Correctness ("Hoare") Triples

Part 1: Definitions and Basic Properties

CS 536: Science of Programming, Spring 2021

2/25 p.5

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 $x < 0$ 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., $\{x > 0\} \text{ if } x > 1 \text{ then } x := x-1 \text{ fi } \ {x > 0}$
• **Example 6**: $\models \{(x = 2*k \lor x = 2*k+1) \land x \geq 0\} x := x/2 \ {x = k \geq 0}$
  (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 $s$ is 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**: $\models \{s = 1+2+\ldots+k\} k := k+1; \ s := s+k \ {s = 1+2+\ldots+k}$
    (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:** \( \models \{ x = c_0 \geq 0 \} \ x := x/2 \ \{ c_0 \geq 0 \land x = c_0 / 2 \} \)

(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 \( x \) 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 \( \Sigma_{\perp} = \Sigma \cup \{ \perp \} \), where \( \Sigma \) 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 \( \Sigma_0 \), if \( \Sigma_0 \subset \Sigma_{\perp} \), then we may have \( \perp \in \Sigma_0 \).
  
  - On the other hand, if \( \Sigma_0 \subset \Sigma \), then \( \perp \notin \Sigma_0 \).

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

- **Definition:** Let \( \Sigma_0 \subset \Sigma_{\perp} \). We say \( \Sigma_0 \) **satisfies** \( p \) if every element of \( \Sigma_0 \) satisfies \( p \). In symbols, \( \Sigma_0 \models p \) iff for all \( \tau \in \Sigma_0 \), \( \tau \models p \). (Note \( \emptyset \models p \), since there exists no \( \tau \in \emptyset \) where \( \tau \models \neg p \).)

- Some consequences of the definition:
  - If \( \perp \in \Sigma_0 \), then \( \Sigma_0 \not\models p \) and \( \Sigma_0 \not\models \neg p \).
  
  - \((\Sigma_0 \models p \land \Sigma_0 \models \neg p) \iff \Sigma_0 = \emptyset \).
  
  - Since \( \perp \not\models p \) (and \( \perp \not\models \neg p \)), we have \( \perp \notin \Sigma_0 \). If \( \tau \not\models \perp \) and \( \tau \models p \) then \( \tau \not\models \neg p \), so \( \tau \notin \Sigma_0 \). So \( \Sigma_0 = \emptyset \).
  
  - If \( \Sigma_0 \) has size \( \geq 2 \) and \( \perp \notin \Sigma_0 \), then \( \Sigma_0 \not\models \neg p \iff \Sigma_0 \models p \).
  
  - 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 \( \Sigma_0 \) has size \( \geq 2 \) and \( \perp \notin \Sigma_0 \), then it is **not** the case that \( \Sigma_0 \not\models p \) iff \( \Sigma_0 \models \neg p \).
  
  - (\( \leftrightarrow \)) If \( \tau \models \neg p \) then \( \tau \not\models p \), so if \( \tau \in \Sigma_0 \), then \( \Sigma_0 \models \neg p \).
  
  - (\( \implies \)) If \( \perp \notin \{ \tau, \tau' \} \subset \Sigma_0 \) where \( \tau \models p \) and \( \tau' \models \neg p \), then \( \tau \not\models \neg p \) (so \( \tau \in \Sigma_0 \) implies \( \Sigma_0 \not\models \neg p \)) and \( \tau' \not\models p \) (so \( \tau' \in \Sigma_0 \) implies \( \Sigma_0 \models p \)). So we have \( \Sigma_0 \not\models p \) and \( \Sigma_0 \not\models \neg p \) 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 \( q \).

---

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, \( \sigma \models_{\text{tot}} \{ p \} \ S \ \{ q \} \) iff \( \sigma \neq \bot \) and (if \( \sigma \models p \) then \( \bot \notin M(S, \sigma) \) and \( M(S, \sigma) \models q \)).
  - 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 \( \bot \notin p \) and \( M(S, \bot) = \{ \bot \} \neq q \), so \( \sigma \models p \) implies \( M(S, \sigma) \models q \) would reduce to (false implies false), which is true.

• Note for \( \sigma \models_{\text{tot}} \{ p \} \ S \ \{ q \} \) we specifically require \( \sigma \neq \bot \) because \( \bot \notin p \), so without banning \( \bot \) explicitly, we'd have \( (\sigma \models p \Rightarrow \ldots) \) turn into (false implies \( \ldots \)), which is true.

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

  - **Notation:** We also write \( \models_{\text{tot}} \{ p \} \ S \ \{ q \} \) 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** (**valid** 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 \( \sigma \neq \bot \) and (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 for all \( \tau \in \Sigma \) and (\( \sigma \models p \) implies \( M(S, \sigma) \models \bot \iff q \)).

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

  - **Definition:** The triple \( \{ p \} \ S \ \{ q \} \) is **partially correct** (**valid** under/for **partial correctness**) iff \( \Sigma \models \{ p \} \ S \ \{ q \} \). I.e, \( \sigma \models \{ p \} \ S \ \{ q \} \) for all states \( \sigma \). **Notation:** We also write \( \models \{ p \} \ S \ \{ q \} \) to mean that the triple is partially correct.

### 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, if \( \sigma \neq \bot \), then
  - \( \sigma \models_{\text{tot}} \{ p \} \ S \ \{ q \} \)
    - iff \( \sigma \models p \) implies \( M(S, \sigma) \models q \)
    - iff \( \sigma \models \neg p \) or \( M(S, \sigma) \models q \)
    - iff \( \sigma \models \neg p \) 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
  - \( \sigma \models \{ p \} \ S \ \{ q \} \)
    - iff \( \sigma \models p \) implies \( M(S, \sigma) - \bot \models q \)
iff \( \sigma \models \neg p \) or \( M(S, \sigma) \cdot \bot \models q \)
iff \( \sigma \models \neg p \) or for every \( \tau \in M(S, \sigma) \), either \( \tau = \bot \) or \( \tau \models q \).

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

### 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 \( \sigma \neq \bot \) 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 \( q \). [2/25]

  - We have \( \sigma \models_{\text{tot}} \{p\} S \{q\} \) iff \( \sigma \models \neg p \) or \( M(S, \sigma) \models q \)
  - So \( \sigma \not\models_{\text{tot}} \{p\} S \{q\} \) iff \( \sigma \models p \) and \( M(S, \sigma) \not\models q \)
    
    \[
    \sigma \models p \text{ and } (\bot \in M(S, \sigma) \text{ or for some } \tau \in M(S, \sigma), \tau \not\models \bot \text{ and } \tau \not\models q) \text{ (i.e., } \tau \not\models \neg q \text{, since } \tau \not\models \bot). \]

- If \( S \) is deterministic, then \( \sigma \models p \) and \( M(S, \sigma) = \{\tau\} \) where \( \tau = \bot \) or \( \tau \models \neg q \).

- If \( S \) is nondeterministic, then \( \sigma \not\models p \) and \( (\bot \in M(S, \sigma) \text{ or for some } \tau \in M(S, \sigma), \tau \models \neg q) \).

  - Another characterization: \( \sigma \models p \) and if \( \bot \not\in M(S, \sigma) \), then for some \( \tau \in M(S, \sigma), \tau \models \neg q \).

#### Unsatisfied Partial Correctness

- For a state \( \sigma \neq \bot \) 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 know \( M(S, \sigma) \cdot \bot \models q \) holds iff for every \( \tau \in M(S, \sigma) \), if \( \tau \not\models \bot \), then \( \tau \models q \)

  - So \( M(S, \sigma) \cdot \bot \models q \) holds iff for some \( \tau \in M(S, \sigma) \), we have \( \tau \models \neg q \) (since \( \tau \not\models q \) with \( \tau \not\models \bot \))

  - 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 \( \bot \) 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 \( M(S, \sigma) \cdot \bot \models q \)).
• **p is a contradiction** (i.e., ⊨ ¬p). Since σ ⊨ p never holds, the implication (if σ ⊨ 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, σ) = {⊥} then M(S, σ) – ⊥ = ∅, and ∅ ⊨ q, so again we get partial correctness of {p} S {q}.

• **q is a tautology** (i.e., ⊨ q). Then for any σ, M(S, σ) – ⊥ ⊨ q, so whether σ 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 σ ⊨tot {p} S {q} means (if σ ⊨ p, then M(S, σ) ⊨ q). (Also, recall that since ⊥ ⊭ q, if M(S, σ) ⊨ q, then ⊥ ∉ M(S, σ).)

• **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, σ) = {⊥}, we know M(S, σ) ⊭ q. So total correctness of {p} S {q} always fails.

• **q is a tautology.** σ ⊨tot {p} S {T} does says something interesting. Since M(S, σ) ⊨ T implies ⊥ ∉ M(S, σ), satisfaction of σ ⊨tot {p} S {T} requires S to **always terminate** under σ. So validity of ⊨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 σ ⊨tot {p} S {q} iff σ ⊨ {p} S {q} and σ ⊨tot {p} S {T}. Again, this means that to show that a triple is totally correct, we can prove partial correctness and termination separately.
Correctness ("Hoare") Triples, pt. 1

CS 536: Science of Programming, Spring 2021

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 practice 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 run a program after an infinite loop or runtime failure.)

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

For Problems 7 – 14, say for each statement whether it's true or false and give a brief explanation. (Just a sentence or two is fine.) Assume \( \sigma \in \Sigma \). (Remember, if \( \sigma \models \) any predicate or triple, then \( \sigma \neq \perp \).)

7. If \( \sigma \models \{ p \} S \{ q \} \), then \( \sigma \models p \).
8. If \( \sigma \not\models \{ p \} S \{ q \} \), then \( \sigma \not\models p \).
9. If \( M(S, \sigma) \cap \{ \bot_d, \bot_e \} \), then \( \sigma \models \{ p \} S \{ q \} \).
10. If $\sigma \models p$ and $M(S, \sigma) \cap \{\bot_d, \bot_e\} \neq \emptyset$, then $\sigma \not\models_{\text{tot}} \{p\} \land \{q\}$.

11. If $\sigma \models \{p\} \land \{q\}$ and $\sigma \models p$, then every state in $M(S, \sigma)$ either $\in \{\bot_d, \bot_e\}$ or satisfies $q$.

12. If $\sigma \models \{p\} \land \{q\}$ and $\sigma \not\models p$, then every state in $M(S, \sigma)$ is either $\in \{\bot_d, \bot_e\}$ or satisfies $\neg q$.

13. For nondeterministic $S$, if $\sigma \not\models \{p\} \land \{q\}$, then $\tau \models \neg q$ for some $\tau \in M(S, \sigma)$ but it's possible for $\xi \models q$ for some $\xi \in M(S, \sigma)$.

14. For nondeterministic $S$, if $\sigma \not\models_{\text{tot}} \{p\} \land \{q\}$, if $\bot \not\in M(S, \sigma)$, then $\tau \models \neg q$ for some $\tau \in M(S, \sigma)$ but it's possible for $\xi \models q$ for some $\xi \in M(S, \sigma)$.

15. Let $S = x := x \ast x; y := y \ast y$ and let $\sigma(x) = \alpha$ and $\sigma(\xi) = \beta$. Verify that $\sigma \models \{x > y > 0\} \land \{x > y > 0\}$. I.e., assume $\sigma$ satisfies the precondition, calculate $M(S, \sigma)$, and verify that $M(S, \sigma) - \bot$ satisfies the postcondition.
Solution to Practice 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 $\tau \not\equiv x > 0$, 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 $y > x$, 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 $S$, $\{F\} S \{q\}$ and $\{p\} S \{T\}$ are valid (satisfied in all states), but neither triple says anything useful about the program $S$.

6. Under total correctness, $\{F\} S \{q\}$ is again valid and doesn't say anything useful about $S$. Under total correctness, however, $\sigma_{\text{tot}} \models \{p\} S \{T\}$ if and only if $S$ always terminates when run in $\sigma$. (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 $\sigma$ that satisfies $p$, then $\sigma \models \{p\} S \{q\}$.

10. True: If $\sigma \models p$, then for $\sigma \models_{\text{tot}} \{p\} S \{q\}$ to hold, we need $M(S, \sigma) = q$. If $M(S, \sigma) \cap \{ \bot_d, \bot_e \} \neq \emptyset$, then $M(S, \sigma) \not\models q$, so $\sigma \not\models_{\text{tot}} \{p\} S \{q\}$.

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 $\sigma$ but $\sigma$ 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.

13. True; if partial correctness fails, it's because (for some execution path), running $S$ terminates satisfying $\neg q$, but for nondeterministic $S$, it's still possible for $S$ to terminate satisfying $q$.

14. True; if total correctness fails, it's because running $S$ can diverge or (if $S$ always converges) $S$ can terminate satisfying $\neg q$, so for nondeterministic $S$, if $S$ always converges, then for some execution path, it terminates satisfying $\neg q$ but can still terminate (along some other path) satisfying $q$.

15. We're given $S = x := x * x; y := y * y$ and $\sigma(x) = \alpha$ and $\sigma(y) = \beta$. For arbitrary $\sigma$,

$$M(S, \sigma) = M(x := x \cdot x; y := y \cdot y, \sigma)$$

$$= M(y := y \cdot y, M(x := x \cdot x, \sigma))$$

$$= M(y := y \cdot y, \sigma[x \mapsto a^2])$$

$$= \{ \sigma[x \mapsto a^2][y \mapsto \beta^2] \}.$$
Since $\sigma(x) = \alpha$ and $\sigma(y) = \beta$, so $\sigma \models x > y > 0$ implies $\alpha > \beta > 0$, which implies $\alpha^2 > \beta^2 > 0$, which
implies $\sigma[x \mapsto \alpha^2][y \mapsto \beta^2] \models x > y > 0$. Thus $\sigma \models \{x > y > 0\} S \{x > y > 0\}$; i.e., if $\sigma \models x > y > 0$ then
$M(S, \sigma) - \bot \models x > y > 0$.

So if $\sigma \models x > y > 0$, then $M(S, \sigma) - \bot \neq \emptyset$ and $\models x > y > 0$. Therefore, $\sigma \models \{x > y > 0\} S \{x > y > 0\}$. 