Proofs of Program Correctness

CS160
Spring 2012

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...
• Testing demonstrates for specific examples that the program seems to be running as intended.
• Testing can only show existence of some bugs rather than exhaustively identifying all of them.
• Verification however...

Program Verification

• We consider a program to be correct if it produces the expected output for every possible input.
• This requires a formal specification of program behavior and techniques for inferring correctness.
• Fortunately, we know about logic.
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

Predicate Logic & 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
Specification of Program Segments

• 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 \( \geq 0 \)
  – **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?

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

\[
(p(x), x = 1) \begin{cases} 
  y = 2; \\
  z = x + y;
\end{cases} (q(z), z = 3)
\]

\( S \): Program Segment

Rule 1: Composition Rule

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

\[
\frac{p\{S_1\}q}{q\{S_2\}r} \quad \therefore p\{S_1; S_2\}r
\]
Rule 2: Conditional Statements

• Given
  
  ```java
  if (condition)
      statement;
  ```

• and
  
  – Initial assertion: \( p \)
  – Final assertion: \( q \)

What does this mean? What must we prove?

Rule 2: Conditional Statements, ...

• Given
  
  ```java
  if (condition)
      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 Example

\textbf{if} (x > y)
\begin{align*}
& y = x;
\end{align*}

- Initial assertion: \textbf{T} (true)
- Final assertion: \texttt{q(y,x)} means $y \geq x$
- Consider the two cases...

\begin{align*}
\text{Conditional Rule} \\
(p \land \text{condition}) \{S\} q \\
(p \land \neg \text{condition}) \rightarrow q \\
\therefore p\{\text{if condition}\} q
\end{align*}
Conditional Rule Example

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

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

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

Conditional Rule Example \((in\ code)\)

```c
input x, y;

. . .

   // Initial: true
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:

\[
\begin{align*}
(p \land \text{condition}) & \{S_1\} q \\
(p \land \neg \text{condition}) & \{S_2\} q \\
\therefore \ & p\{\text{if condition } S_1 \ \text{else } S_2\} q
\end{align*}
\]

Conditional Rule 2a Example

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

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

\[
(p(x), \text{true}) \left\{ \begin{array}{l}
\quad \text{if } (x < 0) \\
\quad \quad \text{abs} = -x; \\
\quad \text{else} \\
\quad \quad \text{abs} = x;
\end{array} \right\} \ (q(abs), \ absto=x)\]
Conditional Rule 2a Example

```c
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

```c
// 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|
Example Method (factorial)

```java
public int factorial(int n) {
    int target;
    if (n < 1) target = 1;
    else target = n;
    int ctr = 1;
    int result = 1;
    while (ctr < target) {
        ctr++;
        result *= ctr;
    }
    return result;
}
```

Task: prove factorial correct

- Prove that the value returned by factorial(n) is n!
- Using:
  - Logic
    - predicate or propositional
  - Arithmetic
    - which we assume to be part of logic
  - Hoare triples
    - Which we briefly introduced earlier
    - But will now expand to cover loops
How to Start

• The Key to Proofs: Decomposition
  – Just like programming!
• Prove properties of pieces of code
  – Straight-line segments of code
  – Control blocks: If statement, loops, etc.
• Put them back together

Rule 1: Composition Rule

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

\[
p\{S_1\}q \\
q\{S_2\}r \\
\therefore p\{S_1;S_2\}r
\]
Back to factorial...

```java
public int factorial(int n) {
    int target;  
    if (n < 1) target = 1;
    else target = n;
    int ctr = 1;
    int result = 1;
    while (ctr < target) {
        ctr++;
        result *= ctr;
    }
    return result;
}
```

Rule 2: Conditional Statements

- **Given**
  
  \[ (p \land \text{condition}) \{S\} q \]

- **and**
  
  \[ (p \land \lnot \text{condition}) \rightarrow q \]

- **Must show that**
  
  \[ \therefore p \{ \text{if condition } S \} q \]

  - when \( p \) is true and \( \text{condition} \) is true then \( q \) is true when \( S \) terminates (case 1)
  - when \( p \) is true and \( \text{condition} \) is false, then \( q \) is true (\( S \) does not execute) (case 2)
Rule 2a: Conditional with Else

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

- Rule is:

\[
\begin{align*}
(p \land \text{condition}) & \{S_1\} q \\
(p \land \neg \text{condition}) & \{S_2\} q \\
\therefore \ p\{\text{if condition } S_1 \text{ else } S_2\} q
\end{align*}
\]

Conditional Rule Example

```c
int target;
if (n < 1) target = 1;
else target = n;
```

- Initial assertion: T (i.e. nothing)
- Condition: less(n, 1)
- Final assertions:
  - geq(target, 1)
  - Eq(n, target) ∨ (less(n,1)∨eq(target, 1))
- Consider the two cases...
• Case 1: condition is true
  – N < 1 (because the condition is true)
  – Target = 1 (because of S1)
  – Postcondition

\[
\textbf{Case 1:} \quad (p \land \text{condition}) \{S_1\}q
\]

\[
(T \land \text{less}(n,1))\{\text{target} = 1\}(\text{eq}(\text{target}, n) \lor (\text{less}(n,1) \land \text{eq}(\text{target}, 1)))
\]

Proof

- eq(target, 1) Assignment
- geq(target, 1) Arithmetic (#1)
- less(n,1) Condition (assumed true)
- less(n,1) \land eq(1, target) Addition (#1, #3)
- eq(n, target) \lor (less(n,1) \land eq(target, 1)) Implication (#4)

Proposition Key

- \text{geq}(x,y) = x \geq y
- \text{less}(x,y) = x < y
- \text{eq}(x,y) = x = y
Case 2: 
\((p \land \neg \text{condition})\{S_2\}q\)

\((T \land geq(n,1))\{\text{target} = n\}(geq(\text{target},1) \lor (\text{less}(n,1) \land eq(\text{target},1)))\)

Proof
\begin{itemize}
\item geq(n, 1) \hspace{3cm} \text{Condition Negation}
\item eq(n, target) \hspace{3cm} \text{Assignment}
\item geq(target, 1) \hspace{3cm} \text{Transitivity (#2, #3)}
\item eq(n, target) \lor (\text{less}(n,1) \land eq(\text{target}, 1)) \hspace{3cm} \text{Implication (#6)}
\end{itemize}

Now back to Rule 2 (if/else)

\[
\begin{align*}
(p \land \text{condition})\{S_1\}q \\
(p \land \neg \text{condition})\{S_2\}q \\
\therefore p\{\text{if condition \ }S_1 \text{ \ else \ } S_2\}q
\end{align*}
\]

- P: T
- Condition: geq(n,1)
- S1: target = 1;
- S2: target = n;
- Q: geq(target, 1),
  eq(n, target) \lor (\text{less}(n,1) \land eq(\text{target}, 1))
Back to factorial, part 2...

```c
struct {
    int ctr = 1;
    int result = 1;
} Code Fragment
```

- How do we pick the preconditions?
  - We don’t need any to prove post-conditions
  - But we will need some for the chain rule later
  - Adopt the post-conditions from part 1 as preconditions

- How do we pick the postconditions?
  - Inspiration (more later...)

Part 2 Formalized

- Preconditions:
  - geq(target, 1)
  - eq(n, target) ∨ (less(n,1)∧eq(target, 1))

- Postconditions:
  - geq(target, 1)
  - eq(n, target) ∨ (less(n,1)∧eq(target, 1))
  - eq(result, ctr!)
Definition of factorial

- To prove this last postcondition formally, we need to define factorial:

\[ n! \equiv \begin{cases} 1, \forall n \leq 1 \\ n \cdot \text{fac}(n-1), \forall n > 1 \end{cases} \]

Part 2 Proof

- \text{geq}(\text{target}, 1) \quad \text{Precondition}
- \text{eq}(\text{n, target}) \lor (\text{less}(\text{n,1}) \land \text{eq}(\text{target, 1})) \quad \text{Precondition}
- \text{eq}(\text{ctr, 1}) \quad \text{Assignment}
- \text{eq}(\text{result, 1}) \quad \text{Assignment}
- \text{eq}(\text{ctr, result!}) \quad \text{Def. of factorial}
Why do I need the extra preconditions?

- Remember the composition rule: $p\{S_1\}q$
  
  $q\{S_2\}r$
  
  $\therefore p\{S_1; S_2\}r$

- In traditional logic, once a statement is true it is always true.
- In code proofs, it could change (e.g. by assignment)
  - You can only assert preconditions as true if not contradicted by the code fragment
  - Therefore you must show that the preconditions are still true after the code...

---

How do 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:

\[
(p \land \text{condition}) \{S\} p \\
\therefore p \{\text{while condition } S\} (\neg \text{condition} \land p)
\]

Note these are both p!

Note both conclusions

Back to factorial, part 2...

```c
int ctr = 1;
int result = 1;
while (ctr < target) {
    ctr++;
    result *= ctr;
}
return result;
```

What is true (a) before the loop and (b) after each iteration of the loop?

Hint: this function is computing factorial...
Loop Invariants

• Initial assertions P:
  – eq(ctr!, result)
    • Before the start of the loop, result = ctr!
    • Because 1 = 1!
  – geq(target, 1)
    • Before the start of the loop, target >= 1
  – eq(n, target) ∨ (less(n,1)∧eq(target, 1))
    • Target is either n or 1 (if n < 1)

• Condition:
  – less(ctr, target)

• Final assertions:
  – eq(ctr, target)
  – eq(ctr!, result)

Part 3 Proof

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

less(ctr, target)                     Condition (assume true)
eq(ctr_1!, result_1)                Precondition
eq(ctr_2, ctr_1+1)                  Assignment
eq(result_2, ctr_2*result_1!)       Assignment
eq(result_2, (ctr_1+1)*result_1!)   Arithmetic
eq(result_2, ctr_2!)                Def. of factorial

Note #1: I use ctr_1 to denote the initial value of ctr, and ctr_2 to denote the value of ctr at end of loop
Now put it all together...

- The preconditions of this proof match the post-conditions of the previous step, so....
- We can assert P:
  - eq(result, ctrl!)
  - eq(n, target) \lor (less(n,1) \land eq(target, 1))
- We can assert the negation of the condition
  - eq(ctrl, target)
    - Slight fudge here...
- So by the associative laws
  - eq(result, ctrl!) \land eq(ctrl, target) \land eq(n, target) or
  - eq(result, ctrl!) \land eq(ctrl, target) \land eq(target, 1) \land less(n,1)
- By our definition of factorial, both cases imply eq(result, n!)
  - QED

What to take away...

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