Correctness ("Hoare") Triples

Part 1: Definitions and Basic Properties

CS 536: Science of Programming, Spring 2023

2023-02-07 pp. 3, 4, 8

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 joins a program's state-transformation semantics with the state-oriented semantics of the 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 */)

  ⇒ ***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 "$\{ p \}$" is like saying "In C, the test in '`if (B) x++;`' is '`if (B)`'" instead of just `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.
  - There's more than one way to understand "after running S", and this will give us two notions of satisfaction.

- ***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 in σ" means "the triple is not buggy in σ", which seems like a rather weak claim.
  - However, "the triple is not satisfied in σ" means "the triple has a bug in σ", which is a pretty strong statement.
- For example, say you're given the triple $\{ x \geq 0 \}\ S\ \{ y^2 \leq x < (y+1)^2 \}$.
  - The triple claims that running the program when x is nonnegative sets y to the integer square root of x.
  - If you run it when x is negative, all bets are off: S could run and terminate with y = some value, it could diverge, it could produce a runtime error. None of these behaviors are bugs because you ran S on a bad input.
- ***Validity*** for correctness triples is analogous to validity of a predicate: The triple must be satisfied in every (well-formed, proper) state.
  - 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.
- ***Notation:*** Analogous to our notation for predicates, for triples
  - $\sigma \models \{ p \}\ S\ \{ q \}$ means σ satisfies the triple.
  - $\sigma \not\models \{ p \}\ S\ \{ q \}$ means σ does not satisfy the triple.
  - $\models \{ p \}\ S\ \{ q \}$ means the triple is valid.
  - $\not\models \{ p \}\ S\ \{ q \}$ means the triple is invalid: $\sigma \not\models \{ p \}\ S\ \{ q \}$ for some σ.

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. (As usual, we'll assume the variables range over ℤ.)
- ***Example 1***: $\models \{x > 0\}\ x := x + 1\ \{ x > 0 \}$. The triple is valid: It's satisfied for all states where $x > 0$.
- ***Example 2***:
  - $\{ x = 1 \} \not\models \{ x > 0 \}\ x := x - 1\ \{ x > 0 \}$: The triple is not satisfied (has a bug) when run with $x = 1$ because it terminates with $x = 0$, not > 0. Thus the triple is not valid: $\not\models \{ x > 0 \}\ x := x - 1$ $\{ x > 0 \}$.
- There are a number of ways to fix the buggy program in Example 2:
  - ***Example 3***: Make the precondition "***stronger***' = "more restrictive". For example, we could use $\models \{ x > 1 \}\ x := x - 1\ \{ x > 0 \}$.
  - ***Example 4***: Make the postcondition "***weaker***" = "less restrictive". For example, we could use $\models \{ x > 0 \}\ x := x - 1\ \{ x > -1 \}$.

- *Example 5*: Change the program. One way is $\{x > 0\}$ ***if*** $x > 1$ ***then*** $x := x - 1$ ***fi*** $\{x > 0\}$.
- Let's have some more complicated examples.
- ***Example 6***: $\models \{x \ge 0 \wedge (x = 2 * k \vee x = 2 * k + 1)\}\ x := x / 2\ \{x = k \ge 0\}$.
  - If x is nonnegative, then the program halves it with truncation.
- ***Example 7***: Assume $sum(0, k)$ yields the sum of the integers 0 through k, then
  $\models \{s = sum(0, k)\}\ s := s + k + 1;\ k := k + 1\ \{s = sum(0, k)\}$.
  - The triple says if $s = sum(0, k)$ when we start, then $s = sum(0, k)$ when we finish.
  - It's ok that s and k are changed by the program because $s = sum(0, k)$ is true in both places relative to the state at that point in time.
  - (Later, we'll use this program as part of a larger program, and we'll augment the conditions with information about how the ending values of k and s are larger than the starting values.)
  - Note we can write $s = 0 + 1 + 2 + \ldots + k$ as an informal equivalent of $s = sum(0, k)$, but it doesn't strictly have the form of a predicate as $s = sum(0, k)$ does.
- ***Example 8***: $\models \{s = sum(0, k)\}\ k := k + 1;\ s := s + k\ \{s = sum(0, k)\}$
  - This has the same specification as Example 7 but the code is different: It increments k first and then update s by adding k (not $k + 1$) to it.)
- ***Example 9***: [Note the invalidity] $\not\models \{s = sum(0, k)\}\ k := k + 1;\ s := s + k + 1\ \{s = sum(0, k)\}$
  - This is like Example 8 but the program doesn't meet its specification. To get validity, the postcondition should be $s = sum(0, k) + 1$. (Or more likely, the code needs to be fixed.)

F. Connecting Starting and Ending Values of Variables

- There are times when we want the postcondition to be able to refer to values that the variables started with.
- Recall Examples 7 and 8: $\models \{s = sum(0, k)\}\ S\ \{s = sum(0, k)\}$ (where S is different in the two examples). Say we want the postcondition to include "k gets larger by 1" somehow. What we can do is create a new variable (call it k_0) whose job it is to refer to the starting value of k, before we run S.
- We'll make the precondition $k = k_0 \wedge s = sum(0, k)$ ("k has some starting value and s is the sum of 0 through k"). We'll make the postcondition $k = k_0 + 1 \wedge s = sum(0, k)$ ("k is one larger than its starting value and s is the sum of 0 through k (for this new value of k)".
- [2023-02-07] We actually did the same thing in Example 6: $\models \{x \ge 0 \wedge (x = 2 * k \vee x = 2 * k + 1)\}$ $x := x / 2\ \{x = k \ge 0\}$. The variable k helps describe the value of x before and after execution. One interesting feature of k and k_0 is that they don't appear in the program, only the specifications. So where do variables appear in correctness triples?
- ***Definition:*** For a triple $\{p\}\ S\ \{q\}$,
  - A variable that appears in S is a ***program variable***. E.g., x is a program variable in $x := 1$. We manipulate them to get work done.

- A variable that appears in p or q is a ***condition variable***. E.g., y in $\{y>0\}$ … $\{\ldots.\}$. We use condition variables to reason about our program. They may or may not also be program variables. (These are not the same kind of condition variables used in distributed programming.)
  - E.g., in $\{y>0\}\ y:=y+1\ \{y>1\}$, y is a program and a condition variable.
  - A ***logical variable*** is a condition variable that is not also a program variable. E.g., c in $\{z\geq c\}\ z:=z+1\ \{z>c\}$. We use them to reason about our program but they don't appear in the program itself. (Note that here, "logical" doesn't mean "Boolean".)
  - A ***logical constant*** is a named constant logical variable. E.g., c in the previous example. Logical constants are great for keeping track of an old value of a variable.
- ***Example 10***: $\models\{x=x_0\geq 0\}\ x:=x/2\ \{x_0\geq 0\wedge x=x_0/2\}$. If x is ≥ 0, then after the assignment $x:=x/2$, the old value of x (which we're calling x_0) was ≥ 0 and x is its old value divided by 2. Here, x is a program and condition variable and x_0 is a logical constant.

G. 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 $\Sigma_\perp=\Sigma\cup\{\perp\}$, where Σ is the set of all (well-formed, proper) states.
  - Then, $\sigma\in\Sigma_\perp$ allows $\sigma=\perp$, but $\sigma\in\Sigma$ implies $\sigma\neq\perp$.
  - Similarly for a set of states Σ_0, if $\Sigma_0\subseteq\Sigma_\perp$, then we may have $\perp\in\Sigma_0$.
  - On the other hand, if $\Sigma_0\subseteq\Sigma$, then $\perp\notin\Sigma_0$.
- ***Notation***: $\Sigma_0-\perp$ means $\Sigma_0\cap\Sigma$, the subset of Σ_0 containing its non-$\perp$ members.
- ***Definition***: Let $\Sigma_0\subseteq\Sigma_\perp$. We say Σ_0 ***satisfies*** p if every element of Σ_0 satisfies p.
  - In symbols, $\Sigma_0\models p$ iff for all $\tau\in\Sigma_0$, $\tau\models p$. It follows that $\Sigma_0\not\models p$ iff $\tau\not\models p$ for some $\tau\in\Sigma_0$.
  - (Note $\varnothing\not\models p$ is clearly false, which means $\varnothing\models p$ is true.)
- Some consequences of the definition:
  - If $\perp\in\Sigma_0$, then $\Sigma_0\not\models p$ and $\Sigma_0\not\models\neg p$.
  - $(\Sigma_0\models p$ and $\Sigma_0\models\neg p)$ iff $\Sigma_0=\varnothing$.
    - Since $\perp\not\models p$ (and $\not\models\neg p$), we have $\perp\notin\Sigma_0$. If $\tau\neq\perp$ and $\tau\models p$ then $\tau\not\models\neg p$, so $\tau\notin\Sigma_0$. So $\Sigma_0=\varnothing$.
  - If $\perp\notin\Sigma_0$ and Σ_0 is a singleton set (it has size = 1), then $\Sigma_0\models p$ iff $\Sigma_0\not\models\neg p$ (and $\Sigma_0\models\neg p$ iff $\Sigma_0\not\models p$). [2023-02-07]
    - Either $\tau\models p$ or $\tau\models\neg p$ but not both, so ($\tau\models p$ and $\tau\not\models\neg p$) or ($\tau\not\models p$ and $\tau\models\neg p$).
  - If $\Sigma_0-\perp$ is not a singleton set then it is possible that $\Sigma_0-\perp\not\models$ both p and $\neg p$.
    - Say we have $\sigma_1,\sigma_2\in\Sigma_0-\perp$ where $\sigma_1\models p$ and $\sigma_2\models\neg p$. For $\Sigma_0-\perp\models p$, we need all its members to satisfy p, but that's false, so $\Sigma_0-\perp\not\models p$. Similarly, $\Sigma_0-\perp\not\models\neg p$ because not all members of $\Sigma_0-\perp$ satisfy $\neg p$.

H. Total Correctness

- Normally, we want our programs to always terminate[1] 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*** σ or σ satisfies the triple under ***total correctness*** iff it's the case that if σ satisfies p, then running S in σ always terminates in a state satisfying q.[2]
- In symbols, $\sigma \models_{\text{tot}} \{p\}\,S\,\{q\}$ iff $\sigma \neq \bot$ and $($if $\sigma \models p$ then $\bot \notin M(S,\sigma)$ and $M(S,\sigma) \models q)$.
  - Note $M(S,\sigma) \models q$ implies $\bot \notin M(S,\sigma)$, so it's redundant to say $\bot \notin M(S,\sigma)$ explicitly, but it's not a bad idea to emphasize it for a while.
  - We require $\sigma \neq \bot$ because we want the implication ($\sigma \models p$ implies $M(S,\sigma) \models q$) to be false when $\sigma = \bot$. Since $M(S,\bot) = \{\bot\} \not\models q$, if we allowed $\bot \models p$ then the implication would become true (since false implies false).
- ***Definition***: The triple $\{p\}\,S\,\{q\}$ is ***totally correct*** (is ***valid*** under ***total correctness***) iff $\sigma \models_{\text{tot}} \{p\}\,S\,\{q\}$ for all $\sigma \in \Sigma$ (Recall Σ is the set of well-formed proper states.) Usually, we'll write $\models_{\text{tot}} \{p\}\,S\,\{q\}$.

I. 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 termination -- i.e., that those errors won't occur.
- ***Definition***: The triple $\{p\}\,S\,\{q\}$ is ***partially correct in*** σ or σ ***satisfies the triple under partial correctness*** iff
  - $\sigma \neq \bot$ and
  - If σ satisfies p, then whenever running S in σ terminates (without error), the final state satisfies q.
- In symbols, $\sigma \models \{p\}\,S\,\{q\}$ 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, $\sigma \models \{p\}\,S\,\{q\}$ iff $\sigma \neq \bot$ and $(\sigma \models p$ implies $M(S,\sigma) - \bot \models q)$.
  - It might help to point out that S not terminating under σ doesn't make partial correctness false.

---

[1] "Terminate" will mean "terminate without error" (Final state $\in \Sigma - \bot$). "Terminate possibly with an error" means we end in $\Sigma_\bot$.

[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.

- Note we must say explicitly that $\bot \not\vDash \{p\}\, S\, \{q\}$ because otherwise the general case would hold: $\bot \not\vDash p$ and $M(S,\sigma) - \bot = \{\bot\} - \bot = \varnothing \vDash q$, so the general case $(\sigma \vDash p$ implies $M(S,\sigma) - \bot \vDash q)$ would be true (i.e., false implies false).
- ***Definition***: The triple $\{p\}\, S\, \{q\}$ is ***partially correct*** (i.e., is ***valid*** under/for ***partial correctness***) iff $\sigma \vDash \{p\}\, S\, \{q\}$ for all states σ. ***Notation***: We usually write $\vDash \{p\}\, S\, \{q\}$ but $\Sigma \vDash \{p\}\, S\, \{q\}$ is also ok.

J. 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 ($\sigma \vDash \neg p$ iff $\sigma \not\vDash p$) and ($\sigma \vDash p$ iff $\sigma \not\vDash \neg p$).
- For total correctness, just generally, if $\sigma \neq \bot$, then

  $\sigma \vDash_{\text{tot}} \{p\}\, S\, \{q\}$
  iff $\sigma \vDash p$ implies $M(S,\sigma) \vDash q$
  iff $\sigma \vDash \neg p$ or $M(S,\sigma) \vDash q$
  iff $\sigma \vDash \neg p$ or $\tau \vDash q$ for every member $\tau \in M(S,\sigma)$
- Under total correctness, if S is deterministic, then $M(S,\sigma) = \{\tau\}$ for some τ, with $\tau \neq \bot$ and $\tau \vDash q$. If S is nondeterministic, we can have multiple $\tau \in M(S,\sigma)$ and none of them can be $\bot$ [Mon 2023-02-06, 14:52] and all of them satisfy q.
- For partial correctness, if $\sigma \neq \bot$, then

  $\sigma \vDash \{p\}\, S\, \{q\}$
  iff $\sigma \vDash p$ implies $M(S,\sigma) - \bot \vDash q$
  iff $\sigma \vDash \neg p$ or $M(S,\sigma) - \bot \vDash q$
  iff $\sigma \vDash \neg p$ or for every $\tau \in M(S,\sigma)$, either $\tau = \bot$ or $\tau \vDash q$.
- Under partial correctness, if S is deterministic, then $M(S,\sigma) = \{\tau\}$ for some τ, and either $\tau = \bot$ or $\tau \vDash q$. If S is nondeterministic, we can have multiple $\tau \in M(S,\sigma)$ and all of them either are some version of $\bot$ or satisfy q.

K. Unsatisfied Correctness Triples

- It's useful to figure out when a state ***doesn't 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 $\sigma \neq \bot$ 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 $\sigma \vDash_{\text{tot}} \{p\}\, S\, \{q\}$ iff $\sigma \vDash \neg p$ or $M(S,\sigma) \vDash q$
  - So $\sigma \not\vDash_{\text{tot}} \{p\}\, S\, \{q\}$ iff $\sigma \vDash p$ and $M(S,\sigma) \not\vDash q$
    iff $\sigma \vDash p$ and $(\bot \in M(S,\sigma)$ or $\tau \not\vDash q$ for some $\tau \in M(S,\sigma))$.
  - (Recall if $\tau \neq \bot$ then $\tau \not\vDash q$ iff $\tau \vDash \neg q$.)

- So breaking down the cases, $\sigma \vDash_{tot} \{p\}\, S\, \{q\}$ means
  - If S is deterministic, then $\sigma \vDash p$ and $M(S, \sigma) = \{\tau\}$ where $\tau = \bot$ or $\tau \vDash \neg q$.
  - If S is nondeterministic, then $\sigma \vDash p$ and $(\bot \in M(S, \sigma)$ or $\tau \vDash \neg q$ for some $\tau \in M(S, \sigma))$.
- Note for nondeterministic S, having $\sigma \nvDash_{tot} \{p\}\, S\, \{q\}$ only says that one $\tau \in M(S, \sigma)$ is $\bot$ or satisfies $\neg q$. This doesn't preclude $M(S, \sigma)$ from having states that satisfy q.

Unsatisfied Partial Correctness

- For a state to not satisfy $\{p\}\, S\, \{q\}$ under partial correctness, either the state is $\bot$ or, it satisfies p and running S in it always terminates in a state satisfying $\neg q$.
  - We have $\sigma \vDash \{p\}\, S\, \{q\}$ iff $\sigma \vDash \neg p$ or $M(S, \sigma) - \bot \vDash q$
  - So $\sigma \nvDash \{p\}\, S\, \{q\}$ iff $\sigma \vDash p$ and $M(S, \sigma) - \bot \nvDash q$
    iff $\sigma \vDash p$ and $\tau \vDash \neg q$ for some $\tau \neq \bot$ in $M(S, \sigma)$.
  - For deterministic S, there's only one τ in $M(S, \sigma)$ and (it must be $\neq \bot$ and) satisfy $\neg q$.
  - For nondeterministic S, we need one $\tau \in M(S, \sigma)$, ($\tau \neq \bot$ and) $\tau \vDash \neg q$.
    - The other $\tau \in M(S, \sigma)$ can be $\bot$ or satisfy q.
    - I.e., at least one path $\langle S, \sigma\rangle \rightarrow^* \langle E, \tau\rangle$ with $\tau \vDash \neg q$, but there can be paths $\langle S, \sigma\rangle \rightarrow^*$ $\langle E, \bot\rangle$ or $\langle S, \sigma\rangle \rightarrow^* \langle E, \tau\rangle$ with $\tau \vDash q$.

L. Three Extreme (Mostly Trivial) Cases

- There are three edge cases where partial correctness occurs for uninformative reasons.. First recall the definition of partial correctness: $\sigma \vDash \{p\}\, S\, \{q\}$ means (if $\sigma \vDash p$, then $M(S, \sigma) - \bot \vDash q$).
  - ***p is a contradiction*** (i.e., $\vDash \neg p$). Since $\sigma \vDash p$ never holds, $M(S, \sigma) - \bot \vDash q$ is irrelevant and partial correctness of $\{p\}\, S\, \{q\}$ always holds. So for example, $\{F\}\, S\, \{q\}$ is valid under partial correctness, for all S and q. (Even $\{F\}\, S\, \{F\}$ and $\{F\}\, S\, \{T\}$.)
  - ***S always fails to terminate***[3]. If $M(S, \sigma) = \{\bot\}$ then $M(S, \sigma) - \bot = \varnothing$, which satisfies q, so we get partial correctness of $\{p\}\, S\, \{q\}$.
  - ***q is a tautology*** (i.e., $\vDash q$). Then for any σ, $M(S, \sigma) - \bot \vDash q$, so ($\sigma \vDash p$ implies $M(S, \sigma) - \bot \vDash q$) is true (so p is irrelevant) and we get partial correctness of $\{p\}\, S\, \{q\}$. So for example, $\{p\}\, S\, \{T\}$ is valid under partial correctness for all p and S. (Even $\{F\}\, S\, \{T\}$.)
- For total correctness, recall $\sigma \vDash_{tot} \{p\}\, S\, \{q\}$ means (if $\sigma \vDash p$, then $M(S, \sigma) \vDash q$). Note $\bot \notin M(S, \sigma)$ because $\bot \notin M(S, \sigma)$ implies $M(S, \sigma) \nvDash q$)
  - ***p is a contradiction***. The argument here is the same as for partial correctness, so for all S and q, we have $\vDash_{tot} \{F\}\, S\, \{q\}$.
  - ***S always fails to terminate***. Since $M(S, \sigma) = \{\bot\}$, we know $M(S, \sigma) \nvDash q$. So total correctness of $\{p\}\, S\, \{q\}$ always fails. I.e., $\sigma \nvDash_{tot} \{T\}\, S\, \{q\}$ for all σ. [2023-02-07]

---

[3] Remember, just "terminate" implicitly includes "without error". "Not terminate" means "Diverges or gets a runtime error".

- ***q is a tautology***. This case is actually useful. Since $M(S, \sigma) \vDash T$ implies $\bot \notin M(S, \sigma)$, satisfaction of $\sigma \vDash_{tot} \{p\}\ S\ \{T\}$ requires S ***to always terminate*** under σ. So validity of $\vDash_{tot} \{p\}\ S\ \{T\}$ happens exactly when S always terminates when started in a state satisfying p.

- ***Lemma:*** $\sigma \vDash_{tot} \{p\}\ S\ \{q\}$ iff $\sigma \vDash \{p\}\ S\ \{q\}$ and $\sigma \vDash_{tot} \{p\}\ S\ \{T\}$.
  - This just says that total correctness is partial correctness plus termination.
  - Partial correctness says that $\langle S, \sigma \rangle \rightarrow^*$ to a final state that $\vDash q$ or is $\bot$). Termination says every $\langle S, \sigma \rangle \rightarrow^*$ to a final state that satisfies true (and thus $\neq \bot$)). So we have total correctness: Every $\langle S, \sigma \rangle \rightarrow^*$ to a final state that $\vDash q$.