Correctness (“Hoare”) Triples

Part 2: Sequencing, Assignment, Strengthening, and Weakening

CS 536: Science of Programming, Spring 2023

2023-02-09 p.3

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.
- To connect correctness triples in sequence, we need to weaken and strengthen conditions.

B. Objectives

At the end of today you should know

- Programs may have many different annotations, and we might prefer one annotation over another (or not), depending on the context.
- Under the right conditions, correctness triples can be joined together.
- One general rule for reasoning about assignments goes "backwards" from the postcondition to the precondition.
- What strength is and what weakening and strengthening are.

C. Examples of Partial and Total Correctness With Loops

- For the following examples, let $W \equiv$ ***while*** $k \neq 0$ ***do*** $k := k-1$ ***od***.
    - ***Example 1:*** $\vDash_{\text{tot}} \{k \geq 0\}\ W\ \{k = 0\}$. If we start in a state with $k \geq 0$, the loop is guaranteed to terminate in a state satisfying $k = 0$.
    - ***Example 2:*** $\vDash \{k = -1\}\ W\ \{k = 0\}$ but $\nvDash_{\text{tot}} \{k = -1\}\ W\ \{k = 0\}$. The triple is partially correct but not totally correct because it diverges if $k = -1$. I.e., we have $\nvDash_{\text{tot}} \{k = -1\}\ W\ \{T\}$. Also note that partial correctness would hold if we substitute any predicate for $k = 0$.
    - ***Example 3***: $\vDash \{T\}\ W\ \{k = 0\}$ but $\nvDash_{\text{tot}} \{T\}\ W\ \{k = 0\}$. The triple is partially correct but not totally correct because it diverges for at least one value of k.
- For the following examples, let $W' \equiv$ ***while*** $k > 0$ ***do*** $k := k-1$ ***od***. (We’re changing the loop test of W so that it terminates immediately when k is negative.)
    - ***Example 4:*** $\vDash_{\text{tot}} \{T\}\ W'\ \{k \leq 0\}$.

- ***Example 5:*** $\vDash_{\text{tot}} \{ k = c_0 \} W' \{ (c_0 \le 0 \rightarrow k = c_0) \wedge (c_0 \ge 0 \rightarrow k = 0) \}$. This is Example 4 with the "strongest" (most precise) postcondition possible. (In general, it's not always possible to find such a postcondition for loop, but it is here.)

D. More Correctness Triple Examples

Same Code, Different Conditions

- The same piece of code can be annotated with conditions in different ways, and there's not always a "best" annotation. An annotation might be the most general one possible (we'll discuss this concept soon), but depending on the context, we might prefer a different annotation.
- As before, let $sum(x, y)$ = the sum of x, $x+1$, $x+2 <$ … y. (If $x > y$, let $sum(x, y) = 0$.) In Examples 9 – 12, we have the same program annotated (with preconditions and postconditions) of various strengths (strength = generality).
- ***Example 9:*** $\{T\}\ i := 0\ ;\ s := 0\ \{ i = 0 \wedge s = 0 \}$.
  - This is the strongest (most precise) annotation for this program.
- ***Example 10:*** $\{T\}\ i := 0\ ;\ s := 0\ \{ i = 0 \wedge s = 0 = sum(0, i) \}$.
  - This adds a summation relationship to i and s when they're both zero.
- ***Example 11:*** $\{ n \ge 0 \}\ i := 0\ ;\ s := 0\ \{ 0 = i \le n \wedge s = 0 = sum(0, i) \}$
  - This limits i to a range of values 0, …, n. We have to include $n \ge 0$ in the precondition if we want to claim $n \ge 0$ in the postcondition.
- ***Example 12:*** $\{ n \ge 0 \}\ i := 0 ; s := 0\ \{ 0 \le i \le n \wedge s = sum(0, i) \}$
  - The postcondition no longer includes i and s being zero, so this postcondition is weaker (less precise) than the postcondition for Example 11. This might seem like a disadvantage but will turn out to be an advantage later.
- The next two examples relate to calculating the midpoint in binary search. Though the code is the same, whether the midpoint is strictly between the left and right endpoints depends on whether or not the endpoints are nonadjacent.
  - ***Example 13:*** $\{ lt < rt \wedge lt \ne rt - 1 \}\ mid := (lt + rt)/2\ \{ lt < mid < rt \}$
  - ***Example 14:*** $\{ lt < rt \}\ mid := (lt + rt)/2\ \{ lt \le mid < rt \}$
- In Examples 13 and 14, the differences in the postcondition have an effect on how to detect that the value being searched for doesn't exist. In Example 13, it's $lt = rt - 1$; in Example 14, it's $lt > rt$.

Use of DeMorgan's Laws

- When a loop terminates, we know that the negation of the loop test holds, so DeMorgan's laws can be useful. Similarly, for a condition, we know that the negation of the test holds just before we execute the false branch.
- ***Example 15:*** Here we search downward for $x \ge 0$ such that $f(x)$ is $\le y$; we stop if we find such an x or if we run out of values to test.

$\{x \geq 0\}$
while $x \geq 0 \wedge f(x) > y$ ***do*** $x := x - 1$ ***od***
$\{x < 0 \vee f(x) \leq y\}$ [2023-02-08 typo] *// Negation of loop test*

- ***Example 16:*** This is Example 15 rephrased as an array search; we search to the left for an index k such that $b[k] \leq y$; we stop if we find one or run out of indexes to test[1].

  $\{k \geq 0\}$
  while $k \geq 0 \wedge b[k] > y$ ***do*** $k := k - 1$ ***od***
  $\{k < 0 \vee b[k] \leq y\}$ *// Negation of loop test*

Joining Two Triples

- To make two statements a sequence, we have to compare the postcondition of the first statement and the precondition of the second. If they're the same, we can make the join.
  - I.e., if we have $\{p\}\, S_1\, \{q\}$and $\{q\}\, S_2\, \{r\}$, then we can form $\{p\}\, S_1; S_2\, \{r\}$ because when S_1 finishes executing, it will satisfy the precondition of S_2.
- ***Example 17:*** We can join these two statements because the postcondition of the first statement matches the precondition of the second. (Note though $s = sum(0, k)$ holds before and after the two assignments, it doesn't hold between.)

Combining	$\{s = sum(0, k)\}\, s := s + k + 1\, \{s = sum(0, k + 1)\}$
and	$\{s = sum(0, k + 1)\}\, k := k + 1\, \{s = sum(0, k)\}$
yields	$\{s = sum(0, k)\}\, s := s + k + 1\ ;\ k := k + 1\, \{s = sum(0, k)\}$

- ***Example 18:*** Alternatively, we can increment k first and then update s.

Combining	$\{s = sum(0, k)\}\, k := k + 1 \{s = sum(0, k - 1)\}$
and	$\{s = sum(0, k - 1)\}\, s := s + k \{s = sum(0, k)\}$
yields	$\{s = sum(0, k)\}\, k := k + 1\ ;\ s := s + k\, \{s = sum(0, k)\}$

Reasoning About Assignments (Technique 1: "Backward")

- There are two general rules for reasoning about assignments.
- The first rule is a goal-directed one that works "backwards", from the postcondition to the precondition.
- ***Assignment Rule 1 ("Backward" assignment:*** If $P(x)$ is a predicate function, then $\{P(e)\}\, v := e\, \{P(v)\}$. It turns out that $P(e)$ is the most general (the so-called "weakest") precondition that works with the assignment $v := e$ and postcondition $P(v)$. We'll study this in the next lecture.
- ***Example 19:*** $\{P(m/2)\}\, m := m/2\, \{P(m)\}$
  - If $P(x) \equiv x > 0$, then this triple expands to $\{m/2 > 0\}\, m := m/2\, \{m > 0\}$.
  - If $P(x) \equiv x^2 > x^3$, then this triple expands to $\{(m/2)^2 > (m/2)^3\}\, m := m/2\, \{m^2 > m^3\}$.
- ***Example 20:*** $\{Q(k+1)\}\, k := k + 1\, \{Q(k)\}$

[1] Don't think I've said before that we can make $p \wedge q$ short-circuiting by using ***if*** p ***then*** T ***else*** q ***fi***.

- If $Q(x) \equiv s = sum(0, x)$ then this triple expands to
  $$\{s = sum(0, k+1)\}\ k := k+1\ \{s = sum(0, k)\}.$$
- ***Example 21:*** $\{R(s+k+1)\}\ s := s+k+1\ \{R(s)\}$
  - If $R(x) \equiv x = sum(0, k+1)$, then this triple expands to
    $$\{s+k+1 = sum(0, k+1)\}\ s := s+k+1\ \{s = sum(0, k+1)\}$$

- In general, for $\{P(e)\}\ v := e\ \{P(v)\}$ to be valid, we need the following lemma:
- ***Assignment Lemma:*** For all σ, if $\sigma \vDash P(e)$ then $M(v := e, \sigma) = \sigma[v \mapsto \sigma(e)] \vDash P(v)$.
  - Intuitively, what this says is that if we want to know that v has property P after binding v to the value of e, we need to know that e has property P beforehand.
  - We won't go into the detailed proof of this lemma, but basically, you work recursively on the structures of $P(v)$ and $P(e)$ simultaneously. The important case is when we encounter an occurrence of v in $P(v)$ and the corresponding occurrence of e in $P(e)$. In $\sigma \vDash P(e)$, the value of e is $\sigma(e)$. In $\sigma[v \mapsto \sigma(e)] \vDash P(v)$, the value of v is also $\sigma(e)$.

E. Stronger and Weaker Predicates

- ***Generalizing the Sequence Rule:*** We've already seen that two triples $\{p\}\ S_1\ \{q\}$and $\{q\}\ S_2\ \{r\}$ can be combined to form the sequence $\{p\}\ S_1 ; S_2\ \{r\}$.
- Say we want to combine two triples that don't have a common middle condition, $\{p\}\ S_1\ \{q\}$ and $\{q'\}\ S_2\ \{r\}$.
  - We can do this iff $q \rightarrow q'$. If S_1 terminates in state τ and $\{p\}\ S_1\ \{q\}$ is valid, then $\tau \vDash q$. If $\vDash q \rightarrow q'$, then $\tau \vDash q'$, so if $\{q'\}\ S_2\ \{r\}$, then if running S_2 in τ terminates, it terminates in a state satisfying r.
  - This reasoning works both for partial and total correctness.
- Our earlier discussion used a special case of the above. When $q \equiv q'$, we get that $\{p\}\ S_1\ \{q\}$ and $\{q\}\ S_2\ \{r\}$ can be joined.
- ***Definition:*** If $p \rightarrow q$ then p is ***stronger than*** q and q is ***weaker than*** p. I.e., the states that satisfy p also satisfy q.
  - (Technically, we should say "stronger than or equal to" and "weaker than or equal to," because if $p \leftrightarrow q$, then p is stronger and weaker than q and vice versa. But it's too much of a mouthful.
- ***Definition***: p is ***strictly stronger*** than q and q is ***strictly weaker*** than p if $(p \rightarrow q) \wedge \neg(q \rightarrow p)$.
- ***Example 22***: $x = 0$ is (strictly) stronger than $x = 0 \vee x = 1$, which is (strictly) stronger than $x \geq 0$.

Predicates and Venn Diagrams

- You can view a predicate as standing for the set of states that satisfy it. In that case, $p \rightarrow q$ means that the set of states for p is $\subseteq$ the set of states for q.[2]
- ***Notation:*** Sometimes we'll abbreviate "*the set of states satisfying p*" to just "*p*".
- Venn diagrams with sets of states can help illustrate comparisons of predicate strength.
- In Figure 1, $p \rightarrow q$ because the set of states for p is $\subseteq$ the set of states for q. It's less obvious, but the contrapositive $\neg q \rightarrow \neg p$ also holds.
  - r has an intersection with q; the part inside q is $q \wedge r$; the part outside q is $\neg q \wedge r$.
  - p has no intersection with r, so neither $p \rightarrow r$ nor $r \rightarrow p$ hold, but all of p is outside r, so $p \rightarrow \neg r$ (and $r \rightarrow \neg p$).

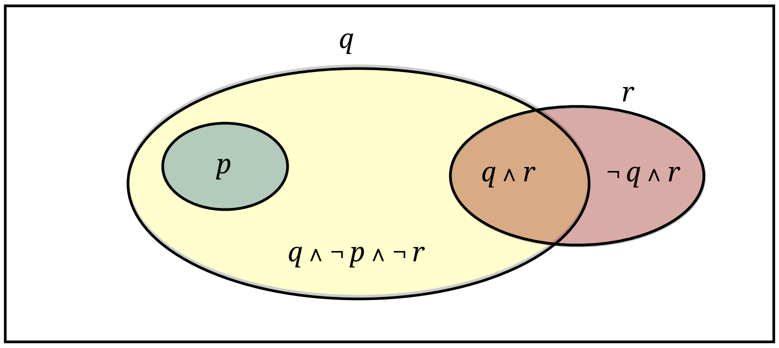

Figure 1: Predicates As Standing for Sets of States

F. Strengthening and Weakening Conditions

- The relationship between stronger and weaker predicates allows us to make certain changes to the conditions of triples "for free" (i.e, we just have to know one implies the other) Both of the following properties are valid for both partial and total correctness of triples (i.e., for $\vDash$ and $\vDash_{tot}$).
- The "always" below means that given an appropriate correctness triple it's sufficient to know $s \rightarrow b$, (so that s and b have *smaller* and *bigger* sets of satisfying states).
  - ***Preconditions can always be strengthened***: If $s \rightarrow b$ and $\{b\}\ S\ \{q\}$, then $\{s\}\ S\ \{q\}$.
  - ***Postconditions can always be weakened***: If $s \rightarrow b$ and $\{p\}\ S\ \{s\}$, then $\{p\}\ S\ \{b\}$.
- With p and q in Figure 1, $p \rightarrow q$, so p is stronger than q.
  - So if q were the precondition of a triple, it could be strengthened to p. We can't strengthen q to r, but we can strengthen it to $q \wedge r$.
  - In the other direction, a postcondition of p or $q \wedge r$ can be weakened to q.

[2] One very old notation for implication is $p \supset q$; it's important not to read that $\supset$ as a superset symbol, since $p \rightarrow q$ means that the set of states for p is $\subseteq$ the set of states for q.

- ***Example 23***: If $\{x \geq 0\}\ S\ \{y = 0\}$ is valid, then so are
  - $\{x \geq 0\}\ S\ \{y = 0 \vee y = 1\}$ *Weakened postcondition*
  - $\{x = 0\}\ S\ \{y = 0\}$ *Strengthened precondition*
  - $\{x = 0\}\ S\ \{y = 0 \vee y = 1\}$ *Strengthened precondition and weakened postcondition*

- ***Example 24***: Since $s = sum(0, k) \Leftrightarrow s + k + 1 = sum(0, k+1)$, we can "strengthen" the precondition of $\{s + k + 1 = sum(0, k+1)\}\ k := k + 1\ \{q\}$ and get $\{s = sum(0, k)\}\ k := k + 1\ \{q\}$. (I put the "strengthen" in quotes here to point out that since the two conditions are $\Leftrightarrow$, we can also do "strengthening" in the other direction, going from $s = sum(0, k)\}$ to $s + k + 1 = sum(0, k+1)$.)

Limitations of Strengthening and Weakening

- If $p \to q$ says that the sets of states satisfying p and q respectively are $\subseteq$, then the implications $q \to q_1$, $q_1 \to q_2$, $q_2 \to q_3$, etc. form a sequence of weaker and weaker predicates because the sets of states get larger and larger. There is a limit, namely, Σ, the set of all states. As a set of states, Σ corresponds to T (true) and is the weakest possible state.
- Similarly, going right to left, the implications ...., $p_3 \to p_2$, $p_2 \to p_1$, $p_1 \to p$ form a sequence of stronger and stronger predicates because their sets of states get smaller and smaller. Here, the limit is the empty set $\varnothing$, the set of states that correspond to F (false), making it the strongest possible state.
- Having F be strongest and T be weakest may be counterintuitive. It may help if you think of the strength of a predicate being the amount of constraints it puts on the set of the states that satisfy it. The strongest predicate, F has contradictory constraints on it, hence is satisfied by no state. The weakest predicate, T, has no constraints on it, hence it is satisfied by every state. In sets of states notation, $\{\ \} \vDash F$ and $\{\Sigma\} \vDash T$.

Can vs Should

- Just because we ***can*** strengthen preconditions and weaken postconditions doesn't mean we ***should***. Recall our edge cases for satisfying correctness triples:
  - $\sigma \vDash \{F\}\ S\ \{q\}$ and $\sigma \vDash_{tot} \{F\}\ S\ \{q\}$ have the strongest possible preconditions.
  - $\sigma \vDash \{p\}\ S\ \{T\}$ has the weakest possible postcondition.
  - $\sigma \vDash_{tot} \{p\}\ S\ \{T\}$ says that S terminates when you start it in p but it says nothing about what the state looks like when it terminates.
- From the programmer's point of view, if $\{p\}\ S\ \{q\}$ has a bug, then strengthening p or weakening q can get rid of the bug without changing S.
  - ***Example 25 (Strengthening the precondition)***
    - If $\{p\}\ S\ \{q\}$ causes an error if $x = 0$, we can tell the user to use $\{p \wedge x \neq 0\}\ S\ \{q\}$.
  - ***Example 26 (Weakening the postcondition)***
    - If $\{p\}\ S\ \{q\}$ causes an error because S terminates satisfying predicate r (where r doesn't imply q) then we can tell the user to use $\{p\}\ S\ \{q \vee r\}$.

Weaker Preconditions and Stronger Postconditions

- From the user's point of view, weaker preconditions and stronger postconditions make triples more useful.

Weaker Preconditions

- Weaker preconditions make code more applicable by increasing the set of starting states.
- Say $\{p\}\ S\ \{r\}$ is valid. If $q \rightarrow p$, then weakening the precondition to get $\{q\}\ S\ \{r\}$ gives the programmer more flexibility in what states to start.
- Unlike strengthening a precondition, however, weakening a precondition requires work because we need to show that the states in q that are not in p also work as precondition states.
  - In symbols, if we show $\{q \wedge \neg p\}\ S\ \{r\}$, then since we already know $\{p\}\ S\ \{r\}$, we can $\vee$ the preconditions. We get $p \vee (q \wedge \neg p) \Leftrightarrow (p \vee q) \wedge (p \vee \neg p) \Leftrightarrow (p \vee q) \wedge T \Leftrightarrow p \vee q$.
  - Then we can take precondition $p \vee q$ and strengthen it to just q, so $\{q\}\ S\ \{r\}$.

Stronger Postconditions

- Stronger postconditions make code more specific by decreasing the set of ending states.
- Say $\{r\}\ S\ \{q\}$ is valid. if $q \rightarrow p$, then we could strengthen postcondition q to get $\{r\}\ S\ \{p\}$.
- But we can't do this unless we know that running S gets us to the part of q that is inside p, (and never to the part of q outside p).
  - If we show $\{r\}\ S\ \{\neg(q \wedge \neg p)\}$, then we can $\wedge$ that with postcondition p and get $q \wedge \neg(q \wedge \neg p) \Leftrightarrow q \wedge (\neg q \vee p) \Leftrightarrow (q \wedge \neg q) \vee (q \wedge p) \Leftrightarrow q \wedge p$.
  - Then we can weaken postcondition $p \wedge q$ to just p.