Correctness ("Hoare") Triples, pt. 1

CS 536: Science of Programming, Fall 2019

9/16

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 `*/`)

  ⇒ *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 $\texttt{x} < 0$ because S calculates $\texttt{sqrt(x)}$
  - And say the triple is $\{\texttt{x} \geq 0\}\ \texttt{y := sqrt(x)}\ \{\texttt{y}^2 \leq \texttt{x} < (\texttt{y+1})^2\}$
  - You can't say this program has a bug when you start in a state with $\texttt{x} < 0$, even though the program fails, because you ran the program on bad input.
- Analogous to $\sigma \vDash p$ and $\vDash p$ for satisfaction and validity of predicates, we'll use the notations $\sigma \vDash \{p\}\ S\ \{q\}$ and $\vDash \{p\}\ S\ \{q\}$ 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**: $\vDash$ {x > 0} x := x+1 {x > 0}. This is satisfied in all states, so the triple is valid.
- **Example 2**: $\nvDash$ {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} $\nvDash$ {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":
    $\vDash$ {x > 1} x := x−1 {x > 0} or $\vDash$ {x−1 > 0} x := x−1 {x > 0}
  - **Example 4**: Make the postcondition "**weaker**" = "less restrictive":
    $\vDash$ {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**: $\vDash$ {(x = 2*k ∨ x = 2*k+1) ∧ x ≥ 0} x := x/2 {x = k ≥ 0}
  (If x is nonnegative and equals 2*k or 2*k+1 before dividing x by 2 then after the division, x equals k, which is nonnegative.)
- **Example 7**: $\vDash$ {s = 1 + 2 + … + k} s := s+k+1**;** k := k+1 {s = 1 + 2 + … + 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**: $\vDash$ {s = 1 + 2 + … + k} k := k+1**;** s := s+k {s = 1 + 2 + … + 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**:
  $\vDash$ {s = 1 + 2 + … + k}
  k := k+1;
  s := s+k+1
  {s = 1 + 2 + … + (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**: $\vDash$ {x = c_0 ≥ 0} x := x/2 {c_0 ≥ 0 ∧ x = c_0/2}
  (If x is ≥ 0, then after the assignment x := x/2, the old value of x (call it c_0) was ≥ 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.)

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 $\Sigma_\perp = \Sigma \cup \{\perp\}$, so $\sigma \in \Sigma_\perp$ 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**: $\Sigma_0 - \perp$ and $\Sigma_0 \cap \Sigma$ both mean Σ_0 less $\perp$: $\Sigma_0 - \perp = \Sigma_0 \cap \Sigma = \{\sigma \in \Sigma_0 \mid \sigma \in \Sigma\} = \{\sigma \in \Sigma_0 \mid \sigma \neq \perp\}$.

- **Definition**: Let $\Sigma_0 \subseteq \Sigma_\perp$. We say Σ_0 **satisfies** p if it is nonempty[1] and every element of Σ_0 satisfies p. In symbols, $\Sigma_0 \vDash p$ iff $\Sigma_0 \neq \varnothing$ and for all $\tau \in \Sigma_0$, $\tau \vDash p$.
- Some consequences of the definition:
  - Since $\perp$ satisfies no predicate, if $\perp \in \Sigma_0$, then $\Sigma_0 \nvDash p$ and $\Sigma_0 \nvDash \neg p$.
  - If $\Sigma_0 \subseteq \Sigma$, we have ($\Sigma_0 \vDash p$ implies $\Sigma_0 \nvDash \neg p$) and ($\Sigma_0 \vDash \neg p$ implies $\Sigma_0 \nvDash p$).
  - The converses ($\Sigma_0 \nvDash \neg p$ implies $\Sigma_0 \vDash p$) and ($\Sigma_0 \nvDash p$ implies $\Sigma_0 \vDash \neg p$) hold if Σ_0 is a singleton set $\subseteq \Sigma$.
    - If $\tau \neq \perp$, then $\tau \vDash p$ iff $\tau \nvDash \neg p$, so if $\Sigma_0 = \{\tau\} \nsubseteq \{\perp\}$, then either ($\Sigma_0 \vDash p$ and $\Sigma_0 \nvDash \neg p$) or ($\Sigma_0 \vDash \neg p$ and $\Sigma_0 \nvDash p$).
  - If $\Sigma_0 \subseteq \Sigma$ and Σ_0 contains more than one state, then it's possible for to have $\Sigma_0 \nvDash p$ and $\Sigma_0 \nvDash \neg p$.
    - If there are τ and $\tau' \in \Sigma_0$ with $\tau \vDash p$ and $\tau' \vDash \neg p$, then $\Sigma_0 \nvDash p$ and $\Sigma_0 \nvDash \neg p$.
    - The converse also holds if Σ_0 does not include $\perp$.
- Since validity means "satisfied in all states", we have $\vDash p$ iff $\Sigma_0 \vDash p$ where $\Sigma_0 = \{\sigma \in \Sigma \mid \sigma \vDash p\}$. Since states that don't $\vDash p$ trivially satisfy $\{p\}$ S $\{q\}$, we get $\Sigma \vDash \{p\}\ S\ \{q\}$.

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** 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 states satisfying q.[2]
- In symbols, $\sigma \vDash_{tot} \{p\}\ S\ \{q\}$ iff $\sigma \in \Sigma$ and (if $\sigma \vDash p$ then $M(S, \sigma) \subseteq \Sigma$ and $M(S, \sigma) \vDash q$).
  - $M(S, \sigma) \subseteq \Sigma$ iff running S in σ always terminates in a state because $M(S, \sigma) \subseteq \Sigma$ iff $\perp \notin M(S, \sigma)$.
  - $M(S, \sigma) \vDash q$ iff $M(S, \sigma) \neq \varnothing$ and for every $\tau \in \Sigma_\perp$, if $\tau \in M(S, \sigma)$, then $\tau \vDash q$.
  - Since $\perp \nvDash q$, we know $M(S, \sigma) \vDash q$ implies $\perp \notin M(S, \sigma)$, which again implies that running S in σ always terminates.
- For total correctness, we can't allow $\sigma = \perp$ because $\perp \nvDash p$ and $M(S, \perp) = \{\perp\} \nvDash q$, so ($\sigma \vDash p$ implies $M(S, \sigma) \vDash q$) would reduce to (false implies false), which is true.
- **Definition**: The triple $\{p\}\ S\ \{q\}$ is **totally correct** (is **valid** under **total correctness**) iff $\sigma \vDash_{tot} \{p\}\ S\ \{q\}$ for all σ. The notation is $\vDash_{tot} \{p\}\ S\ \{q\}$. (And again, $\vDash_{tot} \{p\}\ S\ \{q\}$ means $\Sigma \vDash_{tot} \{p\}\ S\ \{q\}$.)

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.

---

[1] If we allowed $\Sigma_0 = \varnothing$ then we would have $\varnothing \vDash p$ and $\varnothing \vDash \neg p$, which just doesn't sound right :-)

[2] The sense of “implies” or "if… then…" used here is not like → (which appears in predicates) or ⇒ (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.

- **Definition**: The triple $\{p\}\ S\ \{q\}$ is **partially correct in** σ or σ satisfies the triple under **partial correctness** iff it's the case that if σ satisfies p, then whenever running S in σ converges to a memory state, that state satisfies q.
- In symbols, $\sigma \vDash \{p\}\ S\ \{q\}$ iff $\sigma \in \Sigma$ and ($\sigma \vDash p$ implies (for every $\tau \in M(S, \sigma)$, if $\tau \in \Sigma$, then $\tau \vDash q$)).
- Equivalently, $\sigma \vDash \{p\}\ S\ \{q\}$ iff $\sigma \in \Sigma$ and ($\sigma \vDash p$ implies ($M(S, \sigma) = \{\bot\}$ or $M(S, \sigma) - \bot \vDash q$)).
  - If running S in σ never terminates (i.e., if $M(S, \sigma)$ only contains flavors of ⊥), then it's vacuously true that (for every $\tau \in M(S, \sigma)$, if $\tau \in \Sigma$, then $\tau \vDash q$)). (Or equivalently, there does not exist a $\tau \in M(S, \sigma)$ with $\tau \in \Sigma$ and $\tau \nvDash q$.)
  - Note we need to include $M(S, \sigma) = \{\bot\}$ because $M(S, \sigma) - \bot \vDash q$ doesn't hold if $M(S, \sigma) = \{\bot\}$.
- As with total correctness, we can't allow $\sigma = \bot$ for partial correctness because $\bot \nvDash p$ and $M(S, \bot) = \{\bot\}$, so ($\sigma \vDash p$ implies ($M(S, \sigma) \subseteq \{\bot\}$ or $M(S, \sigma) - \bot \vDash q$) would reduce to (false implies false or $\varnothing \vDash q$), 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 $\sigma \vDash \{p\}\ S\ \{q\}$ for all σ. The notation is $\vDash \{p\}\ S\ \{q\}$. (And $\vDash \{p\}\ S\ \{q\}$ iff $\Sigma \vDash \{p\}\ S\ \{q\}$.)

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 ($\sigma \vDash \neg p$ iff $\sigma \nvDash p$) and ($\sigma \vDash p$ iff $\sigma \nvDash \neg p$).
- For total correctness

  $\sigma \vDash_{tot} \{p\}\ S\ \{q\}$

  iff $\sigma \neq \bot$ and ($\sigma \vDash p$ implies $M(S, \sigma) \vDash q$)

  iff $\sigma \neq \bot$ and ($\sigma \vDash \neg p$ or $M(S, \sigma) \vDash q$)
  - If $\sigma \neq \bot$, then $\sigma \vDash_{tot} \{p\}\ S\ \{q\}$ iff ($\sigma \vDash \neg p$ or $M(S, \sigma) \vDash q$).
- For partial correctness,

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

  iff $\sigma \neq \bot$ and ($\sigma \vDash p$ implies (for every $\tau \in M(S, \sigma)$, $\tau = \bot$ or $\tau \vDash q$))

  iff $\sigma \neq \bot$ and ($\sigma \vDash \neg p$ or (for every $\tau \in M(S, \sigma)$, $\tau = \bot$ or $\tau \vDash q$))

  iff $\sigma \neq \bot$ and ($\sigma \vDash \neg p$ or ($M(S, \sigma) - \bot \vDash q$))
  - If $\sigma \neq \bot$, then $\sigma \vDash \{p\}\ S\ \{q\}$ iff ($\sigma \vDash \neg p$ or $M(S, \sigma) - \bot$ is $\varnothing$ or $\vDash q$))

J. 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 $\sigma \in \Sigma$ 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. In symbols,

$\sigma \vDash_{tot} \{p\}\ S\ \{q\}$

iff $\sigma \neq \perp$ and ($\sigma \vDash \neg p$ or $M(S, \sigma) \vDash q$)

iff $\sigma \neq \perp$ and ($\sigma \neq \perp$ implies ($\sigma \vDash \neg p$ or $M(S, \sigma) \vDash q$))

$\sigma \nvDash_{tot} \{p\}\ S\ \{q\}$

iff $\sigma = \perp$ or not ($\sigma \neq \perp$ implies ($\sigma \vDash \neg p$ or $M(S, \sigma) \vDash q$))

iff $\sigma = \perp$ or ($\sigma \neq \perp$ and $\sigma \nvDash \neg p$ and $M(S, \sigma) \nvDash q$))

iff $\sigma = \perp$ or ($\sigma \neq \perp$ and $\sigma \vDash p$ and $M(S, \sigma) \nvDash q$))

iff $\sigma = \perp$ or ($\sigma \neq \perp$ and $\sigma \vDash p$ and ($\perp \in M(S, \sigma)$ or for some $\tau \in M(S, \sigma)$, $\tau \vDash \neg q$)

- If S is deterministic, then $M(S, \sigma)$ has just one member, so for $\sigma \neq \perp$, we have $\sigma \nvDash_{tot} \{p\}\ S\ \{q\}$ iff $\sigma \vDash p$ and ($M(S, \sigma) = \{\perp\}$ or $M(S, \sigma) = \{\tau\} \subseteq \Sigma$ with $\tau \vDash \neg q$). In English, σ satisfies p and running S in σ either doesn't terminate or it terminates in a state in which q is false.

Unsatisfied Partial Correctness

- 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

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

  iff $\sigma \neq \perp$ and ($\sigma \vDash p$ implies ($M(S, \sigma) \subseteq \{\perp\}$ or $M(S, \sigma) - \perp \vDash q$)

  iff $\sigma \neq \perp$ and ($\sigma \nvDash p$ or $M(S, \sigma) \subseteq \{\perp\}$ or $M(S, \sigma) - \perp \vDash q$)

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

  iff $\sigma = \perp$ or ($\sigma \vDash p$ and $M(S, \sigma) \nsubseteq \{\perp\}$ and $M(S, \sigma) - \perp \nvDash q$)

  iff $\sigma = \perp$ or ($\sigma \vDash p$ and $M(S, \sigma) - \perp \neq \emptyset$ and $M(S, \sigma) - \perp \nvDash q$)

  iff $\sigma = \perp$ or ($\sigma \vDash p$ and there is a $\tau \in M(S, \sigma)$ where $\tau \neq \perp$ and $\tau \nvDash q$)

  iff $\sigma = \perp$ or ($\sigma \vDash p$ and for there is a $\tau \in M(S, \sigma)$ where $\tau \vDash \neg q$)

  - From this last line, if S is deterministic, then $M(S, \sigma) = \{\tau\} \vDash \neg q$.
  - On the other hand, if S is nondeterministic, though, we can't conclude $M(S, \sigma) \vDash \neg q$. Though there is a $\tau \vDash \neg q$, it's still possible for $M(S, \sigma) - \tau$ to be nonempty and contain a state that satisfies q.
- For this last transition, since $M(S, \sigma) \neq \emptyset$, to get $M(S, \sigma) \nsubseteq \{\perp\}$, we must have $M(S, \sigma) - \perp \neq \emptyset$. Since $M(S, \sigma) - \perp \nvDash q$, there must be a state τ in $M(S, \sigma) - \perp$ that doesn't satisfy q. Since τ isn't $\perp$, we know $\tau \nvDash q$ iff $\tau \vDash \neg q$.
- If S is deterministic, then $M(S, \sigma)$ has just one member, so for $\sigma \neq \perp$, we have $\sigma \nvDash \{p\}\ S\ \{q\}$ iff $\sigma \vDash \neg p$ and $M(S, \sigma) = \{\tau\} \subseteq \Sigma$ and $\tau \vDash \neg q$. In English, σ satisfies p and running S in σ terminates in a state in which q is false.

Correctness ("Hoare") Triples, pt. 1

CS 536: Science of Programming, Fall 2019

A. Why

- To specify a program's correctness, we need to know its precondition (what must be true before executing it) and its postcondition (what should be true after it).

B. Objectives

At the end of this activity you should be able to

- Recognize syntactically correct correctness triples.
- Say whether a correctness triple is satisfied, given information about whether the current state satisfies the precondition, whether the statement terminates, and if it does, whether the terminating state satisfies the postcondition.

C. Questions

For all the questions below, you can assume (unless otherwise said) that $\sigma \in \Sigma$, not $\Sigma_\perp$. (I.e., we're not trying to start our program after an infinite loop or failure.)

1. For a loop-free program (a program that doesn't include any loops) without runtime errors, is there any difference between partial and total correctness?
2. Say we're given $\sigma \vDash \{\mathtt{x} > 0\}\ S\ \{\mathtt{y} > \mathtt{x}\}$ for all σ and we're given a state τ where $\tau(\mathtt{x}) = -3$. Do we know what S will do if we run in τ? Must it terminate? (With or without a runtime error?) Diverge? Must $\mathtt{y} > \mathtt{x}$ afterwards? How about $\mathtt{y} \le \mathtt{x}$?
3. For which σ does $\sigma \vDash \{\mathtt{x} > 1\}\ \mathtt{y} := \mathtt{x*x}\ \{\mathtt{y} > \mathtt{x}\}$ hold? Is this triple valid?
4. For which σ does $\sigma \vDash \{\mathtt{x} > 0\}\ \mathtt{y} := \mathtt{x*x}\ \{\mathtt{y} > \mathtt{x}\}$ hold? Is this triple valid?
5. Under partial correctness, does $\sigma \vDash \{\mathtt{F}\}\ S\ \{q\}$ hold for all σ, q, and S? What about $\sigma \vDash \{p\}\ S\ \{\mathtt{T}\}$? Do these triples say anything interesting about S?
6. Repeat the previous question under total correctness: Does $\sigma \vDash_{tot} \{\mathtt{F}\}\ S\ \{q\}$ always hold? Does $\sigma \vDash_{tot} \{p\}\ S\ \{\mathtt{T}\}$? Do these triples say anything interesting about S?

For Questions 7 – 12, specify for each statement whether it's true or false and give a brief explanation. (Just a sentence or two is fine.) Assume $\sigma \in \Sigma$. (And remember, $\sigma \vDash$ *anything* implies $\sigma \neq \perp$.)

7. If $\sigma \vDash \{p\}\ S\ \{q\}$, then $\sigma \vDash p$.
8. If $\sigma \nvDash \{p\}\ S\ \{q\}$, then $\sigma \nvDash p$.
9. If $M(S, \sigma) \subseteq \{\perp_d, \perp_e\}$, then $\sigma \vDash \{p\}\ S\ \{q\}$.
10. If $\sigma \vDash p$ and $M(S, \sigma) \cap \{\perp_d, \perp_e\} \neq \varnothing$, then $\sigma \nvDash_{tot} \{p\}\ S\ \{q\}$.
11. If $\sigma \vDash \{p\}\ S\ \{q\}$ and $\sigma \vDash p$, then every state in $M(S, \sigma)$ either $\in \{\perp_d, \perp_e\}$ or satisfies q.
12. If $\sigma \vDash \{p\}\ S\ \{q\}$ and $\sigma \nvDash p$, then every state in $M(S, \sigma)$ is either $\in \{\perp_d, \perp_e\}$ or satisfies $\neg q$.

Solution to Activity 8 (Hoare Triples, pt 1)

1. No: For a loop-free, failure-free program, there's no difference between partial and total correctness.
2. No to all the questions: The triple only tells us what will happen if the precondition is satisfied. Since $\tau \not\models \mathtt{x} > \mathtt{0}$, the triple doesn't say anything about what will happen when you run S; it might cause an error or terminate in a state, and that state might satisfy $\mathtt{y} > \mathtt{x}$, but it might not.
3. All states satisfy the triple, so the triple is valid.
4. States in which x = 1 do not satisfy the triple; states in which x > 1 set y appropriately and do satisfy the triple. States in which x < 1 satisfy the triple trivially.
5. Under partial correctness, for all S, $\{\mathtt{F}\}\ S\ \{q\}$ and $\{p\}\ S\ \{\mathtt{T}\}$ are valid (satisfied in all states), but neither triple says anything useful about the program S.
6. Under total correctness, $\{\mathtt{F}\}\ S\ \{q\}$ is again valid and doesn't say anything useful about S. Under total correctness, however, $\sigma_{tot} \models \{p\}\ S\ \{\mathtt{T}\}$ if and only if S always terminates when run it in σ. (I.e., it never goes into an infinite loop or fails at runtime.)
7. False; $\sigma \models \{p\}\ S\ \{q\}$ does not imply $\sigma \models p$. (It doesn't imply $\sigma \not\models p$ either.)
8. False; if $\sigma \in \Sigma$ and $\sigma \not\models \{p\}\ S\ \{q\}$, then $\sigma \models p$ (and $M(S, \sigma) \cap \Sigma \models \neg q$).
9. True; under partial correctness, if S always causes an error when run in a σ that satisfies p, then $\sigma \models \{p\}\ S\ \{q\}$.
10. True: If $\sigma \models p$, then for $\sigma \models_{tot} \{p\}\ S\ \{q\}$ to hold, we need $M(S, \sigma) \models q$. If $M(S, \sigma) \cap \{\bot_d, \bot_e\} \neq \varnothing$, then $M(S, \sigma) \not\models q$, so $\sigma \not\models_{tot} \{p\}\ S\ \{q\}$.
11. True; if $\{p\}\ S\ \{q\}$ is partially correct and we run S in a state satisfying p, then either S causes an error or terminates in a state satisfying q.
12. False; if a triple is satisfied in σ but σ doesn't satisfy the precondition, then all possibilities can happen: S might diverge, it might cause a runtime error, and even if it terminates, the final state might satisfy q but it doesn't have to.

