Hoare Examples & Proof Theory

COS 441 Slides 11
Agenda

• The last several lectures:
  – Denotational semantics of formulae in Haskell
  – Reasoning using Hoare Logic

• This lecture:
  – Exercises
  – A further introduction to the mathematical notation used in programming languages research
EXERCISES
Which Implications are Valid?

• Assume all formulae and states are well-formed.
• An implication $P \Rightarrow Q$ is valid if $P$ describes fewer (or the same) states as $Q$
• Which implications are valid?
  – false $\Rightarrow$ true
  – true $\Rightarrow$ false
  – true $\Rightarrow$ true
  – false $\Rightarrow$ false
  – false $\Rightarrow$ $P$ (for any formula $P$)
  – $P \Rightarrow$ false (for any formula $P$)
  – $P \Rightarrow$ true (for any formula $P$)
  – true $\Rightarrow$ $P$ (for any formula $P$)
  – $x = x+1 \Rightarrow$ true
  – $x = x+1 \Rightarrow y = y+1$
  – $5 = 5 \Rightarrow 6 > 3$
  – $x > y \Rightarrow x < y$
  – $B \& A \Rightarrow A$ (for any $A$, $B$)
  – $A \Rightarrow A || B$ (for any $A$)
  – true $\&\&$ false $\Rightarrow$ true $||$ false
Which Triples are Valid?

1. { false } skip { true }
2. { false } skip { false }
3. { true } skip { false }
4. { true } skip { true }
5. { x = x+1 } skip { y = y+1 }
6. { true } skip { 0 = 3 }
7. { 2 = 2 } skip { 5 = 5 }
8. { 8 > 3 } skip { false }
<table>
<thead>
<tr>
<th>Triple</th>
<th>Valid/Invalid</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>(false) skip (true)</td>
<td>yes</td>
<td>any triple with false precondition</td>
</tr>
<tr>
<td>(false) skip (false)</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>(true) skip (false)</td>
<td>no</td>
<td>postcondition can’t be made true</td>
</tr>
<tr>
<td>(true) skip (true)</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>(x = x+1) skip (y = y+1)</td>
<td>yes</td>
<td>precondition is equivalent to false</td>
</tr>
<tr>
<td>(true) skip (0 = 3)</td>
<td>no</td>
<td>0 = 3 is equivalent to false</td>
</tr>
<tr>
<td>(2 = 2) skip (5 = 5)</td>
<td>yes</td>
<td>equivalent to (true) skip (true)</td>
</tr>
<tr>
<td>(8 &gt; 3) skip (false)</td>
<td>no</td>
<td>equivalent to (true) skip (false)</td>
</tr>
</tbody>
</table>
Fill in the Pre-conditions

{ ? }
y = x;

y = x + x + y;

{ y = 3*x }
Fill in the Pre-conditions

\{ true \}
\{ x + x + x = 3*x \}

\textcolor{red}{y = x;}
\{ x + x + y = 3*x \}

\textcolor{red}{y = x + x + y;}
\{ y = 3*x \}

\textit{simplify using the rule of consequence}
Fill in the Pre-conditions

```plaintext
{ ? }
z = x + 2;
y = z + z;
x = z + y
{x > z & y = 3 }
```
Fill in the Pre-conditions

\[
\begin{align*}
\{ 2 \times x &= -1 \} & \quad \text{false if we are dealing with integers} \\
\{ (x+2) + (x+2) &= 3 \} & \quad \text{no integer solution!} \\
\end{align*}
\]

\[
z = x + 2;
\]

\[
\{ z + z = 3 \} & \quad \text{simplify using the rule of consequence part-way through}
\]

\[
\{ \text{true} \& z + z = 3 \}
\]

\[
\{ z + (z + z) > z \& z + z = 3 \}
\]

\[
y = z + z;
\]

\[
\{ z + y > z \& y = 3 \}
\]

\[
x = z + y
\]

\[
\{ x > z \& y = 3 \}
\]
{ ? }
if ( x - y < 0 ) then {
    z = x
}
else {
    z = y
}
{ z <= y & z <= x }
Fill in the Pre-conditions

{ ? }
if ( x - y < 0 ) then {

    z = x
    { z <= y & z <= x }
}

} else {

    z = y
    { z <= y & z <= x }

}
{ z <= y & z <= x }
Fill in the Pre-conditions

{ ? }
if ( x - y < 0 ) then {
    { x <= y & x <= x }
    z = x
    { z <= y & z <= x }
} else {
    { y <= y & y <= x }
    z = y
    { z <= y & z <= x }
}
{ z <= y & z <= x }
{ ? }
if ( x - y < 0 ) then {
    \{ x <= y \} 
    \{ x <= y & x <= x \} 
    z = x 
    \{ z <= y & z <= x \}
} else {
    \{ y <= x \} 
    \{ y <= y & y <= x \} 
    z = y 
    \{ z <= y & z <= x \}
}
{ z <= y & z <= x }
if (x - y < 0) then {
    {x <= y}
    {x <= y & x <= x}
    z = x
    {z <= y & z <= x}
} else {
    {y <= x}
    {y <= y & y <= x}
    z = y
    {z <= y & z <= x}
}
{z <= y & z <= x}

if rule:
   If {e < 0 & ?} C1 {Q} and {~(e < 0) & ?} C2 {Q} then {?} if e < 0 then C1 else C2 {Q}

we need to find ? such that:

(x - y < 0) & ? => x <= y

and

~(x - y < 0) & ? => y <= x
\[ \{ \ ? \ \} \]

if (\( x - y < 0 \)) then \{ \\
  \{ x \leq y \} \\
  \{ x \leq y \& x \leq x \} \\
  z = x \\
  \{ z \leq y \& z \leq x \} \\
\}

else \{ \\
  \{ y \leq x \} \\
  \{ y \leq y \& y \leq x \} \\
  z = y \\
  \{ z \leq y \& z \leq x \} \\
\}

\{ z \leq y \& z \leq x \}

if rule:

\[
\begin{align*}
\text{If } \{ e < 0 \& \? \} \text{ C1 } \{ Q \} \text{ and } \{ \neg (e < 0) \& \? \} \text{ C2 } \{ Q \} \\
\text{then } \{ \ ? \ \} \text{ if } e < 0 \text{ then C1 else C2 } \{ Q \}
\end{align*}
\]

we need to find \( ? \) such that:

\[
\begin{align*}
(x-y < 0) \& \? & \Rightarrow & x \leq y \\
\neg (x-y < 0) \& \? & \Rightarrow & y \leq x
\end{align*}
\]

x – y < 0 already implies x \leq y

\( \neg (x – y < 0) \) already implies y \leq x

Anything for \( ? \) works, including true.
{ ? }

if ( x > 0 ) then {

    x = x + 1

} else {

    x = z

}

{ even (x) }
Fill in the Pre-conditions

```plaintext
{ ? }
if ( x > 0 ) then {

    x = x+1
    { even(x) }
}

else {

    x = z
    { even(x) }
}

{ even(x) }
```
{ ? }
if ( x > 0 ) then {
    { even(x+1) }
    x = x+1
    { even(x) }
} else {
    { even(z) }
    x = z
    { even(x) }
}
{ even (x) }
Fill in the Pre-conditions

\[
\{ \ ? \ \} \\
\text{if } ( x > 0 ) \text{ then } \{ \\
\quad \{ \text{even}(x+1) \} \\
\quad x = x+1 \\
\quad \{ \text{even}(x) \} \\
\}\text{ else } \{ \\
\quad \{ \text{even}(z) \} \\
\quad x = z \\
\quad \{ \text{even}(x) \} \\
\}\} \{ \text{even} (x) \}
\]

\[
\text{if rule:} \\
\text{If } \{ e > 0 \ & \ ? \} \ C1 \{ Q \} \text{ and } \{ \neg(e > 0) \ & \ ? \} \ C2 \{ Q \} \text{ then } \{ \ ? \} \text{ if } e < 0 \text{ then } C1 \text{ else } C2 \{ Q \}
\]

we need to find ? such that:

\[
x > 0 \ & \ ? \Rightarrow \text{even}(x+1)
\]

and

\[
\neg(x > 0) \ & \ ? \Rightarrow \text{even}(z)
\]
if ( x > 0 ) then {
    { even(x+1) }
    x = x+1
    { even(x) }
} else {
    { even(z) }
    x = z
    { even(x) }
}
{ even (x) }

if rule:

If { e > 0 & ? } C1 { Q } and { ~(e > 0) & ? } C2 { Q } then { ? } if e < 0 then C1 else C2 { Q }

we need to find ? such that:

x > 0 & ? => even(x+1)

and

~(x > 0) & ? => even(z)

? could be odd(x) & even(z)
if ( x > 0 ) then {
  { even(x+1) }
  x = x+1
  { even(x) }
} else {
  { even(z) }
  x = z
  { even(x) }
}
{ even (x) }

if rule:

If { e > 0 & ? } C1 { Q } and { ~(e > 0) & ? } C2 { Q }
then { ? } if e < 0 then C1 else C2 { Q }

we need to find ? such that:

x > 0 & odd(x) & even(z)
  =>  even(x+1)

and

~(x > 0) & odd(x) & even(z)
  =>  even(z)

? could be odd(x) & even(z)
AN INTRODUCTION TO
PROOF THEORY
Semantics So Far

- Relatively speaking, the semantics of expressions is simple
  - it is given by a simple partial function
  - $e_1, e_2$ are any expressions (they are “metavariables”)
  - $s$ is any state ($s$ is also a “metavariable”)

$$[[ e_1 + e_2 ]]s = [[ e_1 ]]s + [[ e_2 ]]s$$

- Semantics of formulae is also easy:

$$[[ true ]]s = true$$
$$[[ false ]]s = false$$
$$[[ f_1 \& f_2 ]]s = [[ f_1 ]]s \& [[ f_2 ]]s$$
• Semantics of formulae:

\[
[[ \text{true} ]]_s = \text{true} \\
[[ \text{false} ]]_s = \text{false} \\
[[ f_1 \& f_2 ]]_s = [[ f_1 ]]_s \& [[ f_2 ]]_s
\]

• In your handout:

\[ s \models f \]

"state \( s \) satisfies formula \( f \)" or "formula \( f \) describes state \( s \)" or "formula \( f \) is true in state \( s \)"

the same as:  \([f]]_s == \text{true}\)

• Some examples:

\[ s \models \text{true} \quad \text{(for any} \ s) \]

\[ [x=3, y=7] \models (x > 1) \& (y = 7) \]
• Relatively speaking, the semantics of expressions is simple
  — it is given by a simple partial function:

  \[[ \text{e}_1 + \text{e}_2 ]\]_s = \[[ \text{e}_1 ]\]_s + \[[ \text{e}_2 ]\]_s

• Hoare proof theory is a little more complicated
  — it was given by a series of “rules”:

Skip:
{ P } skip { P }

Assignment:
{ F[e/x] } x = e { F }

Consequence:
If \( P' \Rightarrow P \) and \{ P \} C \{ Q \} and \( Q \Rightarrow Q' \)
then \{ P' \} C \{ Q' \}

While:
If \( P \Rightarrow I \) and \{ e > 0 & I \} C \{ I \} and \( I & \neg (e > 0) \Rightarrow Q \)
then \{ P \} while (e > 0) do C \{ Q \}

Sequence:
if \{ F1 \} C1 \{ F2 \} and \{ F2 \} C2 \{ F3 \}
then \{ F1 \} C1; C2 \{ F3 \}

If:
If \{ e > 0 & P \} C1 \{ Q \} and \{ \neg (e > 0) & P \} C2 \{ Q \}
then \{ P \} if e > 0 then C1 else C2 \{ Q \}
Inference Rules

• Looking at the rules, they decompose into base cases (axioms):

  **Skip:**
  \[
  \{ P \} \text{skip} \{ P \}
  \]

  **Assignment:**
  \[
  \{ F[e/x] \} x = e \{ F \}
  \]

• And inductive cases that appeal to smaller proofs of Hoare triple validity:

  **Consequence:**
  \[
  \text{If } P' \Rightarrow P \text{ and } \{ P \} \subset C \{ Q \} \text{ and } Q \Rightarrow Q' \text{ then } \{ P' \} \subset C \{ Q' \}
  \]

  **Sequence:**
  \[
  \text{if } \{ F_1 \} C_1 \{ F_2 \} \text{ and } \{ F_2 \} C_2 \{ F_3 \} \text{ then } \{ F_1 \} C_1; C_2 \{ F_3 \}
  \]

  **While:**
  \[
  \text{If } P \Rightarrow I \text{ and } \{ e > 0 \wedge I \} \subset C \{ I \} \text{ and } I \wedge \lnot(e > 0) \Rightarrow Q \text{ then } \{ P \} \subset \text{while}(e > 0) \text{do} C \{ Q \}
  \]

  **If:**
  \[
  \text{If } \{ e > 0 \wedge P \} \subset C_1 \{ Q \} \text{ and } \{ \lnot(e > 0) \wedge P \} \subset C_2 \{ Q \} \text{ then } \{ P \} \text{if} e > 0 \text{ then} C_1 \text{ else} C_2 \{ Q \}
  \]

• When I say “smaller proofs of Hoare triple validity”, what I mean is a smaller number of uses of the above inference rules.
I’ve been careful to write all of the inference rules for Hoare logic in a suggestive format:

Sequence:

if \{ F_1 \} C_1 \{ F_2 \} and \{ F_2 \} C_2 \{ F_3 \}
then \{ F_1 \} C_1; C_2 \{ F_3 \}
Inference rules

• I’ve been careful to write all of the inference rules for Hoare logic in a suggestive format:

  Sequence:
  if \{ F_1 \} C_1 \{ F_2 \} and \{ F_2 \} C_2 \{ F_3 \}
  then \{ F_1 \} C_1; C_2 \{ F_3 \}

• PL researchers use the following notation:

  horizontal line means “if”

  \{ F_1 \} C_1 \{ F_2 \} \quad \{ F_2 \} C_2 \{ F_3 \}
  { F_1 \} C_1; C_2 \{ F_3 \}
Inference rules

• I’ve been careful to write all of the inference rules for Hoare logic in a suggestive format:

   Sequence:
   
   if \{ F1 \} C1 { F2 } and \{ F2 \} C2 { F3} then \{ F1 \} C1; C2 { F3 }

• PL researchers use the following notation:

   horizontal line means "if"

   metavariables can be replaced by any (well-formed) element of the right sort

   premises

   conclusion
Inference rules

• I’ve been careful to write all of the inference rules for Hoare logic in a suggestive format:

  PL researchers use the following notation:

  {F1} C1; C2 {F3}

• PL researchers use the following notation:

  horizontal line means “if”

  metavariables can be replaced by any (well-formed) element of the right sort
PL researchers use the following notation:

\[
\begin{align*}
\{ F_1 \} & C_1 \{ F_2 \} \\
\{ F_2 \} & C_2 \{ F_3 \} \\
\{ F_1 \} & C_1; C_2 \{ F_3 \}
\end{align*}
\]

(Seq)

Example instance of the rule:

\[
\begin{align*}
\{ x = 4 \} & x = x+2 \{ x = 6 \} \\
\{ x = 6 \} & x = x+1 \{ x = 7 \} \\
\{ x = 4 \} & x = x+2; x = x+1 \{ x = 7 \}
\end{align*}
\]

(Seq)

metavariables can be replaced by any (well-formed) element of the right sort
Complete Hoare Rules

**Axioms**

<table>
<thead>
<tr>
<th>{ P } skip { P }</th>
<th>(skip)</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ F [e/x] } x = e { F }</td>
<td>(assign)</td>
</tr>
</tbody>
</table>

| P' => P | { P } C { Q } | Q => Q' | (consequence) |
|---------|----------------|----------|
| { P' } C { Q' } | |

| P => I | { e > 0 & I } C { I } | I & ~(e > 0) => Q | (while) |
|--------|------------------------|------------------|
| { P } while (e > 0) do C { Q } | |

| { F1 } C1 { F2 } | { F2 } C2 { F3} | (seq) |
|------------------|-----------------|
| { F1 } C1; C2 { F3 } | |

| { e > 0 & P } C1 { Q } | { ~(e > 0) & P } C2 { Q } | (if) |
|------------------------|-------------------------|
| { P } if e > 0 then C1 else C2 { Q } | |
A random bunch of boxes and arrows is not a consistent, well-defined notation for proofs:

```
if x > 0 then
  skip;
else
  y = 1;

{ x > 0 || y = 1 }
{ x > 0 & true }
{ ~(x > 0) & true }
{ true }
```
Building Proofs

• Build proofs by stringing together a collection of rules
• Valid axioms are at the top
• Valid rule instances connect premises to conclusions

\[
x = 4 \implies x + 2 = 6 \quad \{ x + 2 = 6 \} x = x+2 \{ x = 6 \}
\]
\[
x = 6 \implies x + 1 = 7 \quad \{ x + 1 = 7 \} x = x+1 \{ x = 7 \}
\]
\[
\{ x = 4 \} x = x+2 \{ x = 6 \} \quad \{ x = 6 \} x = x+1 \{ x = 7 \}
\]
\[
\{ x = 4 \} x = x+2; x = x+1 \{ x = 7 \}
\]
Building Proofs

There wasn’t space on the slide, but putting a name next to each horizontal line indicates the rule that was used:

- \( x = 4 \Rightarrow x + 2 = 6 \)
  - \( \{ x + 2 = 6 \} x = x + 2 \{ x = 6 \} \)
- \( x = 6 \Rightarrow x + 1 = 7 \)
  - \( \{ x + 1 = 7 \} x = x + 1 \{ x = 7 \} \)

- \( x = 4 \Rightarrow x + 2 = 6 \)
  - \( \{ x = 4 \} x = x + 2 \{ x = 6 \} \)
  - \( \{ x = 4 \} x = x + 2; x = x + 1 \{ x = 7 \} \)

Diagram:

- (assign)
- (seq)
- (consequence)
Building Proofs Bottom-up

- Start with the Hoare Triple you want to prove at the bottom of your page:

\{ \text{odd}(x) \& \text{even}(z) \} \text{ if } x > 0 \text{ then } x = x + 1 \text{ else } x = z \{ \text{even}(x) \}
Building Proofs Bottom-up

• Consider the rules that apply.
• Typically:
  – the rule for the kind of statement
  – the rule of consequence
• Use the rule you choose to generate premises.
• Write the premises above the line
• Continue until you have axioms

\[\{\text{odd}(x) \& \text{even}(z) \& x>0\} \Rightarrow x = x+1 \{\text{even}(x)\}\]
\[\{\text{odd}(x) \& \text{even}(z) \& \neg(x>0)\} \Rightarrow x = z \{\text{even}(x)\}\]
\[\{\text{odd}(x) \& \text{even}(z)\} \text{ if } x > 0 \text{ then } x=x+1 \text{ else } x=z \{\text{even}(x)\}\]
There wasn’t space on the slide, but putting a name next to each horizontal line indicates the rule that was used:

odd(x) & even(z) & x>0 => even(x+1)
Building Proofs Bottom-up

There wasn’t space on the slide, but putting a name next to each horizontal line indicates the rule that was used:

\[
\text{odd}(x) \& \text{even}(z) \& x>0 \Rightarrow \text{even}(x+1)
\]

axiom for assignment, so we can stop this branch of the proof:
There wasn’t space on the slide, but putting a name next to each horizontal line indicates the rule that was used:

- `odd(x) & even(z) & x>0 => even(x+1)`
- `odd(x) & even(z) & ~(x>0) => odd(x)`
- Axiom for assignment
More Generally

• Proof systems tell us how to conclude certain kinds of *propositions* (aka *assertions* or *properties*) from a set of *rules*

• The propositions are typically called *judgements*
  – eg: \{ P \} C \{ Q \} is the Hoare Triple judgement

• The rules are typically called *inference rules*:

  \[ J_1 \quad J_2 \quad \ldots \quad J_n \quad \text{cond}_1 \quad \ldots \quad \text{cond}_k \quad J \]
More Generally

• Proof systems tell us how to conclude certain kinds of *propositions* (aka *assertions* or *properties*) from a set of *rules*.

• The propositions are typically called *judgements*.
  – eg: $\{ P \} C \{ Q \}$ is the Hoare Triple judgement.

• The rules are typically called *inference rules*:

$$ J_1 \quad J_2 \quad \ldots \quad J_n \quad \text{cond}_1 \quad \ldots \quad \text{cond}_k \quad J $$

judgement we are defining

simple logical conditions

conclusion
More Generally

- Proof systems tell us how to conclude certain kinds of *propositions* (aka *assertions* or *properties*) from a set of *rules*.
- The propositions are typically called *judgements*.
  - eg: \{ P \} C \{ Q \} is the Hoare Triple judgement.
- The rules are typically called *inference rules*.
- A *formal proof* stitches together a finite number of valid rules, ending with *axioms*:

\[
\begin{array}{cccc}
  & J3 & J4 & J6 \\
J1 & J2 & \text{cond} & J
\end{array}
\]
SUMMARY!
PL researchers often describe programming languages using *judgements* and *rules*

The rules for Hoare Logic look like this:

\[
\begin{align*}
\{ P \} \text{skip} \{ P \} & \quad \text{(skip)} \\
\{ F [e/x] \} x = e \{ F \} & \quad \text{(assign)} \\
\end{align*}
\]

\[
\frac{P' \Rightarrow P \quad \{ P \} C \{ Q \} \quad Q \Rightarrow Q'}{\{ P' \} C \{ Q' \}} \quad \text{(consequence)}
\]

Proofs stitch together a series of rules

- in a valid proof
  - the proof tops out with valid instances of one of the axioms
  - every step from premises to conclusion is a valid instance of one of the inference rules