Announcements

- First section tomorrow!
- Homework 0 due today (Wednesday) at 10 pm
  - Heads up: no late days for this one!
- Quiz 1 due tomorrow (Thursday) at 10 pm
- Homework 1 due Monday at 10 pm
  - Will be posted by tomorrow
- Message board
  - Use “needs-answer” tag on questions that need an answer
- Collaboration policy clarification

Overview

- Motivation
- Reasoning Informally
- Hoare Logic
- Weaker and Stronger Statements
- Variable Renaming

Note: This lecture has very helpful notes on the course website!
### Formalization and Reasoning

Geometry gives us incredible power
- Lets us represent shapes symbolically
- Provides basic truths about these shapes
- Gives rules to combine small truths into bigger truths

Geometric proofs often establish *general* truths

![Geometric diagram](image)

\[ a^2 + b^2 = c^2 \]
\[ p + q + r = 180 \]

### Why Reason About Programs?

Essential complement to *testing*
- Testing shows specific result for a specific input

*Proof* shows general result for entire class of inputs
- *Guarantee* code works for *any* valid input
- Can only prove correct code, proving uncovers bugs
- Provides deeper understanding of why code is correct

Precisely stating assumptions is essence of spec
- “Callers must not pass *null* as an argument”
- “Callee will always return an unaliased object”

### Formalization and Reasoning

Formal reasoning provides tradeoffs
+ Establish truth for many (possibly infinite) cases
+ Know properties ahead of time, before object exists
- Requires abstract reasoning and careful thinking
- Need basic truths and rules for combining truths

Today: develop formal reasoning for programs
- What is true about a program’s state as it executes?
- How do basic constructs change what’s true?
- Two flavors of reasoning: *forward* and *backward*

### Reasoning About Programs

- Formal reasoning tells us what’s true of a program’s state as it executes, given an initial assumption or a final goal
- What are some things we might want to know about certain programs?
  - If \( x > 0 \) initially, then \( y == 0 \) when loop exits
  - Contents of array \( arr \) refers to are sorted
  - Except at one program point, \( x + y == z \)
  - For all instances of \( Node \ n \),
    \[ n.next == null \lor n.next.prev == n \]
  - ...
Why Reason About Programs?

“Today a usual technique is to make a program and then to test it. *While program testing can be a very effective way to show the presence of bugs, it is hopelessly inadequate for showing their absence.* The only effective way to raise the confidence level of a program significantly is to give a convincing proof of its correctness.”

-- Dijkstra (1972)

Overview

- Motivation
- Reasoning Informally
- Hoare Logic
- Weaker and Stronger Statements
- Variable Renaming

Reasoning Informally

- Re-explain to your neighbor (groups of 3-4)
- TAs may have some useful insights!
- Then share interesting thoughts/questions from your discussions.
Our Approach

Hoare Logic, an approach developed in the 70’s
- Focus on core: assignments, conditionals, loops
- Omit complex constructs like objects and methods

Today: the basics for assign, sequence, if in 3 steps
1. High-level intuition for forward and backward reasoning
2. Precisely define assertions, preconditions, etc.
3. Define weaker/stronger and weakest precondition

Next lecture: loops

How Does This Get Used?

Current practitioners rarely use Hoare logic explicitly
- For simple program snippets, often overkill
- For full language features (aliasing) gets complex
- Shines for developing loops with subtle invariants
  - See Homework 0, Homework 2

Ideal for introducing program reasoning foundations
- How does logic “talk about” program states?
- How can program execution “change what’s true”?
- What do “weaker” and “stronger” mean in logic?

All essential for specifying library interfaces!

Informal Notation Warning

- The slides in this section have informal notation
- You will need to use more formal notation on your homework (after hw0)

Forward Reasoning Example

Suppose we initially know (or assume) \( w > 0 \)

```plaintext
// w > 0
x = 17;
// w > 0  ∧  x == 17
y = 42;
// w > 0  ∧  x == 17  ∧  y == 42
z = w + x + y;
// w > 0  ∧  x == 17  ∧  y == 42  ∧  z > 59
...
```

Then we know various things after, e.g., \( z > 59 \)
Backward Reasoning Example

Suppose we want \( z < 0 \) at the end

\[
// \ w + 17 + 42 < 0 \\
x = 17; \\
// \ w + x + 42 < 0 \\
y = 42; \\
// \ w + x + y < 0 \\
z = w + x + y; \\
// \ z < 0
\]

Then initially we need \( w < -59 \)

For the assertion after this statement to be true, what must be true before it?

Forward vs. Backward

Forward Reasoning
- Determine what follows from initial assumptions
- Useful for ensuring an invariant is maintained

Backward Reasoning
- Determine sufficient conditions for a certain result
- Desired result: assumptions need for correctness
- Undesired result: assumptions needed to trigger bug

Conditionals

// initial assumptions
if(...) {
    ... // also know condition is true
} else {
    ... // also know condition is false
}
// either branch could have executed

Key ideas:
1. The precondition for each branch includes information about the result of the condition
2. The overall postcondition is the disjunction ("or") of the postconditions of the branches
Conditional Example (Fwd)

// x >= 0
z = 0;
// x >= 0 ∧ z == 0
if(x != 0) {
    // x >= 0 ∧ z == 0 ∧ x != 0 (so x > 0)
    z = x;
    // ... ∧ z > 0
} else {
    // x >= 0 ∧ z == 0 ∧ !(!x!=0) (so x == 0)
    z = x + 1;
    // ... ∧ z == 1
}
// ( ... ∧ z > 0) ∨ (... ∧ z >= 1) (so z > 0)

Overview

- Motivation
- Reasoning Informally
- Hoare Logic
- Weaker and Stronger Statements
- Variable Renaming

Our Approach

**Hoare Logic**, an approach developed in the 70’s
- Focus on core: assignments, conditionals, loops
- Omit complex constructs like objects and methods

Today: the basics for **assign, sequence, if** in 3 steps
1. High-level intuition for forward and backward reasoning
2. Precisely define assertions, preconditions, etc.
3. Define weaker/stronger and weakest precondition

Next lecture: loops
Notation and Terminology

Precondition: “assumption” before some code
Postcondition: “what holds” after some code

Conventional to write pre/postconditions in “{...}”

\{ w < -59 \}
x = 17;
\{ w + x < -42 \}

Preconditions and Postconditions are two types of Formal Assertions.

Notation and Terminology

Note the “{...}” notation is NOT Java

Within pre/postcondition “=” means mathematical equality, like Java’s “==” for numbers

\{ w > 0 \land x = 17 \}
y = 42;
\{ w > 0 \land x = 17 \land y = 42 \}

Assertion Semantics (Meaning)

An assertion (pre/postcondition) is a logical formula that can refer to program state (variables)

Given a variable, a program state tells you its value
  • Or the value for any expression with no side effects

An assertion holds on a program state if evaluating the assertion using the program state produces true
  • An assertion represents the set of state for which it holds

Hoare Triples

A Hoare triple is code wrapped in two assertions

\{ P \} S \{ Q \}

• P is the precondition
• S is the code (statement)
• Q is the postcondition

Hoare triple \{ P \} S \{ Q \} is valid if:
• For all states where P holds, executing S always produces a state where Q holds
• “If P true before S, then Q must be true after”
• Otherwise the triple is invalid
Hoare Triple Examples

Valid or invalid?
- Assume all variables are integers without overflow

\{ x \neq 0 \} y = x*x; \{ y > 0 \} \quad \text{valid}

\{ z \neq 1 \} y = z*z; \{ y \neq z \} \quad \text{invalid}

\{ x \geq 0 \} y = 2*x; \{ y > x \} \quad \text{invalid}

\{ \text{true} \} (\text{if}(x > 7)\{ y=4; \} \text{else} \{ y=3; \}) \{ y < 5 \} \quad \text{valid}

\{ \text{true} \} (x = y; z = x;) \{ y=z \} \quad \text{valid}

\{ x=7 \land y=5 \}
\{ \text{tmp} = x; x=\text{tmp}; y=x; \} \quad \text{invalid}

\{ y=7 \land x=5 \}

The General Rules

So far, we decided if a Hoare trip was valid by using our informal understanding of programming constructs

Now we’ll show a general rule for each construct
- The basic rule for assignments (they change state!)
- The rule to combine statements in a sequence
- The rule to combine statements in a conditional
- The rule to combine statements in a loop [next time]

Aside: assert in Java

A Java assertion is a statement with a Java expression

\text{assert} (x > 0 \&\& y < x);

Similar to our assertions
- Evaluate with program state to get true or false

Different from our assertions
- Java assertions work at \text{run-time}
- Raise an exception if this execution violates assert
  - … unless assertion checking disable (discuss later)

This week: we are \textit{reasoning} about the code \textit{statically} (before run-time), not checking a particular input

Basic Rule: Assignment

\{ P \} x = e; \{ Q \}

Let \( Q' \) be like \( Q \) except replace \( x \) with \( e \)

Triple is valid if:
- The basic rule for assignments (they change state!)
- The rule to combine statements in a sequence
- The rule to combine statements in a conditional
- The rule to combine statements in a loop [next time]

Example: \{ z > 34 \} y = z + 1; \{ y > 1 \}
- \( Q' \) is \{ z + 1 > 1 \}
Combining Rule: Sequence

\( \{ P \} S_1; S_2 \{ Q \} \)

Triple is valid iff there is an assertion \( R \) such that both the following are valid:
- \( \{ P \} S_1 \{ R \} \)
- \( \{ R \} S_2 \{ Q \} \)

Example:
\[
\{ z \geq 1 \} \\
y = z + 1; \\
w = y \times y; \\
\{ w > y \}
\]

Let \( R = \{ y > 1 \} \)
1. Show \( \{ z \geq 1 \} \ y = z + 1 \{ y > 1 \} \)
   Use basic assign rule:
   \( z \geq 1 \) implies \( z + 1 > 1 \)
2. Show \( \{ y > 1 \} \ w = y \times y \{ w > y \} \)
   Use basic assign rule:
   \( y > 1 \) implies \( y \times y > y \)

Combining Rule: Conditional

\( \{ P \} \text{if}(b) S_1 \text{else} S_2 \{ Q \} \)

Triple is valid iff there are assertions \( Q_1, Q_2 \) such that:
- \( \{ P \wedge \neg b \} S_1 \{ Q_1 \} \) is valid
- \( \{ P \wedge b \} S_2 \{ Q_2 \} \) is valid
- \( Q_1 \wedge \neg Q_2 \) implies \( Q \)

Example:
\[
\{ \text{true} \} \\
\text{if}(x > 7) \\
y = x; \\
\text{else} \\
y = 20; \\
\{ y > 5 \}
\]

Let \( Q_1 = \{ y > 7 \} \) and \( Q_2 = \{ y = 20 \} \)
1. Show \( \{ \text{true} \wedge x > 7 \} \ y = x \{ y > 7 \} \)
2. Show \( \{ \text{true} \wedge x \leq 7 \} \ y = 20 \{ y = 20 \} \)
3. Show \( y > 7 \wedge y = 20 \) implies \( y > 5 \)

What if we change the code in a way that changes \( Q_2 \) to “\( y=4 \)”

\( y > 7 \text{ OR } y = 20 \)
\( y > 5 \)
\( y > 7 \text{ OR } y = 4 \)

Let \( Q_1 = \{ y > 7 \} \) and \( Q_2' = \{ y = 4 \} \)
1. Show \( \{ \text{true} \wedge x > 7 \} \ y = x \{ y > 7 \} \)
2. Show \( \{ \text{true} \wedge x \leq 7 \} \ y = 20 \{ y = 20 \} \)
3. Show \( y > 7 \wedge y = 20 \) implies \( y > 5 \)
Overview

- Motivation
- Reasoning Informally
- Hoare Logic
  - Weaker and Stronger Statements
  - Variable Renaming

Our Approach

Hoare Logic, an approach developed in the 70's
- Focus on core: assignments, conditionals, loops
- Omit complex constructs like objects and methods

Today: the basics for assign, sequence, if in 3 steps
1. High-level intuition for forward and backward reasoning
2. Precisely define assertions, preconditions, etc.
3. Define weaker/stronger and weakest precondition

Next lecture: loops

Weaker vs. Stronger

If $P_1$ implies $P_2$ (written $P_1 \Rightarrow P_2$) then:
- $P_1$ is stronger than $P_2$
- $P_2$ is weaker than $P_1$

Whenever $P_1$ holds, $P_2$ is guaranteed to hold
- So it is at least as difficult to satisfy $P_1$ as $P_2$
- $P_1$ holds on a subset of the states where $P_2$ holds
- $P_1$ puts more constraints on program states
- $P_1$ is a “stronger” set of obligations / requirements
**Weaker vs. Stronger Examples**

- **x = 17** is stronger than **x > 0**.
- **x is prime** neither stronger nor weaker than **x is odd**.
- **x is prime / \ x > 2** is stronger than **x is odd / \ x > 2**.

**Strength and Hoare Logic**

Suppose:
- \{P\} S \{Q\} and
- P is weaker than some P\(_1\) and
- Q is stronger than some Q\(_1\)

Then \{P\(_1\)\} S \{Q\} and \{P\} S \{Q\(_1\)\} and \{P\(_1\)\} S \{Q\(_1\)\}

Example:
- P is x >= 0
- P\(_1\) is x > 0
- S is y = x+1
- Q is y > 0
- Q\(_1\) is y >= 0

"Wiggle Room"

**Weakest Precondition**

wp(x = e, Q) is Q with each x replaced by e

- Example: wp(x = y*y; x > 4) is y*y > 4, i.e., |y| > 2

wp(S\(_1\); S\(_2\), Q) is wp(S\(_1\), wp(S\(_2\), Q))
- i.e., let R be wp(S\(_2\), Q) and overall wp is wp(S\(_1\), R)
- Example: wp((y=x+1; z=y+1;), z > 2) is (x + 1) + 1 > 2, i.e., x > 0

wp(if b S\(_1\) else S\(_2\), Q) is this logical formula:

(b ∧ wp(S\(_1\), Q)) ∨ (!b ∧ wp(S\(_2\), Q))
- In any state, b will evaluate to either true or false...
- You can sometimes then simplify the result
Simple Examples

If $S$ is $x = y*y$ and $Q$ is $x > 4$, then $wp(S,Q)$ is $y*y > 4$, i.e., $|y| > 2$.

If $S$ is $y = x + 1; z = y - 3;$ and $Q$ is $z = 10$, then $wp(S,Q)$...

$= wp(y = x + 1; z = y - 3; , z = 10)$
$= wp(y = x + 1; , wp(z = y - 3; , z = 10))$
$= wp(y = x + 1; , y-3 = 10)$
$= wp(y = x + 1; , y = 13)$
$= x+1 = 13$
$= x = 12$

Bigger Example

$S$ is if (x < 5) {
  x = x*x;
} else {
  x = x+1;
}
$Q$ is $x >= 9$

$wp(S, x >= 9)$

= (x < 5 ∧ wp(x = x*x;, x >= 9))
  ∨ (x >= 5 ∧ wp(x = x+1;, x >= 9))
  = (x < 5 ∧ x*x >= 9)
  ∨ (x >= 5 ∧ x+1 >= 9)
  = (x <= -3)  ∨ (x >= 3 ∧ x < 5)
  ∨ (x >= 8)

“Correct”

If $wp(S, Q)$ is true, then executing $S$ will always produce a state where $Q$ holds, since true holds for every program state.

If our program state only has one variable, $x$, we can think of the true precondition as an assertion that holds for all values of $x$. 
Oops! Forward Bug…

With forward reasoning, our intuitive rule for assignment is wrong:
• Changing a variable can affect other assumptions

Example:

\[
\begin{align*}
\{ \text{true} \} \\
& w = x + y; \\
& \{ w = x + y; \} \\
& x = 4; \\
& \{ w = x + y \land x = 4 \} \\
& y = 3; \\
& \{ w = x + y \land x = 4 \land y = 3 \}
\end{align*}
\]

But clearly we do not know \( w = 7 \) (!!!)

Fixing Forward Assignment

When you assign to a variable, you need to replace all other uses of the variable in the post-condition with a different “fresh” variable, so that you refer to the “old contents”

Corrected example:

\[
\begin{align*}
\{ \text{true} \} \\
& w = x + y; \\
& \{ w = x + y; \} \\
& x = 4; \\
& \{ w = x_1 + y \land x = 4 \} \\
& y = 3; \\
& \{ w = x_1 + y_1 \land x = 4 \land y = 3 \}
\end{align*}
\]

Useful Example: Swap

Name initial contents so we can refer to them in the post-condition

Just in the formulas: these “names” are not in the program

Use these extra variables to avoid “forgetting” “connections”

\[
\begin{align*}
& \{ x = x_{pre} \land y = y_{pre} \} \\
& \text{tmp} = x; \\
& \{ x = x_{pre} \land y = y_{pre} \land \text{tmp}=x \} \\
& x = y; \\
& \{ x = y \land y = y_{pre} \land \text{tmp}=x_{pre} \} \\
& y = \text{tmp}; \\
& \{ x = y_{pre} \land y = \text{tmp} \land \text{tmp}=x_{pre} \}
\end{align*}
\]

Announcements
Announcements

• First section tomorrow!
• Homework 0 due today (Wednesday) at 10 pm
  • Heads up: no late days for this one!
• Quiz 1 due tomorrow (Thursday) at 10 pm
• Homework 1 due Monday at 10 pm
  • Will be posted by tomorrow
• Read the Formal Reasoning Notes
  • posted on the course website
• Friday’s lecture is on one of the hardest topics