CS4004/CS4504: FORMAL VERIFICATION

Lectures 15: Hoare Logic

Vasileios Koutavas

School of Computer Science and Statistics
Trinity College Dublin
So far we have seen:

→ Rules for assignment, conditional, sequence and implication.

\[
\begin{align*}
(G[E/x]) & \quad x = E \quad \text{Asg} \\
(F \land B) & \quad C_1 \quad G \quad (F \land \neg B) \quad C_2 \quad G \quad \text{Cond} \\
(F) & \quad \text{if } B \text{ then } C_1 \text{ else } C_2 \quad G \\
(F) & \quad C_1 \quad \eta \quad C_2 \quad G \quad \text{Comp} \\
(F) & \quad C_1 ; C_2 \quad G \\
(F) & \quad C \quad G \quad \text{Imp} \\
(G \land B) & \quad C \quad G \quad \text{While} \\
(G) & \quad \text{while } B \{ C \} \quad G \land \neg B
\end{align*}
\]

→ Implication rule: allows us to transform the logical formulas in proofs.

→ Partial while rule: proves correctness without termination.
Proof technique: start from the post condition of the program and work backwards through the code. Find the weakest precondition of each of the commands.

→ **Assignment:** \((\eta?) \ x = E \ (G)\)
  set \(\eta = G[E/x]\)

→ **Composition:** \(\langle \eta? \rangle \ C_1 ; C_2 \ (G)\)
  find weakest precondition \(\langle \eta_1? \rangle \ C_2 \ (G)\)
  then find weakest precondition \(\langle \eta? \rangle \ C_1 \ (\eta_1)\)

→ **Conditional:** \(\langle \eta? \rangle \ if \ B \ then \ C_1 \ else \ C_2 \ (G)\)
  find weakest precondition \(\langle \eta_1? \rangle \ C_1 \ (G)\)
  find weakest precondition \(\langle \eta_2? \rangle \ C_1 \ (G)\)
  set \(\eta = (B \rightarrow \eta_1) \land (\neg B \rightarrow \eta_2)\)

\[\frac{(G[E/x]) \ x = E \ (G)}{(G[E/x])} \text{ ASG} \quad \frac{(F \land B) \ C_1 \ (G)}{(F)} \text{ if } B \text{ then } C_1 \text{ else } C_2 \ (G) \text{ COND} \]

\[\frac{(F) \ C_1 \ (\eta)}{(F)} \text{ C_1; C_2 (G)} \text{ COMP} \]
while loop: \((F?)\) while \(B\) \{\(C\) \(G\)\)

guess invariant \(F\)
Prove \(\vdash_{AR} F \land \neg B \rightarrow G\)
find weakest precondition \((\eta?)\) \(C\) \((F)\)
Prove \(\vdash_{AR} F \land B \rightarrow \eta\)

\[
\frac{(G \land B) \ C \ (G) \\
(G) \ \text{while} \ B \ \{C\} \ (G \land \neg B)}{(G) \ \text{while} \ B \ \{C\} \ (G \land \neg B)}
\]
Total Correctness
We define a **variant** $E$ in the hoare triple for while:

\[
\frac{(G \land B \land (0 \leq E = E_0)) \land C (G \land (0 \leq E < E_0))}{(G \land (0 \leq E)) \text{ while } B \{C\} (G \land \neg B)}
\]

**Collatz Conjecture:** the following program $C$ satisfies $\vdash_{\text{tot}} (j \vdash 0 < x \vdash j) C (j \vdash \top \vdash j)$ (i.e., the program terminates for all positive inputs)

```c
int c = x;
while (c != 1) {
    if (c % 2 == 0) { c = c / 2; }
    else { c = 3 * c + 1; }
}
```
We define a variant $E$ in the hoare triple for while:

$$
\frac{(G \land B \land (0 \leq E = E_0)) \land C (G \land (0 \leq E < E_0))}{(G \land (0 \leq E)) \text{ while } B \{C\} (G \land \neg B)}
$$

A variant is not always easy to find:

**Collatz Conjecture:** the following program $C$ satisfies $\vdash_{\text{tot}} (0 < x) C (\top)$ (i.e., the program terminates for all positive inputs)

```plaintext
c = x;
while (c != 1) {
    if ( c % 2 == 0 ) { c = c / 2; } 
    else { c = 3 * c + 1; }
}
```
EXAMPLE

We have proven \( \vdash_{\text{par}} (|x > 0|) \) Fact1 (\(|y = x!|\))

Now prove that \( \vdash_{\text{tot}} (|x > 0|) \) Fact1 (\(|T|\))

The above two imply \( \vdash_{\text{tot}} (|x > 0|) \) Fact1 (\(|y = x!|\))

Fact1 is the program:

\[
\begin{align*}
y &= 1; \\
z &= 0; \\
\text{while} \ (z \neq x) \{ \\
&\quad z = z + 1; \\
&\quad y = y \times z; \\
\}
\end{align*}
\]

\[
\frac{(G[E/x]) \ x = E (G)}{\text{ASG}} \quad \quad \frac{\begin{array}{c}
\{ F \} \ C_1 (G) \\
\{ F \} \ C_2 (G)
\end{array}}{\text{COMP}} \quad \quad \frac{\begin{array}{c}
\{ F \} \ C (G) \\
\{ F \} \ C (G')
\end{array}}{\text{IMPL}}
\]

\[
\frac{\begin{array}{c}
\{ F \} \ C_1 (\eta) \\
\{ F \} \ C_2 (G)
\end{array}}{\text{COMP}} \quad \quad \frac{\begin{array}{c}
\{ F \} \ C (G) \\
\{ F \} \ C (G')
\end{array}}{\text{IMPL}}
\]

\[
\frac{\begin{array}{c}
\{ F \} \ C_1 (\eta) \\
\{ F \} \ C_2 (G)
\end{array}}{\text{COMP}} \quad \quad \frac{\begin{array}{c}
\{ F \} \ C (G) \\
\{ F \} \ C (G')
\end{array}}{\text{IMPL}}
\]

\[
\frac{(G \land B \land (0 \leq E = E_0)) \ C (G \land (0 \leq E < E_0))}{\text{WHILE}}
\]
Prove that $\vdash_{\text{tot}} (\neg x = x_0 \land (x \geq 0))$ Fact2 $(y = x_0!)$ when Fact2 is the program:

```lang-plaintext
y = 1;
while (x != 0) {
    y = y * x;
    x = x - 1;
}
```

\[
\begin{align*}
\frac{(G[E/x])}{x = E (G)} & \quad \text{ASG} \\
\frac{(F) C_1 (\eta) \quad (\eta) C_2 (G)}{(F) C_1; C_2 (G)} & \quad \text{COMP} \\
\frac{(F) C_1 (G)}{\vdash_{\text{AR}} F' \rightarrow F} & \quad \text{IMPL} \\
\frac{(F) C (G)}{\vdash_{\text{AR}} G \rightarrow G'} & \quad \text{IMPL} \\
\frac{(G \land B \land (0 \leq E = E_0))}{(G \land (0 \leq E)) \quad \text{WHILE}} \\
\frac{(G \land (0 \leq E))}{(G \land \neg B)}
\end{align*}
\]
Here we assume our language has a \texttt{max} function. Consider the program \texttt{Max}: 

\begin{verbatim}
  k := 1;
  m := s[0];
  while (k != |s|) {
    m := max(m, s[k]);
    k := k + 1;
  }
\end{verbatim}

→ What does \texttt{Max} do?
→ Give correctness specification(s).
→ Give invariant(s).
→ Prove the specifications.
Let's assume $|s| = 4$.

→ when $k = 1$ and enter the while loop:

$$m = s[0]$$
Let's assume $|s| = 4$.

→ when $k = 1$ and enter the while loop:

$$m = s[0]$$

→ when $k = 2$ and enter the while loop:

$$m = \max \begin{pmatrix} s[0] \\ s[1] \end{pmatrix}$$
Let's assume $|s| = 4$.

$\rightarrow$ when $k = 1$ and enter the while loop:

$$m = s[0]$$

$\rightarrow$ when $k = 2$ and enter the while loop:

$$m = \max \begin{pmatrix} s[0] \\ s[1] \end{pmatrix}$$

$\rightarrow$ when $k = 3$ and enter the while loop:

$$m = \max \begin{pmatrix} s[0] \\ s[1] \\ s[2] \end{pmatrix}$$

$\rightarrow$ when $k = 4$ we exit the while loop:

$$m = \max \begin{pmatrix} s[0] \\ s[1] \\ s[2] \\ s[3] \end{pmatrix}$$
Let's assume $|s| = 4$.

→ when $k = 1$ and enter the while loop:

$$m = s[0]$$

→ when $k = 2$ and enter the while loop:

$$m = \max\begin{pmatrix} s[0] \\ s[1] \end{pmatrix}$$

→ when $k = 3$ and enter the while loop:

$$m = \max\begin{pmatrix} s[0] \\ s[1] \\ s[2] \end{pmatrix}$$

→ when $k = 4$ we exit the while loop:

$$m = \max\begin{pmatrix} s[0] \\ s[1] \\ s[2] \\ s[3] \end{pmatrix}$$
Different correctness aspects can be specified.
Different correctness aspects can be specified.

→ “At the end of Max, $m$ contains a number in $s$."

$G_1 \text{ def } i : ((0 < i < j \land s[j] = m) \land s[i]) \rightarrow \text{"At the end of Max, } m \text{ contains a number in } s."$

$G_2 \text{ def } i : ((0 < i < j \land s[j] \neq m) \land s[i]) \rightarrow \text{Give invariant(s) of the while loop. Prove} (j_0 < j_0 < j) \text{Max}(j G_i j), \text{for } i = 1; 2.
Different correctness aspects can be specified.

→ “At the end of Max, \( m \) contains a number in \( s \).”

\[
G_1 \overset{\text{def}}{=} \exists i. \ ((0 \leq i < |s|) \land (m = s[i]))
\]
Different correctness aspects can be specified.

→ "At the end of Max, \( m \) contains a number in \( s \)."

\[
G_1 \triangleq \exists i. ((0 \leq i < |s|) \land (m = s[i]))
\]

→ "At the end of Max, \( m \) contains a number greater than or equal to the any number in \( s \)."
Different correctness aspects can be specified.

→ “At the end of Max, $m$ contains a number in $s$.”

$$G_1 \overset{\text{def}}{=} \exists i. \ ((0 \leq i < |s|) \land (m = s[i]))$$

→ “At the end of Max, $m$ contains a number greater than or equal to the any number in $s$.”

$$G_2 \overset{\text{def}}{=} \forall i. \ ((0 \leq i \leq j < |s|) \rightarrow (m \geq s[i]))$$

→ Give invariant(s) of the while loop.

→ Prove $(0 < |s|) \ \text{Max} \ (|G_i|)$, for $i = 1, 2$. 
Here we assume our language has a \texttt{min} function.

Consider the program \texttt{MinSum}:

\begin{verbatim}
   k := 1;
   t := s[0];
   m := s[0];
   while (k != |s|) {
      t := min(t + s[k], s[k]);
      m := min(m, t);
      k := k + 1;
   }
\end{verbatim}

→ What does \texttt{MinSum} do?
→ Give correctness specification(s).
→ Give invariant(s).
→ Prove the specifications.
WHAT DOES minsum DO?

Let’s assume $|s| = 4$.

→ when $k = 1$ and enter the while loop:

\[ t = s[0] \]
\[ m = s[0] \]
WHAT DOES minsum DO?

Let’s assume $|s| = 4$.

→ when $k = 1$ and enter the while loop:

$t = s[0]$  \hspace{1cm} m = s[0]

→ when $k = 2$ and enter the while loop:

$t = \min\left(\frac{s[0]}{s[1]} + \frac{s[1]}{s[1]}\right)$  \hspace{1cm} m = \min\left(\frac{s[0]}{s[0]} + \frac{s[1]}{s[1]}\right)$
WHAT DOES $\text{minsum}$ DO?

Let’s assume $|s| = 4$.

$\rightarrow$ when $k = 1$ and enter the while loop:
$$t = s[0]$$
$$m = s[0]$$

$\rightarrow$ when $k = 2$ and enter the while loop:
$$t = \min\left( \begin{array}{cc} s[0] & + & s[1] \\ s[1] & + & s[2] \end{array} \right)$$
$$m = \min\left( \begin{array}{c} s[0] \\ s[1] \end{array} \right) + \begin{array}{c} s[1] \\ s[2] \end{array}$$

$\rightarrow$ when $k = 3$ and enter the while loop:
$$t = \min\left( \begin{array}{cc} s[0] & + & s[1] & + & s[2] \\ s[1] & + & s[2] & + & s[3] \end{array} \right)$$
$$m = \min\left( \begin{array}{ccc} s[0] & + & s[1] & + & s[2] \\ s[1] & + & s[2] & + & s[3] \end{array} \right)$$
Let’s assume $|s| = 4$.

→ when $k = 1$ and enter the while loop:

\[ t = s[0] \quad \text{and} \quad m = s[0] \]

→ when $k = 2$ and enter the while loop:

\[ t = \min \left( \frac{s[0]}{s[1]} + \frac{s[1]}{s[1]} \right) \quad \text{and} \quad m = \min \left( \frac{s[0]}{s[0]} + \frac{s[1]}{s[1]} \right) \]

→ when $k = 3$ and enter the while loop:

\[ t = \min \left( \frac{s[0]}{s[1]} + \frac{s[1]}{s[1]} + \frac{s[2]}{s[2]} \right) \quad \text{and} \quad m = \min \left( \frac{s[0]}{s[0]} + \frac{s[1]}{s[1]} + \frac{s[2]}{s[2]} \right) \]

→ when $k = 4$ and exit the while loop:

\[ t = \min \left( \frac{s[0]}{s[1]} + \frac{s[1]}{s[1]} + \frac{s[2]}{s[2]} + \frac{s[3]}{s[3]} \right) \quad \text{and} \quad m = \min \left( \frac{s[0]}{s[0]} + \frac{s[1]}{s[1]} + \frac{s[2]}{s[2]} + \frac{s[3]}{s[3]} \right) \]
**WHAT DOES minsum DO?**

**MinSum** computes in variable \( m \) the value:

\[
\min \left( s[0] \right. \\
\left. s[0] + s[1] \\
\left. s[1] + s[2] \\
\left. s[2] + s[3] \right)
\]

i.e., it computes the minimum sum of an interval \([i,j]\) in \( s \).
We can specify different aspects of the correctness of MinSum.
How can we specify the correctness of minsum?

We can specify different aspects of the correctness of MinSum.

→ “At the end of MinSum, m contains the sum of some interval [i,j] in s.”
HOW CAN WE SPECIFY THE CORRECTNESS OF \textbf{minsum}?

We can specify different aspects of the correctness of \textbf{MinSum}.

→ “At the end of \textbf{MinSum}, \( m \) contains the sum of some interval \([i,j]\) in \( s \).”

\[
G_1 \overset{\text{def}}{=} \exists i, j. \left( 0 \leq i \leq j < |s| \right) \land \left( m = \sum_{k=i}^{j} s[k] \right)
\]
We can specify different aspects of the correctness of MinSum.

→ “At the end of MinSum, \( m \) contains the sum of some interval \([i,j]\) in \( s \).”

\[
G_1 \overset{\text{def}}{=} \exists i,j. \left( 0 \leq i \leq j < |s| \right) \land \left( m = \sum_{k=i}^{j} s[k] \right)
\]

→ “At the end of MinSum, \( m \) contains a number smaller than or equal to the sum of any interval \([i,j]\) in \( s \).”
How can we specify the correctness of \texttt{minsum}?

We can specify different aspects of the correctness of \texttt{MinSum}.

\begin{itemize}
  \item “At the end of \texttt{MinSum}, \( m \) contains the sum of some interval \([i,j]\) in \( s \).”
  \begin{equation}
  G_1 \overset{\text{def}}{=} \exists i,j. \left( (0 \leq i \leq j < |s|) \land (m = \sum_{k=i}^{j} s[k]) \right)
  \end{equation}
  \item “At the end of \texttt{MinSum}, \( m \) contains a number smaller than or equal to the sum of any interval \([i,j]\) in \( s \).”
  \begin{equation}
  G_2 \overset{\text{def}}{=} \forall i,j. \left( (0 \leq i \leq j < |s|) \rightarrow (m \leq \sum_{k=i}^{j} s[k]) \right)
  \end{equation}
  \item Give invariant(s) of the while loop.
  \item Prove \((0 < |s|) \texttt{MinSum} (\{G_i\}), \text{ for } i = 1, 2.\)
Consider the program \texttt{LinSearch}:

\begin{verbatim}
var ind := 0;
found := 0;
while(ind < |s| && found==0) {
    if (s[ind] == n) {
        found := 1;
    } else {
        ind := ind + 1;
    }
}
\end{verbatim}

→ Give correctness specification(s).
→ Give invariant(s).
→ Prove the specifications.
Consider the program BinSearch:

lo := 0;
hi := |s| - 1;
found := 0;
while ((lo <= hi) & (found = 0)) {
    mid := (hi - lo) / 2;
    if (s[mid] < x) then {
        lo := mid + 1
    } else {
        if (s[mid] > x) then
            hi := mid - 1
        else
            found := 1
    }
}

→ Give correctness specification(s).
→ Give invariant(s).
→ Prove the specifications.