CSE 331
Software Design & Implementation

Autumn 2022

Section 1 – Code Reasoning

Administrivia

- HW1 due next Wednesday at **11PM**.
  - Last two questions have loops: covered tomorrow in lecture.
- Please read the syllabus and watch the video commentary!
- Any questions before we dive in?
  - Admin stuff or straight-line reasoning?

Some Section Reminders

- Each week, we will introduce the homework assignment for the following week
- We will often practice topics presented in lecture
- Section is not recorded, but materials are posted
- We do not take attendance, but you should attend as much as possible

Agenda

- Introductions?
- Review logical reasoning about code with Floyd Logic
- Review logical strength of assertions (weaker *vs.* stronger)
- Practice logical reasoning and determining stronger/weaker assertions
- Talk about reasoning through conditionals, forward and backward

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 <u>pre</u>condition to <u>post</u>condition

- Backward reasoning
  - What facts need to be true to reach a goal?
  - Go from <u>post</u>condition to <u>pre</u>condition

Floyd Logic: Validity by Reasoning

- Checking validity of **{{*P*}} *S* {{*Q*}}**
  - Valid iff, starting from any state satisfying ***P***, executing ***S*** results in a state satisfying ***Q***

- Forward reasoning:
  - Reason from ***P*** to strongest postcondition **{{*P*}} *S* {{*R*}}**
  - Check that ***R*** implies ***Q*** (i.e., ***Q*** is weaker)

- Backward reasoning:
  - Reason from ***Q*** to get weakest precondition **{{*R*}} *S* {{*Q*}}**
  - Check that ***P*** implies ***R*** (i.e., ***P*** is stronger)

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

Assignment Statements

- Reasoning about `x = y;`

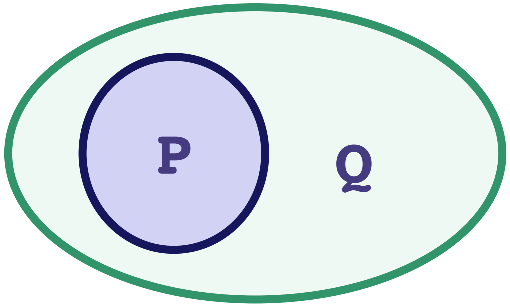
- Forward reasoning:
  - add “x = y” as a new fact
  - (also rewrite any existing references to “x” to use new value)

- Backward reasoning:
  - replace all instances of “x” in the postcondition with “y”

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

- Let’s do 1, 3, and 7 on the worksheet
  - If you have extra time, work on 5
- Find a partner or small group and work with them
- Let us know if you feel stuck
- We’ll walk through some solutions afterwards

Worksheet – problem 3 (backwards)

```
{{ x + 3 * b - 4 > 0 }}
a = x + b;
{{ a + 2 * b - 4 > 0 }}
c = 2 * b - 4;
{{ a + c > 0 }}
x = a + c;
{{ x > 0 }}
```

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 equally strong as `{{ 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 equally strong as	`{{ 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 equally strong as `{{ y < 0.00023 }}`

`{{ x = y * z }}` is **incomparable** with `{{ y = x / z }}`

`{{ is_prime(y) }}` is **incomparable** with `{{ is_odd(y) }}`

If Statements

Forward reasoning

{{ P }}
```
if (cond)
```
{{ P and `cond` }}
```
S1
```
{{ P1 }}
```
else
```
{{ P and not `cond` }}
```
S2
```
{{ P2 }}

{{ P1 or P2 }}

Worksheet – problem 2 (example)

```
{{ true }}
if (x>0) {
  {{ x > 0 }}
  y = 2*x;
  {{ x > 0 ∧ y = 2x }}
} else {
  {{ x <= 0 }}
  y = -2*x;
  {{ x <= 0 ∧ y = -2x }}
}
{{ (x > 0 ∧ y = 2x) ∨ (x <= 0 ∧ y = -2x) }}
⇒ {{ y = 2|x| }}
```

If Statements

Backward reasoning

```
{{ cond and Q1 or
   not cond and Q2 }}
if (cond)
  {{ Q1 }}
  S1
  {{ Q }}
else
  {{ Q2 }}
  S2
  {{ Q }}
{{ Q }}
```

Worksheet – problem 4 (example)

```
{{ 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

- Now, let’s try number 6 on the worksheet
  - You can do forward or backward

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) }}
```

Questions?

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