Correctness ("Hoare") Triples, v1.1

Part 2: Sequencing, Assignment, Strengthening, and Weakening

CS 536: Science of Programming, Fall 2021

A. Why

• To specify a program's correctness, we need to know its precondition and postcondition (what should be true before and after executing it).
• The semantics of a verified program combines its program semantics rule with the state-oriented semantics of its specification predicates.
• To connect correctness triples in sequence, we need to weaken and strengthen conditions.

B. Objectives

At the end of today you should know

• Programs may have many different annotations, and we might prefer one annotation over another (or not), depending on the context.
• Under the right conditions, correctness triples can be joined together.
• One general rule for reasoning about assignments goes "backwards" from the postcondition to the precondition.
• What strength is and what weakening and strengthening are.

C. Examples of Partial and Total Correctness With Loops

• For the following examples, let \( W = \text{while } k 
eq 0 \text{ do } k := k - 1 \text{ od} \).

  • Example 1: \( \vdash_{\text{tot}} \{k \geq 0\} W \{k = 0\} \). If we start in a state where \( k \) is \( \geq 0 \), then the loop is guaranteed to terminate in a state satisfying \( k = 0 \).

  • Example 2: \( \vdash \{k = -1\} W \{k = 0\} \) but \( \nvdash_{\text{tot}} \{k = -1\} W \{k = 0\} \). The triple is partially correct but not totally correct because it diverges if \( k = -1 \). I.e., we have \( \nvdash_{\text{tot}} \{k = -1\} W \{T\} \). Also note that partial correctness would hold if we substitute any predicate for \( k = 0 \).

  • Example 3: \( \vdash \{T\} W \{k = 0\} \) but \( \nvdash_{\text{tot}} \{T\} W \{k = 0\} \). The triple is partially correct but not totally correct because it diverges for at least one value of \( k \).

• For the following examples, let \( W' = \text{while } k > 0 \text{ do } k := k - 1 \text{ od} \). (We're changing the loop test of \( W \) so that it terminates immediately when \( k \) is negative.)

  • Example 4: \( \vdash_{\text{tot}} \{T\} W' \{k \leq 0\} \).

  • Example 5: \( \vdash_{\text{tot}} \{k = c_0\} W' \{(c_0 \leq 0 \rightarrow k = c_0) \land (c_0 \geq 0 \rightarrow k = 0)\} \). This is Example 4 with the "strongest" (most precise) postcondition possible.
D. More Correctness Triple Examples

Same Code, Different Conditions

• The same piece of code can be annotated with conditions in different ways, and there's not always a "best" annotation. An annotation might be the most general one possible (we'll discuss this concept soon), but depending on the context, we might prefer a different annotation.

• Below, let \( \text{sum}(x, y) = x + (x+1) + (x+2) + ... + y \). (If \( x > y \), let \( \text{sum}(x, y) = 0 \).) In Examples 9 – 12, we have the same program annotated (with preconditions and postconditions) of various strengths (strength = generality).

  • Example 9: \( \{ T \} : i := 0; s := 0 \{ i = s = 0 \} \).
    - This is the strongest (most precise) annotation for this program.

  • Example 10: \( \{ T \} : i := 0; s := 0 \{ i = s = 0 = \text{sum}(0, i) \} \).
    - This adds a summation relationship to \( i \) and \( s \) when they're both zero.

  • Example 11: \( \{ n \geq 0 \} : i := 0; s := 0 \{ 0 \leq i \leq n \land s = \text{sum}(0, i) \} \).
    - This limits \( i \) to a range of values \( 0, ..., n \). There's no way in the postcondition to know \( n \geq 0 \) unless we assume it in the precondition.

  • Example 12: \( \{ n \geq 0 \} : i := 0; s := 0 \{ 0 \leq i \leq n \land s = \text{sum}(0, i) \} \).
    - The postcondition no longer includes \( i = s = 0 \), which might seem like a disadvantage but will turn out to be an advantage later.

• The next two examples relate to calculating the midpoint in binary search. Though the code is the same, whether the midpoint is strictly between the left and right endpoints depends on whether or not the endpoints are nonadjacent.

  • Example 13: \( \{ L < R \land L \neq R - 1 \} : M := (L + R)/2 \{ L < M < R \} \).
  • Example 14: \( \{ L < R \} : M := (L + R)/2 \{ L \leq M \leq R \} \).

• DeMorgan's laws can apply when a test is a conjunction or disjunction.

  • Example 15: Here we search downward for a nonnegative \( x \) where \( f(x) \) is \( \leq y \); we stop if \( x \) goes negative or we find an \( x \) with \( f(x) \leq y \).
    \begin{verbatim}
{x \geq 0}
while x \geq 0 \land f(x) > y do x := x-1 od
{x < 0 \lor f(x) \leq y}
\end{verbatim}

  • Example 16: This is Example 15 rephrased as an array search; as long as we have a legal index \( i \) and \( b[i] \) isn't \( \leq y \), we move left. We stop if the index becomes illegal or we find an index with \( b[i] \leq y \).
    \begin{verbatim}
{0 \leq i}
while i \geq 0 \land b[i] > y do i := i-1 od
{i < 0 \lor b[i] \leq y}
\end{verbatim}
Joining Two Triples

- If the postcondition of one triple matches the precondition of another, we can join their programs
and form a sequence: If \( \{ p_1 \} S_1 \{ p_2 \} \) and \( \{ p_2 \} S_2 \{ p_3 \} \) are satisfied in a state, so is \( \{ p_1 \} S_1 S_2 \{ p_3 \} \).

- **Example 17**: Note: \( s = \text{sum}(0, k) \) holds before and after the two assignments below, but not between them.

  Combining
  \[
  \begin{align*}
  \{ s = \text{sum}(0, k) \} & \quad s := s + k + 1 \quad \{ s = \text{sum}(0, k+1) \} \\
  \{ s = \text{sum}(0, k+1) \} & \quad k := k + 1 \quad \{ s = \text{sum}(0, k) \}
  \end{align*}
  \]
  yields
  \[
  \{ s = \text{sum}(0, k) \} \quad s := s + k + 1 \quad k := k + 1 \quad \{ s = \text{sum}(0, k) \}
  \]

- **Example 18**: Alternatively, we can increment \( k \) first and then update \( s \).

  Combining
  \[
  \begin{align*}
  \{ s = \text{sum}(0, k) \} & \quad k := k + 1 \quad \{ s = \text{sum}(0, k-1) \} \\
  \{ s = \text{sum}(0, k-1) \} & \quad s := s + k \quad \{ s = \text{sum}(0, k) \}
  \end{align*}
  \]
  yields
  \[
  \{ s = \text{sum}(0, k) \} \quad k := k + 1 \quad s := s + k \quad \{ s = \text{sum}(0, k) \}
  \]

Reasoning About Assignments (Technique 1: “Backward”)

- There are two general rules for reasoning about assignments.
- The first rule is a goal-directed one that works “backwards” — from the postcondition to the
precondition.

**Assignment Rule 1 (“Backward” assignment)**: If \( P(x) \) is a predicate function, then \( \{ P(e) \} v := e \quad \{ P(v) \} \).

It turns out that \( P(e) \) is the most general (the so-called “weakest”) precondition that works with
the assignment \( v := e \) and postcondition \( P(v) \). We'll study this in the next lecture.

- **Example 19**: \( \{ P(m/2) \} m := m/2 \quad \{ P(m) \} \)

  - If \( P(x) \equiv x > 0 \), then this triple expands to \( \{ m/2 > 0 \} m := m/2 \quad \{ m > 0 \} \).

- **Example 20**: \( \{ Q(k+1) \} k := k + 1 \quad \{ Q(k) \} \)

  - If \( Q(x) \equiv s = \text{sum}(0, x) \) then this triple expands to \( \{ s = \text{sum}(0, k+1) \} k := k + 1 \quad \{ s = \text{sum}(0, k) \} \).

- **Example 21**: \( \{ R(s+k+1) \} s := s + k + 1 \quad \{ R(s) \} \)

  - If \( R(x) \equiv x = \text{sum}(0, k+1) \), then this triple expands to

    \[
    \{ s + k + 1 = \text{sum}(0, k+1) \} \quad s := s + k + 1 \quad \{ s = \text{sum}(0, k+1) \}
    \]

In general, for \( \{ P(e) \} v := e \quad \{ P(v) \} \) to be valid, we need the following lemma:

**Assignment Lemma**: For all \( \sigma \), if \( \sigma \models P(e) \) then \( M(v := e, \sigma) = \sigma[v \mapsto \sigma(e)] \models P(v) \).

  - Intuitively, what this says is that if we want to know that \( v \) has property \( P \) after binding \( v \) to the
value of \( e \), we need to know that \( e \) has property \( P \) beforehand.

We won't go into the detailed proof of this lemma, but basically, you work recursively on the
structures of \( P(v) \) and \( P(e) \) simultaneously. The important case is when we encounter an occurrence
of \( v \) in \( P(v) \) and the corresponding occurrence of \( e \) in \( P(e) \). In \( \sigma \models P(e) \), the value of \( e \) is \( \sigma(e) \). In
\( \sigma[v \mapsto \sigma(e)] \models P(v) \), the value of \( v \) is also \( \sigma(e) \). So intuitively, the truth value of \( P(e) \) in \( \sigma \) should match
the truth value of \( P(v) \) in \( \sigma[v \mapsto \sigma(e)] \).
E. Stronger and Weaker Predicates

- **Generalizing the Sequence Rule:** We've already seen that two triples \( \{p\} S_1 \{q\} \) and \( \{q\} S_2 \{r\} \) can be combined to form the sequence \( \{p\} S_1 ; S_2 \{r\} \).

- Say we want to combine two triples that don't have a common middle condition, \( \{p\} S_1 \{q\} \) and \( \{q'\} S_2 \{r\} \).
  - We can do this iff \( q \rightarrow q' \). If \( S_1 \) terminates in state \( \tau \) and \( \{p\} S_1 \{q\} \) is valid, then \( \tau \models q \). If \( \tau \models q' \), so if \( \{q'\} S_2 \{r\} \), then if running \( S_2 \) in \( \tau \) terminates, it terminates in a state satisfying \( r \).
  - This reasoning works both for partial and total correctness.

- Our earlier rule with \( q = q' \) is a special case of this more general rule (since \( q \) always implies itself).

- **Definition:** If \( q \rightarrow r \) then \( q \) is **stronger than** \( r \) and \( r \) is **weaker than** \( q \).
  - I.e., the set of states that satisfy \( q \) is \( \subseteq \) the set of states that satisfy \( r \).
  - (Technically, we should say “stronger than or equal to” and “weaker than or equal to,” since we can have \( q \rightarrow r \), but it’s too much of a mouthful. Then “strictly stronger” means “stronger than or equal to and not equal to;” strictly weaker is similar.)

- **Note:** One notation for implication is \( p \supset q \); it’s important not to read that \( \supset \) as a superset symbol, since the set of states for \( p \) is \( \subseteq \) the set of states for \( q \).

- **Example 22:** \( x = 0 \) is stronger than \( x = 0 \lor x = 1 \), which is stronger than \( x \geq 0 \).

- A predicate corresponds to the set of states that satisfy it, so we can use Venn diagrams with sets of states to display comparisons of predicate strength.
  - In Figure 1, \( p \rightarrow q \) because the set of states for \( p \) is \( \subseteq \) the set of states for \( q \). We don’t have \( p \rightarrow r, q \rightarrow r, r \rightarrow p, \) or \( r \rightarrow q \), but we do have \( p \rightarrow \neg r \) (all of \( p \) is outside \( r \)), \( r \rightarrow \neg p \), and \( \neg q \rightarrow \neg p \).

![Venn Diagram](image.png)

**Figure 1: Predicates As Sets of States**
F. Strengthening and Weakening Conditions

- This idea of freely going from stronger to weaker predicates applies to individual triples, not just sequences. Both of the following properties are valid for both partial and total correctness of triples (i.e., for $\models$ and $\models_{\text{tot}}$).

  - **Preconditions can always be strengthened**: If $p_0 \rightarrow p_1$ and $\{p_1\} S \{q\}$, then $\{p_0\} S \{q\}$.

  - **Postconditions can always be weakened**: If $q_0 \rightarrow q_1$ and $\{p\} S \{q_0\}$, then $\{p\} S \{q_1\}$.

- **Example 23**: If $\{x \geq 0\} S \{y = 0\}$ is valid, then so are
  - $\{x \geq 0\} S \{y = 0 \lor y = 1\}$ (Weakened postcondition)
  - $\{x = 0\} S \{y = 0\}$ (Strengthened precondition)
  - $\{x = 0\} S \{y = 0 \lor y = 1\}$ (Strengthened precondition and weakened postcondition)

- Note these rules still apply if $p_0 \not\rightarrow p_1$ or $q_0 \not\rightarrow q_1$.

- **Example 24**: Let $p_0 \equiv s = \text{sum}(0, k)$ and let $p_1 \equiv s+k+1 = \text{sum}(0, k+1)$.
  Since $p_0 \leftrightarrow p_1$, we can “strengthen” $\{p_1\} S \{q\}$ to $\{p_0\} S \{q\}$, where $q \equiv s = \text{sum}(0, k)$.

- Just because we can strengthen preconditions and weaken postconditions doesn’t mean we should. In the limit, strengthening preconditions and weakening postconditions gives us uninteresting correctness triples:
  - $\sigma \models \{F\} S \{q\}$ and $\sigma \models_{\text{tot}} \{F\} S \{q\}$ have the strongest possible preconditions
  - $\sigma \models \{p\} S \{T\}$ has the weakest possible postcondition
  - $\sigma \models_{\text{tot}} \{p\} S \{T\}$ has the weakest possible postcondition but it does tell us that $S$ terminates when you start it in $p$.

- From the programmer’s point of view, if $\{p\} S \{q\}$ has a bug, then strengthening $p$ or weakening $q$ can get rid of the bug without changing $S$.

  - **Example 25**: If $\{p\} S \{q\}$ causes an error if $x = 0$, we can tell the user to use $\{p \land x \neq 0\} S \{q\}$.

  - **Example 26**: If $\{p\} S \{q\}$ causes an error because $S$ terminates with $y = 1$ (and $y = 1$ does not imply $q$), we can tell the user to use $\{p\} S \{q \lor y = 1\}$.

- From the user’s point of view, weaker preconditions and stronger postconditions make triples more useful:
  - **Weaker preconditions make code more applicable** by increasing the set of starting states. If $p \rightarrow q$ then going from $\{p\} S \{r\}$ to $\{q\} S \{r\}$ provides more information because it increases the set of states that we say $S$ can start in and end with $r$. Basically, we combine $\{p\} S \{r\}$ and $\{\neg p \land q\} S \{r\}$ to get $\{q\} S \{r\}$.

  - **Stronger postconditions make code more specific** by decreasing the set of ending states: If we know $q \rightarrow r$, then $\{p\} S \{r\}$ implies $\{p\} S \{q\}$. This second triple has a smaller set of states we say $S$ can end in, so the second triple provides better information. Here we take $\{p\} S \{r\}$, show that $\{p\} S \{r \land \neg q\}$ never holds, so we can remove that part of the postcondition and get $\{p\} S \{q\}$.