

Class 04: Programs, Conditions, and Hoare Triples

A. Why?

The simple deterministic programming language we'll begin with is the foundation for the parallel and nondeterministic extensions we'll see in later chapters. To discuss correctness of our programs, we'll phrase requirements as predicates known as "conditions." Finally, we'll combine conditions and programs to form "correctness triples" (a.k.a Hoare triples).

B. Outcomes

By the end of the class you should

- Know the basic syntax of our simple deterministic programming language.
- Have an intuitive knowledge of the state-transforming semantics of our simple programming language and be familiar with the notation $\mathcal{M}\llbracket S \rrbracket(\sigma)$.
- Know what it means for a predicate to be satisfied in a state and be familiar with the notation $\sigma \models p$.
- Know the syntax of correctness triples (a.k.a. Hoare triples) as $\{p\} S \{q\}$ and that its precondition and postcondition are p and q respectively.
- Understand what it means for a program to meet its specification in a given state (notation $\sigma \models \{p\} S \{q\}$).
- Understand what it means for a correctness triple to be valid (notation $\models \{p\} S \{q\}$).

C. Quiz 1 (30 min)

D. Simple Programs

- Our programming language has five kinds of statements and no declarations
 - **skip** — does nothing
 - Assignment: $var := expr$
 - We'll always work with type-correct programs, and we'll assume we know the types of variables without bothering to write declarations. E.g., in $x := 17$, x must be an integer variable.
 - Sequence: $S_1 ; S_2$ where S_1 and S_2 are statements. Semicolon is a separator.
 - Longer sequences are read left-to-right: $S_1 ; S_2 ; S_3$, for example.
 - Conditional
 - **if B then S_1 else S_2 fi** where B is a boolean expression and S_1 and S_2 are statements. (Can be nested.)
 - **if B then S_1 fi** is an abbreviation for **if B then S_1 else skip fi**.
 - Loop
 - **while B do S_1 od** where B is a boolean expression and S_1 is a statement.

E. Semantics of Programs and the $\mathcal{M}\llbracket S \rrbracket(\sigma)$ Notation

- A program in our language is just a statement.
- The meaning of a statement S is that if you run it in some state, then S changes the state to some new state: S is a state transformer.
- Notation: $\mathcal{M}\llbracket S \rrbracket(\sigma)$ is “the meaning of statement S in state σ ” a.k.a. “the state we end in if we start in state σ and run S ”
 - Example: $\mathcal{M}\llbracket x := x+1; y := x \rrbracket(\sigma)$ should use $\sigma[x := \alpha+1][y := \alpha]$ where $\alpha = \sigma(x)$.
 - **Warning:** In general in this course, $()$, $[\]$, $\{ \}$, and $\llbracket \ \rrbracket$ are not interchangeable.
- However, $\mathcal{M}\llbracket S \rrbracket(\sigma)$ yields a set of states, not just a state.
 - Returning a set of states lets us know when a program doesn’t terminate. If S **diverges** at σ (goes into an infinite loop when started in state σ), then $\mathcal{M}\llbracket S \rrbracket(\sigma) = \emptyset$. Since S is deterministic, $\mathcal{M}\llbracket S \rrbracket(\sigma)$ is \emptyset or has exactly one member.
- Extend notation to a set of states: If X is a set of states, then $\mathcal{M}\llbracket S \rrbracket(X)$ is the set of states we can end in if we start in a state in X .
 - I.e., $\mathcal{M}\llbracket S \rrbracket(X)$ is the union of $\mathcal{M}\llbracket S \rrbracket(\sigma)$ for all $\sigma \in X$.
 - Note if $X = \emptyset$, then $\mathcal{M}\llbracket S \rrbracket(X) = \emptyset$.

F. Definition of $\mathcal{M}\llbracket S \rrbracket(\sigma)$

- The definition of $\mathcal{M}\llbracket S \rrbracket(\sigma)$ is recursive/inductive on the structure of the statement S .
 - (Note: The text uses a more-complicated definition of $\mathcal{M}\llbracket S \rrbracket(\sigma)$ based on something called “operational semantics” — we won’t look at this until parallel programs.)
 - Base case 1: S is **skip**. In English: **skip** doesn’t actually change the state; the meaning of **skip** is the null transformation on states.
 - In symbols, $\mathcal{M}\llbracket \text{skip} \rrbracket(\sigma) = \{\sigma\}$.
 - Base case 2: S is a simple assignment $x := s$. In English: To evaluate an assignment, we evaluate s in the current state and copy its value to memory as the new value of x .
 - In symbols, $\mathcal{M}\llbracket x := s \rrbracket(\sigma) = \{\sigma[x := \sigma(s)]\}$.
 - Base case 3: S is an array assignment $b[s_1] := s_2$.
 - We’re putting this off for a while.
 - Inductive case 1: S is a sequence $S_1 ; S_2$. In English: First we run S_1 ; if S_1 doesn’t terminate then $S_1 ; S_2$ doesn’t terminate. Otherwise, we run S_2 in the state we get after S_1 terminates.
 - In symbols, $\mathcal{M}\llbracket S_1 ; S_2 \rrbracket(\sigma) = \mathcal{M}\llbracket S_2 \rrbracket(\mathcal{M}\llbracket S_1 \rrbracket(\sigma))$.
 - I.e., $\mathcal{M}\llbracket S_1 ; S_2 \rrbracket(\sigma) = \mathcal{M}\llbracket S_2 \rrbracket(\sigma')$ where $\sigma' \in \mathcal{M}\llbracket S_1 \rrbracket(\sigma)$. If $\mathcal{M}\llbracket S_1 \rrbracket(\sigma)$ is \emptyset then $\mathcal{M}\llbracket S_1 ; S_2 \rrbracket(\sigma)$ is \emptyset .

- Inductive case 2: S is a conditional **if** B **then** S_1 **else** S_2 **fi**. In English: The meaning of a conditional is either the meaning of the true branch or the false branch, depending on the value of the test.
 - In symbols, $\mathcal{M}\llbracket S \rrbracket(\sigma) = \mathcal{M}\llbracket S_1 \rrbracket(\sigma)$ if $\sigma(B) = \mathbf{true}$; it = $\mathcal{M}\llbracket S_2 \rrbracket(\sigma)$ if $\sigma(B) = \mathbf{false}$.
- Inductive case 3: S is a loop **while** B **do** S_1 **od**. (This is the tricky case.)
 - In English: First evaluate the test B ; if the test is false, we're done. Otherwise, evaluate the loop body and repeat the test. Continue until some test yields **false**. If none of the tests yields false, we're in an infinite loop. (Also, if the loop body goes into an infinite loop, then we don't terminate either.)
 - Define the set of states that hold each time we test B : Let $\tau_0 = \sigma$, let $\tau_1 \in \mathcal{M}\llbracket S_1 \rrbracket(\tau_0)$, $\tau_2 \in \mathcal{M}\llbracket S_1 \rrbracket(\tau_1)$, etc.
 - Note that if any of the states in the τ sequence is \emptyset , then all the states from then on are \emptyset .
 - Let k be the smallest natural number such that $\tau_k(B) = \mathbf{false}$. Then $\mathcal{M}\llbracket \mathbf{while} \ B \ \mathbf{do} \ S_1 \ \mathbf{od} \rrbracket(\sigma) = \{\tau_k\}$.
 - If no such k exists, then the meaning of the loop is \emptyset .

G. Conditions and Satisfaction

- Programs are only correct relative to a specification.
- We'll use predicate logic formulas to specify what memory should look like before and after executing a program.
- **Conditions** (a.k.a. **assertions**) are just predicates used to describe what properties states should have.
 - Notation: Typical conditions are p, q, r, \dots
- A **precondition** specifies a condition that should be true before running a program.
- A **postcondition** specifies a condition that should be true after running a program.
- Example: We might have $x > 0$ as a precondition, $x := x+1$ as the program, and $x > 1$ as the postcondition.
- A condition is **satisfied** in some state σ if it is true in that state.
 - Notation: $\sigma \models p$ (“ σ satisfies p ,” or “ p is true in σ ”).
 - The “ \models ” symbol is the “double turnstile”.
 - If σ does not satisfy p , then we can write $\sigma \not\models p$. Note if $\sigma \models \neg p$ then $\sigma \not\models p$.
 - If the condition is a simple boolean expression B , then $\sigma \models B$ iff $\sigma(B) = \mathbf{true}$.
 - E.g., if $\sigma_0(x) = 3$ then $\sigma_0 \models x > 0$. If $\sigma_1(x) = \sigma_1(y)-1$, then $\models \sigma_1 \models x < y$.
- We can ask about p being satisfied in every state in a set of states X .
 - Notation: $X \models p$ means $\sigma \models p$ for all $\sigma \in X$. (So if $X = \emptyset$, then $X \models p$ automatically.)

- Rather than give a formal definition of $\sigma \models p$, I'm going to rely on your intuition for what it means for a predicate to be true. As some examples, $\sigma \models 3 < 4$ for any σ ; $\sigma \models x > 0$ iff $\sigma(x)$ is positive, $\sigma \models p \vee q$ iff $\sigma \models p$ or $\sigma \models q$ (or both).

H. Syntax of Correctness Triples

- A **correctness triple** (a.k.a. “**Hoare triple**,” after C.A.R. Hoare) combines a precondition, postcondition, and program.
- Syntax for correctness triple: $\{p\} S \{q\}$ has precondition p , postcondition q , and program S .
- My pet peeve: Claiming that the precondition is $\{p\}$ and the postcondition is $\{q\}$
 - Here the braces are comment symbols (not part of p or q).
 - Think $/* p */ S /* q */$. The precondition is p , not $/* p */$
 - In **if** B **then** S **fi, the test is B , not **if** B **then**.**
 - $\{p\}$ is a set of conditions with one element.

I. Semantics of Partial Correctness Triples

- A correctness triple is **satisfied** in some state if the program meets its specification in that state.
- Have to distinguish between **total correctness** (no infinite loops) and **partial correctness** (correct if there are no infinite loops).
- **For partial correctness:** In English, the triple $\{p\} S \{q\}$ is satisfied in state σ when (1) if the precondition is true in σ , and (2) if S terminates when you run it in σ , then (3) the terminal state satisfies the postcondition.
 - Notation: $\sigma \models \{p\} S \{q\}$ iff $(\sigma \models p)$ implies $((\mathcal{M}\llbracket S \rrbracket(\sigma) \neq \emptyset) \text{ implies } (\mathcal{M}\llbracket S \rrbracket(\sigma) \models q))$.
- An equivalent formulation:
 - $\sigma \models \{p\} S \{q\}$ iff $(\sigma \not\models p)$ or $(\mathcal{M}\llbracket S \rrbracket(\sigma) = \emptyset)$ or $(\mathcal{M}\llbracket S \rrbracket(\sigma) \models q)$.
 - This makes it easy to see that if σ doesn't satisfy p or if S doesn't terminate, then $\sigma \models \{p\} S \{q\}$ automatically.
- Examples:
 - Let p be $x > 0$, let q be $x = 2$, and let S be **while** $x \neq 2$ **do** $x := x-1$ **od**.
 - Let $\sigma_0(x) = 0$. Then $\sigma_0 \not\models p$ so it doesn't matter whether $\mathcal{M}\llbracket S \rrbracket(\sigma) =$ or $\neq \emptyset$ or $\mathcal{M}\llbracket S \rrbracket(\sigma) \models$ or $\not\models q$. So $\sigma_0 \models \{p\} S \{q\}$.
 - Let $\sigma_1(x) = 1$, then $\sigma_1 \models p$ and $\mathcal{M}\llbracket S \rrbracket(\sigma) = \emptyset$, so $\sigma_1 \models \{p\} S \{q\}$.
 - Let $\sigma_2(x) = 2$, then $\sigma_2 \models p$ and $\mathcal{M}\llbracket S \rrbracket(\sigma) = \{\sigma_2[x := 2]\}$ is not empty and $\sigma_2[x := 2] \models q$. So $\sigma_2 \models \{p\} S \{q\}$.

J. Valid Partial Correctness Triples

- A correctness triple is **valid** if it is satisfied in every (type-correct) state.

- If your correctness triple is valid, then your program is correct.
- If a correctness triple $\{p\} S \{q\}$ is invalid, then for some σ , the triple is unsatisfied.
 - In English, there's some state σ that satisfies p , but if you run S in σ , you end with q not being satisfied. I.e., in state σ , our program has a bug.
 - In symbols, $\sigma \not\models \{p\} S \{q\}$ iff $\sigma \models p$ and $\mathcal{M}[S](\sigma) \neq \emptyset$ and $\mathcal{M}[S](\sigma) \not\models q$.
- Example: For $\{x > 0\} x := x+1 \{x > 2\}$, if $\sigma(x) = 1$, then $\sigma \models x > 0$, $\mathcal{M}[x := x+1](\sigma) = \{\sigma[x := 2]\} \neq \emptyset$, but $\{\sigma[x := 2]\} \not\models x > 2$.

Activity/Homework: Meanings of Programs; Satisfaction; Trivial Triples

A. Why?

To understand how programs work, we must understand what they mean. To know if a program meets its specification, we have to know when predicates are satisfied.

B. Outcomes

By the end of the activity you should

- Be able to calculate $\mathcal{M}\llbracket S \rrbracket(\sigma)$ for short programs S .
- Be able to check whether or not a predicate is satisfied in a state. (I.e., is $\sigma \models p$?)

C. Questions

We'll do some of these questions as an activity; do the rest as homework.

Meanings of Programs

1. If $\sigma_0(x) = 5$ and $\sigma_0(y) = 9$, what is $\mathcal{M}\llbracket x := x+1 ; y := y * x \rrbracket(\sigma_0)$?
2. For the same σ_0 , what is $\mathcal{M}\llbracket y := y * x ; x := x+1 \rrbracket(\sigma_0)$?
3. Let σ_1 be a state and let $S \equiv \mathbf{if } v > w \mathbf{ then } w := w*2 \mathbf{ else } v := v*3 \mathbf{ fi}$. What is $\mathcal{M}\llbracket S \rrbracket(\sigma_1)$? Note: you'll have two cases, depending on the relationship between $\sigma_1(v)$ and $\sigma_1(w)$. The new values of v and w will involve $\sigma_1(v)$ and $\sigma_1(w)$.
4. Let $\Omega \equiv \mathbf{while true do skip od}$. For any σ , what is the sequence of test states (τ_0, τ_1, \dots) for $\mathcal{M}\llbracket \Omega \rrbracket(\sigma)$? What set is the value of $\mathcal{M}\llbracket \Omega \rrbracket(\sigma)$?
5. Let $W \equiv \mathbf{while } x \neq 0 \mathbf{ do } x := x-1 \mathbf{ od}$. Let $\sigma'(x) = 3$. What is the sequence of test states for $\mathcal{M}\llbracket W \rrbracket(\sigma')$? What is the value of $\mathcal{M}\llbracket W \rrbracket(\sigma')$?
6. Let $\sigma''(x) = -4$. What is the sequence of test states for $\mathcal{M}\llbracket W \rrbracket(\sigma'')$? [Same W as in the previous question.] What is the value of $\mathcal{M}\llbracket W \rrbracket(\sigma'')$?

Predicate Satisfaction

7. Give an example of a state that $\models 0 < m < n$. Also, give an example of a state that $\not\models 0 < m < n$. (The state should be “proper” — define type-correct values for m and n — even though those values don't satisfy $0 < m < n$.)
8. For an existential, $\sigma \models \exists x \in T . p$ iff there's some value (of type T) for x that makes σ updated at x with α satisfy p . E.g., $\sigma \models \exists x . x^2 < 1$ (where x ranges over the integers); we can use 0 for the “witness” α and find $\sigma[x := 0] \models x^2 < 1$. In general, it's possible to have many witnesses. What are the possible witnesses α that can be used to show $\sigma \models \exists x . x^2 > 1$?

9. If we have a set of states X , then “ X satisfies p ” (written $X \models p$) means that every state in X satisfies p . (And if $X = \emptyset$, then $X \models p$ automatically.) Now, in general, $\sigma \models p \rightarrow q$ iff $\sigma \models p$ implies that $\sigma \models q$. Question: Say $\sigma \models p \rightarrow q$ for all states σ . If $X \models p$, then does $X \models q$ also?

Trivial Partial Correctness Triples

10. Remember, $\sigma \models \{p\} S \{q\}$ iff $(\sigma \not\models p)$ or $(M[[S]](\sigma) = \emptyset)$ or $M[[S]](\sigma) \models q$. There are three cases in which $\sigma \models \{p\} S \{q\}$ kind of trivially:

- (a) When $\sigma \not\models p$ for every σ .
- (b) When $M[[S]](\sigma) = \emptyset$ for every σ .
- (c) When $\tau \models q$ for every state τ (because then the ones $\in M[[S]](\sigma)$ must $\models q$ too).

Give an example of a p that fits case (a). Give an example of an S that fits case (b).

Give an example of a q that fits case (c).