Program Verification

HE Mingxin, Max
CS104: program07 @ yeah.net
CS108: mxhe1 @ yeah.net

I2ML(H) Spring 2023 (CS104|CS108)

Exercises 14 : Reading and More

Record your time spent (in 0.1 hours) with brief tasks and durations in your learning log by hand writing!

1) Read textB-ch04-ProgramVerification.pdf (in two weeks)

Outline

Key Points

Key points to learn:

- Give reasons for performing formal verification rather than testing.
- Define a Hoare triple.
- Define partial correctness.
- Define total correctness.

Program Correctness

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

How do we show that a program works correctly?

- Walk through the code
- 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.

Quiz: t = x; x = y; y = t; execute an exchange.

Can we exchange 2 variables without extra space?

Quiz: t = x; x = y; y = t; execute an exchange.

Can we exchange 2 variables without extra space?

Look at the Code:

```
int x, y, t;
…
t = x;
x = y;
y = t;
```

Why do we believe the code exchange the values of 2 variables?

Quiz: t = x; x = y; y = t; execute an exchange.

Can we exchange 2 variables without extra space?

Look at the Code:

```
int x, y, t;
…
t = x;
x = y;
y = t;
```

Why do we believe the code exchange the values of 2 variables?

Prove it:

```
// x=A & y=B; pre-cond.
t = x;
// x=A & y=B & t=A
x = y;
// x=B & y=B & t=A
y = t;
// x=B & y=A & t=A; post-cond.
```

Compare the pre-condition and the post-condition, we may conclude that IT IS TRUE.

Solution 1: Use additions

Look at the Code:

```
int x, y;
…
x = x + y;
y = x - y;
x = x - y;
```

Are you sure?

Solution 1: Use additions

Look at the Code:

```
int x, y;
…
x = x + y;
y = x - y;
x = x - y;
```

Are you sure?

Prove it:

```
// x=A & y=B;  pre-cond.
x = x + y;
// x=A+B & y=B
y = x - y;
// x=A+B & y=A
y = x - y;
// x=B & y=A;  post-cond.
```

Compare the pre-condition and the post-condition, we may conclude that IT IS TRUE.

Solution 1: Use additions

Look at the Code:

```
int x, y;
…
x = x + y;
y = x - y;
x = x - y;
```

Are you sure?

Prove it:

```
// x=A & y=B;  pre-cond.
x = x + y;
// x=A+B & y=B
y = x - y;
// x=A+B & y=A
y = x - y;
// x=B & y=A;  post-cond.
```

Compare the pre-condition and the post-condition, we may conclude that IT IS TRUE.

Q: Any Problem?

Solution 1: Use additions

Look at the Code:

```
int x, y;
…
x = x + y;
y = x - y;
x = x - y;
```

Are you sure?

Prove it:

```
// x=A & y=B;  pre-cond.
x = x + y;
// x=A+B & y=B
y = x - y;
// x=A+B & y=A
y = x - y;
// x=B & y=A;  post-cond.
```

Compare the pre-condition and the post-condition, we may conclude that IT IS TRUE.

Q: Any Problem? A: May have Overflow!

Solution 2: Use Bit-wise Exclusive OR (XOR) ^

Look at the Code:

```
int x, y;
…
x = x ^ y;
y = x ^ y;
x = x ^ y;
```

Are you sure?

on XOR:

```
1^b => ~b
0^b => b
b^b => 0
a^b^b => a
```

Solution 2: Use Bit-wise Exclusive OR (XOR) ^

Look at the Code:

```
int x, y;
…
x = x ^ y;
y = x ^ y;
x = x ^ y;
```

Are you sure?

on XOR:

```
1^b => ~b
0^b => b
b^b => 0
a^b^b => a
```

Prove it:

```
// x=A & y=B;  pre-cond.
x = x ^ y;
// x=A^B & y=B
y = x ^ y;
// x=A^B & y=A
x = x ^ y;
// x=B & y=A;  post-cond.
```

Compare the pre-condition and the post-condition, we may conclude that IT IS TRUE.

Solution 2: Use Bit-wise Exclusive OR (XOR) ^

Look at the Code:

```
int x, y;
…
x = x ^ y;
y = x ^ y;
x = x ^ y;
```

Are you sure?

on XOR:

```
1^b => ~b
0^b => b
b^b => 0
a^b^b => a
```

Prove it:

```
// x=A & y=B;  pre-cond.
x = x ^ y;
// x=A^B & y=B
y = x ^ y;
// x=A^B & y=A
x = x ^ y;
// x=B & y=A;  post-cond.
```

Compare the pre-condition and the post-condition, we may conclude that IT IS TRUE.

No Overflow Anymore!

Why is testing not sufficient?

True/False

1. We can use testing to show that there exists a bug in a program.
2. We can use testing to show that there does NOT exist a bug in a program.

(A) True and True
(B) True and False
(C) False and True
(D) False and False
(E) I don't know.

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?

- Formally specifying software 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 too large number
- Mars Climate Orbiter, 1999
  - Destroyed on entering atmosphere of Mars
  - Reason: mixture of pounds and kilograms
- Power black-out, 2003
  - 50 million people in Canada and US without power
  - Reason: programming error
- Royal Bank, 2004
  - Financial transactions disrupted for 5 days
  - Reason: programming error

Without formal verification, what could go wrong?

- UK Child Support Agency, 2004
  - Overpaid 1.9 million people, underpaid 700,000, cost to taxpayers over $ 1 billion
  - Reason: more than 500 bugs reported
- Science (a prestigious scientific journal), 2006
  - Retraction of research papers due to erroneous research results
  - Reason: program incorrectly flipped the sign (+ to -) 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 min
  - Reason: programming error

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.

Imperative programs

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

State at the “while” test:

1. $z = 0$, $y = 1$
2. $z = 1$, $y = 1$
3. $z = 2$, $y = 2$
4. $z = 3$, $y = 6$
5. $z = 4$, $y = 24$

Formal specification

Consider the following specification:

Given an integer x as input, the program will compute an integer y whose square is less than x.

Does this specification provide sufficient information for us to verify the correctness of the program?

Formal specification

Two important components of a specification:

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

Tony Hoare

- Sir Charles Antony Richard Hoare. British computer scientist.
- Won Turing award in 1980.
- Developed the QuickSort algorithm and the Hoare logic for verifying program correctness.

Hoare Triples

A Hoare Triple consists of

- $(\!| P |\!)$ — precondition
- C — code or program
- $(\!| Q |\!)$ — postcondition

The meaning of the Hoare triple $(\!| P |\!)\, C\, (\!| Q |\!)$:

If the state of program C before execution satisfies P,
then the ending state of C after execution will satisfy Q.

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 integer,
compute a number whose square is less than x

might be expressed as

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

Specification is NOT behaviour

Consider two programs C_1 and C_2.

Listing 1: C_1

```
y = 0;
```

Listing 2: C_2

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

Is the Hoare triple $(\!|\,(x > 0)\,|\!)\; C_1\; (\!|\,((y * y) < x)\,|\!)$ satisfied?

(A) Yes

(B) No

(C) Not enough information to tell

Specification is NOT behaviour

Consider two programs C_1 and C_2.

Listing 3: C_1

```
y = 0;
```

Listing 4: C_2

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

Is the Hoare triple $(\!|\,(x > 0)\,|\!)\; C_2\; (\!|\,((y * y) < x)\,|\!)$ satisfied?

(A) Yes

(B) No

(C) Not enough information to tell

Partial Correctness

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

if and only if

- for every state s_1 that satisfies condition P,
- if execution of C starting from state s_1 terminates in a state s_2,
- then state s_2 satisfies condition Q.

CQ Verifying Partial Correctness

Consider the Hoare triple $\llparenthesis\, (x > 0) \,\rrparenthesis\; C_1 \;\llparenthesis\, ((y * y) < x) \,\rrparenthesis$.

If we run C_1 starting with the state $x = 5, y = 5$,
C_1 terminates in the state $x = 5, y = 0$.

Is the Hoare triple satisfied under partial correctness?

(A) Yes

(B) No

(C) Not enough information to tell.

CQ Verifying Partial Correctness

Consider the Hoare triple $(\!|\,(x > 0)\,|\!)\ C_2\ (\!|\,((y * y) < x)\,|\!)$.

If we run C_2 starting with the state $x = 5, y = 5$,
C_2 terminates in the state $x = 5, y = 3$.

Is the Hoare triple satisfied under partial correctness?

(A) Yes

(B) No

(C) Not enough information to tell.

CQ Verifying Partial Correctness

Consider the Hoare triple $\llparenthesis\, (x > 0) \,\rrparenthesis\; C_3 \;\llparenthesis\, ((y * y) < x) \,\rrparenthesis$.

If we run C_3 starting with the state $x = -3, y = 5$,
C_3 terminates in the state $x = -3, y = 0$.

Is the Hoare triple satisfied under partial correctness?

(A) Yes

(B) No

(C) Not enough information to tell.

CQ Verifying Partial Correctness

Consider the Hoare triple $\llparenthesis\,(x > 0)\,\rrparenthesis\; C_4 \;\llparenthesis\,((y * y) < x)\,\rrparenthesis$.

If we run C_4 starting with the state $x = 2, y = 5$,
C_4 does not terminate.

Is the Hoare triple satisfied under partial correctness?

(A) Yes

(B) No

(C) Not enough information to tell.

Total Correctness

A triple $(\!| P |\!) \; C \; (\!| Q |\!)$ is satisfied under total correctness

if and only if

- for every state s_1 that satisfies condition P,
- execution of C starting from state s_1 terminates in a state s_2,
- and state s_2 satisfies condition Q.

Total Correctness = Partial Correctness + Termination

CQ Verifying Partial and Total Correctness

Is the following Hoare triple satisfied under partial and/or total correctness?

$(\!|\,(x = 1)\,|\!)$
`y = x;`
$(\!|\,(y = 1)\,|\!)$

(A) Neither satisfied.

(B) Only partial correctness satisfied.

(C) Total correctness satisfied.

CQ Verifying Partial and Total Correctness

Is the following Hoare triple satisfied under partial and/or total correctness?

$(\!|\, (x = 1) \,|\!)$
`y = x;`
$(\!|\, (y = 2) \,|\!)$

(A) Neither satisfied.

(B) Only partial correctness satisfied.

(C) Total correctness satisfied.

CQ Verifying Partial and Total Correctness

Is the following Hoare triple satisfied under partial and/or total correctness?

$(\!|\ (x = 1)\ |\!)$

```
while (1) {
  x = 0
};
```

$(\!|\ (x > 0)\ |\!)$

(A) Neither satisfied.

(B) Only partial correctness satisfied.

(C) Total correctness satisfied.

CQ Verifying Partial and Total Correctness

Is the following Hoare triple satisfied under partial and/or total correctness?

$(\!|\, (x \geq 0) \,|\!)$

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

$(\!|\, (y = x!) \,|\!)$

(A) Neither satisfied.

(B) Only partial correctness satisfied.

(C) Total correctness satisfied.

CQ Verifying Partial and Total Correctness

Is the following Hoare triple satisfied under partial and/or total correctness?

$(\!|\, true \,|\!)$

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

$(\!|\, (y = x!) \,|\!)$

(A) Neither satisfied.

(B) Only partial correctness satisfied.

(C) Total correctness satisfied.

CQ Difference between Partial and Total Correctness

For the following Hoare triple, what is the most important difference between partial and total correctness?

$$\llparenthesis\, P \,\rrparenthesis\; C \;\llparenthesis\, Q \,\rrparenthesis$$

(A) One requires the starting state to satisfy P and the other one doesn't.

(B) One requires the program C to terminate and the other one doesn't.

(C) One requires the terminating state to satisfy Q and the other one doesn't.

(D) There is no difference.

Summary

Key points learnt:

- Give reasons for performing formal verification rather than testing.
- Define a Hoare triple.
- Define partial correctness.
- Define total correctness.