Proofs of Program Correctness
(not in textbook)

TOPICS

• Program Verification
• PreConditions and PostConditions
• Composition Rule
• Conditional Rule
• Loop Invariants

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 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 Proof

• 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.
Simple Example

• Assume that our proof system already includes rules of arithmetic.
• Consider the following code:

\[
\begin{align*}
y &= 2; \\
z &= x + y;
\end{align*}
\]

• 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:

\[
\begin{align*}
p \{ S_1 \} q \\
q \{ S_2 \} r \\
\therefore p \{ S_1; S_2 \} r
\end{align*}
\]

Rule 2: Conditional Statements

• Given
  \[
  \text{if (condition)} \\
  \text{statement;}
  \]
• and
  – Initial assertion: \( p \)
  – Final assertion: \( q \)

What does this mean?
What must we prove?
Rule 2: Conditional Statements, ...

- Given
  
  \[
  \text{if (condition)}
  \text{statement;}
  \]

- with Initial assertion: \( p \) and Final assertion: \( q \)

- Must show that
  
  - when \( p \) is true and \( \text{condition} \) is true then \( q \) is true
    when \( S \) terminates: \( (p \land \text{condition})[S]q \)
  
  - when \( p \) is true and \( \text{condition} \) is false, then \( q \) is true
    \( (S \) does not execute) \( (p \land \neg \text{condition}) \rightarrow q \)

Conditional Rule

\[
(p \land \text{condition})[S]q \\
(p \land \neg \text{condition}) \rightarrow q \\
\therefore p \{\text{if condition} S\} q
\]

Conditional Rule Example

\[
\text{if (x > y)} \\
y = x;
\]

- Initial assertion: \( T \) (true)
- Final assertion: \( q(y, x) \) means \( y \geq x \)

\[
(p(x), \text{true}) \begin{cases} 
\text{if (x > y)} \\
y = x;
\end{cases} \\
(q(y,x), y \geq x)
\]

Conditional Rule Example (in code)

\[
\text{input x, y;}
\]

- // Initial: true
  
  \[
  \text{if (x > y)} \\
y = x;
  \]

  // Final: \( y \geq x \)

- Initial assertion: \( T \) (true)
- Final assertion: \( q(y, x) \) means \( y \geq x \)
Rule 2a: Conditional with Else

if (condition)
  S1;
else
  S2;

• Rule is:
  \[
  \frac{(p \land \text{condition})(S_1)q}{(p \land \neg \text{condition})(S_2)q} \implies p\{\text{if condition } S, \text{ else } S_2\}q
  \]

Conditional Rule 2a Example

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

• Initial assertion: \(T\) \text{ (true)}
• Final assertion: \(q(abs), abs=|x|\)

// 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

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

• Initial assertion: \(T\) \text{ (true)}
• Final assertion: \(q(abs), abs=|x|\)

Consider the two cases...
How to we prove loops correct?

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

Rule 3: Loop Invariant

while (condition)
S;

- Rule:

\[
\begin{align*}
(p \land \text{condition}) \{S\} p \\
\therefore p\{\text{while condition } S\}{\neg\text{condition } \land p}
\end{align*}
\]

Note these are both p! Note both conclusions

Loop Invariant Example

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

What to take away...

- Correctness proofs
  - Yes, they are long
  - But doable if you know logic and arithmetic
  - Necessary for safety-critical systems
  - Termination is still unproved
- Loop Invariants
  - Common program analysis technique
  - Used for documentation, debugging