Program Verification (Rosen, Sections 5.5)

TOPICS

- Propositional Logic
- Logical Operations
- Equivalences
- Predicate Logic

Proofs about Programs

- Why make you study logic?
- Why make you do proofs?
- Because we want to prove properties of *programs*
  - In particular, we want to prove properties of variables at specific points in a program

Isn't testing enough?

- Assuming the program compiles, we perform some amount of testing.
- Testing shows that for specific examples the program seems to be running as intended.
- Testing can only show existence of some bugs but cannot exhaustively identify all of them.
- Verification can be used to prove the correctness of the program with any input.

Program Verification

- We consider a program to be *correct* if it produces the expected output **for all possible inputs**.
- Domain of input values can be very large, how many possible values of an integer?
- Instead we can formally specify program behavior, then use techniques for inferring correctness.
- For example, we can use logic techniques.

Program Correctness Proofs

- Two parts:
  - Correct answer when the program terminates (called *partial correctness)*
  - The program does terminate
- We will only do part 1
  - Prove that a method is correct if it terminates
- Part 2 has been shown to be impossible!

Predicate Logic and Programs

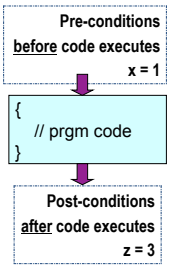
- Variables in programs are like variables in predicate logic:
  - They have a domain of discourse (data type)
  - They have values (drawn from the data type)
- Variables in programs are different from variables in predicate logic:
  - Their values change over time

Assertions

- Two parts:
  - ***Initial Assertion:*** a statement of what must be true about the input values or values of variables at the beginning of the program segment
    - E.g Method that determines the sqrt of a number, requires the input (parameters) to be >= 0
  - ***Final Assertion***: a statement of what must be true about the output values or values of variables at the end of the program segment
    - E.g. What is the output/final result after a call to the method?

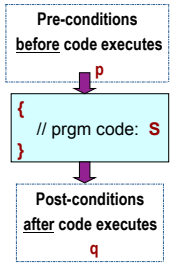
Preconditions and PostConditions

- ***Initial Assertion:*** sometimes called the **precondition**
- ***Final Assertion***: sometimes called the **postcondition**
- **Note**: these assertions can be represented as propositions or predicates. For simplicity, we will write them generally as propositions.



Hoare Triple

- "A program, or program segment, **S**, is said to be **partially correct** with respect to the initial assertion **p** and the final assertion **q** if, whenever **p** is true for the input values of **S** and **S** terminates, then **q** is true for the output values of **S**." [Rosen 7th edition, p. 372]
- Notation: **p{S}q**

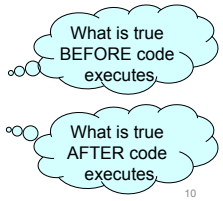


Simple Example

- Assume that our proof system already includes rules of arithmetic…
- Consider the following code:

```
y = 2;
z = x + y;
```

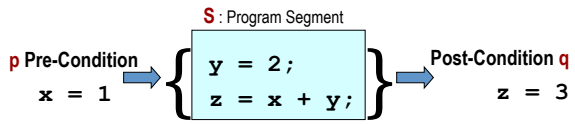
- Initial Assertion: *p(x), x =1*
- Final assertion: *q(z), z =3*



Simple Example, continued

- Here {S} contains two code statements
- So, **p {S} q** for this example is

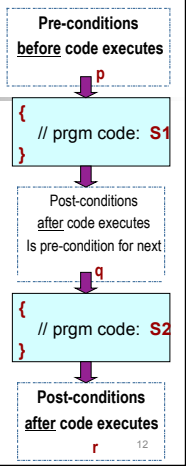
(p(x), x =1) { `y = 2;` `z = x + y;` } *(q(z), z =3)*



Rule 1: Composition Rule

- Once we prove correctness of program segments, we can combine the proofs together to prove correctness of an entire program.

$$\frac{p\{S_1\}q \quad q\{S_2\}r}{\therefore p\{S_1;S_2\}r}$$



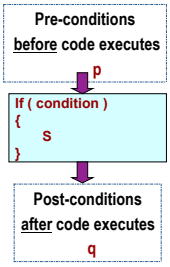
Rule 2: Conditional Statements

- Given

```
if (condition)
    statement;
```

- and
  - Initial assertion: p
  - Final assertion: q

What does this mean?
What must we prove?



Rule 2: Conditional Statements

- Given

```
if (condition)
    statement;
```

- with Initial assertion: p and Final assertion: q
- Must show that
  - when p is true and *condition* is true then q is true when S terminates: $(p \wedge condition)\{S\}q$
  - when p is true and *condition* is false, then q is true (S does not execute) $(p \wedge \neg condition) \rightarrow q$

Conditional Rule

$$\frac{(p \wedge condition)\{S\}q \qquad (p \wedge \neg condition) \rightarrow q}{\therefore p\{\text{if } condition\ S\}q}$$

Conditional Rule: Example

```
if (x > y)
   y = x;
```

- Initial assertion: **T** *(true)*
- Final assertion: **q(y,x)** means **y≥x**
- Consider the two cases…

Conditional Rule Example

```
if (x > y)
  y = x;
```

- Initial assertion: **T** *(true)*
- Final assertion: **q(y,x)** means **y≥x**

(p(x), true) { `if (x > y) y = x;` } *(q(y,x), y≥x)*

Conditional Rule: Code

```
input x, y;
  . . .
       // Initial: true
 if (x > y)
   y = x;
       // Final:  y>=x
  . . . .
```

- Initial assertion: **T** *(true)*
- Final assertion: **q(y,x)** means **y≥x**

Conditional Rule: Example

- Verify the the program segment

```
if (x % 2 == 1)
      x = x + 1
```

- Is correct with respect to the initial assertion T and the final assertion x is even

Rule 2a: Conditional with Else

```
if (condition)
  S1;
else
  S2;
```

- Rule is:

$$\frac{(p \wedge condition)\{S_1\}q \quad (p \wedge \neg condition)\{S_2\}q}{\therefore p\{\text{if } condition\ S_1 \text{ else } S_2\}q}$$

Conditional Rule 2a: Example

```
if (x < 0)
   abs = -x;
else
   abs = x;
```

- Initial assertion: **T** ***(true)***
- Final assertion: **q(abs), abs=|x|**

(p(x), true) { `if (x < 0) abs = -x; else abs = x;` } ***(q(abs), abs=|x|)***

Conditional Rule 2a: Example

```
if (x < 0)
   abs = -x;
else
   abs = x;
```

- Initial assertion: **T** ***(true)***
- Final assertion: **q(abs), abs=|x|**
- Consider the two cases…

Conditional Rule 2a: Code

```
   //  true
if (x < 0)
   abs = -x;   // x < 0 -> abs = |x|
else
   abs = x;    // x >= 0 -> abs = x
    //  abs = |x|
```

- Initial assertion: **T**
- Final assertion: **q(abs), abs=|x|**

Conditional Rule 2a: Example

Verify the the program segment

```
if (balance > 100)
      newBalance = balance * 1.02
else
      newBalance = balance * 1.005
```

Is correct with respect to the initial assertion balance > 0 and the final assertion (balance > 100 ^ newBalance = balance * 1.02) v (balance <= 100 ^ newBalance = balance * 1.005)

How to we prove loops correct?

- General idea: *loop invariant*
- Find a property that is true before the loop
- Show that it must still be true after every iteration
- Therefore it is true after the loop

Rule 3: Loop Invariant

```
while (condition)
  S;
```

- Rule:

Note these are both p!

$$\frac{(p \wedge condition)\{S\}p}{\therefore p\{\text{while } condition\ \ S\}(\neg condition \wedge p)}$$

Note both conclusions

Loop Invariant: Example

- Initial assertion: n >= 1

```
i = 1;
factorial = 1;
// i <= n and factorial = i!
while (i < n) {
        i++;
        factorial *= i;
}
// i <= n and factorial = i!
```

- Final assertion q(factorial, n) is factorial = n!

Loop Invariant: Example

- What is the loop invariant?
  - factorial = i!
  - i <= n
- Evaluate program segment
  - Before entering loop
  - If loop terminates, p true and i < n false
  - After loop, factorial = i! and i <= n is true, but i < n false, so i >= n, so
    - i = n and factorial = i! = n!