CSE 331
Software Design & Implementation

Winter 2020

Section 1 – Code Reasoning

Administrivia

- HW1 due next Tuesday.
- Any questions before we dive in?
  - What are the most interesting/confusing/puzzling things so far in the course?

Agenda

- Review logical reasoning about code with Hoare Logic
- Practice both forward and backward modes
  - Just assignment, conditional (“if-then-else”), and sequence
  - Logical rules from yesterday’s lecture/notes
- Review logical strength of assertions (weaker *vs.* stronger)
- Practice determining stronger/weaker assertions

Why reason about code?

- Prove that code is correct
- Understand *why* code is correct
- Diagnose why/how code is *not* correct
- Specify code behavior

Logical reasoning about code

- Determine facts that hold of program state between statements
  - "Fact" ~ assertion (logical formula over program state, informally "value(s) of some/all program variables)
  - Driven by assumption (precondition) or goal (postconditon)

- Forward reasoning – What facts follow from initial assumptions?
  - Go from pre condition to post condition
  - Use Hoare-triple inference rules pretty much directly

- Backward reasoning – What facts need to be true to reach a goal?
  - Go from post condition to pre condition
  - Use weakest-precondition transformer wp(S, Q) to get most general possible set of facts

Inference rules of Hoare Logic

- Assignment: $\{P\}$ `x = e;` $\{Q\}$ is valid iff
  - $P \Rightarrow Q'$ where Q' is Q with each **`x`** replaced by **`e`**

- Sequence: $\{P\}\ S_1;S_2\ \{Q\}$ is valid iff there is some R such that
  - $\{P\}\ S_1\ \{R\}$ is valid
  - $\{R\}\ S_2\ \{Q\}$ is valid

- Conditional: $\{P\}$ `if (b)` S_1 `else` S_2 $\{Q\}$ is valid iff
  - $\{P \wedge \mathtt{b}\}\ S_1\ \{Q_1\}$ is valid
  - $\{P \wedge \mathtt{!b}\}\ S_2\ \{Q_2\}$ is valid
  - $Q_1 \vee Q_2 \Rightarrow Q$

Inference rules of Hoare Logic

- Assignment: `{P} x = e; {Q}` is valid iff
  - $P \Rightarrow Q'$ where Q' is Q with each `x` replaced by `e`
  - **Example:**

```
{e = 2}
x = e;
{x = 2}
```

Inference rules of Hoare Logic

- Sequence: $\{P\}\ S_1; S_2\ \{Q\}$ is valid iff there is some R such that
  - $\{P\}\ S_1\ \{R\}$ is valid
  - $\{R\}\ S_2\ \{Q\}$ is valid
  - **Example:**

```
P ⟶ {true}
        x = 2;
R ⟶ {x = 2}
        y = 3;
Q ⟶ {x = 2 & y = 3}
```

Inference rules of Hoare Logic

- Conditional: $\{P\}$ `if (b)` S_1 `else` S_2 $\{Q\}$ is valid iff
  - $\{P \wedge b\}\, S_1\, \{Q_1\}$ is valid
  - $\{P \wedge !b\}\, S_2\, \{Q_2\}$ is valid
  - $Q_1 \vee Q_2 \Rightarrow Q$

Example:

```
{true}
if (x > 0) {
    {true & x % 2 = 0}
    y = 5;
    {y = 5 & x > 0}
} else {
    {true & x <= 0}
    y = 5;
    {y = 5 & x <= 0}
}
{y = 5 & x > 0 or
 y = 5 & x <= 0} => {y = 5}
```

Implication (=>)

- Logic formulas with *and* (&, &&, or ∧), *or* (|, ||, or ∨) and *not* (! or ¬) have the same meaning they do in programs

- Implication might be a bit new, but the basic idea is pretty simple. Implication p=>q is true as long as q is always true whenever p is

p	q	p => q
T	T	T
T	F	F
F	T	T
F	F	T

Conditionals, more closely

Forward reasoning

$\{P\}$
`if (b)`
    $\{P \wedge \mathtt{b}\}$
    S_1
    $\{Q_1\}$
`else`
    $\{P \wedge \mathtt{!b}\}$
    S_2
    $\{Q_2\}$
$\{Q_1 \vee Q_2\}$

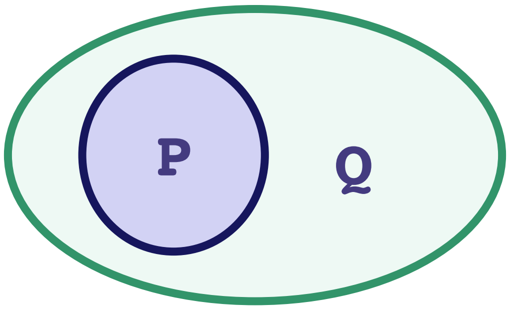
Backward reasoning

$\{\ (\mathtt{b} \wedge P_1) \vee (\mathtt{!b} \wedge P_2)\ \}$
`if (b)`
    $\{P_1\}$
    S_1
    $\{Q\}$
`else`
    $\{P_2\}$
    S_2
    $\{Q\}$
$\{Q\}$

Weaker vs. stronger

Formal definition:

- If $P \Rightarrow Q$, then
  - Q is weaker than P
  - P is stronger than Q



Intuitive definition:

- "Weak" means unrestrictive; a weaker assertion has a larger set of possible program states (*e.g.*, **`x != 0`**)
- "Strong" means restrictive; a stronger assertion has a smaller set of possible program states (*e.g.*, **`x = 1`** or **`x > 0`** are both stronger than **`x != 0`**).

Worksheet

- Take ~10 minutes to get where you can
- Find a partner and work with them
- Let me know if you feel stuck
- We’ll walk through some solutions afterwards

Worksheet – problem 2

```
{ true }
if (x>0) {
  { x > 0 }
  abs = x;
  { x > 0 ∧ abs = x }
} else {
  { x <= 0 }
  abs = -x;
  { x <= 0 ∧ abs = -x }
}
{ (x > 0 ∧ abs = x) ∨ (x <= 0 ∧ abs = -x) }
⇒ { abs = |x| }
```

Worksheet – problem 4

```
{ y > 15 ∨ (y <= 5 ∧ y + z > 17) }
if (y > 5) {
  { y > 15 }
  x = y + 2
  { x > 17 }
} else {
  { y + z > 17 }
  x = y + z;
  { x > 17 }
}
{ x > 17 }
```

Worksheet – problem 6 (forward)

```
{ true }
if (x < y) {
  { true ∧ x < y }
  m = x;
  { x < y ∧ m = x }
} else {
  { true ∧ x >= y }
  m = y;
  { x >= y ∧ m = y }
}
{ (x < y ∧ m = x) ∨ (x >= y ∧ m = y) }
⇒ { m = min(x, y) }
```

Worksheet – problem 6 (backward)

```
{ true } ⇔
{ (x <= y ∧ x < y) ∨ (y <= x ∧ x >= y) }
if (x < y) {
  { x = min(x, y) } ⇔ { x <= y }
  m = x;
  { m = min(x, y) }
} else {
  { y = min(x, y) } ⇔ { x >= y }
  m = y;
  { m = min(x, y) }
}
{ m = min(x, y) }
```

Worksheet – problem 7

`{ y > 23 }`	`{ y >= 23 }`
`{ y = 23 }`	`{ y >= 23 }`
`{ y < 0.23 }`	`{ y < 0.00023 }`
`{ x = y * z }`	`{ y = x / z }`
`{ is_prime(y) }`	`{ is_odd(y) }`

Worksheet – problem 7

{ y > 23 }	is stronger than	{ y >= 23 }
{ y = 23 }		{ y >= 23 }
{ y < 0.23 }		{ y < 0.00023 }
{ x = y * z }		{ y = x / z }
{ is_prime(y) }		{ is_odd(y) }

Worksheet – problem 7

`{ y > 23 }`	is stronger than	`{ y >= 23 }`
`{ y = 23 }`	is stronger than	`{ y >= 23 }`
`{ y < 0.23 }`		`{ y < 0.00023 }`
`{ x = y * z }`		`{ y = x / z }`
`{ is_prime(y) }`		`{ is_odd(y) }`

Worksheet – problem 7

`{ y > 23 }`	is stronger than	`{ y >= 23 }`
`{ y = 23 }`	is stronger than	`{ y >= 23 }`
`{ y < 0.23 }`	is weaker than	`{ y < 0.00023 }`
`{ x = y * z }`		`{ y = x / z }`
`{ is_prime(y) }`		`{ is_odd(y) }`

Worksheet – problem 7

`{ y > 23 }`	is stronger than	`{ y >= 23 }`
`{ y = 23 }`	is stronger than	`{ y >= 23 }`
`{ y < 0.23 }`	is weaker than	`{ y < 0.00023 }`
`{ x = y * z }`	is **incomparable** with	`{ y = x / z }`
`{ is_prime(y) }`		`{ is_odd(y) }`

Worksheet – problem 7

`{ y > 23 }`	is stronger than	`{ y >= 23 }`
`{ y = 23 }`	is stronger than	`{ y >= 23 }`
`{ y < 0.23 }`	is weaker than	`{ y < 0.00023 }`
`{ x = y * z }`	is **incomparable** with	`{ y = x / z }`
`{ is_prime(y) }`	is **incomparable** with	`{ is_odd(y) }`

Weakest precondition, casually

- Backward reasoning lets us figure out what assumptions we need at the beginning to reach our goal at the end
- The idea of weakest precondition is to formalize this process with mechanical rules, one for each kind of statement:
  - Assignment: Substitute expression for variable
  - Sequence: Chain the process from last to first
  - Conditional: Merge branches as cases of condition
- Bonus: That formalized process is guaranteed to yield the most lenient (least restrictive) assumption necessary

Weakest precondition, formally

- If P^* is the weakest precondition, then
  - $\{P^*\} S \{Q\}$
  - $P \Rightarrow P^*$ for all P such that $\{P\} S \{Q\}$

- The "predicate transformer" wp(S, Q) gives a mechanical process to derive the weakest precondition:
  - wp(`x = e;`, Q) is Q with each **`x`** replaced by **`e`**
  - wp(S_1 ; S_2, Q) is wp(S_1, wp(S_2, Q))
  - wp(**`if (b)`** S_1 **`else`** S_2, Q) is (**`b`** $\wedge$ wp(S_1,Q)) $\vee$ (**`!b`** $\wedge$ wp(S_2,Q))

Weakest precondition – example

wp($\mathbf{x = y*y}$, $\mathbf{x > 4}$)

= $\mathbf{y*y > 4}$

= $|\mathbf{y}| > 2$

wp($\mathbf{x = e;}$, Q) is Q with each $\mathbf{x}$ replaced by $\mathbf{e}$

wp(S_1 ; S_2, Q) is wp(S_1, wp(S_2, Q))

wp($\mathbf{y = x+1; z = y-3}$, $\mathbf{z = 10}$)

= wp($\mathbf{y = x+1}$, wp($\mathbf{z = y-3}$, $\mathbf{z = 10}$))

= wp($\mathbf{y = x+1}$, $\mathbf{y-3 = 10}$)

= $\mathbf{(x+1)-3 = 10}$

= $\mathbf{x-2 = 10}$

= $\mathbf{x = 12}$

Questions?

- What is the most surprising thing about this?
- What is the most confusing thing?
- What will need a bit more thinking to digest?