Proof Rules and Proofs for Correctness Triples

Part 2: Conditional and Iterative Statements

CS 536: Science of Programming, Fall 2022

A. Why?

- Proof rules give us a way to establish truth with textually precise manipulations
- We need inference rules for compound statements such as conditional and iterative.

B. Outcomes

At the end of this topic you should know

- The rules of inference for ***if-else*** statements.
- The rule of inference for ***while*** statements.
- The impracticality of the *wp* and *sp* for loops; the definition and use of loop invariants.

C. Rules for Conditionals

- There are two popular ways to characterize correctness for ***if-else*** statements

If-Else Conditional Rule 1

- The *sp*-oriented basic rule is

1. $\{p \wedge B\}\ S_1\ \{q_1\}$
2. $\{p \wedge \neg B\}\ S_2\ \{q_2\}$
3. $\{p\}$ ***if*** B ***then*** S_1 ***else*** S_2 ***fi*** $\{q_1 \vee q_2\}$ ***if-else*** *1, 2*[1]

- In proof tree form:

$$\frac{\{p \wedge B\}\ S_1\ \{q_1\} \qquad \{p \wedge \neg B\}\ S_2\ \{q_2\}}{\{p\}\ \textbf{\textit{if}}\ B\ \textbf{\textit{then}}\ S_1\ \textbf{\textit{else}}\ S_2\ \textbf{\textit{fi}}\ \{q_1 \vee q_2\}}\ \textbf{\textit{if-else}}$$

- The rule says that
  - If running the true branch S_1 in a state satisfying p and B establishes q_1,
  - And running the false branch S_2 in a state satisfying p and $\neg B$ establishes q_2,
  - Then you know that running the ***if-else*** in a state satisfying p establishes $q_1 \vee q_2$.

[1] The rule name can be *conditional* or *if-else;* your choice. A postcondition $q_1 \vee q_1$ can be abbreviated to q_1.

- ***Example 1***: Here's a proof of $\{T\}$ ***if*** $x \geq 0$ ***then*** $y := x$ ***else*** $y := -x$ ***fi*** $\{y \geq 0\}$. We need
  - $\{x \geq 0\}\ y := x\ \{y \geq 0\}$ for the true branch (line 1 below).
  - $\{x < 0\}\ y := -x\ \{y \geq 0\}$ for the false branch (lines 2 – 4 below).

1.	$\{x \geq 0\}\ y := x\ \{y \geq 0\}$	*assignment (backward)*
2.	$\{x < 0\}\ y := -x\ \{x < 0 \wedge y = -x\}$	*assignment (forward)*
3.	$x < 0 \wedge y = -x \rightarrow y \geq 0$	*predicate logic*
4.	$\{x < 0\}\ y := -x\ \{y \geq 0\}$	*postcondition weakening, 2, 3*
5.	$\{T\}$ ***if*** $x \geq 0$ ***then*** $y := x$ ***else*** $y := -x$ ***fi*** $\{y \geq 0\}$	***if-else*** *1, 4*

- The proof above used forward assignment; backward assignment works also: Lines 2 – 4 become

2.	$\{-x \geq 0\}\ y := -x\ \{y \geq 0\}$	*assignment (forward)*
3.	$x < 0 \rightarrow -x \geq 0$	*predicate logic*
4.	$\{x < 0\}\ y := -x\ \{y \geq 0\}$	*precondition strengthening 3, 2*

If-Else Conditional Rule 2

- ***Conditional rule 2:*** An equivalent, more goal-oriented / *wp*-oriented conditional rule is:

1.	$\{p_1\}\ S_1\ \{q_1\}$	
2.	$\{p_2\}\ S_2\ \{q_2\}$	
3.	$\{p_0\}$ ***if*** B ***then*** S_1 ***else*** S_2 ***fi*** $\{q_1 \vee q_2\}$ *where* $p_0 \equiv (B \rightarrow p_1) \wedge (\neg B \rightarrow p_2)$	***if-else***

- If we add a preconditioning strengthening step of $p \rightarrow (B \rightarrow p_1) \wedge (\neg B \rightarrow p_2)$ to the rule above, we get the same effect as the old precondition $(p \wedge B \rightarrow p_1) \wedge (p \wedge \neg B \rightarrow p_2)$.
- We can derive this second version of the conditional rule using the first version. The assumptions below become the antecedents of the derived rule above; the conclusion below becomes the consequent of the derived rule above.

1.	$\{p_1\}\ S_1\ \{q_1\}$	*assumption 1*
2.	$p_0 \wedge B \rightarrow p_1$ where $p_0 \equiv (p \wedge B \rightarrow p_1) \wedge (p \wedge \neg B \rightarrow p_2)$	*predicate logic*
3.	$\{p_0 \wedge B\}\ S_1\ \{q_1\}$	*precondition strengthening 2, 1*
4.	$\{p_2\}\ S_2\ \{q_2\}$	*assumption 2*
5.	$p_0 \wedge \neg B \rightarrow p_2$	*predicate logic*
6.	$\{p_0 \wedge \neg B\}\ S_2\ \{q_2\}$	*precondition strengthening 5, 4*
7.	$\{p_0\}$ ***if*** B ***then*** S_1 ***else*** S_2 ***fi*** $\{q_1 \vee q_2\}$	***if-else*** *3, 6*

If-Then Statement Rule

- An ***if-then*** statement is an ***if-else*** with $\{p \wedge \neg B\}$ ***skip*** $\{p \wedge \neg B\}$ as the false branch.

1.	$\{p \wedge B\}\ S_1\ \{q_1\}$	
2.	$\{p \wedge \neg B\}$ ***skip*** $\{p \wedge \neg B\}$	*skip*
3.	$\{p\}$ ***if*** B ***then*** S_1 ***fi*** $\{q_1 \vee (p \wedge \neg B)\}$	***if-else*** *1, 2*

Nondeterministic Conditionals

- Perhaps surprisingly, the proof rules for nondeterministic conditionals are almost exactly the same as for deterministic conditionals.

Nondeterministic if-fi rule 1: (sp-like)

1. $\{p \wedge B_1\}\ S_1\ \{q_1\}$
2. $\{p \wedge B_2\}\ S_2\ \{q_2\}$
3. $\{p\}$ ***if*** $B_1 \rightarrow S_1 \ \square\ B_2 \rightarrow S_2$ ***fi*** $\{q_1 \vee q_2\}$ ***if-fi*** *1, 2*

Nondeterministic if-fi rule 1: (wp-like)

1. $\{p_1\}\ S_1\ \{q_1\}$
2. $\{p_2\}\ S_2\ \{q_2\}$
3. $\{p_0\}$ ***if*** $B_1 \rightarrow S_1 \ \square\ B_2 \rightarrow S_2$ ***fi*** $\{q_1 \vee q_2\}$ ***if-fi*** *1, 2*
   where $p_0 \equiv (p \wedge B_1 \rightarrow p_1) \wedge (p \wedge B_2 \rightarrow p_2)$

D. Problems With Calculating the wp or sp of a Loop

- What is $wp(W, q)$ for a typical loop $W \equiv$ ***while*** B ***do*** S ***od***? It turns out that some $wp(W, q)$ have no finite representation. ($sp(W, p)$ has the same problem.)
  - Let's look at the general problem of $wp(W, q)$.
  - First, define w_k to be the weakest precondition of ***W*** and ***q*** that requires exactly ***k*** iterations.
    - Let $w_0 \equiv \neg B \wedge q$ and for all $k \geq 0$, define $w_{k+1} \equiv B \wedge wp(S, w_k)$.
  - If we know that W will run for, say, ≤ 3 iterations, then $wp(W, q) \Leftrightarrow w_0 \vee w_1 \vee w_2 \vee w_3$.
  - But in general, W might run for any number of iterations, so $wp(W, q) \Leftrightarrow w_0 \vee w_1 \vee w_2 \vee \ldots$.
  - If this infinitely-long disjunction collapses somehow, then we can write $wp(W, q)$ finitely.
  - E.g., if $w_{k+1} \rightarrow w_k$ when $k \geq 5$, then $wp(W, q) \Leftrightarrow w_0 \vee w_1 \vee w_2 \vee w_3 \vee w_4 \vee w_5$.
  - Or, if there's a predicate function $P(k) \Leftrightarrow w_k$ (i.e., if the w_k are parameterized by k), then $wp(W, q) \Leftrightarrow \exists n . P(n)$.

E. Using Invariants to Approximate the wp and sp With Loops

Basic notions

- If we can't calculate $wp(S, q)$ or $sp(p, W)$ exactly, the best we can do is to approximate it.
- The simplest approximation is a predicate p that implies all the w_k.
  - If $p \Rightarrow w_k$ for all k, then $p \Rightarrow w_0 \vee w_1 \vee w_2 \vee \ldots$, so $p \Rightarrow wp(S, q)$.

- ***Definition:*** A ***loop invariant for*** $W \equiv$ ***while*** B do S ***od*** is a predicate p such that $\vDash \{p \wedge B\}\ S\ \{p\}$. It follows that $\vDash \{p\}\ W\ \{p \wedge \neg B\}$.[2]
  - Under partial correctness, if W terminates, it must terminate satisfying $p \wedge \neg B$.
  - Note this is for partial correctness only: To get total correctness, we'll need to prove that the loop terminates, and we'll address that problem later.
- ***Notation***: To indicate a loop's invariant, we'll add it as an extra clause: {***inv*** p} ***while*** B ***do*** S ***od***. This declares that p is not only a precondition of the loop, it's an invariant.

Need Useful Invariants

- Not all invariants are useful. E.g., any tautology is an invariant: $\{T \wedge B\}\ S\ \{T\}$, so $\{T\}\ W\ \{T \wedge \neg B\}$. (For that matter, contradictions are invariants too, but they're even less useful.)
- The key is to find an invariant that:
  1. Can be established using simple loop initialization code: $\{p_0\}$ *initialization code* $\{p\}$.
  2. Can serve as a precondition and postcondition of a loop iteration: $\{p \wedge B\}$ *loop body* $\{p\}$.
  3. When combined with $\neg B$ and loop termination code, implies the postcondition we want: $\{p \wedge \neg B\}$ *termination code* $\{q\}$. If $p \wedge \neg B \rightarrow q$, then we don't need any termination code.
- There's no general algorithm for generating useful invariants. In a future class, we'll look at some heuristics for trying to find them.

Semantics of Invariants

- How do invariants fit in with the semantics of loops?
- Recall if we take the loop $W \equiv \{$***inv*** $p\}$ ***while*** B ***do*** S ***od*** and run it in state σ_0, then one iteration takes us to state σ_1, the next to σ_2, and so on: $\sigma_{k+1} = M(S, \sigma_k)$ for all k, and $M(W, \sigma_0)$ is the first σ_k that satisfies $\neg B$; if there is no such state, then we write $\bot_d \in M(W, \sigma_0)$[3]
- The invariant p must be satisfied by every possible $\tau_0, \tau_1, \ldots$, which implies that it's an approximation to various *wp* and *sp* for the loop and loop body:

Predicate	***Approximates***	***Because***
p	the *wp* of the loop	$p \rightarrow wp(W, p \wedge \neg B)$
$p \wedge B$	the *wp* of the loop body	$p \wedge B \rightarrow wp(S, p)$
$p \wedge \neg B$	the *sp* of the loop	$sp(p, W) \rightarrow p \wedge \neg B$
p	the *sp* of the loop body	$sp(S, p \wedge B) \rightarrow p$

---

[2] We've been using *"p"* as a generic name for a predicate. From now on, it may or may not stand for a loop invariant, depending on the context.

[3] If W is nondeterministic, it's a bit more complicated: For each possible sequence of τ_k, $M(W, \tau_0)$ either contains the first τ_k that satisfies $\neg B$ or $\bot_d$ if that sequence can be continued infinitely.

Loop Initialization and Cleanup

- The purpose of loop initialization code is to establish the loop invariant: $\{p_0\}$ *initialization code* $\{p\}$. Typically, we initialize any variables that appear fresh in the invariant; e.g., $\{n \geq 0\}\ k := 0\ \{0 \leq k < n\}$.
- If $p \wedge \neg B \rightarrow q$, the desired postcondition for the loop, then no cleanup is necessary, otherwise we need loop termination code: $\{p \wedge \neg B\}$ *termination code* $\{q\}$.

F. While Loop Rule; Loop Invariant Example

- The proof rule for a loop only has one antecedent, which requires us to have a loop invariant.

1.	$\{\ p \wedge B\ \}\ S\ \{\ p\ \}$	
2.	$\{$***inv*** $p\ \}$ ***while*** B ***do*** S ***od*** $\{\ p \wedge \neg B\ \}$	loop (or ***while***), 1

- As a triple, the loop behaves like $\{p\}$ ***while*** B ***do*** S ***od*** $\{p \wedge \neg B\}$, so any precondition strengthening is relative to p, and any postcondition weakening is relative to $p \wedge \neg B$.

Example 2: Correctness of a Loop Body Using an Invariant

- We want to show that the loop W establishes $s = sum(0, n)$, given
  - $p \equiv 0 \leq k \leq n \wedge s = sum(0, k)$
  - $W \equiv$ ***while*** $k < n$ ***do*** $k := k+1;\ s := s+k$ ***od***
- First, let's write out a full proof of correctness for this program, then we can analyze its parts:

1.	$\{p[s+k/s]\}\ s := s+k\ \{p\}$	*assignment (backward)*
2.	$\{p[s+k/s][k+1/k]\}\ k := k+1\ \{p[s+k/s]\}$	*assignment (backward)*
3.	$\{p[s+k/s][k+1/k]\}\ k := k+1;\ s := s+k\ \{p\}$	*sequence 2, 1*
4.	$p \wedge k < n \rightarrow p[s+k/s][k+1/k]$	*predicate logic*
5.	$\{p \wedge k < n\}\ k := k+1;\ s := s+k\ \{p\}$	*precondition str 4, 3*
6.	$\{$***inv*** $p\}\ W\ \{p \wedge k \geq n\}$	*loop 5*
7.	$p \wedge k \geq n \rightarrow s = sum(0, n)$	*predicate logic*
8.	$\{$***inv*** $p\}\ W\ \{s = sum(0, n)\}$	*postcondition weakening 6, 7*

- The key requirement is showing that p is indeed invariant (line 5). Using the loop rule will let us conclude $\{$***inv*** $p\}\ W\ \{p \wedge k \geq n\}$ (line 6).
- Once the loop terminates, we know $p \wedge k \geq n$ holds, but our final goal is to show $s = sum(0, n)$. It turns out that postcondition weakening is sufficient (we don't need any cleanup code). This completes the loop
- Turning back to the loop body $\{p \wedge k < n\}\ k := k+1;\ s := s+k\ \{p\}$, since this is a sequence, we need to show correctness of each assignment statement (lines 1 and 2) and combine them into a sequence (line 3).
  - We use the backward assignment rule twice, but the proof can certainly be done with forward assignment (see Example 3 below). The structure of the triple makes it easy to infer that backward assignment is being used, so “backward” can be omitted.

    - When we combine the assignments to form the sequence (line 3), the resulting precondition is $p[s+k/s][k+1/k]$, so we use precondition strengthening to get $p \wedge k < n$, which is the form required by the loop rule.
- A reminder: The implication in line 4, $p \wedge k < n \rightarrow p[s+k/s][k+1/k]$, is a predicate logic obligation. We're concentrating on correctness triples, which is why we're omitting formal proofs of the obligations. Still, it's good to convince ourselves that the implication is correct:
- First, let's expand the substitutions used. For $p \wedge k < n \rightarrow p[s+k/s][k+1/k]$, we get
    - $p[s+k/s] \equiv (0 \le k \le n \wedge s = sum(0, k))[s+k/s] \equiv 0 \le k \le n \wedge s+k = sum(0, k)$
    - $p[s+k/s][k+1/k] \equiv (0 \le k \le n \wedge s+k = sum(0, k))\,[k+1/k] \equiv 0 \le k+1 \le n \wedge s+k+1 = sum(0, k+1)$
    - $(p \wedge k < n) \equiv (0 \le k \le n \wedge s = sum(0, k) \wedge k < n)$
- So $p \wedge k < n \rightarrow p[s+k/s][k+1/k]$ expands to an implication that's easy to see is correct.

  $0 \le k \le n \wedge s = sum(0, k) \wedge k < n) \rightarrow 0 \le k+1 \le n \wedge s+k+1 = sum(0, k+1)$
- There's also an obligation in line 7, $(p \wedge k \ge n \rightarrow s = sum(0, n))$ but this one is easier to see: $p \wedge k \ge n$ implies $k \le n \wedge k \ge n$, so $k = n$. Along with $s = sum(0, k)$ from p, we get $s = sum(0, n)$.

Example 3: Correctness of the Same Loop Body Using sp

- Above, we showed correctness of the loop body using *wp*; it's also possible to prove correctness using *sp* instead. We have to replace lines 1 – 5 of the proof above, but lines 6 – 8 don't change because they don't rely on how the loop body was proved to be correct.

1.	$\{ p \wedge k < n \}\ k := k+1\ \{ p_1 \}$	assignment
	where $p_1 \equiv (p \wedge k < n)[k_0/k] \wedge k = k\,[k_0/k]$	
2.	$\{ p_1 \}\ s := s+k\ \{ p_2 \}$	assignment
	where $p_2 \equiv p_1[s_0/s] \wedge s = s_0+k$	
3.	$\{ p \wedge k < n\}\ k := k+1;\ s := s+k\ \{ p_2 \}$	sequence 1, 2
4.	$p_2 \rightarrow p$	predicate logic
5.	$\{ p \wedge k < n\}\ k := k+1;\ s := s+k\ \{ p \}$	postcondition weak. 4, 3

- Here are the expansions of p_1 and p_2 used in the new proof:
    - $p_1 \equiv (p \wedge k < n)[k_0/k] \wedge k = k\,[k_0/k]$

      $\equiv ((0 \le k \le n \wedge s = sum(0, k)) \wedge k < n)[k_0/k] \wedge k = k\,[k_0/k]$

      $\equiv 0 \le k_0 \le n \wedge s = sum(0, k_0) \wedge k_0 < n \wedge k = k_0+1$
    - $p_2 \equiv p_1[s_0/s] \wedge s = s_0+k$

      $\equiv (0 \le k_0 \le n \wedge s = sum(0, k_0) \wedge k_0 < n \wedge k = k_0+1)\,[s_0/s] \wedge s = s_0+k$

      $\equiv 0 \le k_0 \le n \wedge s_0 = sum(0, k_0) \wedge k_0 < n \wedge k = k_0+1 \wedge s = s_0+k$

Example 4: Another Loop Example

- Here's a simple loop program that calculates $s = sum(0, n) = 0+1+\ldots+n$ where $n \geq 0$. (If $n < 0$, define $sum(0, n) = 0$.) Note the loop invariant appears explicitly.

  $\{n \geq 0\}$
  $k := 0;\ s := 0;$
  $\{$**inv** $p_1 \equiv 0 \leq k \leq n \wedge s = sum(0, k)\}$
  while $k < n$ ***do***
  $\quad s := s+k+1;$
  $\quad k := k+1$
  od
  $\{\ s = sum(0, n)\}$

- Informally, to see that this program works, we need
  - $\{n \geq 0\}\ k := 0;\ s := 0\ \{\ p_1 \equiv 0 \leq k \leq n \wedge s = sum(0, k)\}$
  - $\{\ p_1 \wedge k < n\}\ s := s+k+1;\ k := k+1\ \{\ p_1\ \}$
  - $p_1 \wedge k \geq n \rightarrow s = sum(0, n)$
- It's straightforward to use *wp* or *sp* to show that the two triples are correct. A bit of predicate logic gives us the implication, which we need to weaken the loop's postcondition to the one we want.
- We'll do a detailed analysis in a little while.

G. Alternative Invariants Yield Different Programs and Proofs

- The invariant, test, initialization code, and body of a loop are all interconnected: Changing one can change them all. For example, we use $s = sum(0, k)$ in our invariant, so we have the loop terminate with $k = n$.
- If instead we use $s = sum(0, k+1)$ or $s = sum(0, k-1)$ in our invariant, we must terminate with $k+1 = n$ or $k-1 = n$ respectively, and we change the increment of s.

- ***Example 5***: Using $s = sum(0, k)$ as the invariant.

  $\{n \geq 0\}$
  $k := 0;\ s := 0;$
  $\{$**inv** $p_1 \equiv 0 \leq k \leq n \wedge s = sum(0, k)\}$
  while $k < n$ ***do***
  $\quad s := s+k+1;$
  $\quad k := k+1$
  od
  $\{s = sum(0, n)\}$

- ***Example 6:*** Using $s = sum(0, k+1)$ as the invariant.

  $\{n > 0\}$
  $k := 0;\ s := 1;$
  $\{$***inv*** $p_2 \equiv 0 \le k < n \wedge s = sum(0, k+1)\}$
  while $k < n-1$ ***do***
      $s := s+k+2;$
      $k := k+1$
  od
  $\{s = sum(0, n)\}$

- ***Example 7:*** Using $s = sum(0, k-1)$ as the invariant.

  $\{n \ge 0\}$
  $k := 1;\ s := 0;$
  $\{$***inv*** $p_2 \equiv 1 \le k \le n+1 \wedge s = sum(0, k-1)\}$
  while $k \le n$ ***do***
      $s := s+k;$
      $k := k+1$
  od
  $\{s = sum(0, n)\}$