Overview  Hoare Triples  Proving assertions  Inductive Annotation  Hoare Logic  Weakest Preconditions  Completeness

Floyd-Hoare Style Program Verification

Deepak D’Souza

Department of Computer Science and Automation
Indian Institute of Science, Bangalore.

31 Oct 2018
Outline of these lectures

1. Overview
2. Hoare Triples
3. Proving assertions
4. Inductive Annotation
5. Hoare Logic
6. Weakest Preconditions
7. Completeness
Floyd-Hoare Style of Program Verification


Floyd-Hoare Logic

- A way of asserting properties of programs.
- Hoare triple: \( \{ A \} P \{ B \} \) asserts that “Whenever program \( P \) is started in a state satisfying condition \( A \), if it terminates, it will terminate in a state satisfying condition \( B \).”
- Example assertion: \( \{ n \geq 0 \} P \{ a = n + m \} \), where \( P \) is the program:
  ```
  int a := m;
  int x := 0;
  while (x < n) {
    a := a + 1;
    x := x + 1;
  }
  ```
- Inductive Annotation (“consistent interpretation”) (due to Floyd)
- A proof system (due to Hoare) for proving such assertions.
- A way of reasoning about such assertions using the notion of “Weakest Preconditions” (due to Dijkstra).
A simple programming language

- `skip`
- `x := e` (assignment)
- `if b then S else T` (if-then-else)
- `while b do S` (while)
- `S ; T` (sequencing)
Programs as State Transformers

View program $P$ as a partial map $[P] : \text{States} \rightarrow \text{States}$. (Assume that $\text{States} = \text{Var} \rightarrow \mathbb{Z}$)

All States

All States

State $s$

$P$

State $t$

\[ \langle x \mapsto 2, \ y \mapsto 10, \ z \mapsto 3 \rangle \]

\[ y := y + 1; \]

\[ z := x + y \]

\[ \langle x \mapsto 2, \ y \mapsto 11, \ z \mapsto 13 \rangle \]
Predicates on States

All States

States satisfying
Predicate A

Eg. $0 \leq x \land x < y$
Assertion of “Partial Correctness” $\{A\} P \{B\}$

$\{A\} P \{B\}$ asserts that “Whenever program $P$ is started in a state satisfying condition $A$, either it will not terminate, or it will terminate in a state satisfying condition $B$.”

All States

$\{10 \leq y\}$

$y := y + 1;$
$z := x + y$

$\{x < z\}$
Mathematical meaning of a Hoare triple

- View program $P$ as a relation

$$[P] \subseteq \text{States} \times \text{States}.$$ 
so that $(s, t) \in [P]$ iff it is possible to start $P$ in the state $s$ and terminate in state $t$.

- As usual here elements of $\text{States}$ are maps from variables to integers.

- $[P]$ is possibly non-deterministic, in case we also want to model non-deterministic assignment etc.

- Then the Hoare triple $\{A\} \ P \ \{B\}$ is true iff for all states $s$ and $t$: whenever $s \models A$ and $(s, t) \in [P]$, then $t \models B$.

- In other words $\text{Post}_{[P]}([A]) \subseteq [B]$. 
Example programs and pre/post conditions

// Pre: true
if (a <= b)
    min := a;
else
    min := b;
// Post: min <= a && min <= b

// Pre: 0 <= n
int a := m;
int x := 0;
while (x < n) {
    a := a + 1;
    x := x + 1;
}
// Post: a = m + n
Floyd’s style of proof: Inductive Annotation

START

$n \in J^+ \ (J^+ \ is \ the \ set \ of \ positive \ integers)$

$i \leftarrow 1$

$n \in J^+ \land i = 1$

$S \leftarrow 0$

$n \in J^+ \land i = 1 \land S = 0$

$n \in J^+ \land i \in J^+ \land i \leq n + 1 \land S = \sum_{j=1}^{i-1} a_j$

$i > n?$

Yes

$n \in J^+ \land i = n + 1 \land S = \sum_{j=1}^{i-1} a_j; \ i.e., \ S = \sum_{j=1}^{n} a_j$

No

HALT

$n \in J^+ \land i \in J^+ \land i \leq n \land S = \sum_{j=1}^{i-1} a_j$

$S \leftarrow S + a_i$

$n \in J^+ \land i \in J^+ \land i \leq n \land S = \sum_{j=1}^{i-1} a_j$

$i \leftarrow i + 1$

$n \in J^+ \land i \in J^+ \land 2 \leq i \leq n + 1 \land S = \sum_{j=1}^{i-1} a_j$
Inductive annotation based proof of a pre/post specification

- Annotate each program point \( i \) with a predicate \( A_i \)

- Successive annotations must be inductive:
  \[ A_i \land [S_i] \implies A'_{i+1}. \]

- Annotation is adequate:
  \( \text{Pre} \implies A_1 \) and \( A_n \implies \text{Post} \).

- Adequate annotation constitutes a proof of \( \{\text{Pre}\} \text{Prog} \{\text{Post}\} \).
Example of inductive annotation

To prove: \( \{ y > 10 \} \ y := y + 1; \ z := x + y \ \{ z > x \} \)
Exercise

Prove using Floyd’s inductive annotation:

\[
\{ n \geq 1 \} \quad P \quad \{ a = n! \},
\]

where \( P \) is the program:

\[
\begin{align*}
x & := n; \\
a & := 1; \\
\text{while } (x \geq 1) \{ \\
& \quad a := a \times x; \\
& \quad x := x - 1
\}
\end{align*}
\]

Assume that factorial is defined as follows:

\[
n! = \begin{cases} 
  n \times (n - 1) \times \cdots \times 1 & \text{if } n \geq 1 \\
  1 & \text{if } n = 0 \\
  -1 & \text{if } n < 0
\end{cases}
\]
Exercise

Prove using Floyd’s inductive annotation:

\[ \{ n \geq 1 \} \ P \ \{ a = n! \} , \]

where \( P \) is the program:

\begin{align*}
S1: & \quad x := n; \\
S2: & \quad a := 1; \\
S3: & \quad \text{while} \ (x \geq 1) \ {\{ } \\
S4: & \quad a := a \times x; \\
S5: & \quad x := x - 1 \\
\end{align*}

Assume that factorial is defined as follows:

\[ n! = \begin{cases} 
  n \times (n-1) \times \cdots \times 1 & \text{if } n \geq 1 \\
  1 & \text{if } n = 0 \\
  -1 & \text{if } n < 0 
\end{cases} \]
Solution

Need a loop invariant $P$ satisfying:

1. $\{ n \geq 1 \} \ S1; \ S2 \ \{ P \}$
2. $\{ P \land (x \geq 1) \} \ S4; \ S5 \ \{ P \}$
3. $(P \land \neg(x \geq 1)) \implies (a = n!)$

A potential $P$: $(x \geq 0) \land (a \times x! = n!)$. 
int a := m;
int x := 0;
while (x < n) {
    a := a + 1;
    x := x + 1;
}

Hoare’s view: Program as a composition of statements
Hoare’s view: Program as a composition of statements

int a := m;
int x := 0;
while (x < n) {
    a := a + 1;
    x := x + 1;
}

S1: int a := m;
S2: int x := 0;
S3: while (x < n) {
    a := a + 1;
    x := x + 1;
}

Program is S1;S2;S3
Proof rules of Hoare Logic

Axiom of Valid formulas:

\[
\begin{align*}
\text{provided } & \models A \quad \text{(i.e. } A \text{ is a valid logical formula, eg. } x > 10 \implies x > 0). \\
\text{Skip: } & \\
\{ A \} \text{ skip } \{ A \} \\
\text{Assignment } & \\
\{ A[e/x] \} \ x := e \ \{ A \}
\end{align*}
\]
Proof rules of Hoare Logic

If-then-else:

\[
\frac{\{P \land b\} S \{Q\}, \{P \land \neg b\} T \{Q\}}{\{P\} \text{ if } b \text{ then } S \text{ else } T \{Q\}}
\]

While (here \(P\) is called a \textit{loop invariant})

\[
\frac{\{P \land b\} S \{P\}}{\{P\} \text{ while } b \text{ do } S \{P \land \neg b\}}
\]

Sequencing:

\[
\frac{\{P\} S \{Q\}, \{Q\} T \{R\}}{\{P\} S; T \{R\}}
\]

Weakening:

\[
P \implies Q, \{Q\} S \{R\}, R \implies T \quad \Rightarrow \quad \{P\} S \{T\}
\]
A predicate $P$ is a **loop invariant** for the while loop:

```plaintext
while (b) {
    S
}
```

If $\{P \land b\} S \{P\}$ holds.

If $P$ is a loop invariant then we can infer that:

```plaintext
\{P\} while b do S \{P \land \neg b\}
```
Some examples to work on

Use the rules of Hoare logic to prove the following assertions:

1. \( \{ x \geq 3 \} \quad x := x + 2 \quad \{ x \geq 5 \} \)
2. \( \{(y \leq 0) \land (-1 < x)\} \quad \text{if} \ (y < 0) \ \text{then} \ x := x + 1 \ \text{else} \ x := y \quad \{ 0 \leq x \} \)
3. \( \{ x \leq 0 \} \quad \text{while} \ (x \leq 5) \ \text{do} \ x := x + 1 \quad \{ x = 6 \} \)
Example proof using Hoare Logic

1. \{n \geq 0\} S1 \quad \{n \geq 0 \land a = m\}
2. \{n \geq 0 \land a = m\} S2 \quad \{n \geq 0 \land a = m \land x = 0\}
3. \{a = m + x \land 0 \leq x \leq n \land x < n\} S4; S5
   \{a = m + x \land 0 \leq x \leq n\}
4. \{a = m + x \land 0 \leq x \leq n\} S3
   \{a = m + x \land 0 \leq x \leq n \land x \geq n\}
5. \{n \geq 0\} S1; S2 \quad \{n \geq 0 \land a = m \land x = 0\} (From Seq rule, 1 and 2)
6. \(n \geq 0 \land a = m \land x = 0\) \implies (a = m + x \land 0 \leq x \leq n) (From logical axiom)
7. \{n \geq 0\} S1; S2 \quad \{a = m + x \land 0 \leq x \leq n\} (From Weakening rule, 5 and 6)
8. \{n \geq 0\} (S1; S2); S3
   \{a = m + x \land 0 \leq x \leq n \land x \geq n\} (From Seq rule, 7, 4)
9. \(a = m + x \land 0 \leq x \leq n \land x \geq n\) \implies (a = m + n)
10. \{n \geq 0\} (S1; S2); S3 \quad \{a = m + n\} (From Weakening rule, 8, 9).

Program is S1; S2; S3

\begin{verbatim}
// pre: n >= 0
S1: int a := m;
S2: int x := 0;
S3: while (x < n) {
  S4:  a := a + 1;
  S5:  x := x + 1;
}
// post: a = m + n
\end{verbatim}
Exercise

Prove using Hoare logic:

\[ \{ n \geq 1 \} \ P \ \{ a = n! \} , \]

where \( P \) is the program:

\[
\begin{align*}
x & := n; \\
a & := 1; \\
\text{while } (x \geq 1) \{ \\
& a := a \times x; \\
& x := x - 1 \\
\}
\end{align*}
\]

Assume that factorial is defined as follows:

\[
n! = \begin{cases} 
  n \times (n - 1) \times \cdots \times 1 & \text{if } n \geq 1 \\
  1 & \text{if } n = 0 \\
  -1 & \text{if } n < 0
\end{cases}
\]
Exercise

Prove using Hoare logic:

\[
\{ n \geq 1 \} \ P \ \{ a = n! \},
\]

where \( P \) is the program:

S1: \( x := n; \)
S2: \( a := 1; \)
S3: \( \text{while} \ (x \geq 1) \ {\{} \)
S4: \( a := a \times x; \)
S5: \( x := x - 1 \)
{\}}

Assume that factorial is defined as follows:

\[
n! = \begin{cases} 
  n \times (n-1) \times \cdots \times 1 & \text{if } n \geq 1 \\
  1 & \text{if } n = 0 \\
  -1 & \text{if } n < 0 
\end{cases}
\]
Need a loop invariant $P$ satisfying:

1. $\{n \geq 1\} \ S1; \ S2 \ \{P\}$
2. $\{P \land (x \geq 1)\} \ S4; \ S5 \ \{P\}$
3. $(P \land \neg(x \geq 1)) \implies (a = n!)$

A potential $P$: $(x \geq 0) \land (a \times x! = n!)$. 
**Soundness and Completeness**

**Soundness:** If our proof system proves \{A\} P \{B\} then \{A\} P \{B\} indeed holds.

**Completeness:** If \{A\} P \{B\} is true then our proof system can prove \{A\} P \{B\}.

- Floyd proof style is sound since any execution must stay within the annotations. Complete because the “collecting” set is an adequate inductive annotation for any program and any true pre/post condition.

- Hoare logic is sound, essentially because the individual rules can be seen to be sound.

- For completeness of Hoare logic, we need weakest preconditions.
Weakest Precondition \( WP(P, B) \)

\( WP(P, B) \) is “a predicate that describes the exact set of states \( s \) such that when program \( P \) is started in \( s \), if it terminates it will terminate in a state satisfying condition \( B \).”

\[
\begin{align*}
\{ x < z \} \\
\text{y := y + 1; } \\
z := x + y; \\
\{ 10 < y \}
\end{align*}
\]
Exercise: Give “weakest” preconditions

1. \(? \) \{ x := x + 2 \} \{ x \geq 5 \}
Exercise: Give “weakest” preconditions

1 \{ x \geq 3 \} \ x := x + 2 \ \{ x \geq 5 \}

2 \{ ? \}
   if (y < 0) then x := x+1 else x := y
   \{ x > 0 \}
Exercise: Give “weakest” preconditions

1 \{ x \geq 3\} \ x := x + 2 \ \{x \geq 5\}

2 \{ (y < 0 \land x > -1) \lor (y > 0)\}
   if (y < 0) then x := x+1 else x := y
   \{x > 0\}

3 \{? \} \text{ while } (x \leq 5) \text{ do } x := x+1 \ \{x = 6\}
Exercise: Give “weakest” preconditions

1. \{ x \geq 3 \} \ x := x + 2 \ { x \geq 5 \}

2. \{ (y < 0 \land x > -1) \lor (y > 0) \} 
   if (y < 0) then x := x+1 else x := y 
   \{ x > 0 \}

3. \{ x \leq 6 \} \ while (x \leq 5) do x := x+1 \ { x = 6 \}
Exercise: How will you define $WP(P, B)$?
Exercise: How will you define $WP(P, B)$?

$WP(P, B) = \{ s \mid \forall t : (s, t) \in [P] \text{ we have } t \models B\}$
Weakest preconditions give us a way to:

- Check inductiveness of annotations

\[ \{A_i\} \text{ } S_i \text{ } \{A_{i+1}\} \text{ iff } A_i \implies WP(S_i, A_{i+1}) \]

- Reduce the amount of user-annotation needed
  - Programs **without loops** don’t need any user-annotation
  - For programs with loops, user only needs to provide **loop invariants**
Checking \{A\} \quad P \quad \{B\} \quad using \quad WP

Check that

\((y > 10) \implies WP(P, z > x)\)
WP rules

- Hoare’s rules for skip, assignment, and if-then-else are already WP rules.
- For Sequencing:

\[ WP(S; T, B) = WP(S, WP(T, B)) \].
Weakest Precondition for while statements

- We can “approximate” \( WP(\text{while } b \text{ do } c) \).
- \( WP_i(w, A) = \) the set of states from which the body \( c \) of the loop is either entered more than \( i \) times or we exit the loop in a state satisfying \( A \).
- \( WP_i \) defined inductively as follows:

\[
\begin{align*}
WP_0 &= b \lor A \\
WP_{i+1} &= (\neg b \land A) \lor (b \land WP(c, WP_i))
\end{align*}
\]

- Then \( WP(w, A) \) can be shown to be the “limit” or least upper bound of the chain \( WP_0(w, A), WP_1(w, A), \ldots \) in a suitably defined lattice (here the join operation is “And” or intersection).
Consider the program $w$ below:

\[
\text{while } (x \geq 10) \text{ do} \\
\quad x := x - 1
\]

- What is the weakest precondition of $w$ with respect to the postcondition $(x \leq 0)$?
- Compute $WP_0(w, (x \leq 0))$, $WP_1(w, (x \leq 0))$, \ldots.
Consider the program $w$ below:

```
while ($x \geq 10$) do
  $x := x - 1$
```

- What is the weakest precondition of $w$ with respect to the postcondition ($x \leq 0$)?
- Compute $WP_0(w, (x \leq 0))$, $WP_1(w, (x \leq 0))$, . . .
Reducing verification to satisfiability: Generating Verification Conditions

To check:

\{ y > 10 \}

y := y + 1;

z := x + y;

\{ x < z \}

Use the weakest precondition rules to generate the verification condition:

\[(y > 10) \implies (y > -1).\]

Check the verification condition by asking a theorem prover / SMT solver if the formula

\[(y > 10) \land \neg(y > -1).\]

is satisfiable.
What about **while** loops?

Pre: $0 \leq n$

```plaintext
int a := m;
int x := 0;
while (x < n) {
    a := a + 1;
    x := x + 1;
}
```

Post: $a = m + n$
What is a “good” loop invariant for this program?

```plaintext
x := 0;
while (x < 10) {
    if (x >= 0)
        x := x + 1;
    else
        x := x - 1;
}
assert(x <= 11);
```
Adequate loop invariant

```plaintext
x := 0;
while (x < 10) {
    if (x >= 0)
        x := x + 1;
    else
        x := x - 1;
}
assert(x <= 11);
```

<table>
<thead>
<tr>
<th>Canonical Invariant</th>
<th>Not-inv</th>
<th>Inv,not-ind</th>
<th>Inv,ind,not-adeq</th>
<th>Inv,ind,adeq</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 ≤ x ≤ 10</td>
<td>5 ≤ x</td>
<td>-1 ≤ x</td>
<td>0 ≤ x ≤ 12</td>
<td>0 ≤ x ≤ 11</td>
</tr>
</tbody>
</table>

```
0 ≤ x ≤ 10  5 ≤ x  -1 ≤ x  0 ≤ x ≤ 12  0 ≤ x ≤ 11
```
An adequate loop invariant needs to satisfy:

- \{ n \geq 0 \} \; a := m; \; x := 0
- \{ a = m + x \land x \leq n \}.

- \{ a = m + x \land x \leq n \land x < n \} \; a := a+1; \; x := x+1 \; \{ a = m + x \land x \leq n \}.

- \{ a = m + x \land x \leq n \land x \geq n \} \; \text{skip} \; \{ a = m + n \}.

Verification conditions are generated accordingly.

Note that \( a = m + x \) is not an adequate loop invariant.
Generating Verification Conditions for a program

```
while (b) {
  assume Pre
  invariant Inv
  S1
}
```

The following VCs are generated:

- \( Pre \land [S_1] \implies Inv' \)
  Or: \( Pre \implies WP(S_1, Inv) \)

- \( Inv \land b \land [S_2] \implies Inv' \)
  Or: \( (Inv \land b) \implies WP(S_2, Inv) \)

- \( Inv \land \neg b \land [S_3] \implies Post' \)
  Or: \( Inv \land \neg b \implies WP(S_3, Post) \)
Relative completeness of Hoare logic

**Theorem (Cook 1974)**

Hoare logic is complete provided the assertion language $L$ can express the WP for any program $P$ and post-condition $B$.

Proof uses WP predicates and proceeds by induction on the structure of the program $P$.

- Suppose $\{A\} \text{ skip } \{B\}$ holds. Then it must be the case that $A \implies B$ is true. By Skip rule we know that $\{B\} \text{ skip } \{B\}$. Hence by Weakening rule, we get that $\{A\} \text{ skip } \{B\}$ holds.

- Suppose $\{A\} \text{ x := e } \{B\}$ holds. Then it must be the case that $A \implies B[e/x]$. By Assignment rule we know that $\{B[e/x]\} \text{ x := e } \{B\}$ is true. Hence by Weakening rule, we get that $\{A\} \text{ x := e } \{B\}$ holds.

- Similarly for sequencing $S;T$.

- Similarly for if-then-else.
Suppose $\{A\}$ while $b$ do $S$ $\{B\}$ holds. Let $P = WP(\text{while } b \text{ do } S, B)$. Then it is not difficult to check that $P$ is a loop invariant for the while statement. I.e $\{P \land b\} S \{P\}$ is true. By induction hypothesis, this triple must be provable in Hoare logic. Hence we can conclude using the While rule, that $\{P\}$ while $b$ do $S$ $\{P \land \neg b\}$. But since $P$ was a valid precondition, it follows that $(P \land \neg b) \implies B$. By the weakening rule, we have a proof of $\{A\}$ while $b$ do $S$ $\{B\}$. 
Features of this Floyd-Hoare style of verification:

- Tries to find a proof in the form of an *inductive annotation*.
- A Floyd-style proof can be used to obtain a Hoare-style proof; and vice-versa.
- Reduces verification (given key annotations) to checking satisfiability of a logical formula (VCs).
- Is flexible about predicates, logic used (for example can add quantifiers to reason about arrays).

Main challenge is the need for user annotation (adequate loop invariants).

Can be increasingly automated (using learning techniques).