Program Verification

CMPSC 461
Programming Language Concepts
Penn State University
Fall 2016

Correctness

A program implements the desired property for all possible inputs

E.g.,

Functional correctness: a program calculates n!

Type safety: no typing errors at run time

Memory safety: no buffer overflow in a program

Security: no information leakage

We will focus on **functional correctness** in this course

Functional Correctness

```
r:=0, i:=0;
while (i<n) {
  r := r+2;
  i ++;
}
```

This code ensures $r = 2 \times n$ when $n \geq 0$?

- ***Testing***: assert `r:= 2*n` and execute with different values of n (cannot cover all inputs in general)

- ***Verification***: prove $r = 2 \times n$ for any possible n

Program and Logics

Is a program correct (e.g., is the result n!)?

We need to formally specify

1) The desired property
2) The behavior of program

Logics as our specification language

Background: Propositional Logic

Proposition: statements that can either be true or false

E.g., $1 > 0$ (true), $0 \times 5 = 0$ (true), $5 - 1 = 5$ (false)

Composed propositions: with prop. p and q,

And: $p \wedge q$ true iff both p and q are true

Or: $p \vee q$ false iff both p and q are false

not: $\neg p$ true iff p is false

E.g., $1 > 0 \wedge 0 \times 5 = 0$ (true)

$0 \times 5 = 0 \vee 5 - 1 = 5$ (true)

Implication

A proposition p implies another proposition q (written as $p \Rightarrow q$) iff $(\neg p \vee q)$

(intuitively, q is true whenever p is true)

E.g., $7 > 5 \Rightarrow 5 > 3$ (true)

$1 > 2 \Rightarrow 2 > 3$ (true, due to the false condition)

$2 > 1 \Rightarrow 2 > 3)$ (false)

Derivation:

Modus Ponens: given $p \Rightarrow q$ and p, we have q

(First-Order) Predicate Logic

A formula can mention bounded variables

For all (Universally quantified): $\forall x.\, p$
E.g., $\forall x.\, x = 5$ (false)

Exists (Existentially quantified): $\exists x.\, p$
E.g., $\exists x.\, x = 5$ (true)

(We assume the domain of integers for simplicity)

(First-Order) Predicate Logic

E.g., $\forall x. (x > 5 \Rightarrow x > 3)$ (true)

$\forall x. (x > 1 \Rightarrow x > 3)$ (false, consider $x = 2$)

$\exists x. (x > 1 \Rightarrow x > 3)$ (true, consider $x = 4$)

Truth value of a formula can be *proved* based on derivation rules from predicate logic and number theory

We only use simple formulas in this course; in general, some tools (e.g., an SMT solver) can automatically tell the truth value of many formulas

Program and Logics

We need to formally specify

1) The desired property

```
int Max(int a, int b) {
  int m;
  if (a>b)  m:=a;
  else      m:=b;
  return m;
}
```

Assignment

Precondition (true)

function symbol in logics

Postcondition $(m = \max(a, b))$

Program and Logics

We need to formally specify

1) The desired property

```
int factorial(int n) {
  int r:=1, i=n;
  while (i>0) {
    r := r*i;
    i --;
  }
  return r;
}
```

Precondition $(\mathrm{n} \geq 0)$

Postcondition $(r = n!)$

Program and Logics

Is a program correct?

We need to formally specify

1) The desired property
2) **The behavior of program**

Informally …

```
x := 5;
y := 1;
```

After second assignment, we know $\{x = 5, y = 1\}$

Why?

1. Initially, we assume nothing
2. After the first assignment, we know $\{x = 5\}$
3. After the second assignment, we know $\{y = 1\}$ is true as well

Formalizing the Reasoning

```
x := 5;
y := 1;
```

1. Initially, we assume nothing
2. After the first assignment, we know $\{x = 5\}$
3. After the second assignment, we know $\{y = 1\}$ is true as well

The reasoning:

$$\{\text{true}\}\texttt{x:=5}\ \{x = 5\}\ \texttt{y:=5}\ \{x = 5 \land y = 1\}$$

Each predicate specifies the assertion that must be true before/after a statement

Hoare Triple

Assertion: a predicate that describes the ***state*** of a program at a point in its execution

Hoare Triple: $\{P\}s\{Q\}$

Precondition P: an assertion before execution

Postcondition Q: an assertion after execution

Program s: program being analyzed

A triple is ***valid*** If we start from a state satisfying P, and execute s, then final state must satisfy Q

Examples

$\{\text{true}\}$`x:=5`$\{x = 5\}$

$\{y = 6\}$`x:=5`$\{x = 5, y = 6\}$

$\{\text{true}\}$`x:=5`$\{x < 10\}$

$\{x = y\}$`x:=x+3`$\{x = y + 3\}$

$\{x = a\}$`if (x<0) then x:=-x` $\{x = |a|\}$

All of these triples are valid

Program Correctness

A program is correct (w.r.t. pre/postcondition) if the corresponding Hoare triple is valid

```
{true}
int m;
if (a>b)   m:=a;
else       m:=b;
{m=max(a,b)}
```

```
{n >= 0}
int r:=1, i:=n;
while (i>0) {
  r := r*i;
  i --;
}
{r = n!}
```

How can we tell the validity of a Hoare triple?

Goal: check if $\{P\}s\{Q\}$ is valid

$\{\text{true}\}\mathrm{x} := 5\{x = 5\}$

$\{\text{true}\}\mathrm{x} := 5\{x = 5 \lor x = 2\}$

$\{\text{true}\}\mathrm{x} := 5\{x > 0\}$

$\{\text{true}\}\mathrm{x} := 5\{x < 10\}$

Observation: some postconditions are more useful

$x = 5 \Rightarrow x = 5 \lor x = 2$

$x = 5 \Rightarrow x > 0$

$x = 5 \Rightarrow x < 10$

Need to compute the ***strongest*** postcondition

Goal: check if $\{P\}s\{Q\}$ is valid
Method 1: check $\text{sp}(s, P) \Rightarrow Q$

Strongest Postcondition

$\text{sp}(s, P)$ is the **strongest postcondition** of s, w.r.t. P
Property: $\{P\}s\{Q\}$ is valid iff $\text{sp}(s, P) \Rightarrow Q$

Hence, validity of a triple $\{P\}s\{Q\}$ is equivalent to the truth value of proposition $\text{sp}(s, P) \Rightarrow Q$

Assignment Rule (Floyd's Axiom)

$$\mathrm{sp}(\mathtt{x:=e}, P) = (\exists v. \left(x = (e[x \leftarrow v])\right) \wedge (P[x \leftarrow v]))$$

Substitute x with v in e

Examples:

$\mathrm{sp}(\mathtt{x:=5}, \mathrm{true}) = (\exists v. (x = 5) \wedge \mathrm{true}) = (x = 5)$

$\mathrm{sp}(\mathtt{x:=x+3}, x = y) = \left(\exists v. (x = v + 3) \wedge (v = y)\right)$

$= (x = y + 3)$

Composition Rule

$$\boxed{\text{sp}(\texttt{s1;s2}, P) = \text{sp}(\texttt{s2}, \text{sp}(\texttt{s1}, P))}$$

```
x := 5;
y := 1;
```

$$\begin{aligned}\text{sp}(\texttt{x:=5;y:=1}, \text{true}) &= \text{sp}(\texttt{y:=1}, \text{sp}(\texttt{x:=5}, \text{true}))\\ &= \text{sp}(\texttt{y:=1}, x = 5) \text{ (previous slide)}\\ &= \left(\exists v. (y = 1) \land (x = 5)\right)\\ &= (y = 1) \land (x = 5)\end{aligned}$$

Composition Rule

$$\text{sp}(\texttt{s1;s2}, P) = \text{sp}(\texttt{s2}, \text{sp}(\texttt{s1}, P))$$

```
x := 5;
x := 2;
```

$$\begin{aligned}
\text{sp}(\texttt{x:=5;x:=2}, \text{true}) &= \text{sp}(\texttt{x:=2}, \text{sp}(\texttt{x:=5}, \text{true})) \\
&= \text{sp}(\texttt{x:=2}, x = 5) \\
&= \big(\exists v. \big(x = (2[x \leftarrow v])\big) \wedge (x = 5[x \leftarrow v])\big) \\
&= \big(\exists v. (x = 2) \wedge (v = 5)\big) \\
&= (x = 2)
\end{aligned}$$

The existential quantifier complicates the formula …