Correctness ("Hoare") Triples, pt. 1: Definitions and Basic Properties

CS 536: Science of Programming, Fall 2020

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

- 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”:
    $\models \{x > 0\} x := x-1 \{x > 0\}$ or $\models \{x > 0\} x := x-1 \{x > 0\}$
  • Example 4: Make the postcondition “weaker” = “less restrictive”:
    $\models \{x > 0\} x := x-1 \{x > 1\}$
• Example 5: Change the program: E.g., $\{x > 0\} \text{if } x > 1 \text{ then } x := x-1 \text{ fi } \{x > 0\}$
• 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: $\models \{s = 1 + 2 + \ldots + k\} s := s+k+1; k := k+1 \{s = 1 + 2 + \ldots + k\}$
  (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: $\models \{s = 1 + 2 + \ldots + k\} s := s+k \{s = 1 + 2 + \ldots + k\}$
  (This is like Example 7 but we increment $k$ first and then update $s$ by adding $k$ (not $k+1$) to it.)
• Example 9:
  $\models \{s = 1 + 2 + \ldots + k\}$
  $\quad k := k+1;$
  $\quad s := s+k+1$
  $\quad \{s = 1 + 2 + \ldots + (k-1) + (k+1)\}$
  (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 $k$.)
• Example 10: $\models \{x = c_0 \geq 0\} x := x/2 \{c_0 \geq 0 \land x = c_0/2\}$
  (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 $x$ = its old value divided by 2. Note $c_0$ is a logical constant — it doesn’t appear inside the program, just in the
proof of correctness.) (“Logical” in the sense of talking about proofs, not boolean logic.) (Maybe “constant logical variable” would be a better name.) $x$ is a program variable (appears in the actual program); logical variables don’t appear in the program, only in the conditions.

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 $\Sigma = \Sigma \cup \{\bot\}$, so $\sigma \in \Sigma$ allows $\sigma = \bot$, but $\sigma \in \Sigma$ implies $\sigma \neq \bot$. Similarly for a set of states $\Sigma_0$, if $\Sigma_0 \subseteq \Sigma_1$, then we may have $\bot \in \Sigma_0$. On the other hand, if $\Sigma_0 \subseteq \Sigma$, then $\bot \notin \Sigma_0$.

- **Notation:** $\Sigma_0 - \bot$ and $\Sigma_0 \cap \Sigma$ both mean $\Sigma_0$ less $\bot$: $\Sigma_0 - \bot = \Sigma_0 \cap \Sigma = \{ \sigma \in \Sigma_0 \mid \sigma \in \Sigma \} = \{ \sigma \in \Sigma_0 \mid \sigma \neq \bot \}$.

- **Definition:** Let $\Sigma_0 \subseteq \Sigma_1$. We say $\Sigma_0$ satisfies $p$ if it is nonempty¹ and every element of $\Sigma_0$ satisfies $p$. In symbols, $\Sigma \models p$ if $\Sigma \neq \emptyset$ and for all $\tau \in \Sigma_0$, $\tau \models p$.

Some consequences of the definition:

- Since $\bot$ satisfies no predicate, if $\bot \in \Sigma_0$, then $\Sigma_0 \not\models p$ and $\Sigma_0 \not\models \neg p$.
- If $\Sigma_0 \subseteq \Sigma$, we have $(\Sigma_0 \models p \iff \Sigma_0 \not\models \neg p)$ and $(\Sigma_0 \not\models \neg p \iff \Sigma_0 \not\models p)$.

- The converses $(\Sigma_0 \not\models \neg p \iff \Sigma_0 \models p)$ and $(\Sigma_0 \not\models p \iff \Sigma_0 \models \neg p)$ hold if $\Sigma_0$ is a singleton set $\subseteq \Sigma$.

- If $\tau \neq \bot$, then $\tau \models p$ iff $\tau \models \neg \neg p$, so if $\Sigma_0 = \{ \tau \} \subseteq \{ \bot \}$, then either $(\Sigma_0 \models p$ and $\Sigma_0 \not\models \neg p)$ or $(\Sigma_0 \not\models \neg p$ and $\Sigma_0 \not\models p)$.

- If $\Sigma_0 \subseteq \Sigma$ and $\Sigma_0$ contains more than one state, then it’s possible for to have $\Sigma_0 \not\models p$ and $\Sigma_0 \not\models \neg p$.

- If there are $\tau$ and $\tau' \in \Sigma_0$ with $\tau \models p$ and $\tau' \models \neg p$, then $\Sigma_0 \not\models p$ and $\Sigma_0 \not\models \neg p$.

- The converse also holds if $\Sigma_0$ does not include $\bot$.

- Since validity means "satisfied in all states", we have $\models p$ if $\Sigma \models p$ where $\Sigma_0 = \{ \sigma \in \Sigma \mid \sigma \models p \}$. Since states that don’t $\models p$ trivially satisfy $\{ p \} \cap \{ q \}$, we get $\Sigma \models \{ p \} \cap \{ q \}$.

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** if it’s the case that if $\sigma$ satisfies $p$, then running $S$ in $\sigma$ always terminates in states satisfying $q$.$^2$

- In symbols, $\sigma \models_{\text{tot}} \{ p \} S \{ q \}$ iff $(\sigma \models p \iff M(S, \sigma) \subseteq \Sigma$ and $M(S, \sigma) \models q)$.

- $M(S, \sigma) \subseteq \Sigma$ iff running $S$ in $\sigma$ always terminates in a state because $M(S, \sigma) \subseteq \Sigma$ iff $\bot \notin M(S, \sigma)$.

¹ If we allowed $\Sigma_0 = \emptyset$ then we would have $\emptyset \models p$ and $\emptyset \not\models \neg p$, which just doesn’t sound right :-)

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.
• \( M(S, \sigma) \models q \) iff \( M(S, \sigma) \neq \emptyset \) and for every \( \tau \in \Sigma, \) if \( \tau \in M(S, \sigma), \) then \( \tau \models q. \)

• Since \( \bot \neq q, \) we know \( M(S, \sigma) \models q \) implies \( \bot \notin M(S, \sigma), \) which again implies that running \( S \) in \( \sigma \) always terminates.

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

• **Definition:** The triple \( \{ p \} S \{ q \} \) is **totally correct** (is valid under total correctness) iff \( \sigma \models_{\text{tot}} \{ p \} S \{ q \} \) for all \( \sigma. \) The notation is \( \models_{\text{tot}} \{ p \} S \{ q \}. \) (And again, \( \models_{\text{tot}} \{ p \} S \{ q \} \) means \( \Sigma \models_{\text{tot}} \{ p \} S \{ q \}. \)

**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, \( \sigma \models \{ p \} S \{ q \} \) iff \( \sigma \in \Sigma \) and \( (\sigma \models p) \) implies (for every \( \tau \in M(S, \sigma), \) if \( \tau \in \Sigma, \) then \( \tau \models q). \)

• Equivalently, \( \sigma \models \{ p \} S \{ q \} \) iff \( \sigma \in \Sigma \) and \( (\sigma \models p) \) implies \( M(S, \sigma) = \{ \bot \} \) or \( M(S, \sigma) \neq \bot \models q). \)

  • If running \( S \) in \( \sigma \) never terminates (i.e., if \( M(S, \sigma) \) only contains flavors of \( \bot \), then it’s vacuously true that (for every \( \tau \in M(S, \sigma), \) if \( \tau \in \Sigma, \) then \( \tau \models q). \)) (Or equivalently, there does not exist a \( \tau \in M(S, \sigma) \) with \( \tau \in \Sigma \) and \( \tau \models q. \))

  • Note we need to include \( M(S, \sigma) = \{ \bot \} \) because \( M(S, \sigma) \neq \bot \models q \) doesn’t hold if \( M(S, \sigma) = \{ \bot \}. \)

• As with total correctness, we can’t allow \( \sigma = \bot \) for partial correctness because \( \bot \neq p \) and \( M(S, \bot) = \{ \bot \}, \) so \( (\sigma \models p) \) implies \( M(S, \sigma) \subseteq \{ \bot \} \) or \( M(S, \sigma) \neq \bot \models q \) would reduce to (false implies false or \( \emptyset \models \bot \)), which is (false implies false or false), which is true.

• **Definition:** The triple \( \{ p \} S \{ q \} \) is **partially correct** (is valid under partial correctness) iff \( \sigma \models \{ p \} S \{ q \} \) for all \( \sigma. \) The notation is \( \models \{ p \} S \{ q \}. \) (And \( \models \{ p \} S \{ q \} \) means \( \Sigma \models \{ p \} S \{ q \}. \)

**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 \( (\sigma \models \neg p \iff \sigma \neq p) \) and \( (\sigma \models p \iff \sigma \neq \neg p). \)

• For total correctness
  \[
  \sigma \models_{\text{tot}} \{ p \} S \{ q \}
  \]
  iff \( \sigma \neq \bot \) and \( (\sigma \models p \text{ implies } M(S, \sigma) \models q) \)
  iff \( \sigma \neq \bot \) and \( (\sigma \models \neg p \text{ or } M(S, \sigma) \models q) \)

• If \( \sigma \neq \bot, \) then \( \sigma \models_{\text{tot}} \{ p \} S \{ q \} \) iff \( (\sigma \models \neg p \text{ or } M(S, \sigma) \models q). \)

• For partial correctness,
  \[
  \sigma \models \{ p \} S \{ q \}
  \]
  iff \( \sigma \neq \bot \) and \( (\sigma \models p \text{ implies (for every } \tau \in M(S, \sigma), \tau = \bot \text{ or } \tau \models q)) \)
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 $\sigma \in \Sigma$ to not satisfy $\{p\} S \{q\}$ under total correctness, it must satisfy $p$ and running $S$ in it can cause an error and/or can yield a final state in which $q$ is false. Since $\sigma \in \Sigma$ implies $\sigma \neq \bot$, we can assume $\sigma \neq \bot$ through the rest of the discussion. Then,

$$\sigma \models_{\text{tot}} \{p\} S \{q\} \text{ iff } (\sigma \models \neg p \text{ or } M(S, \sigma) \models q)$$

So $\sigma \models_{\text{tot}} \{p\} S \{q\} \text{ iff } \neg (\sigma \models \neg p \text{ or } M(S, \sigma) \models q)$.

- If $S$ is deterministic, then $M(S, \sigma)$ has just one member, so for deterministic $S$, we find $\sigma \models_{\text{tot}} \{p\} S \{q\}$ if $\sigma \models p$ and $(M(S, \sigma) = \{\bot\}$ or $M(S, \sigma) = \{\tau\}$ with $\tau \models \neg q$). In English, $\sigma$ satisfies $p$ and running $S$ in $\sigma$ either doesn't terminate or it terminates in a state in which $q$ is false.

Unsatisfied Partial Correctness

- $(M(S, \sigma) = \{\bot\}$ or $M(S, \sigma) - \bot \models q)$.

- For a state $\sigma \in \Sigma$ 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

$$\sigma \models \{p\} S \{q\}$$

iff $\sigma \models p$ implies $(M(S, \sigma) = \{\bot\}$ or $M(S, \sigma) - \bot \models q)$

iff $\sigma \models p$ and $M(S, \sigma) = \{\bot\}$ or $M(S, \sigma) - \bot \models q)$

So $\sigma \models \{p\} S \{q\}$

iff $\sigma \models p$ and $M(S, \sigma) \neq \{\bot\}$ and $M(S, \sigma) - \bot \models q)$

iff $\sigma \models p$ and for some $\tau \in M(S, \sigma)$, $\tau \neq \bot$ and $\tau \models q)$

- From this last line, if $S$ is deterministic, then $\sigma \models \{p\} S \{q\}$ iff $\sigma \models p$ and $(M(S, \sigma) = \{\tau\} \models \neg q$ where $\tau \in \Sigma$.

- If $S$ is nondeterministic, it's useful to go back and study $M(S, \sigma) - \bot \models q$.

  - Recall that for any $\Sigma' \subseteq \Sigma$, the definition was $\Sigma' \models q$ iff $(\Sigma' \neq \emptyset$ and $\tau \models q$ for every $\tau \in \Sigma').$

  - So $\Sigma' \models q$ iff $(\Sigma' = \emptyset$ or $\tau \models q$ for some $\tau \in \Sigma').$

  - So $M(S, \sigma) - \bot \models q$ iff $(M(S, \sigma) - \bot = \emptyset$ or $\tau \models q$ for some $\tau \in M(S, \sigma) - \bot$.

  - For that one $\tau \in M(S, \sigma) - \bot$, we know $\tau \models q$ if $\tau \models \neg q$, but this doesn't say anything about the rest of $M(S, \sigma) - \bot$. I.e., for other $\gamma \in M(S, \sigma) - \bot - \{\tau\},$ we can have $\gamma \models q$ or $\gamma \models \neg q$.

  - Summarizing, if $S$ is nondeterministic, $\sigma \models \{p\} S \{q\}$ iff for some $\tau \in M(S, \sigma)$, $\tau \models \neg q$, it's still possible for there to be $\gamma \in M(S, \sigma)$ where $\gamma \models q$.
• In English, if σ doesn't satisfy \( \{ p \} S \{ q \} \) for partial correctness, if \( S \) is nondeterministic, then some execution of \( S \) under σ ends in a state satisfying \( \neg q \). Even so, it's possible for \( S \) under σ to end in a state satisfying \( q \).

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

• There are three extreme 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 \( M(S, \sigma) = \perp \) or \( M(S, \sigma) - \perp \models q \)).
  
  - **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.
  
  - **S always causes an error.** (\( M(S, \sigma) = \perp \) for all \( \sigma \).) Then the implication (if \( ... \) then \( M(S, \sigma) = \perp \) or \( ... \)) always holds, so partial correctness of \( \{ p \} S \{ q \} \) always holds.
  
  - **q is a tautology** (i.e., \( \models q \)). Then for any \( \sigma \), \( M(S, \sigma) - \perp \models q \) (unless \( M(S, \sigma) = (\perp) \)). Either way, partial correctness of \( \{ p \} S \{ q \} \) holds.

• For total correctness, recall \( \sigma \models_{tot} \{ p \} S \{ q \} \) means (if \( \sigma \models p \), then \( \perp \in M(S, \sigma) \) and \( M(S, \sigma) \models q \)\(^3\)).
  
  - **p is a contradiction** (i.e., \( \models \neg p \)). As with partial correctness, since \( \sigma \not\models p \), the implication (if \( \sigma \models p \), then \( ... \)) holds, so we get total correctness of \( \{ p \} S \{ q \} \). So this case is trivial in the same way as for partial correctness.
  
  - **S always causes an error.** Then (if \( ... \) then \( \perp \notin M(S, \sigma) \) and \( ... \)) never holds, so total correctness never holds. Unlike partial correctness (where if \( S \) always causes an error, partial correctness holds), we get the opposite conclusion: total correctness never holds.
  
  - **q is a tautology.** This actually implies something interesting. Unless \( \perp \in M(S, \sigma) \), we know \( M(S, \sigma) \models q \) holds, so total correctness reduces to (if \( \sigma \models p \), then \( \perp \notin M(S, \alpha) \) and true). I.e., \( \sigma \models_{tot} \{ p \} S \{ q \} \) reduces to \( \sigma \models_{tot} \{ p \} S \{ T \} \). So \( \sigma \models_{tot} \{ p \} S \{ T \} \) iff (if \( \sigma \models p \), then \( S \) terminates under \( \sigma \)).
  
  - **Total correctness is partial correctness plus termination:** Since \( \sigma \models_{tot} \{ p \} S \{ T \} \) holds iff (\( \sigma \models p \) implies \( \perp \notin M(S, \sigma) \)) and partial correctness \( \sigma \models \{ p \} S \{ q \} \) holds iff (\( \sigma \models p \) implies \( M(S, \sigma) = \perp \) or \( M(S, \sigma) - \perp \models q \)), their conjunction holds iff total correctness holds: \( \sigma \models_{tot} \{ p \} S \{ q \} \) iff \( \sigma \models \{ p \} S \{ q \} \) and \( \sigma \models_{tot} \{ p \} S \{ T \} \).

---

\(^3\) \( M(S, \sigma) = q \) implies \( \perp \notin M(S, \alpha) \), but saying \( \perp \notin M(S, \alpha) \) explicitly doesn't hurt here.