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 => Q is valid if P describes fewer (or the same) states as Q
- Which implications are valid?
  - false => true
  - true => false
  - true => true
  - false => false
  - false => P (for any formula P)
  - P => false (for any formula P)
  - P => true (for any formula P)
  - true => P (for any formula P)
  - x = x+1 => true
  - x = x+1 => y = y+1
  - 5 = 5 => 6 > 3
  - x > y => x < y
  - B & A => A (for any A, B)
  - A => A || B (for any A)
  - true && false => 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 }

Which Triples are Valid?

1. { false } skip { true }	yes (any triple with false precondition)
2. { false } skip { false }	yes
3. { true } skip { false }	no (postcondition can't be made true)
4. { true } skip { true }	yes
5. { x = x+1 } skip { y = y+1 }	yes (precondition is equivalent to false)
6. { true } skip { 0 = 3 }	no 0 = 3 is equivalent to false
7. { 2 = 2 } skip { 5 = 5 }	yes, equivalent to { true } skip { true }
8. { 8 > 3 } skip { false }	no, equivalent to { true } skip { false }

Fill in the Pre-conditions

{ ? }

y = x;

y = x + x + y;

{ y = 3*x }

Fill in the Pre-conditions

{ true } ← simplify using the rule of consequence

{ x + x + x = 3*x }

y = x;

{ x + x + y = 3*x }

y = x + x + y;

{ y = 3*x }

Fill in the Pre-conditions

{ ? }

z = x + 2;

y = z + z;

x = z + y

{ x > z & y = 3 }

Fill in the Pre-conditions

{ 2*x = -1 } ← false if we are dealing with integers no integer solution!

{ (x+2) + (x+2) = 3 }

z = x + 2;

{ z + z = 3 } ← simplify using the rule of consequence part-way through

{ 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 }

Fill in the Pre-conditions

```
{ ? }
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 }
```

Fill in the Pre-conditions

```
{ ? }
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 }
```

rule of consequence

rule of consequence

Fill in the Pre-conditions

```
{ ? }
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
```

Fill in the Pre-conditions

```
{ ? }
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

x – y < 0 already implies x <= y
~(x – y < 0) already implies y <= x
Anything for ? works, including true.

Fill in the Pre-conditions

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

        x = x+1

} else {

        x = z

}
{ even (x) }
```

Fill in the Pre-conditions

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

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

        x = z
        { even(x) }
}
{ even (x) }
```

Fill in the Pre-conditions

```
{ ? }
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

```
{ ? }
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)

Fill in the Pre-conditions

```
{ ? }
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)

Fill in the Pre-conditions

```
{ ? }
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
  - e1, e2 are any expressions (they are “metavariables”)
  - s is any state (s is also a “metavariable”)

[[e1 + e2]]s = [[e1]]s + [[e2]]s

- Semantics of formulae is also easy:

[[true]]s = true
[[false]]s = false
[[f1 & f2]]s = [[f1]]s & [[f2]]s

Semantics So Far

- Semantics of formulae:

  [[true]]s = true
  [[false]]s = false
  [[f1 & f2]]s = [[f1]]s & [[f2]]s

- In your handout:

  s |= 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 == true

- Some examples:

  s |= true (for any s)

  [x= 3, y= 7] |= (x > 1) & (y = 7)

Semantics So Far

- Relatively speaking, the semantics of expressions is simple
  - it is given by a simple partial function:

[[e1 + e2]]s = [[e1]]s + [[e2]]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' => P and { P } C { Q } and Q => Q'
then { P' } C { Q' }

While:

If P => I and { e > 0 & I } C { I } and I & ~(e > 0) => 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 { ~(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 } skip { P }

Assignment:

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

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

Consequence:

If P’ => P and { P } C { Q } and Q => Q’
then { P’ } C { Q’ }

While:

If P => I and { e > 0 & I } C { I } and I & ~(e > 0) => 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 { ~(e > 0) & P } C2 { Q }
then { P } if e > 0 then C1 else C2 { Q }

- When I say “smaller proofs of Hoare triple validity”, what I mean is a smaller number of uses of the above inference rules

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 }

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"

premises

$$\frac{\{\ F1\ \}\ C1\ \{\ F2\ \} \qquad \{\ F2\ \}\ C2\ \{\ F3\ \}}{\{\ F1\ \}\ C1;\ C2\ \{\ F3\ \}}$$

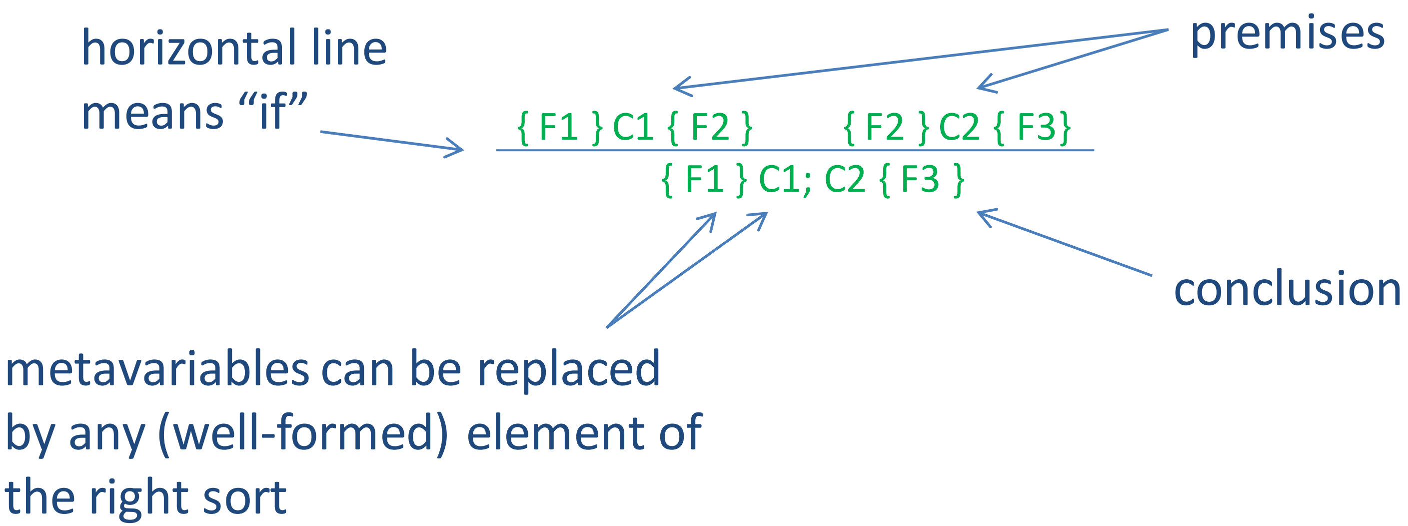
conclusion

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:



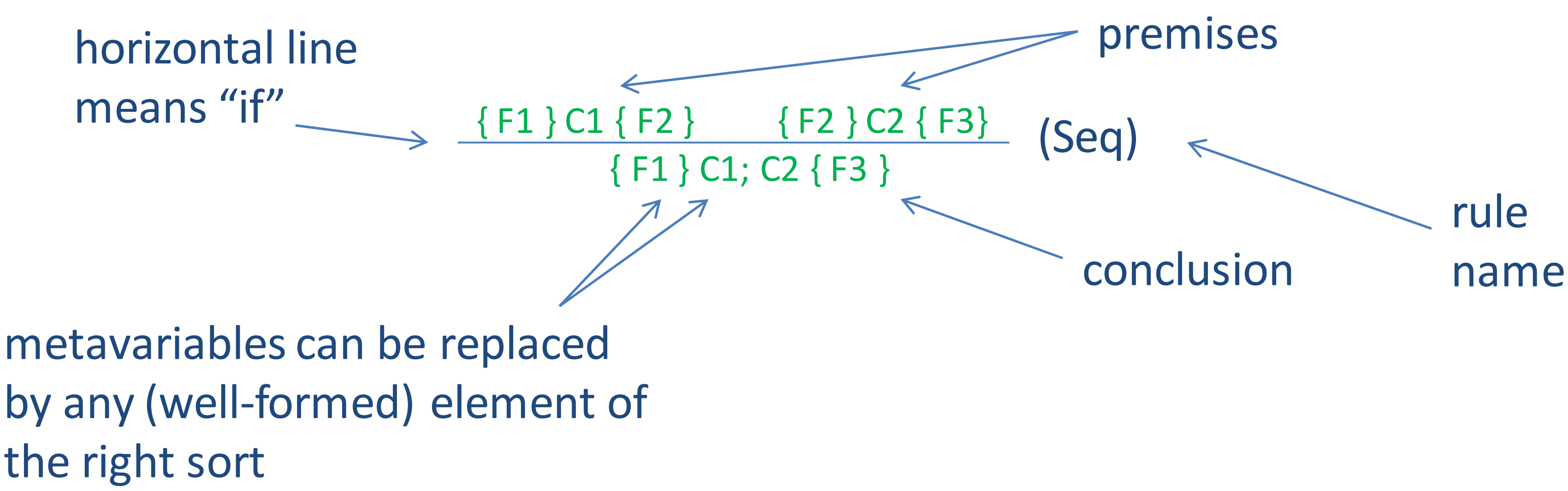
Inference rules

- 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:



Inference rules

- PL researchers use the following notation:

premises

$$\frac{\{\ F1\ \}\ C1\ \{\ F2\ \} \qquad \{\ F2\ \}\ C2\ \{\ F3\ \}}{\{\ F1\ \}\ C1;\ C2\ \{\ F3\ \}}\ (\text{Seq})$$

conclusion

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

- Example instance of the rule:

$$\frac{\{\ x = 4\ \}\ x = x{+}2\ \{\ x = 6\ \} \qquad \{\ x = 6\ \}\ x = x{+}1\ \{\ x = 7\ \}}{\{\ x = 4\ \}\ x = x{+}2;\ x = x{+}1\ \{\ x = 7\ \}}\ (\text{Seq})$$

Complete Hoare Rules

axioms

$$\frac{}{\{ P \}\ \text{skip}\ \{ P \}}\ \text{(skip)}$$

$$\frac{}{\{ F[e/x] \}\ x = e\ \{ F \}}\ \text{(assign)}$$

inductive rules

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

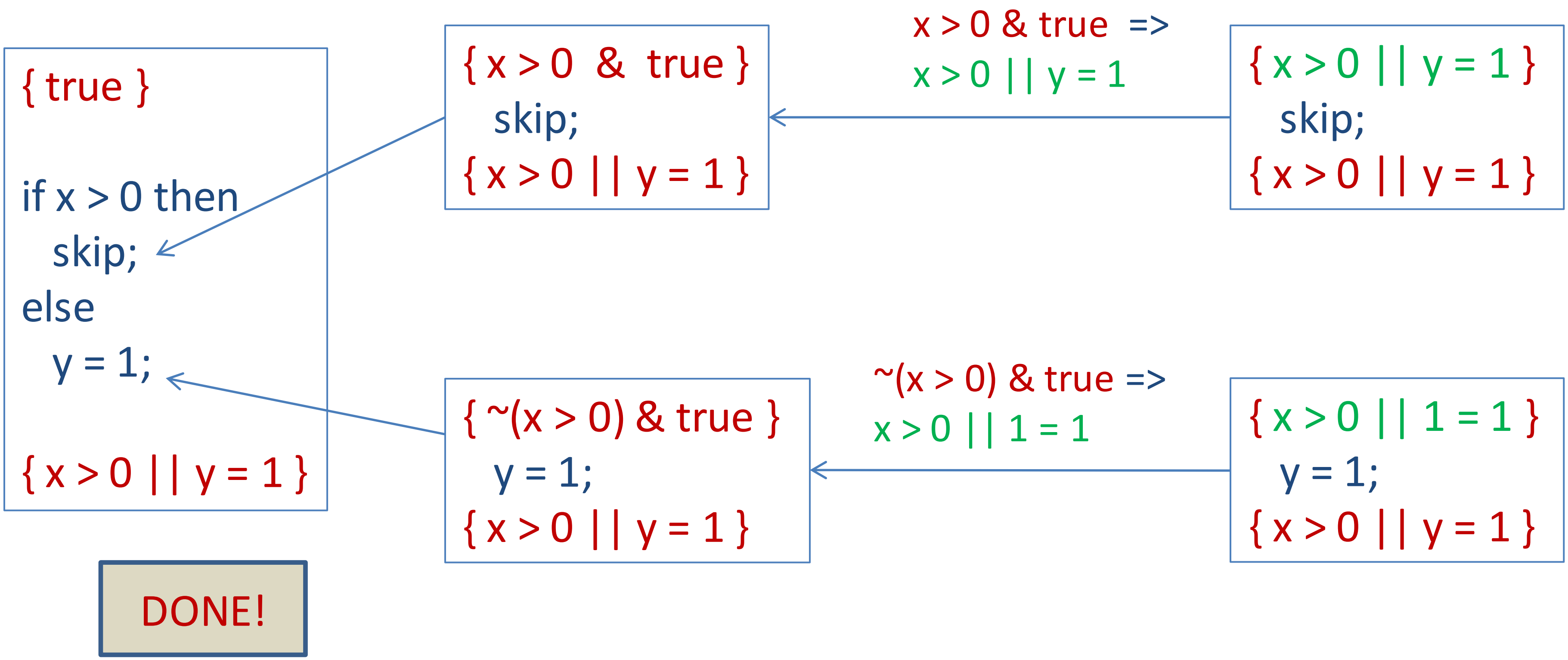
$$\frac{P \Rightarrow I \quad \{ e > 0\ \&\ I \}\ C\ \{ I \} \quad I\ \&\ \sim(e > 0) \Rightarrow Q}{\{ P \}\ \text{while}\ (e > 0)\ \text{do}\ C\ \{ Q \}}\ \text{(while)}$$

$$\frac{\{ F1 \}\ C1\ \{ F2 \} \quad \{ F2 \}\ C2\ \{ F3 \}}{\{ F1 \}\ C1;\ C2\ \{ F3 \}}\ \text{(seq)}$$

$$\frac{\{ e > 0\ \&\ P \}\ C1\ \{ Q \} \quad \{ \sim(e > 0)\ \&\ P \}\ C2\ \{ Q \}}{\{ P \}\ \text{if}\ e > 0\ \text{then}\ C1\ \text{else}\ C2\ \{ Q \}}\ \text{(if)}$$

Building Proofs

- A random bunch of boxes and arrows is not a consistent, well-defined notation for proofs:



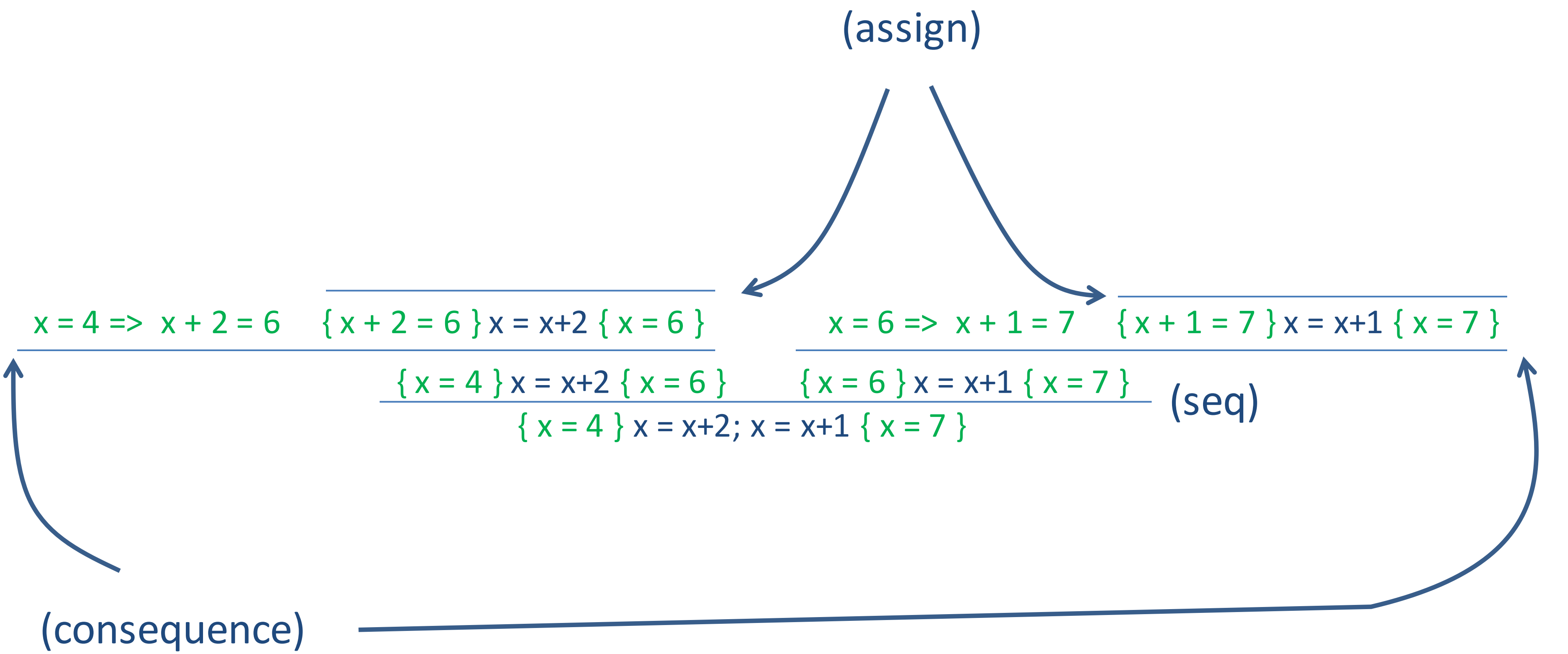
Building Proofs

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

$$\dfrac{\dfrac{x = 4 => x + 2 = 6 \qquad \dfrac{}{\{ x + 2 = 6 \}\ x = x+2\ \{ x = 6 \}}}{\{ x = 4 \}\ x = x+2\ \{ x = 6 \}} \qquad \dfrac{x = 6 => x + 1 = 7 \qquad \dfrac{}{\{ x + 1 = 7 \}\ x = x+1\ \{ x = 7 \}}}{\{ x = 6 \}\ x = x+1\ \{ x = 7 \}}}{\{ x = 4 \}\ x = x+2;\ x = x+1\ \{ x = 7 \}}$$

Building Proofs

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



Building Proofs Bottom-up

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

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

$$\dfrac{\dfrac{}{\{odd(x)\ \&\ even(z)\ \&\ x>0\}\ x = x+1\ \{even(x)\}} \qquad \dfrac{}{\{odd(x)\ \&\ even(z)\ \&\ \sim(x>0)\}\ x = z\ \{even(x)\}}}{\{odd(x)\ \&\ even(z)\}\ \text{if } x > 0 \text{ then } x=x+1 \text{ else } x=z\ \{even(x)\}}$$

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:

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

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

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

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

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:

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

axiom for assignment,
so we can stop this branch of the proof

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

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

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

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

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:

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

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

axiom for assignment

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

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

{even(z)} x = z { even(x) }

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

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

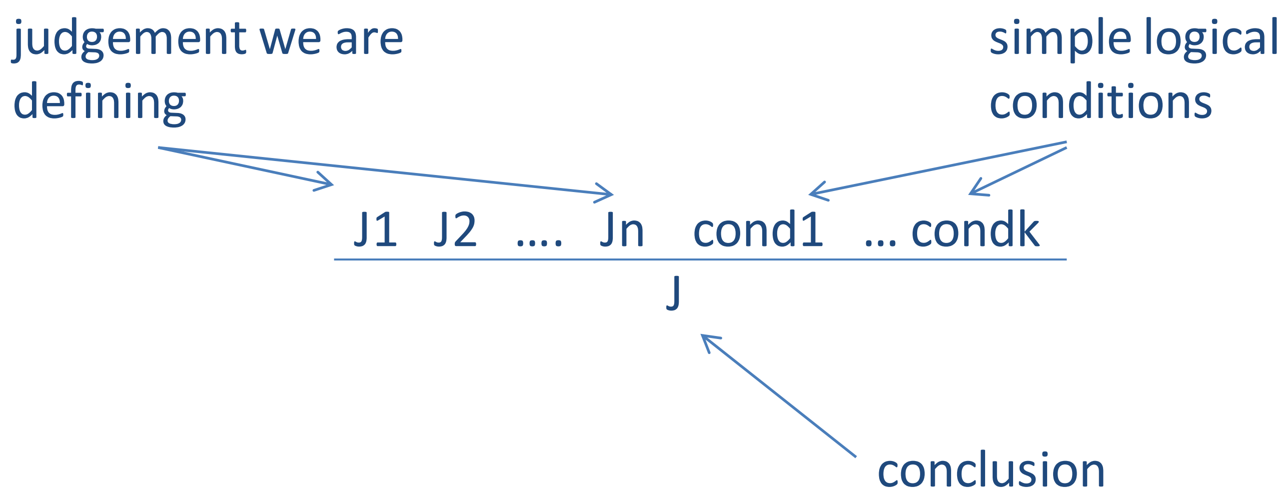
More Generally

- 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*:

$$\frac{J1 \quad J2 \quad .... \quad Jn \quad cond1 \quad ... \; condk}{J}$$

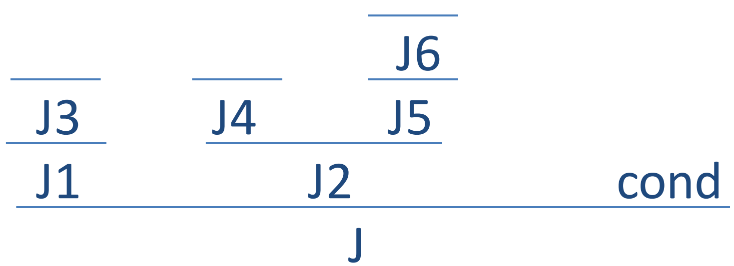
More Generally

- 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*:



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*:



SUMMARY!

Summary

- PL researchers often describe programming languages using *judgements* and *rules*
- The rules for Hoare Logic look like this:

$$\frac{}{\{ P \}\ \text{skip}\ \{ P \}}\ (\text{skip}) \qquad \frac{}{\{ F\ [e/x]\ \}\ x = e\ \{ F \}}\ (\text{assign})$$

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

....

- 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