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.

$$\frac{}{(\!|G[E/x]|\!)\ x = E\ (\!|G|\!)}\ \textsc{Asg} \qquad \frac{(\!|F \land B|\!)\ C_1\ (\!|G|\!) \qquad (\!|F \land \neg B|\!)\ C_2\ (\!|G|\!)}{(\!|F|\!)\ \texttt{if}\ B\ \texttt{then}\ C_1\ \texttt{else}\ C_2\ (\!|G|\!)}\ \textsc{Cond}$$

$$\frac{(\!|F|\!)\ C_1\ (\!|\eta|\!) \qquad (\!|\eta|\!)\ C_2\ (\!|G|\!)}{(\!|F|\!)\ C_1; C_2\ (\!|G|\!)}\ \textsc{Comp}$$

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

$$\frac{\vdash_{AR} F' \to F \qquad (\!|F|\!)\ C\ (\!|G|\!) \qquad \vdash_{AR} G \to G'}{(\!|F'|\!)\ C\ (\!|G'|\!)}\ \textsc{Impl}$$

→ Partial while rule: proves correctness without termination.

$$\frac{(\!|G \land B|\!)\ C\ (\!|G|\!)}{(\!|G|\!)\ \texttt{while}\ B\ \{C\}\ (\!|G \land \neg B|\!)}\ \textsc{While}$$

WEAKEST PRECONDITION

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:** $(\!|\eta?|\!)\ C_1; C_2\ (\!|G|\!)$
  find weakest precondition $(\!|\eta_1?|\!)\ C_2\ (\!|G|\!)$
  then find weakest precondition $(\!|\eta?|\!)\ C_1\ (\!|\eta_1|\!)$
- **Conditional:** $(\!|\eta?|\!)$ `if` B `then` C_1 `else` C_2 $(\!|G|\!)$
  find weakest precondition $(\!|\eta_1?|\!)\ C_1\ (\!|G|\!)$
  find weakest precondition $(\!|\eta_2?|\!)\ C_1\ (\!|G|\!)$
  set $\eta = (B \to \eta_1) \wedge (\neg B \to \eta_2)$

$$\frac{}{(\!|G[E/x]|\!)\ x = E\ (\!|G|\!)}\ \textsc{Asg} \qquad \frac{(\!|F \wedge B|\!)\ C_1\ (\!|G|\!) \qquad (\!|F \wedge \neg B|\!)\ C_2\ (\!|G|\!)}{(\!|F|\!)\ \texttt{if}\ B\ \texttt{then}\ C_1\ \texttt{else}\ C_2\ (\!|G|\!)}\ \textsc{Cond}$$

$$\frac{(\!|F|\!)\ C_1\ (\!|\eta|\!) \qquad (\!|\eta|\!)\ C_2\ (\!|G|\!)}{(\!|F|\!)\ C_1; C_2\ (\!|G|\!)}\ \textsc{Comp}$$

- **while loop:** $(\!|F?|\!)$ `while` $B\ \{C\}\ (\!|G|\!)$
  guess invariant F
  Prove $\vdash_{\mathrm{AR}} F \wedge \neg B \rightarrow G$
  find weakest precondition $(\!|\eta?|\!)\ C\ (\!|F|\!)$
  Prove $\vdash_{\mathrm{AR}} F \wedge B \rightarrow \eta$

$$\frac{(\!|G \wedge B|\!)\ C\ (\!|G|\!)}{(\!|G|\!)\ \texttt{while}\ B\ \{C\}\ (\!|G \wedge \neg B|\!)}\ \textsc{While}$$

Total Correctness

We define a variant E in the hoare triple for while:

$$\frac{(\!|G \land B \land (0 \leq E = E_0)|\!)\ C\ (\!|G \land (0 \leq E < E_0)|\!)}{(\!|G \land (0 \leq E)|\!)\ \texttt{while}\ B\ \{C\}\ (\!|G \land \neg B|\!)}\ \textsc{While}$$

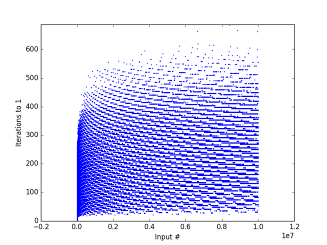
We define a variant E in the hoare triple for while:

$$\frac{(\!|G \wedge B \wedge (0 \leq E = E_0)|\!)\ C\ (\!|G \wedge (0 \leq E < E_0)|\!)}{(\!|G \wedge (0 \leq E)|\!)\ \texttt{while}\ B\ \{C\}\ (\!|G \wedge \neg B|\!)}\ \textsc{While}$$

A variant is not always easy to find:

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

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



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:

```
y = 1;
z = 0;
while (z != x) {
  z = z + 1;
  y = y * z;
}
```

$$\frac{}{(\!|G[E/x]|\!)\ x = E\ (\!|G|\!)}\ \textsc{Asg} \qquad \frac{(\!|F \wedge B|\!)\ C_1\ (\!|G|\!) \qquad (\!|F \wedge \neg B|\!)\ C_2\ (\!|G|\!)}{(\!|F|\!)\ \texttt{if}\ B\ \texttt{then}\ C_1\ \texttt{else}\ C_2\ (\!|G|\!)}\ \textsc{Cond}$$

$$\frac{(\!|F|\!)\ C_1\ (\!|\eta|\!) \qquad (\!|\eta|\!)\ C_2\ (\!|G|\!)}{(\!|F|\!)\ C_1; C_2\ (\!|G|\!)}\ \textsc{Comp} \qquad \frac{\vdash_{\text{AR}} F' \rightarrow F \qquad (\!|F|\!)\ C\ (\!|G|\!) \qquad \vdash_{\text{AR}} G \rightarrow G'}{(\!|F'|\!)\ C\ (\!|G'|\!)}\ \textsc{Impl}$$

$$\frac{(\!|G \wedge B \wedge (0 \leq E = E_0)|\!)\ C\ (\!|G \wedge (0 \leq E < E_0)|\!)}{(\!|G \wedge (0 \leq E)|\!)\ \texttt{while}\ B\ \{C\}\ (\!|G \wedge \neg B|\!)}\ \textsc{While}$$

Prove that $\vdash_{\text{tot}} (\!|(x = x_0) \land (x \geq 0)|\!)$ Fact2 $(\!|y = x_0!|\!)$ when Fact2 is the program:

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

$$\frac{}{(\!|G[E/x]|\!)\ x = E\ (\!|G|\!)}\ \textsc{Asg} \qquad \frac{(\!|F \land B|\!)\ C_1\ (\!|G|\!) \qquad (\!|F \land \neg B|\!)\ C_2\ (\!|G|\!)}{(\!|F|\!)\ \texttt{if}\ B\ \texttt{then}\ C_1\ \texttt{else}\ C_2\ (\!|G|\!)}\ \textsc{Cond}$$

$$\frac{(\!|F|\!)\ C_1\ (\!|\eta|\!) \qquad (\!|\eta|\!)\ C_2\ (\!|G|\!)}{(\!|F|\!)\ C_1; C_2\ (\!|G|\!)}\ \textsc{Comp} \qquad \frac{\vdash_{AR} F' \to F \qquad (\!|F|\!)\ C\ (\!|G|\!) \qquad \vdash_{AR} G \to G'}{(\!|F'|\!)\ C\ (\!|G'|\!)}\ \textsc{Impl}$$

$$\frac{(\!|G \land B \land (0 \leq E = E_0)|\!)\ C\ (\!|G \land (0 \leq E < E_0)|\!)}{(\!|G \land (0 \leq E)|\!)\ \texttt{while}\ B\ \{C\}\ (\!|G \land \neg B|\!)}\ \textsc{While}$$

Here we assume our language has a `max` function. Consider the program `Max`:

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

- → What does `Max` do?
- → Give correctness specification(s).
- → Give invariant(s).
- → Prove the specifications.

WHAT DOES `max` DO?

Let's assume $|s| = 4$.

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

$$m = s[0]$$

WHAT DOES max DO?

Let's assume $|s| = 4$.

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

$$m = s[0]$$

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

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

WHAT DOES max DO?

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:

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

WHAT DOES max DO?

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`."

Different correctness aspects can be specified.

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

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

Different correctness aspects can be specified.

- "At the end of `Max`, m contains a number in `s`."

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

- "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`."

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

- "At the end of `Max`, m contains a number greater than or equal to the any number in `s`."

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

- Give invariant(s) of the while loop.
- Prove $(\!|0 < |s||\!)$ `Max` $(\!|G_i|\!)$, for $i = 1, 2$.

Here we assume our language has a `min` function.

Consider the program `MinSum`:

```
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;
}
```

- What does `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:

$$t = s[0] \qquad\qquad m = s[0]$$

WHAT DOES `minsum` DO?

Let's assume $|s| = 4$.

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

$$t = s[0] \qquad\qquad m = s[0]$$

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

$$t = min \begin{pmatrix} s[0] & + & s[1] \\ & & s[1] \end{pmatrix} \qquad\qquad m = min \begin{pmatrix} s[0] & & \\ s[0] & + & s[1] \\ & & s[1] \end{pmatrix}$$

WHAT DOES `minsum` DO?

Let's assume $|s| = 4$.

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

$$t = s[0] \qquad\qquad m = s[0]$$

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

$$t = min \begin{pmatrix} s[0] & + & s[1] \\ & & s[1] \end{pmatrix} \qquad\qquad m = min \begin{pmatrix} s[0] & & \\ s[0] & + & s[1] \\ & & s[1] \end{pmatrix}$$

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

$$t = min \begin{pmatrix} s[0] & + & s[1] & + & s[2] \\ & & s[1] & + & s[2] \\ & & & & s[2] \end{pmatrix} \qquad m = min \begin{pmatrix} s[0] & & & & \\ s[0] & + & s[1] & & \\ & & s[1] & & \\ s[0] & + & s[1] & + & s[2] \\ & & s[1] & + & s[2] \\ & & & & s[2] \end{pmatrix}$$

WHAT DOES `minsum` DO?

Let's assume $|s| = 4$.

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

$$t = s[0] \qquad\qquad m = s[0]$$

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

$$t = min \begin{pmatrix} s[0] & + & s[1] \\ & & s[1] \end{pmatrix} \qquad m = min \begin{pmatrix} s[0] & & \\ s[0] & + & s[1] \\ & & s[1] \end{pmatrix}$$

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

$$t = min \begin{pmatrix} s[0] & + & s[1] & + & s[2] \\ & & s[1] & + & s[2] \\ & & & & s[2] \end{pmatrix} \qquad m = min \begin{pmatrix} s[0] & & & & \\ s[0] & + & s[1] & & \\ & & s[1] & & \\ s[0] & + & s[1] & + & s[2] \\ & & s[1] & + & s[2] \\ & & & & s[2] \end{pmatrix}$$

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

$$t = min \begin{pmatrix} s[0] & + & s[1] & + & s[2] & + & s[3] \\ & & s[1] & + & s[2] & + & s[3] \\ & & & & s[2] & + & s[3] \\ & & & & & & s[3] \end{pmatrix} \qquad m = min \begin{pmatrix} s[0] & & & & & & \\ s[0] & + & s[1] & & & & \\ & & s[1] & & & & \\ s[0] & + & s[1] & + & s[2] & & \\ & & s[1] & + & s[2] & & \\ & & & & s[2] & & \\ s[0] & + & s[1] & + & s[2] & + & s[3] \\ & & s[1] & + & s[2] & + & s[3] \\ & & & & s[2] & + & s[3] \\ & & & & & & s[3] \end{pmatrix}$$

`MinSum` computes in variable `m` the value:

$$
min \begin{pmatrix}
s[0] & & & & & & \\
s[0] & + & s[1] & & & & \\
 & & s[1] & & & & \\
s[0] & + & s[1] & + & s[2] & & \\
 & & s[1] & + & s[2] & & \\
 & & & & s[2] & & \\
s[0] & + & s[1] & + & s[2] & + & s[3] \\
 & & s[1] & + & s[2] & + & s[3] \\
 & & & & s[2] & + & s[3] \\
 & & & & & & s[3]
\end{pmatrix}
$$

i.e., it computes the mimum sum of an interval [i,j] in s.

HOW CAN WE SPECIFY THE CORRECTNESS OF `minsum`?

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`.”

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`."

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

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`."

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

- "At the end of `MinSum`, m contains a number smaller than or equal to the sum of any interval [i,j] in `s`."

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`."

$$G_1 \stackrel{\text{def}}{=} \exists i, j.\ \left((0 \le i \le j < |s|) \wedge (m = \sum_{k=i}^{j} s[k]) \right)$$

- "At the end of `MinSum`, m contains a number smaller than or equal to the sum of any interval [i,j] in `s`."

$$G_2 \stackrel{\text{def}}{=} \forall i, j.\ \left((0 \le i \le j < |s|) \rightarrow (m \le \sum_{k=i}^{j} s[k]) \right)$$

- Give invariant(s) of the while loop.
- Prove $(\!|0 < |s||\!)$ `MinSum` $(\!|G_i|\!)$, for $i = 1, 2$.

Consider the program `LinSearch`:

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

- → 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.