Warm-Up Problem

Let Σ be a set of well-formed Predicate logic formulas. Let α, β be well-formed Predicate logic formulas. Prove or disprove the following.

If

$$\Sigma \vdash (\alpha \vee \beta)$$

then

$$\Sigma \cup \{(\neg\alpha)\} \vDash \beta$$

Program Verification

Carmen Bruni

Lecture 18

Based on slides by Jonathan Buss, Lila Kari, Anna Lubiw and Steve Wolfman with thanks to B. Bonakdarpour, A. Gao, D. Maftuleac, C. Roberts, R. Trefler, and P. Van Beek

Last Time

- Soundness and Completeness of Natural Deduction for Predicate Logic.

Last Time

- Soundness and Completeness of Natural Deduction for Predicate Logic.
- But before we leave Predicate Logic, one final review question!

Learning Goals

- Give reasons for performing formal verification vs testing.
- Define a Hoare Triple.
- Define Partial Correctness.
- Prove that a Hoare triple is satisfied under partial correctness for a program containing assignment and conditional statements.

Program Correctness

Does a program satisfy its specification? (Does it do what it is supposed to do?

How do show that a program works correctly?

- Testing (black box and white box)
- Formal verification

Techniques for verifying program correctness

Testing

- Check a program for carefully chosen inputs.
- Cannot be exhaustive in general.

Formal Verification:

- State a specification formally.
- Prove that a program satisfies the specification for all inputs.

Why is testing not sufficient?

Testing can be a very effective way to show the presence of bugs, but it is hopelessly inadequate for showing their absence.

E. Dijkstra, 1972.

Why formally specify and verify programs

- Discover and reduce bugs especially for safety-critical software and hardware
- Documentation facilitates collaboration and code re-use.

What is being done in practice?

What do we do in practice?

- Formal software specification is widespread.
- Formally verifying software is less widespread.
- Hardware verification is common.

Without formal verification, what could go wrong?

- Therac-25, X-ray, 1985
  - Overdosing patients during radiation treatment, 5 dead
  - Reason: race condition between concurrent tasks
- AT&T, 1990
  - Long distance service fails for 9 hours.
  - Reason: wrong BREAK statement in C code
- Patriot-Scud, 1991
  - 28 dead and 100 injured
  - Reason: rounding error
- Pentium Processor, 1994
  - The division algorithm is incorrect.
  - Reason: incomplete entries in a look-up table

Without formal verification, what could go wrong?

- Ariane 5, 1996
  - Exploded 37 seconds after takeoff
  - Reason: data conversion of a number that was too large
- Mars Climate Orbiter, 1999
  - destroyed on entering atmosphere of Mars
  - Reason: mixture of pounds and kilograms
- Power Blackout, 2003
  - 50 million people in Canada and USA without power
  - Reason: Programming error (race condition)
- Royal Bank, 2004
  - Financial transactions disrupted for 5 days
  - Reason: Programming error

Without formal verification, what could go wrong?

- Science [prestigious journal], 2006
  - Retraction of research papers due to erroneous research results.
  - Reason: Program incorrectly flipped positive sign to negative on data.
- Toyota Prius, 2007
  - 160,000 hybrid vehicles recalled due to stalling unexpectedly
  - Reason: Programming error
- Knight Capital Group, 2012
  - High frequency trading system lost $440 million in 30 minutes
  - Reason: Programming Error
- Spectre and Meltdown (Intel Chipset), 2018
  - Chipset vulnerabilities that could allow hackers to access private data
  - Reason: Hardware bug

The process of formal verification

1. Convert an informal description R of requirements for a program into a logical formula φ_R.
2. Write a program P which is meant to satisfy the requirements R above.
3. Prove that program P satisfies the formula φ_R.

We will consider only the third part in this course.

Our programming language

We will use a subset of C/C++ and Java.

Core features of our language:

- integer and Boolean expressions
- assignment statements
- conditional statements
- while-loops
- arrays

Imperative programs

- A program manipulates variables.
- The state of a program consists of the values of variables at a particular time in the program execution.
- A sequence of commands modify the state of the program.
- Given inputs, the program produce outputs.

Example

We shall use the following code as an example.

Compute the factorial of input `x` and store in `y`.

```
y = 1;
z = 0;

while (z != x) {
    z = z + 1;
    y = y * z;
}
```

Example

Compute the factorial of input `x` and store in `y`.

```
    y = 1;
    z = 0;
⟶  while (z != x) {
        z = z + 1;
        y = y * z;
    }
```

State at the "`while`" test:

- Initial state s_0: z=0, y=1
- Next state s_1: z=1, y=1
- State s_2: z=2, y=2
- State s_3: z=3, y=6
- State s_4: z=4, y=24

$\vdots$

Example

Compute the factorial of input `x` and store in `y`.

```
    y = 1;
    z = 0;
⟶  while (z != x) {
        z = z + 1;
        y = y * z;
    }
```

State at the "`while`" test:

- Initial state s_0: z=0, y=1
- Next state s_1: z=1, y=1
- State s_2: z=2, y=2
- State s_3: z=3, y=6
- State s_4: z=4, y=24

$\vdots$

Note: the order of "`z = z + 1`" and "`y = y * z`" matters!

Formal specification

Two important components of a specification:

- The state **before** the program executes
- The state **after** the program executes

We want to develop a notion of proof that will allow us to prove that a program C satisfies the specification given by the precondition P and the postcondition Q. This is done using Hoare triples which differs from our usual notion of proof in two fundamental ways:

- program instructions (actions), rather than propositions.
- a sense of time: before execution versus after execution.

Hoare Triples

Our assertions about programs will have the form

$\{\!\{ P \}\!\}$	— precondition (a predicate formula)
C	— program or code
$\{\!\{ Q \}\!\}$	— postcondition (a predicate formula)

The meaning of the triple $\{\!\{ P \}\!\}\ C\ \{\!\{ Q \}\!\}$:

> If program C is run starting in a state that satisfies P, then the resulting state after the execution of C will satisfy Q.

An assertion $\{\!\{ P \}\!\}$ C $\{\!\{ Q \}\!\}$ is called a Hoare triple.

Specification of a Program

A *specification* of a program C is a Hoare triple with C as the second component: $(\!| P |\!)$ C $(\!| Q |\!)$.

Example. The requirement

> If the input x is a positive number, compute a number whose square is less than x

might be expressed as

$$(\!| (x > 0) |\!) \ C \ (\!| ((y \times y) < x) |\!) \ .$$

Specification Is Not Behaviour

A triple, such as $(\!| (x > 0) |\!)$ C $(\!| ((y \times y) < x) |\!)$, specifies neither a unique program C nor a unique behaviour.

For example, both $(\!| (x > 0) |\!)$ C_1 $(\!| ((y \times y) < x) |\!)$ and $(\!| (x > 0) |\!)$ C_2 $(\!| ((y \times y) < x) |\!)$ hold:

C_1:

```
y = 0 ;
```

C_2:

```
y = 0 ;
while (y * y < x) {
    y = y + 1 ;
    }
y = y - 1 ;
```

Logical variables

Sometimes the pre- and postconditions require additional variables that do not appear in the program.

These are called **logical variables**.

Example.

$\{ ((x = x_0) \wedge (x_0 \geq 0)) \}$

```
y = 1;
while (x != 0) {
    y = y * x;
    x = x - 1;
}
```

$\{ (y = x_0!) \}$

Logical variables

Sometimes the pre- and postconditions require additional variables that do not appear in the program.

These are called **logical variables**.

Example.

$\{ ((x = x_0) \wedge (x_0 \geq 0)) \}$

```
y = 1;
while (x != 0) {
    y = y * x;
    x = x - 1;
}
```

$\{ (y = x_0!) \}$

For a Hoare triple, its set of logical variables are those variables that are free in P or Q and do not occur in C.

Partial correctness

A triple $(\!| P |\!)\ C\ (\!| Q |\!)$ is satisfied under partial correctness, denoted

$$\models_{\mathtt{par}} (\!| P |\!)\ C\ (\!| Q |\!)\ ,$$

if and only if

> for every state s that satisfies condition P,
>
> if execution of C starting from state s terminates in a state s',
>
> then state s' satisfies condition Q.

Example

1. The Hoare triples
   $(\!| (x_0 = x) |\!)$
   `x = 3 ;`
   $(\!| (x = 3) |\!)$
   and
   $(\!| (x = x_0) |\!)$
   `x = 3 ;`
   $(\!| (x = 3) |\!)$
   are satisfied under partial correctness.
2. The Hoare triple
   $(\!| (x_0 = x) |\!)$
   `x = 2 ;`
   $(\!| (x = 1) |\!)$
   is **not** satisfied under partial correctness.

Proving Partial Correctness

Recall the definition of Partial Correctness:

> For every starting state which satisfies P and for which C terminates, the final state satisfies Q.

How do we show this, if there are a large or infinite number of possible states?

Answer: **Inference rules** (proof rules)

> Each construct in our programming language will have a rule.

Presentation of a Proof

A full proof will have one or more conditions before and after each code statement. Each statement makes a Hoare triple with the preceding and following conditions. Each triple (postcondition) has a justification that explains its correctness.

$\{\!|$ *program precondition* $|\!\}$
`y = 1;`
$\{\!|$ ... $|\!\}$ $\langle$*justification*$\rangle$
`while (x != 0) {`
    $\{\!|$ ... $|\!\}$ $\langle$*justification*$\rangle$
    `y = y * x;`
    $\{\!|$ ... $|\!\}$ $\langle$*justification*$\rangle$
    `x = x - 1;`
    $\{\!|$ ... $|\!\}$ $\langle$*justification*$\rangle$
`}`
$\{\!|$ *program postcondition* $|\!\}$ $\langle$*justification*$\rangle$

Inference Rule for Assignment

$$\frac{}{\{\!|\, Q[E/x] \,|\!\}\ \ (\mathtt{x} = E)\ \ \{\!|\, Q \,|\!\}} \text{ (assignment)}$$

Intuition:

$Q(x)$ will hold after assigning (the value of) E to x if Q was true of that value beforehand.

Assignment: Example

Example.

$$\vdash_{par} (\!| ((y+1)=7) |\!) \ \mathtt{x} \ = \ \mathtt{y} \ + \ \mathtt{1} \ (\!| (x=7) |\!)$$

by one application of the assignment rule.

Note: This $\vdash_{par} (\!| P |\!) \ C \ (\!| Q |\!)$ symbol will mean there is a proof using the assignment (and the other) rule(s).

This $\vDash_{\mathtt{par}} (\!| P |\!) \ C \ (\!| Q |\!)$ symbol will mean that the Hoare triple is true under the definition. Yes there is a soundness and completeness result that will hold here but this will be the only mention.

More Examples for Assignment

Example 1.

$\{\!\!\{ (y = 2) \}\!\!\}$	$\{\!\!\{ Q[E/x] \}\!\!\}$
`x = y ;`	$x = E;$
$\{\!\!\{ (x = 2) \}\!\!\}$	$\{\!\!\{ Q \}\!\!\}$

Example 2.

$\{\!\!\{ (0 < 2) \}\!\!\}$	$\{\!\!\{ Q[E/x] \}\!\!\}$
`x = 2 ;`	$x = E;$
$\{\!\!\{ (0 < x) \}\!\!\}$	$\{\!\!\{ Q \}\!\!\}$

Examples of Assignment

Example 3.

$\{\!|\ ((x+1)=2)\ |\!\}$ $\quad$ $\{\!|\ (x=2)[(x+1)/x]\ |\!\}$

```
x = x + 1 ;
```

$\{\!|\ (x=2)\ |\!\}$

Example 4.

$\{\!|\ ((x+1)=(n+1))\ |\!\}$ $\quad$ (equivalent to $\{\!|\ (x=n)\ |\!\}$)

```
x = x + 1;
```

$\{\!|\ (x=(n+1))\ |\!\}$

Note about Examples

In program correctness proofs, we usually work backwards from the postcondition:

???	$\{\!\mid Q[E/x] \mid\!\}$
x = y ;	$x = E;$
$\{\!\mid (x > 0) \mid\!\}$	$\{\!\mid Q \mid\!\}$

Inference Rules with Implications

Rule of "Precondition strengthening":

$$\frac{P \to P' \qquad \{\!|\, P' \,|\!\}\ C\ \{\!|\, Q \,|\!\}}{\{\!|\, P \,|\!\}\ C\ \{\!|\, Q \,|\!\}}\ \textit{(implied)}$$

Rule of "Postcondition weakening":

$$\frac{\{\!|\, P \,|\!\}\ C\ \{\!|\, Q' \,|\!\} \qquad Q' \to Q}{\{\!|\, P \,|\!\}\ C\ \{\!|\, Q \,|\!\}}\ \textit{(implied)}$$

Example of use:

$\{\!|\, (y = 6) \,|\!\}$

$\{\!|\, ((y + 1) = 7) \,|\!\}$ implied

```
x = y + 1
```

$\{\!|\, (x = 7) \,|\!\}$ assignment

Inference Rule for Sequences of Instructions

$$\frac{(\!| P |\!)\ C_1\ (\!| Q |\!), \quad (\!| Q |\!)\ C_2\ (\!| R |\!)}{(\!| P |\!)\ C_1; C_2\ (\!| R |\!)}\ \textit{(composition)}$$

In order to prove $(\!| P |\!)\ C_1; C_2\ (\!| R |\!)$, we need to find a midcondition Q for which we can prove $(\!| P |\!)\ C_1\ (\!| Q |\!)$ and $(\!| Q |\!)\ C_2\ (\!| R |\!)$.

(In our examples, the midcondition will usually be determined by a rule, such as assignment. But in general, a midcondition might be very difficult to determine.)

Proof Format: Annotated Programs

Interleave program statements with assertions, each justified by an inference rule.

The composition rule is implicit.

Assertions should hold true whenever the program reaches that point in its execution.

Proof Format: Annotated Programs

If implied inference rule is used, we must supply a proof of the implication.

- We'll do these proofs after annotating the program.

Each assertion should be an instance of an inference rule.
Normally,

- Don't simplify the assertions in the annotated program.
- Do the simplification while proving the implied conditions.

Example: Composition of Assignments

To show: the following is satisfied under partial correctness.

We work bottom-up for assignments...

$$\{\!|\, ((x = x_0) \wedge (y = y_0)) \,|\!\}$$

```
t = x ;
```

```
x = y ;
```

```
y = t ;
```

$$\{\!|\, ((x = y_0) \wedge (y = x_0)) \,|\!\}$$

Example: Composition of Assignments

To show: the following is satisfied under partial correctness.

We work bottom-up for assignments...

$$\{\!|\ ((x = x_0) \wedge (y = y_0))\ |\!\}$$

```
t = x ;
```

```
x = y ;
```
$\{\!|\ ((x = y_0) \wedge (t = x_0))\ |\!\}$ $\quad P_2$ is $\{\!|\ P[t/y]\ |\!\}$

```
y = t ;
```
$\{\!|\ ((x = y_0) \wedge (y = x_0))\ |\!\}$ $\quad$ assignment $\{\!|\ P\ |\!\}$

Example: Composition of Assignments

To show: the following is satisfied under partial correctness.

We work bottom-up for assignments...

$$\{\!|\ ((x = x_0) \wedge (y = y_0))\ |\!\}$$

```
t = x ;
```
$\{\!|\ ((y = y_0) \wedge (t = x_0))\ |\!\}$ P_3 is $\{\!|\ P_2[y/x]\ |\!\}$

```
x = y ;
```
$\{\!|\ ((x = y_0) \wedge (t = x_0))\ |\!\}$ assignment

```
y = t ;
```
$\{\!|\ ((x = y_0) \wedge (y = x_0))\ |\!\}$ assignment

Example: Composition of Assignments

To show: the following is satisfied under partial correctness.

We work bottom-up for assignments...

$$\{\!|\, ((x = x_0) \wedge (y = y_0)) \,|\!\}$$
$$\{\!|\, ((y = y_0) \wedge (x = x_0)) \,|\!\} \qquad \{\!|\, P_3[x/t] \,|\!\}$$
`t = x ;`
$$\{\!|\, ((y = y_0) \wedge (t = x_0)) \,|\!\} \qquad \text{assignment}$$
`x = y ;`
$$\{\!|\, ((x = y_0) \wedge (t = x_0)) \,|\!\} \qquad \text{assignment}$$
`y = t ;`
$$\{\!|\, ((x = y_0) \wedge (y = x_0)) \,|\!\} \qquad \text{assignment}$$

Example: Composition of Assignments

To show: the following is satisfied under partial correctness.

We work bottom-up for assignments...

$$
\begin{array}{ll}
(\!| ((x = x_0) \wedge (y = y_0)) |\!) & \\
(\!| ((y = y_0) \wedge (x = x_0)) |\!) & \text{implied } \textit{[proof required]} \\
\texttt{t = x ;} & \\
(\!| ((y = y_0) \wedge (t = x_0)) |\!) & \text{assignment} \\
\texttt{x = y ;} & \\
(\!| ((x = y_0) \wedge (t = x_0)) |\!) & \text{assignment} \\
\texttt{y = t ;} & \\
(\!| ((x = y_0) \wedge (y = x_0)) |\!) & \text{assignment}
\end{array}
$$

Finally, show $(\!| ((x = x_0) \wedge (y = y_0)) |\!)$ implies $(\!| ((y = y_0) \wedge (x = x_0)) |\!)$.