Correctness ("Hoare") Triples

Part 1: Definitions and Basic Properties

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.

B. Objectives

At the end of today you should know

• The syntax of correctness triples (a.k.a. Hoare triples).

• What it means for a correctness triples to be satisfied or to be valid.

• That a state in which a correctness triple is not satisfied is a state where the program has a bug.

C. Correctness Triples ("Hoare Triples")

• A correctness triple (a.k.a. "Hoare triple," after C.A.R. Hoare) is a program $S$ plus its specification predicates $p$ and $q$.

  • The precondition $p$ describes what we're assuming is true about the state before the program begins.

  • The postcondition $q$ describes what should be true about the state after the program terminates.

  • Syntax of correctness triples: $\{p\} S \{q\}$ (Think of it as /* $p$ */ $S$ /* $q$ */)

    $\Rightarrow$ Note: The braces are not part of the precondition or postcondition $\Leftarrow$

• The precondition of $\{p\} S \{q\}$ is $p$, not $\{p\}$. Similarly the postcondition is $q$, not $\{q\}$. Saying "The precondition is $\{p\}$" is like saying "In C, the test in 'if (B) x++;' is 'if (B)'".

D. Satisfaction and Validity of a Correctness Triple

• Informally, for a state to satisfy $\{p\} S \{q\}$, it must be that if we run $S$ in a state that satisfies $p$, then after running $S$, we should be in a state that satisfies $q$. For a triple to be valid, it must be satisfied in all states.

• Important: If we start in a state that doesn't satisfy $p$, we claim nothing about what happens when you run $S$. 
• In some sense, “the triple is satisfied” means “the triple is not buggy”.
• Say you (as the user) have been told not to run \( S \) when \( x < 0 \) because \( S \) calculates \( \sqrt{x} \).
  - And say the triple is \( \{ x \geq 0 \} y := \sqrt{x} \ (y^2 \leq x < (y+1)^2) \).
  - You can’t say this program has a bug when you start in a state with \( x < 0 \), even though the program fails, because you ran the program on bad input.
• Analogous to \( \sigma \models p \) and \( \models p \) for satisfaction and validity of predicates, we’ll use the notations \( \sigma \models \{ p \} S \{ q \} \) and \( \models \{ p \} S \{ q \} \) for satisfaction and validity.

### E. Simple Informal Examples of Correctness

• Before going to the formal definitions of partial and total correctness, let’s look at some simple examples, informally.
  - **Example 1:** \( \models \{ x > 0 \} x := x + 1 \{ x > 0 \} \). This is satisfied in all states, so the triple is valid.
  - **Example 2:** \( \not\models \{ x > 0 \} x := x - 1 \{ x > 0 \} \). This is not satisfied (= “has a bug”) in the state where \( x \) is 1. (That is, \( \{ x = 1 \} \not\models \{ x > 0 \} x := x - 1 \{ x > 0 \} \).) So this triple is not valid because it has a bug.
  - There are a number of ways to fix the buggy program in Example 2:
    - **Example 3:** Make the precondition “stronger” = “more restrictive”: \( \models \{ x > 1 \} x := x - 1 \{ x > 0 \} \) or \( \models \{ x - 1 > 0 \} x := x - 1 \{ x > 0 \} \).
    - **Example 4:** Make the postcondition “weaker” = “less restrictive”: \( \models \{ x > 0 \} x := x - 1 \{ x > -1 \} \).
    - **Example 5:** Change the program: E.g., \( \{ x > 0 \} \) if \( x > 1 \) then \( x := x - 1 \) fi \( \{ x > 0 \} \).
  - **Example 6:** \( \models \{ (x = 2k \lor x = 2k+1) \land x \geq 0 \} x := x/2 \{ x = k \geq 0 \} \)
    (If \( x \) is nonnegative and equals \( 2k \) or \( 2k+1 \) before dividing \( x \) by 2 then after the division, \( x \) equals \( k \), which is nonnegative.)
  - **Example 7:** \( \models \{ s = 1 + 2 + \ldots + k \} s := s + k + 1; \quad k := k + 1 \{ s = 1 + 2 + \ldots + k \} \)
    (If \( s = \) the sum of \( 1 \) through \( k \), then after adding \( k + 1 \) to \( s \) and 1 to \( k \), \( s \) is still the sum of \( 1 \) through \( k \).)
  - **Example 8:** \( \models \{ s = 1 + 2 + \ldots + k \} \quad k := k + 1; \quad s := s + k \{ s = 1 + 2 + \ldots + k \} \)
    (This is like Example 7 but we increment \( k \) first and then update \( s \) by adding \( k \) (not \( k+1 \)) to it.)
  - **Example 9:**
    \[ \models \{ s = 1 + 2 + \ldots + k \} \]
    \[ k := k + 1; \]
    \[ s := s + k + 1 \]
    \[ \{ s = 1 + 2 + \ldots + (k - 1) + (k+1) \} \]
    (This is like Example 8 but we increment \( k \) and then add \( k \) (not \( k+1 \)) to \( s \). Hope it’s okay that \( s \) is not the sum of \( 1 \) through \( k \).)
  - **Definition:** For a triple \( \{ p \} S \{ q \} \), a variable that appears in \( S \) is a **program variable**; a variable that appears in \( p \) or \( q \) is a **condition variable**. A **logical variable** is a condition variable that is not also
a program variable: It appears in the logical reasoning about the program but not the program itself. ("Logical" in this context doesn't mean "Boolean").

**Example 10:** \( \forall \{ x = c_0 \geq 0 \} \ x := x/2 \ \{ c_0 \geq 0 \land x = c_0 / 2 \} \)

(If \( x \) is \( \geq 0 \), then after the assignment \( x := x/2 \), the old value of \( x \) (call it \( c_0 \)) was \( \geq 0 \) and \( x \) is its old value divided by 2. Note \( c_0 \) is a **logical constant**, a logical variable that is a named constant.

### F. Having a Set of States that Satisfy a Predicate

- Before looking at the definitions of program correctness, it will help if we extend the notion of a single state satisfying a predicate to having a set of states satisfying a predicate.

**Notation:** Recall that \( \Sigma_\bot = \Sigma \cup \{ \bot \} \), where \( \Sigma \) is the set of all (well-formed, proper) states.

- Then, \( \sigma \in \Sigma_\bot \) allows \( \sigma = \bot \), but \( \sigma \in \Sigma \) implies \( \sigma \neq \bot \).

- Similarly for a set of states \( \Sigma_0 \), if \( \Sigma_0 \subset \Sigma_\bot \), then we may have \( \bot \in \Sigma_0 \).

- On the other hand, if \( \Sigma_0 \subset \Sigma \), then \( \bot \not\in \Sigma_0 \).

**Notation:** \( \Sigma_0 \not\equiv \bot \) means \( \Sigma_0 \cap \Sigma \), which is the set of all non-\( \bot \) members of \( \Sigma_0 \).

**Definition:** Let \( \Sigma_0 \subset \Sigma_\bot \). We say \( \Sigma_0 \) satisfies \( p \) if every element of \( \Sigma_0 \) satisfies \( p \). In symbols, \( \Sigma_0 \vdash p \) iff for all \( \tau \in \Sigma_0 \), \( \tau \models p \). (Note \( \varnothing \vdash p \), since there exists no \( \tau \in \varnothing \) where \( \tau \models p \).)\(^1\)

Some consequences of the definition:

- If \( \bot \in \Sigma_0 \), then \( \Sigma_0 \not\equiv p \) and \( \Sigma_0 \not\equiv \neg p \).

- \( (\Sigma_0 \vdash p \land \Sigma_0 \vdash \neg p) \) iff \( \Sigma_0 = \varnothing \).

- Since \( \bot \not\equiv p \) (and \( \not\equiv \neg p \)), we have \( \bot \not\in \Sigma_0 \). If \( \tau \not\in \bot \) and \( \tau \models p \) then \( \tau \models \neg p \), so \( \tau \not\in \Sigma_0 \). So \( \Sigma_0 = \varnothing \).

- If \( \Sigma_0 \) has size \( \geq 2 \) and \( \bot \not\in \Sigma_0 \), then \( \Sigma_0 \not\equiv \neg p \) iff \( \Sigma_0 \models p \).

- Either \( \tau \models p \) or \( \tau \models \neg p \) but not both, so \( (\tau \models p \land \tau \models \neg p) \) or \( (\tau \not\models p \land \tau \models \neg p) \).

- If \( \Sigma_0 \) has size \( \geq 2 \) and \( \bot \not\in \Sigma_0 \), then it is **not** the case that \( \Sigma_0 \not\equiv p \) iff \( \Sigma_0 \equiv \neg p \).

- \( (\iff) \) If \( \tau \models \neg p \) then \( \tau \not\equiv p \), so if \( \tau \in \Sigma_0 \), then \( \Sigma_0 \not\equiv p \).

- \( (\implies) \) If \( \bot \not\in \{ \tau, \tau' \} \subset \Sigma_0 \) where \( \tau \models p \) and \( \tau' \models \neg p \), then \( \tau \not\equiv \neg p \) (so \( \tau \in \Sigma_0 \) implies \( \Sigma_0 \not\equiv \neg p \)) and \( \tau' \not\equiv \neg p \) (so \( \tau' \in \Sigma_0 \) implies \( \Sigma_0 \not\equiv p \)). So we have \( \Sigma_0 \not\equiv p \) and \( \Sigma_0 \not\equiv \neg p \) simultaneously.

### G. Total Correctness

- Normally, we want our programs to always terminate in states satisfying their postcondition (assuming we start in a state satisfying the precondition). This property is called **total correctness**.

**Definition:** The triple \( \{ p \} \ S \ { q \} \) is **totally correct in** \( \sigma \) or \( \sigma \) satisfies the triple under **total correctness** iff it’s the case that if \( \sigma \) satisfies \( p \), then running \( S \) in \( \sigma \) always terminates in states satisfying \( q \).\(^2\)

---

1 If you run across an old set of these notes, you should know I changed how the notation works in F'20.

2 The sense of "implies" or "if... then..." used here is not like \( \rightarrow \) (which appears in predicates) or \( \Rightarrow \) (which is a relationship between predicates). It’s "if... then" at a semantic level: If this triple is satisfied or if this set is nonempty, then ... holds.
• In symbols, $\sigma \vdash_{\text{tot}} \{ p \} S \{ q \}$ iff $\sigma \neq \bot$ and (if $\sigma \models p$ then $\bot \notin M(S, \sigma)$ and $M(S, \sigma) \models q$).
  
  • The $\bot \notin M(S, \sigma)$ clause is redundant because $M(S, \sigma) \models q$ implies $\bot \notin M(S, \sigma)$.

• For total correctness, we can’t allow $\sigma = \bot$ because $\bot \not\models p$ and $M(S, \bot) = \{ \bot \} \models q$, so $\sigma \models p$ implies $M(S, \sigma) \models q$ would reduce to (false implies false), which is true.

• Note for $\sigma \vdash_{\text{tot}} \{ p \} S \{ q \}$ we specifically require $\sigma \neq \bot$ because $\bot \not\models p$, so without banning $\bot$ explicitly, we’d have ($\sigma \models p \Rightarrow \ldots$) turn into (false implies ...), which is true.

• **Definition:** The triple $\{ p \} S \{ q \}$ is **totally correct** (is valid under total correctness) iff $\Sigma \vdash_{\text{tot}} \{ p \} S \{ q \}$. I.e., $\sigma \vdash_{\text{tot}} \{ p \} S \{ q \}$ for all $\sigma \in \Sigma$ (Recall $\Sigma$ is the set of well-formed proper states.)

  • **Notation:** We also write $\vdash_{\text{tot}} \{ p \} S \{ q \}$ to mean that the triple is totally correct.

### H. Partial vs Total Correctness

• It turns out that reasoning about total correctness can be broken up into two steps: Determine “partial” correctness, where we ignore the possibility of divergence or runtime errors, and then show that those errors won’t occur.

• **Definition:** The triple $\{ p \} S \{ q \}$ is **partially correct in** $\sigma$ or $\sigma$ satisfies the triple under **partial correctness** iff it’s the case that if $\sigma$ satisfies $p$, then whenever running $S$ in $\sigma$ converges to a memory state, that state satisfies $q$.

• In symbols, $\sigma \vdash \{ p \} S \{ q \}$ iff $\sigma \neq \bot$ and ($\sigma \models p$ implies (for every $\tau \in M(S, \sigma)$, if $\tau \in \Sigma$, then $\tau \models q$)).

• Equivalently, $\sigma \vdash \{ p \} S \{ q \}$ iff $\sigma \in \Sigma$ and ($\sigma \models p$ implies $M(S, \sigma) \models \bot \equiv q$).

• As with total correctness, we can’t allow $\sigma = \bot$ for partial correctness because $\bot \not\models p$, which would make ($\sigma \models p \Rightarrow \ldots$) true.

• **Definition:** The triple $\{ p \} S \{ q \}$ is **partially correct** (i.e., is valid under/partial correctness) iff $\Sigma \vdash \{ p \} S \{ q \}$, i.e., $\sigma \vdash \{ p \} S \{ q \}$ for all states $\sigma$. **Notation:** We also write $\vdash \{ p \} S \{ q \}$.

### I. More Phrasings of Total and Partial Correctness

• An equivalent way to understand partial and total correctness uses the property that if $\sigma \neq \bot$, then ($\sigma \models \neg p$ iff $\sigma \not\models p$) and ($\sigma \models p$ iff $\sigma \not\models \neg p$).

• For total correctness, if $\sigma \neq \bot$, then
  
  • $\sigma \vdash_{\text{tot}} \{ p \} S \{ q \}$
    
    • if $\sigma \models p$ implies $M(S, \sigma) \models q$
    
    • if $\sigma \models \neg p$ or $M(S, \sigma) \models q$
    
    • if $\sigma \models \neg p$ or $\tau \models q$ for every $\tau \in M(S, \sigma)$

  • If $S$ is deterministic, then for some $\tau$, $M(S, \sigma) = \{ \tau \}$ and $\tau \models q$ (so we know $\tau \neq \bot$).

  • If $S$ is nondeterministic, then for every $\tau \in M(S, \sigma)$, we have ($\tau \neq \bot$ and) $\tau \models q$.

• For partial correctness, if $\sigma \neq \bot$, then
  
  • $\sigma \vdash \{ p \} S \{ q \}$
    
    • if $\sigma \models p$ implies $M(S, \sigma) \models \bot \equiv q$
iff $\sigma \models \neg p$ or $M(S, \sigma) \cdot \perp \models q$
iff $\sigma \models \neg p$ or for every $\tau \in M(S, \sigma)$, either $\tau = \perp$ or $\tau \models q$.

- If $S$ is deterministic, then there is only one $\tau$ in $M(S, \sigma)$, and either $\tau = \perp$ or $\tau \models q$.

### J. Unsatisfied Correctness Triples

- It’s useful to figure out when a state **does not satisfy** a triple because not satisfying a triple tells you that there’s some sort of bug in the program.

#### Unsatisfied Total Correctness

- For a state $\sigma \neq \perp$ to not satisfy $\{p\} S \{q\}$ under total correctness, it must satisfy $p$ and running $S$ in it can cause an error or one of its final states does not satisfy $q$.
  - We have $\sigma \models_{\text{tot}} \{p\} S \{q\}$ iff $\sigma \models \neg p$ or $M(S, \sigma) = q$
  - So $\sigma \not\models_{\text{tot}} \{p\} S \{q\}$ iff $\sigma \models p$ and $M(S, \sigma) \neq q$
    - iff $\sigma \models p$ and $(\perp \in M(S, \sigma)$ or for some $\tau \in M(S, \sigma), \tau \neq \perp$ and $\tau \not\models q$ (i.e., $\tau \models \neg q$, since $\tau \neq \perp$).
- If $S$ is deterministic, then $\sigma \models p$ and $M(S, \sigma) = \{\tau\}$ where $\tau = \perp$ or $\tau \models \neg q$.
- If $S$ is nondeterministic, then $\sigma \models p$ and $(\perp \in M(S, \sigma)$ or for some $\tau \in M(S, \sigma), \tau \models \neg q$).
  - Another characterization: $\sigma \models p$ and if $\perp \not\in M(S, \sigma)$, then for some $\tau \in M(S, \sigma), \tau \models \neg q$.

#### Unsatisfied Partial Correctness

- For a state $\sigma \neq \perp$ to not satisfy $\{p\} S \{q\}$ under partial correctness, it must satisfy $p$ and running $S$ in it always terminates in a state satisfying $\neg q$. In symbols
  - We have $\sigma \models \{p\} S \{q\}$ iff $\sigma \models \neg p$ or $M(S, \sigma) \cdot \perp \models q$
  - So $\sigma \not\models \{p\} S \{q\}$ iff $\sigma \models p$ and $M(S, \sigma) \cdot \perp \not\models q$
    - We know $M(S, \sigma) \cdot \perp \models q$ holds iff for every $\tau \in M(S, \sigma)$, if $\tau \neq \perp$, then $\tau \models q$
    - So $M(S, \sigma) \cdot \perp \models q$ holds iff for some $\tau \in M(S, \sigma)$, we have $\tau \models \neg q$ (since $\tau \not\models q$ with $\tau \neq \perp$)
    - Substituting back, $\sigma \not\models \{p\} S \{q\}$ iff $\sigma \models p$ and $\tau \models \neg q$ for some $\tau \in M(S, \sigma)$.
- If $S$ is deterministic, then we need $\sigma \models p \land M(S, \sigma) = \{\tau\}$ where $\tau \models \neg q$.
- If $S$ is nondeterministic, $M(S, \sigma)$ can include $\perp$ and states that satisfy $q$, but there must be at least one state in $M(S, \sigma)$ that satisfies $\neg q$.
  - **Note**: If $S$ is nondeterministic and partial correctness of $\{p\} S \{q\}$ fails under $\sigma$, it’s possible that some execution paths of $S$ don’t terminate or terminate in states satisfying $q$, but there must be some execution path that ends in a state satisfying $\neg q$.

### K. Three Extreme (Mostly Trivial) Cases

- There are three edge cases where partial correctness occurs for uninformative reasons.. First recall the definition of partial correctness: $\sigma \models \{p\} S \{q\}$ means (if $\sigma \models p$, then $M(S, \sigma) \cdot \perp \models q$).
• **p is a contradiction** (i.e., $\models \neg p$). Since $\sigma \models p$ never holds, the implication (if $\sigma \models p$ then ...) always holds, so partial correctness of $\{p\} S \{q\}$ always holds. So for example, $\{F\} S \{q\}$ is valid under partial correctness, for all $S$ and $q$.

• **S always causes an error.** If $M(S, \sigma) = \{\bot\}$ then $M(S, \sigma) - \bot = \emptyset$, and $\emptyset \models q$, so again we get partial correctness of $\{p\} S \{q\}$.

• **q is a tautology** (i.e., $\models q$). Then for any $\sigma$, $M(S, \sigma) - \bot \models q$, so whether $\sigma$ satisfies $p$ or not, we get partial correctness of $\{p\} S \{q\}$. So for example, $\{p\} S \{T\}$ is valid under partial correctness for all $p$ and $S$.

• For total correctness, recall $\sigma \models_{\text{tot}} \{p\} S \{q\}$ means (if $\sigma \models p$, then $M(S, \sigma) \models q$). (Also, recall that since $\bot \not\models q$, if $M(S, \sigma) \models q$, then $\bot \notin M(S, \sigma)$.)

• **p is a contradiction.** The argument here is the same as for partial correctness, so for all $S$ and $q$, the triple $\{F\} S \{q\}$ is valid under total correctness.

• **S always causes an error.** Since $M(S, \sigma) = \{\bot\}$, we know $M(S, \sigma) \models \bot \not\models q$. So total correctness of $\{p\} S \{q\}$ always fails.

• **q is a tautology.** $\sigma \models_{\text{tot}} \{p\} S \{T\}$ does says something interesting. Since $M(S, \sigma) \models T$ implies $\bot \not\in M(S, \sigma)$, satisfaction of $\sigma \models_{\text{tot}} \{p\} S \{T\}$ requires $S$ to **always terminate** under $\sigma$. So validity of $\models_{\text{tot}} \{p\} S \{T\}$ happens when $S$ always terminates when started in a state satisfying $p$.

• As a general principle, since total correctness is partial correctness plus termination, we have $\sigma \models_{\text{tot}} \{p\} S \{q\}$ iff $\sigma \models \{p\} S \{q\}$ and $\sigma \models_{\text{tot}} \{p\} S \{T\}$. Again, this means that to show that a triple is totally correct, we can prove partial correctness and termination separately.