Correctness Triples (Continue)

(Weak and Strong)

- If conditions (predicates) $p \Rightarrow q$, then $p$ is **stronger** than $q$ and $q$ is **weaker** than $p$.
  - Technically, it should be "stronger or equal to" and "weaker or equal to". Because if $p \Leftrightarrow q$, then are equally strong.

- A condition is **stronger** if it is "more restricted". For example, we can consider $x > 1$ is stronger than $x > 0$.
  - Making the precondition *stronger* can be helpful when we try to "fix" an invalid triple. For example, $\nvDash \{x > 0\}\ x := x - 1\ \{x > 0\}$ but $\vDash \{x > 1\}\ x := x - 1\ \{x > 0\}$.
  - Also, strengthening the precondition of valid triple will give us another valid triple.

- A condition is **weaker** if it is "less restricted". For example, we can consider $x > -1$ is weaker than $x > 0$.
  - Making the postcondition *weaker* can be helpful when we try to fix an invalid triple. For example, $\nvDash \{x > 0\}\ x := x - 1\ \{x > 0\}$ but $\vDash \{x > 0\}\ x := x - 1\ \{x > -1\}$.
  - Also, weakening the postcondition of valid triple will give us another valid triple.

(Satisfaction by a collection of states)

- In the previous class, we learned that $\sigma \vDash \{p\}\ S\ \{q\}$ means "**if** $\sigma$ satisfies $p$, **then** after running $S$ in $\sigma$ we can get a state $\tau$ who satisfies $q$". What if $M(S, \sigma)$ contains more than one state? What if $M(S, \sigma)$ contains some states that satisfy $q$ and some states don't?

  $\Sigma_\perp = \Sigma \cup \{\perp\}$, where $\Sigma$ is the set of all (well-formed) states.
  - Let $\Sigma_0$ be a subset of $\Sigma_\perp$ (in other word, $\Sigma_0 \subseteq \Sigma_\perp$), then $\Sigma_0 - \perp = \Sigma_0 \cap \Sigma$.
  - Let $\Sigma_0 \subseteq \Sigma_\perp$, then we say $\Sigma_0 \vDash p$, if every state in $\Sigma_0 \vDash p$.

1. True or False.
   a. Let $\sigma \in \Sigma$, then $\sigma \neq \perp$. True
   b. Let $\Sigma_0 \subseteq \Sigma_\perp$, then $\perp \in \Sigma_0$. False, $\perp$ is allowed to be in such $\Sigma_0$, but not necessarily.
   c. Let $\Sigma_0 \vDash p$, then $\perp \notin \Sigma_0$ True, $\perp$ doesn't satisfy any predicate.
   d. $\Sigma_0 \nvDash p \Leftrightarrow \exists \tau \in \Sigma_0 . \tau \nvDash p$ True
   e. If $\perp \in \Sigma_0$, then $\Sigma_0 \vDash \neg p$ False. If $\perp \in \Sigma_0$, then $\Sigma_0 \nvDash p$ and $\Sigma_0 \nvDash \neg p$
   f. $\emptyset \vDash p$ (an empty set of states) True, "every state" in $\emptyset$ satisfies $p$
   g. $\Sigma_0 \vDash p \wedge \Sigma_0 \vDash \neg p \Leftrightarrow \Sigma_0 = \emptyset$ True
   h. Let $\Sigma_0 - \perp = \{\tau\}$, then $\Sigma_0 - \perp \vDash p$ or $\Sigma_0 - \perp \vDash \neg p$ True
   i. $\Sigma_0 - \perp \vDash p$ or $\Sigma_0 - \perp \vDash \neg p$
      False, it is possible that some states in $\Sigma_0 - \perp$ satisfy $p$ and some satisfy $\neg p$, then $\Sigma_0 - \perp \nvDash p$ and $\Sigma_0 - \perp \nvDash \neg p$.

(Total Correctness and Partial Correctness)

- The triple $\{p\}\ S\ \{q\}$ is **totally correct in** $\sigma$ (or $\sigma$ **satisfies the triple under total correctness**), written as $\sigma \vDash_{tot} \{p\}\ S\ \{q\}$, if and only if it is the case that "if $\sigma$ satisfies $p$, then running $S$ in $\sigma$ always terminates (without error) in states satisfying $q$".

- In other words, $\sigma \vDash_{tot} \{p\}\, S\, \{q\} \Leftrightarrow (\sigma(p) \neq \bot) \wedge (\sigma \vDash p \rightarrow M(S,\sigma) \vDash q)$.
  - Here we include $\sigma(p) \neq \bot$ because, we don't want $\sigma \vDash p \rightarrow M(S,\sigma) \vDash q \Leftrightarrow T$ when $\sigma(p) = \bot$. We usually assume that $\sigma(p) \neq \bot$ is a given.
  - $M(S,\sigma) \vDash q$ implies that $\bot \notin M(S,\sigma)$.

- The triple $\{p\}\, S\, \{q\}$ is **totally correct** (or the triple is **valid under total correctness**) if and only if $\sigma \vDash_{tot} \{p\}\, S\, \{q\}$ for all $\sigma \in \Sigma$ (Recall that $\Sigma$ is the set of well-formed states). We write $\vDash_{tot} \{p\}\, S\, \{q\}$.

- The triple $\{p\}\, S\, \{q\}$ is **partially correct in** $\sigma$ (or $\sigma$ **satisfies the triple under partial correctness**), written as $\sigma \vDash \{p\}\, S\, \{q\}$, if and only if it is the case that "if $\sigma$ satisfies $p$, then if running $S$ in $\sigma$ can terminate without an error, it terminates in states satisfying $q$".
  - In other words, $\sigma \vDash \{p\}\, S\, \{q\} \Leftrightarrow (\sigma(p) \neq \bot) \wedge (\sigma \vDash p \rightarrow \forall \tau \in M(S,\sigma).\, \tau \neq \bot \rightarrow \tau \vDash q)$;
    or equivalently, $\sigma \vDash \{p\}\, S\, \{q\} \Leftrightarrow (\sigma(p) \neq \bot) \wedge (\sigma \vDash p \rightarrow M(S,\sigma) - \bot \vDash q)$.

- The triple $\{p\}\, S\, \{q\}$ is **partially correct** (or the triple is **valid under partial correctness**) if and only if $\sigma \vDash \{p\}\, S\, \{q\}$ for all $\sigma \in \Sigma$. We write $\vDash \{p\}\, S\, \{q\}$.

- The only difference between two correctness is the case "executing $S$ in $\sigma$ ends with $\bot$". If $S$ can always terminate in $\sigma$, then total correctness and partial correctness are the same.

2. If $\sigma \vDash p$ and $M(S,\sigma) = \{\bot\}$, then:
   a. Does $\sigma \vDash_{tot} \{p\}\, S\, \{q\}$? No
   b. Does $\sigma \vDash \{p\}\, S\, \{q\}$? Yes

3. Finish the following equalities, assume that $\sigma(p) \neq \bot$.
   a. $\sigma \vDash_{tot} \{p\}\, S\, \{q\} \Leftrightarrow \sigma \vDash p \rightarrow M(S,\sigma) \vDash q$
      $\Leftrightarrow \sigma \nvDash p \vee M(S,\sigma) \vDash q$
      $\Leftrightarrow \sigma \nvDash p \vee \forall \tau \in M(S,\sigma).\, \tau \vDash q$

   b. $\sigma \vDash \{p\}\, S\, \{q\} \Leftrightarrow \sigma \vDash p \rightarrow M(S,\sigma) - \bot \vDash q$
      $\Leftrightarrow \sigma \nvDash p \vee M(S,\sigma) - \bot \vDash q$
      $\Leftrightarrow \sigma \nvDash p \vee \forall \tau \in M(S,\sigma).\, \tau = \bot \vee \tau \vDash q$

   c. $\sigma \nvDash_{tot} \{p\}\, S\, \{q\} \Leftrightarrow \sigma \vDash p \wedge M(S,\sigma) \nvDash q$
      $\Leftrightarrow \sigma \vDash p \wedge \exists \tau \in M(S,\sigma).\, \tau = \bot \vee \tau \nvDash q$

   d. $\sigma \nvDash \{p\}\, S\, \{q\} \Leftrightarrow \sigma \vDash p \wedge M(S,\sigma) - \bot \nvDash q$
      $\Leftrightarrow \sigma \vDash p \wedge \exists \tau \in M(S,\sigma).\, \tau \neq \bot \wedge \tau \nvDash q$

4. True or False:
   a. $\vDash \{F\}\, S\, \{q\}$ True, nothing can pass the precondition.
   b. $\vDash \{p\}\, S\, \{T\}$ True, for any state $\sigma$, $\forall \tau \in M(S,\sigma).\, \tau = \bot \vee \tau \vDash T$
   c. $\vDash_{tot} \{F\}\, S\, \{q\}$ True, nothing can pass the precondition.
   d. $\vDash_{tot} \{p\}\, S\, \{T\}$ False, it is not true for some $\sigma$ such that $\bot \in M(S,\sigma)$

- From Example 4, we can say that $\sigma \vDash_{tot} \{p\}\, S\, \{q\} \Leftrightarrow (\sigma \vDash \{p\}\, S\, \{q\}) \wedge \bot \notin M(S,\sigma)$

5. Let $W \equiv \textbf{while } k \neq 0 \textbf{ do } k \coloneqq k - 1 \textbf{ od}$. Decide true or false.
   a. $\vDash_{tot} \{k \geq 0\}\, W\, \{k = 0\}$ True.
   b. $\vDash \{k = -1\}\, W\, \{k = 0\}$ True. $W$ will diverge in a state with $k = -1$.
   c. $\vDash_{tot} \{k = -1\}\, W\, \{k = 0\}$ False.
   d. $\vDash \{T\}\, W\, \{k = 0\}$ True. If $k < 0$ then $W$ diverges or else $W$ ends with $k = 0$.
   e. $\vDash_{tot} \{T\}\, W\, \{k = 0\}$ False.

- Remember that our goal is to reason whether a program works correctly. "Good" (not just valid) triples can help us to understand the conditions that are "*required*" before the program and conditions "*provided*" after the program. Here, let's use some examples to show (or preview) some rules that can help us to find good triples. We will see these rules in future classes.

6. Let $W' \equiv \textbf{while } k > 0 \textbf{ do } k \coloneqq k - 1 \textbf{ od}$.
   a. Is it true that $\vDash_{tot} \{T\}\, W' \{k \leq 0\}$? Yes.
   b. Rewrite the above triple with a stronger postcondition:
      $$\{k = c_0\}\, W'\, \{(c_0 > 0 \to k = 0) \wedge (c_0 \leq 0 \to k = c_0)\}.$$

      - Here $c_0$ is a **logical variable**, since it also represents a constant, it is also a **logical constant**. A logical variable/constant is some named variable/constant that is only used in predicates, we use them to reason about our program.
      - It is not always easy to find a strong post condition for loop. But here, we happen to get one. We get it by *negating the loop condition* and "*forward assignment*". We will look at these two rules in the future.

7. Consider the following triples. What precondition $p$ can make this triple valid?
   a. $\{p\}\, m \coloneqq k\, \{x = m\}$.
      Since $x$ is bind with $m$ after assigning $k$ to $m$, thus we can say the precondition is $x = k$.

   b. $\{p\}\, k \coloneqq k + 1\, \{P(k)\}$ where $P(k)$ is a predicate function.
      Consider that $k = k_0 + 1$ after the assignment statement, the before the assignment statement we should have $k = k_0$; to make the triple valid, we can let $p \equiv P(k + 1)$.

- **[Backward Assignment Rule]** If $P(x)$ is a predicate function, then $\{P(e)\}\, v \coloneqq e\, \{P(v)\}$. It turns out that $P(e)$ is the *weakest precondition* that works with the assignment statement $v \coloneqq e$ and postcondition $P(v)$.