HOARE LOGIC

VERIFICATION CONDITION GENERATION

- We have already seen that the weakest pre-condition operator can be used to prove Hoare Triples:
  - $\{P\}c\{Q\}$ iff $P \Rightarrow wp(Q, c)$
- Finding exact *wp* for loops is hard. We will instead use the loop invariant as an approximate *wp*.
  - $awp(Q, \text{while(F)@I do c}) = \text{I}$
  - Does this always hold?
- Also need to show that following side-conditions hold:
  - $\{\text{I} \wedge \text{F}\}\text{c}\{\text{I}\}$
  - $\text{I} \wedge \neg\text{F} \Rightarrow Q$

RELATION BETWEEN AWP AND WP

- Let us formally define *awp*:
  - $\forall \sigma \in awp(Q, c) \,.\, \forall \sigma' \,.\, (\sigma, c) \hookrightarrow^* (\sigma', skip) \rightarrow \sigma' \in Q$
  - Homework: Prove that this holds for $awp(Q,$ while(F)@I do c) = I, when the side-conditions hold.
- We defined $wp(Q, c) \triangleq \{\sigma \mid \forall \sigma' \,.\, (\sigma, c) \hookrightarrow^* (\sigma', \text{skip}) \rightarrow \sigma' \in Q\}$
  - $awp(Q, c) \subseteq wp(Q, c)$

RELATION BETWEEN AWP AND WP

- Let us formally define awp:
  - $\forall \sigma \in awp(Q, c) \,.\, \forall \sigma' \,.\, (\sigma, c) \hookrightarrow^* (\sigma', skip) \rightarrow \sigma' \in Q$
  - Homework: Prove that this holds for $awp(Q,$ while(F)@I do c) = I, when the side-conditions hold.
- We defined $wp(Q, c) \triangleq \{\sigma \mid \forall \sigma' \,.\, (\sigma, c) \hookrightarrow^* (\sigma', \text{skip}) \rightarrow \sigma' \in Q\}$
  - $awp(Q, c) \subseteq wp(Q, c)$
- $awp(i \geq 0,$ while(i < n)@(i >= 0) do i := i+1;) = ???

RELATION BETWEEN AWP AND WP

- Let us formally define awp:
  - $\forall \sigma \in awp(Q, c)\,.\,\forall \sigma'\,.\,(\sigma, c) \hookrightarrow^* (\sigma', skip) \rightarrow \sigma' \in Q$
  - Homework: Prove that this holds for $awp(Q,$ while(F)@I do c) = I, when the side-conditions hold.
- We defined $wp(Q, c) \triangleq \{\sigma \mid \forall \sigma'\,.\,(\sigma, c) \hookrightarrow^* (\sigma', \text{skip}) \rightarrow \sigma' \in Q\}$
  - $awp(Q, c) \subseteq wp(Q, c)$
- $awp(i \geq 0,$ while(i < n)@(i >= 0) do i := i+1;) = $i \geq 0$

RELATION BETWEEN AWP AND WP

- Let us formally define *awp*:
  - $\forall \sigma \in awp(Q, c) \,.\, \forall \sigma' \,.\, (\sigma, c) \hookrightarrow^* (\sigma', skip) \rightarrow \sigma' \in Q$
  - Homework: Prove that this holds for $awp(Q,$ while(F)@I do c) = I, when the side-conditions hold.
- We defined $wp(Q, c) \triangleq \{\sigma \mid \forall \sigma' \,.\, (\sigma, c) \hookrightarrow^* (\sigma', \text{skip}) \rightarrow \sigma' \in Q\}$
  - $awp(Q, c) \subseteq wp(Q, c)$
- $awp(i \geq 0,$ while(i < n)@(i >= 0) do i := i+1;) = $i \geq 0$
  - $wp(i \geq 0,$ while(i < n)@(i >= 0) do i := i+1;) = ???

RELATION BETWEEN AWP AND WP

- Let us formally define awp:
  - $\forall \sigma \in awp(Q, c) \,.\, \forall \sigma' \,.\, (\sigma, c) \hookrightarrow^* (\sigma', skip) \to \sigma' \in Q$
  - Homework: Prove that this holds for $awp(Q,$ while(F)@I do c) = I, when the side-conditions hold.
- We defined $wp(Q, c) \triangleq \{\sigma \mid \forall \sigma' \,.\, (\sigma, c) \hookrightarrow^* (\sigma', \text{skip}) \to \sigma' \in Q\}$
  - $awp(Q, c) \subseteq wp(Q, c)$
- $awp(i \geq 0,$ while(i < n)@(i >= 0) do i := i+1;) = $i \geq 0$
  - $wp(i \geq 0,$ while(i < n)@(i >= 0) do i := i+1;) =$n \geq 0 \vee i \geq 0$

VC GENERATION - I

- We define $VC(Q, c)$ to collect the side-conditions needed for verifying that Q holds after execution of c.
- For while(F)@I do c, there are two side-conditions:
  - $\{\mathrm{I} \wedge \mathrm{F}\}\mathrm{c}\{\mathrm{I}\}$
  - $\mathrm{I} \wedge \neg \mathrm{F} \Rightarrow Q$
- $\{\mathrm{I} \wedge \mathrm{F}\}\mathrm{c}\{\mathrm{I}\}$ is valid if $\mathrm{I} \wedge \mathrm{F} \Rightarrow awp(\mathrm{I}, \mathrm{c})$.
  - c may contain loops, so we also need to consider $VC(\mathrm{I}, \mathrm{c})$.
- Hence,
  $VC(\mathrm{Q}, \text{while(F)@I do c}) \triangleq (\mathrm{I} \wedge \neg \mathrm{F} \Rightarrow \mathrm{Q}) \wedge (\mathrm{I} \wedge \mathrm{F} \Rightarrow awp(\mathrm{I}, \mathrm{c})) \wedge VC(\mathrm{I}, \mathrm{c})$

VC GENERATION - II

- $VC(Q, \text{x:=e}) \triangleq \mathit{true}$
  - Also defined as *true* for all simple program commands (assert, assume, havoc).
- $VC(Q, c_1; c_2) \triangleq ???$

VC GENERATION - II

- $VC(Q, \text{x:=e}) \triangleq true$
    - Also defined as *true* for all simple program commands (assert, assume, havoc).
- $VC(Q, c_1; c_2) \triangleq VC(Q, c_2) \wedge VC(awp(Q, c_2), c_1)$

VC GENERATION - II

- $VC(Q, \text{x:=e}) \triangleq true$
    - Also defined as *true* for all simple program commands (assert, assume, havoc).
- $VC(Q, c_1; c_2) \triangleq VC(Q, c_2) \wedge VC(awp(Q, c_2), c_1)$
- $VC(Q, \text{if(F) then c}_1 \text{ else c}_2) \triangleq ???$

VC GENERATION - II

- $VC(Q, \text{x:=e}) \triangleq true$
  - Also defined as *true* for all simple program commands (assert, assume, havoc).
- $VC(Q, c_1; c_2) \triangleq VC(Q, c_2) \wedge VC(awp(Q, c_2), c_1)$
- $VC(Q, \text{if(F) then } \mathrm{c}_1 \text{ else } \mathrm{c}_2) \triangleq VC(Q, \mathrm{c}_1) \wedge VC(Q, \mathrm{c}_2)$

VC GENERATION - III

- $awp(Q, c) \triangleq wp(Q, c)$ except for while loops, for which $awp(Q,$ while(F)@I do c) = I.
- Putting it all together, $\{P\}c\{Q\}$ is valid if the following FOL formula is valid:
    - $(P \rightarrow awp(Q, c)) \wedge VC(Q, c)$

RELATION BETWEEN AWP AND HOARE TRIPLES

- What is the relation between $awp(Q, c)$ and validity of the Hoare Triple $\{P\}c\{Q\}$?
  - Is it possible that $P \rightarrow awp(Q, c)$ is valid and $\{P\}c\{Q\}$ is not valid?
  - Is it possible that $\{P\}c\{Q\}$ is valid and $\neg(P \rightarrow awp(Q, c))$ is satisfiable?
  - How about $\neg(P \rightarrow wp(Q, c))$?

VC GENERATION

SOUNDNESS AND COMPLETENESS

- Is the VC generation procedure sound?
  - Yes. Prove this!
- Is the VC generation procedure complete?
  - No. It is not even relatively complete.
  - The annotated loop invariant may not be strong enough.
- Can the VC generation procedure be fully automated?
  - Yes. Whole point of the exercise!

EXAMPLE

$\{true\}$

```
i := 1;
sum := 0;
while(i <= n) do
   j := 1;
   while(j <= i) do
      sum := sum + j; j := j + 1;
   i := i + 1;
```

$\{\mathtt{sum} \geq 0\}$

EXAMPLE

$\{true\}$

```
i := 1;
sum := 0;
while(i <= n)@(sum ≥ 0) do
    j := 1;
    while(j <= i)@(sum ≥ 0 ∧ j ≥ 0) do
        sum := sum + j; j := j + 1;
    i := i + 1;
```

$\{\texttt{sum} \geq 0\}$

- $VC(sum \geq 0,$outer loop$)$:
  - $sum \geq 0 \wedge i > n \rightarrow sum \geq 0$
  - $sum \geq 0 \wedge i \leq n \rightarrow sum \geq 0 \wedge 1 \geq 0$
  - $VC(sum \geq 0,$inner loop$)$

EXAMPLE

$\{true\}$

```
i := 1;
sum := 0;
while(i <= n)@(sum ≥ 0) do
    j := 1;
    while(j <= i)@(sum ≥ 0 ∧ j ≥ 0) do
        sum := sum + j; j := j + 1;
    i := i + 1;
```

$\{\mathtt{sum} \geq 0\}$

- $VC(sum \geq 0,$inner loop):
  - $sum \geq 0 \wedge j \geq 0 \wedge j > i \rightarrow sum \geq 0$
  - $sum \geq 0 \wedge j \geq 0 \wedge j \leq i \rightarrow sum + j \geq 0 \wedge j + 1 \geq 0$

EXAMPLE

$\{true\}$

```
i := 1;
sum := 0;
while(i <= n)@(sum ≥ 0) do
    j := 1;
    while(j <= i)@(sum ≥ 0 ∧ j ≥ 0) do
        sum := sum + j; j := j + 1;
    i := i + 1;
```

$\{\texttt{sum} \geq 0\}$

- Final Formula:
  - $true \rightarrow 0 \geq 0 \wedge VC(sum \geq 0,$outer loop$)$

ADDING FUNCTIONS TO IMP

p = F*

F = function $f(\mathtt{x}_1, \ldots, \mathtt{x}_n)\{\mathtt{c}\}$

c = x := exp | x := havoc

= | assume(F) | assert(F)

= | skip | c; c | if(F) then c else c | while(F) do c

= | x := $f(\mathtt{exp}_1, \ldots, \mathtt{exp}_n)$ | return exp

MODULAR VERIFICATION

- Each function is annotated with a pre-condition and a post-condition.
- Pre-condition specifies what is expected of the function's arguments
  - Formula in FOL whose free variables are the formal parameters of the function.
- Post-condition describes the function's return value
  - Formula in FOL whose free variables are the formal parameters and a special variable called ret.
- Together, pre-condition and post-condition specify a *contract.*
  - If the function is called with values which obey the pre-condition, then the output of the function will obey the post-condition.

VERIFYING FUNCTION CONTRACT

```
function f(x1,…,xn)
   requires(Pre)
   ensures(Post)
   {Body;}
```

- The function contract can be verified by proving the validity of the Hoare Triple $\{Pre\}\ Body\ \{Post\}$

VERIFYING FUNCTION CALLS

- The function body may have calls to other functions (or even itself)
    - $\{P\}x := f(e_1, \ldots, e_n)\{Q\}$
- If we can guarantee that the function's pre-condition holds before the call, then we can assume that the function's post-condition will hold after the call.
- We model the function call as follows:

VERIFYING FUNCTION CALLS

- The function body may have calls to other functions (or even itself)
  - $\{P\}x := f(e_1, \ldots, e_n)\{Q\}$
- If we can guarantee that the function's pre-condition holds before the call, then we can assume that the function's post-condition will hold after the call.
- We model the function call as follows:

```
assert(Pre[e1/x1,…,en/xn]);
assume(Post[tmp/ret,e1/x1,…,en/xn]);
y := tmp;
```

VERIFYING FUNCTION CALLS

- The function body may have calls to other functions (or even itself)
  - $\{P\}x := f(e_1, \ldots, e_n)\{Q\}$
- If we can guarantee that the function's pre-condition holds before the call, then we can assume that the function's post-condition will hold after the call.
- We model the function call as follows:

```
assert(Pre[e1/x1,…,en/xn]);
assume(Post[tmp/ret,e1/x1,…,en/xn]);
y := tmp;
```

- Why do we have to use tmp?
- What is the generated VC?

VERIFYING FUNCTION CALLS

- The function body may have calls to other functions (or even itself)
  - $\{P\} x := f(e_1, \ldots, e_n) \{Q\}$
- If we can guarantee that the function's pre-condition holds before the call, then we can assume that the function's post-condition will hold after the call.
- We model the function call as follows:

```
assert(Pre[e1/x1,…,en/xn]);
assume(Post[tmp/ret,e1/x1,…,en/xn]);
y := tmp;
```

- Why do we have to use tmp?
- What is the generated VC? $P \rightarrow (Pre \wedge (Post \rightarrow Q[tmp/y]))$