiv [E_4/\mathtt{F1}, E_8/\mathtt{C}]S_4()$
	BCC	ZCOEF	$S_4 \equiv$ if $\mathtt{C} == 0$ $S_5()$ else $S_7()$
	CLC		$S_5 \equiv [0/\mathtt{C}]S_6()$
	ADC	F2	$S_6 \equiv [E_3/\mathtt{A}, E_7/\mathtt{C}, E_9/\mathtt{Z}]S_7()$
ZCOEF	ROR	A	$S_7 \equiv [E_2/\mathtt{A}, E_6/\mathtt{C}]S_8()$
	ROR	LOW	$S_8 \equiv [E_1/\mathtt{LOW}, E_5/\mathtt{C}]S_9(\mathtt{X}, \mathtt{LOW}, \mathtt{C})$
	DEX		$S_9(\mathtt{X}, \mathtt{LOW}, \mathtt{C}) \equiv [\mathtt{X}-\mathtt{1}/\mathtt{X}]S_{10}(\mathtt{X}, \mathtt{LOW}, \mathtt{C})$
	BNE	LOOP	$S_{10}(\mathtt{X}) \equiv$ if $\mathtt{Z}$==0 $S_3()$ else {LOW+256*A == F1SAVE*F2}

$Q = \{$ LOW + 256*A = F1SAVE*F2 $\}$

E_1: LOW/2 + C*128

E_2: A/2 + C*128

E_3: A+F2+C mod 256

E_4: F1/2 + C*128

E_9: A+F2+C mod 256 == 0

E_5: LOW mod 2

E_6: A mod 2

E_7: (A+F2+C)/256

E_8: F1 mod 2

Proving Programs Correct

Collapse non-branching instructions:

{ F1=F1SAVE & F1<256 & F2<256 & LOW<256 }

	LDX	#bits	$S_1 \equiv [8/\texttt{X}, 0/\texttt{A}] S_3()$
	LDA	#0	
LOOP	ROR	F1	$S_3 \equiv [E_4/\texttt{F1}, E_8/\texttt{C}] S_4()$
	BCC	ZCOEF	$S_4 \equiv$ if $\texttt{C} == 0$ $S_5()$ else $S_7()$
	CLC		$S_5 \equiv [0/\texttt{C}] S_6()$
	ADC	F2	$S_6 \equiv [E_3/\texttt{A}, E_7/\texttt{C}, E_9/\texttt{Z}] S_7()$
ZCOEF	ROR	A	$S_7 \equiv [E_2/\texttt{A}, E_6/\texttt{C}] S_8()$
	ROR	LOW	$S_8 \equiv [E_1/\texttt{LOW}, E_5/\texttt{C}] S_9(\texttt{X}, \texttt{LOW}, \texttt{C})$
	DEX		$S_9(\texttt{X}, \texttt{LOW}, \texttt{C}) \equiv [\texttt{X} - \texttt{1}/\texttt{X}] S_{10}(\texttt{X}, \texttt{LOW}, \texttt{C})$
	BNE	LOOP	$S_{10}(\texttt{X}) \equiv$ if $\texttt{Z}$==0 $S_3()$ else {`LOW+256*A == F1SAVE*F2`}

$Q = \{$ LOW + 256*A = F1SAVE*F2 $\}$

E_1:	`LOW`/2 + `C`*128	E_5:	`LOW` mod 2
E_2:	`A`/2 + `C`*128	E_6:	`A` mod 2
E_3:	`A`+`F2`+`C` mod 256	E_7:	(`A`+`F2`+`C`)/256
E_4:	`F1`/2 + `C`*128	E_8:	`F1` mod 2
E_9:	`A`+`F2`+`C` mod 256 == 0		

Proving Programs Correct

Collapse non-branching instructions:

{ F1=F1SAVE & F1<256 & F2<256 & LOW<256 }

	LDX	#bits	$S_1 \equiv [8/\mathtt{X}, 0/\mathtt{A}]S_3()$
	LDA	#0	
LOOP	ROR	F1	$S_3 \equiv [E_4/\mathtt{F1}, E_8/\mathtt{C}]S_4()$
	BCC	ZCOEF	$S_4 \equiv \text{if } \mathtt{C} == 0\ S_5() \text{ else } S_7()$
	CLC		$S_5 \equiv [0/\mathtt{C}]S_6()$
	ADC	F2	$S_6 \equiv [E_3/\mathtt{A}, E_7/\mathtt{C}, E_9/\mathtt{Z}]S_7()$
ZCOEF	ROR	A	$S_7 \equiv [E_2/\mathtt{A}, E_6/\mathtt{C}]S_8()$
	ROR	LOW	$S_8 \equiv [E_1/\mathtt{LOW}, E_5/\mathtt{C}, \mathtt{X}-\mathtt{1}/\mathtt{X}]S_{10}(\mathtt{X}, \mathtt{LOW}, \mathtt{C})$
	DEX		
	BNE	LOOP	$S_{10}(\mathtt{X}) \equiv \text{if } \mathtt{Z}{==}0\ S_3()$ else $\{\mathtt{LOW+256*A\ ==\ F1SAVE*F2}\}$

$Q = \{\ \text{LOW} + 256{*}\text{A} = \text{F1SAVE}{*}\text{F2}\ \}$

E_1: $\mathtt{LOW}/2 + \mathtt{C}^*128$

E_2: $\mathtt{A}/2 + \mathtt{C}^*128$

E_3: $\mathtt{A}{+}\mathtt{F2}{+}\mathtt{C} \bmod 256$

E_4: $\mathtt{F1}/2 + \mathtt{C}^*128$

E_5: $\mathtt{LOW} \bmod 2$

E_6: $\mathtt{A} \bmod 2$

E_7: $(\mathtt{A}{+}\mathtt{F2}{+}\mathtt{C})/256$

E_8: $\mathtt{F1} \bmod 2$

E_9: $\mathtt{A}{+}\mathtt{F2}{+}\mathtt{C} \bmod 256 == 0$

Proving Programs Correct

Collapse non-branching instructions:

{ F1=F1SAVE & F1<256 & F2<256 & LOW<256 }

	LDX	#bits	$S_1 \equiv [8/\mathtt{X}, 0/\mathtt{A}]S_3()$
	LDA	#0	
LOOP	ROR	F1	$S_3 \equiv [E_4/\mathtt{F1}, E_8/\mathtt{C}]S_4()$
	BCC	ZCOEF	$S_4 \equiv$ if $\mathtt{C} == 0\ S_5()$ else $S_7()$
	CLC		$S_5 \equiv [0/\mathtt{C}]S_6()$
	ADC	F2	$S_6 \equiv [E_3/\mathtt{A}, E_7/\mathtt{C}, E_9/\mathtt{Z}]S_7()$
ZCOEF	ROR	A	$S_7 \equiv [E_2/\mathtt{A}, E_6/\mathtt{C}, E_1/\mathtt{LOW}, \mathtt{X}-\mathtt{1}/\mathtt{X}]S_{10}()$
	ROR	LOW	
	DEX		
	BNE	LOOP	$S_{10}(\mathtt{X}) \equiv$ if $\mathtt{Z}{==}0\ S_3()$ else $\{\mathtt{LOW+256*A\ ==\ F1SAVE*F2}\}$

$Q = \{$ LOW + 256*A = F1SAVE*F2 $\}$

E_1:	$\mathtt{LOW}/2 + \mathtt{C}$*128	E_5:	$\mathtt{LOW}$ mod 2
E_2:	$\mathtt{A}/2 + \mathtt{C}$*128	E_6:	$\mathtt{A}$ mod 2
E_3:	$\mathtt{A}+\mathtt{F2}+\mathtt{C}$ mod 256	E_7:	$(\mathtt{A}+\mathtt{F2}+\mathtt{C})/256$
E_4:	$\mathtt{F1}/2 + \mathtt{C}$*128	E_8:	$\mathtt{F1}$ mod 2
E_9:	$\mathtt{A}+\mathtt{F2}+\mathtt{C}$ mod 256 == 0		

Proving Programs Correct

Collapse non-branching instructions:

{ F1=F1SAVE & F1<256 & F2<256 & LOW<256 }

	LDX	#bits	$S_1 \equiv [8/\texttt{X}, 0/\texttt{A}]S_3()$
	LDA	#0	
LOOP	ROR	F1	$S_3 \equiv [E_4/\texttt{F1}, E_8/\texttt{C}]S_4()$
	BCC	ZCOEF	$S_4 \equiv \text{if } \texttt{C} == 0\ S_5() \text{ else } S_7()$
	CLC		$S_5 \equiv [0/\texttt{C}, E_3/\texttt{A}, E_7/\texttt{C}, E_9/\texttt{Z}]S_7()$
	ADC	F2	
ZCOEF	ROR	A	$S_7 \equiv [E_2/\texttt{A}, E_6/\texttt{C}, E_1/\texttt{LOW}, \texttt{X}-\texttt{1}/\texttt{X}]S_{10}()$
	ROR	LOW	
	DEX		
	BNE	LOOP	$S_{10}(\texttt{X}) \equiv \text{if } \texttt{Z}{=}{=}0\ S_3()$ else {`LOW+256*A == F1SAVE*F2`}

$Q = \{\ \text{LOW} + 256{*}\text{A} = \text{F1SAVE}{*}\text{F2}\ \}$

E_1:	$\texttt{LOW}/2 + \texttt{C}{*}128$	E_5:	$\texttt{LOW} \bmod 2$
E_2:	$\texttt{A}/2 + \texttt{C}{*}128$	E_6:	$\texttt{A} \bmod 2$
E_3:	$\texttt{A+F2+C} \bmod 256$	E_7:	$(\texttt{A+F2+C})/256$
E_4:	$\texttt{F1}/2 + \texttt{C}{*}128$	E_8:	$\texttt{F1} \bmod 2$
E_9:	$\texttt{A+F2+C} \bmod 256 == 0$		

Proving Programs Correct

Put conditionals in proper form:

{ F1=F1SAVE & F1<256 & F2<256 & LOW<256 }

	LDX	#bits	$S_1 \equiv [8/\mathtt{X}, 0/\mathtt{A}]S_3()$
	LDA	#0	
LOOP	ROR	F1	$S_3 \equiv [E_4/\mathtt{F1}, E_8/\mathtt{C}]S_4()$
	BCC	ZCOEF	$S_4 \equiv \text{if } \mathtt{C} == 0 \ S_5() \text{ else } S_7()$
	CLC		$S_5 \equiv [0/\mathtt{C}, E_3/\mathtt{A}, E_7/\mathtt{C}, E_9/\mathtt{Z}]S_7()$
	ADC	F2	
ZCOEF	ROR	A	$S_7 \equiv [E_2/\mathtt{A}, E_6/\mathtt{C}, E_1/\mathtt{LOW}, \mathtt{X}-\mathtt{1}/\mathtt{X}]S_{10}()$
	ROR	LOW	
	DEX		
	BNE	LOOP	$S_{10}(\mathtt{X}) \equiv (\neg\mathtt{Z} \wedge S_3())\vee$ $(\mathtt{Z} \wedge \{\mathtt{LOW} + \mathtt{256} * \mathtt{A} == \mathtt{F1SAVE} * \mathtt{F2}\})$

Q = { LOW + 256*A = F1SAVE*F2 }

E_1:	LOW/2 + C*128	E_5:	LOW mod 2
E_2:	A/2 + C*128	E_6:	A mod 2
E_3:	A+F2+C mod 256	E_7:	(A+F2+C)/256
E_4:	F1/2 + C*128	E_8:	F1 mod 2
E_9:	A+F2+C mod 256 == 0		

Proving Programs Correct

Put conditionals in proper form:

{ F1=F1SAVE & F1<256 & F2<256 & LOW<256 }

	LDX	#bits	$S_1 \equiv [8/\mathtt{X}, 0/\mathtt{A}]S_3()$
	LDA	#0	
LOOP	ROR	F1	$S_3 \equiv [E_4/\mathtt{F1}, E_8/\mathtt{C}]S_4()$
	BCC	ZCOEF	$S_4 \equiv (\neg\mathtt{C} \wedge S_5()) \vee (\mathtt{C} \wedge S_7())$
	CLC		$S_5 \equiv [0/\mathtt{C}, E_3/\mathtt{A}, E_7/\mathtt{C}, E_9/\mathtt{Z}]S_7()$
	ADC	F2	
ZCOEF	ROR	A	$S_7 \equiv [E_2/\mathtt{A}, E_6/\mathtt{C}, E_1/\mathtt{LOW}, \mathtt{X}-\mathtt{1}/\mathtt{X}]S_{10}()$
	ROR	LOW	
	DEX		
	BNE	LOOP	$S_{10}(\mathtt{X}) \equiv (\neg\mathtt{Z} \wedge S_3())\vee$ $(\mathtt{Z} \wedge \{\mathtt{LOW} + \mathtt{256} * \mathtt{A} == \mathtt{F1SAVE} * \mathtt{F2}\})$

$Q = \{$ LOW + 256*A = F1SAVE*F2 $\}$

E_1:	LOW/2 + C*128	E_5:	LOW mod 2
E_2:	A/2 + C*128	E_6:	A mod 2
E_3:	A+F2+C mod 256	E_7:	(A+F2+C)/256
E_4:	F1/2 + C*128	E_8:	F1 mod 2
E_9:	A+F2+C mod 256 == 0		

Proving Programs Correct

Create recursive functions representing weakest preconditions:

$S_7 \equiv [E_2/\texttt{A}, E_6/\texttt{C}, E_1/\texttt{LOW}, \texttt{X} - \texttt{1}/\texttt{X}]S_{10}$

Proving Programs Correct

Create recursive functions representing weakest preconditions:

$$S_7 \equiv [E_2/\mathtt{A}, E_6/\mathtt{C}, E_1/\mathtt{LOW}, \mathtt{X}-\mathtt{1}/\mathtt{X}](\\ (\neg\mathtt{Z} \wedge S_3) \vee (\mathtt{Z} \wedge \{\mathtt{LOW}+\mathtt{256}*\mathtt{A}=\mathtt{F1SAVE}*\mathtt{F2}\}))$$

Proving Programs Correct

Create recursive functions representing weakest preconditions:

$$
\begin{aligned}
S_7 \equiv [&E_2/\mathtt{A}, E_6/\mathtt{C}, E_1/\mathtt{LOW}, \mathtt{X}-\mathtt{1}/\mathtt{X}](\\
&(\neg\mathtt{Z} \wedge [E_4/\mathtt{F1}, E_8/\mathtt{C}]((\neg\mathtt{C} \wedge S_5) \vee (\mathtt{C} \wedge S_7))) \vee \\
&(\mathtt{Z} \wedge \{\mathtt{LOW} + \mathtt{256} * \mathtt{A} = \mathtt{F1SAVE} * \mathtt{F2}\}))
\end{aligned}
$$

Proving Programs Correct

Create recursive functions representing weakest preconditions:

$$
\begin{array}{l}
S_7 \equiv [E_2/\mathtt{A}, E_6/\mathtt{C}, E_1/\mathtt{LOW}, \mathtt{X}-\mathtt{1}/\mathtt{X}](\\
\qquad (\neg \mathtt{Z} \wedge [E_4/\mathtt{F1}, E_8/\mathtt{C}](\\
\qquad\qquad (\neg \mathtt{C} \wedge [0/\mathtt{C}, E_3/\mathtt{A}, E_7/\mathtt{C}, E_9/\mathtt{Z}]S_7) \vee \\
\qquad\qquad (\mathtt{C} \wedge S_7))) \vee \\
\qquad (\mathtt{Z} \wedge \{\mathtt{LOW} + \mathtt{256} * \mathtt{A} = \mathtt{F1SAVE} * \mathtt{F2}\}))
\end{array}
$$

Proving Programs Correct

Create recursive functions representing weakest preconditions:

$$
\begin{aligned}
S_7 \equiv [&E_2/\mathtt{A}, E_6/\mathtt{C}, E_1/\mathtt{LOW}, \mathtt{X}-1/\mathtt{X}](\\
&(\neg\mathtt{Z} \wedge [E_4/\mathtt{F1}, E_8/\mathtt{C}](\\
&\quad(\neg\mathtt{C} \wedge [0/\mathtt{C}, E_3/\mathtt{A}, E_7/\mathtt{C}, E_9/\mathtt{Z}]S_7) \vee\\
&\quad(\mathtt{C} \wedge S_7))) \vee\\
&(\mathtt{Z} \wedge \{\mathtt{LOW} + 256 * \mathtt{A} = \mathtt{F1SAVE} * \mathtt{F2}\}))
\end{aligned}
$$

```
(DEFUN WP-ZCOEF (F1 C LOW A F1SAVE F2 X)
  (IF (EQUAL (DEC X) 0)
      (EQUAL (+ (* (+ (* C 128) (FLOOR A 2)) 256)
                (+ (* (MOD A 2) 128) (FLOOR LOW 2)))
             (* F1SAVE F2))
      (WP-ZCOEF
         (+ (* (MOD LOW 2) 128) (FLOOR F1 2))
         (* (MOD F1 2) (FLOOR (+ (+ (* C 128) (FLOOR A 2)) F2) 256))
         (+ (* (MOD A 2) 128) (FLOOR LOW 2))
         (+ (* (- 1 (MOD F1 2)) (+ (* C 128) (FLOOR A 2)))
            (* (MOD F1 2) (MOD (+ (+ (* C 128) (FLOOR A 2)) F2) 256)))
         F1SAVE
         F2
         (DEC X))))
```