

Class 05: Syntactic Substitution; Introduction to Proof Rules

A. Why?

To reason about assignments, we need to understand syntactic substitution. To reason about programs, we use proof rules (rules of logic) especially designed for the purposes

B. Outcomes

By the end of the class you should

- Know why we need syntactic substitution and how to do them.
- Know what a verification proof is.
- Know the proof rules for the textbook's system PD (Partial correctness of Deterministic programs).

C. Results of Quiz 1

D. Review Homework

E. Satisfying a Correctness Triple With Assignment

- We know $\mathcal{M}[x := s](\sigma) = \{\sigma[x := \sigma(s)]\}$. When does $\sigma \models \{p\} x := s \{q\}$?
- Try some examples:
 - $\sigma \models \{0 < n\} x := 0 \{x < n\}$
 - $\sigma \models \{x+1 < n\} x := x+1 \{x < n\}$
 - $\sigma \models \{y/z < n\} x := y/z \{x < n\}$
- In general, if our postcondition is some predicate $q(x)$, then we need $q(s)$ beforehand.
 - $\sigma \models \{q(s)\} x := s \{q(x)\}$
 - If $q(x) \equiv x < n$, then $q(x+1) \equiv x+1 < n$, $q(0) \equiv 0 < n$, and $q(y/z) \equiv y/z < n$.
- Taking $q(x)$ and figuring out $q(s)$ is called **syntactic substitution**.
- Notation (textbook): $q[x := s]$ is **q with s substituted for x** .
 - Also pronounced “ **q with s for x** ”, “ **q with x replaced by s** ”, “**Substitute s for x in q** ” (A common notation is $q[s/x]$.)
 - Can also substitute within an expression: $s_1[x := s_2]$.
- Substitution may need/add parentheses
 - Example: $(u < v * 2)[v := z+1]$ is $u < (z+1) * 2$, not $u < z+1 * 2$
 - Think of $u < v * 2$ as $u < (v) * 2$
- Note *predicate* $[var := expr]$ and *expr* $[var := expr]$ notation resembles *state* $[var := value]$ notation.

- Decipher from context: $s[v := z+1]$ is substitution; $\sigma[v := \sigma(z+1)]$ is state update; but $s[v := \sigma(z+1)]$ and $\sigma[v := z+1]$ are bad notation.
- So general rule is $\sigma \models \{q[x := s]\} x := s \{q\}$

F. Performing Syntactic Substitution

- Syntactic substitution is mostly pretty straightforward
 - In $p[x := s]$ or $s_1[x := s]$, you find every occurrence of x inside p or s_1 and replace it with s (or maybe (s) , to keep precedences correct).
 - If there's no occurrence of x in p then $p[x := s] \equiv p$ (similar for $s_1[x := s]$).
- If $ArrayB \equiv \forall j: 0 < j < k \rightarrow x < b[j] < x^2$, then
 - $ArrayB[x := x+1] \equiv \forall j: 0 < j < k \rightarrow x+1 < b[j] < (x+1)^2$.
 - $ArrayB[k := k+1] \equiv \forall j: 0 < j < k+1 \rightarrow x < b[j] < x^2$
 - $ArrayB[x := x+1][k := k+1] \equiv (\forall j: 0 < j < k \rightarrow x+1 < b[j] < (x+1)^2)[k := k+1]$
 $\equiv \forall j: 0 < j < k+1 \rightarrow x+1 < b[j] < (x+1)^2$
 - $ArrayB[k := k+1][x := x+1] \equiv (\forall j: 0 < j < k+1 \rightarrow x < b[j] < x^2)[x := x+1]$
 $\equiv \forall j: 0 < j < k+1 \rightarrow x+1 < b[j] < (x+1)^2 \equiv ArrayB[x := x+1][k := k+1]$
- **Bound** and **free** variables
 - Note that in $ArrayB$, substituting for j doesn't make sense.
 - Example: What would $(\forall j: 0 < j < k \rightarrow \dots)[j := j+1]$ mean?
 - The difference is that j is "bound" to the $\forall j$ but the occurrences of k and x are "free", and we only substitute for free occurrences of a variable.
- Free and bound occurrences
 - An **occurrence of a variable is free** if the occurrence is not within the scope of a quantifier over the variable. The **occurrence of a variable is bound** if it is within the scope of a quantifier over the variable.
- Examples:
 - In $y < 0 \rightarrow \forall x: x * x \geq y$, the two occurrences of x are both bound but the two occurrences of y are both free.
 - In $x > 1 \wedge \forall x: x * x \geq 0$, the first occurrence of x is free; the second and third are bound.
- Free and bound variables
 - A **variable is free** in a predicate (= **is a free variable** of the predicate) if it has a free occurrence. A **variable is bound** in a predicate (= **is a bound variable** of the predicate) if it has a bound occurrence.
- Examples:
 - In $y < 0 \rightarrow \forall x: x * x \geq y$, y is free and x is bound.
 - In $x > 1 \wedge \forall x: x * x \geq 0$, x is **both free and bound**.

- Substitution into quantified predicates
 - Below we use \forall , but the rules are equally valid for \exists .
 - $(\forall x: p)[x := t] \equiv \forall x: p$.
 - We only substitute for free occurrences of x , and there aren't any.
 - $(\forall x: p)[u := t] \equiv \forall x: (p[u := t])$ if $u \neq x$ and x does not occur in t .
 - Example: $(\forall x: x = u \vee x \neq u)[u := y-z]$ is $\forall x: ((x = u \vee x \neq u)[u := y-z])$ is $(\forall x: x = y-z \vee x \neq y-z)$
 - The hard case is $(\forall x: p)[u := t]$ where $u \neq x$ but x occurs (free) in t . (The quantifier would “capture” the free occurrences of x in t and make them bound.)
 - To handle it, first we **rename the bound variable** x to some unused (“fresh”) variable, say v . Then we use the previous case for substitution.
 - $(\forall x: p)$ becomes $(\forall v: p[x := v])$.
 - So $(\forall x: p)[u := t]$ becomes $(\forall v: p[x := v])[u := t]$, which $\equiv (\forall v: p[x := v][u := t])$
 - Example: $(\exists x: x < y)[y := x-3] \equiv (\exists v: (x < y)[x := v])[y := x-3]$
 $\equiv (\exists v: v < y)[y := x-3] \equiv \exists v: v < x-3$
 - Example:

$$(y \geq 2 \wedge \exists x: y < x \wedge x < y * y)[y := x/2]$$

$$\equiv (y \geq 2)[y := x/2] \wedge (\exists x: y < x \wedge x < y * y)[y := x/2]$$

$$\equiv x/2 \geq 2 \wedge (\exists z: y < z \wedge z < y * y)[y := x/2] \quad (\text{rename } \exists x \text{ to } \exists z)$$

$$\equiv x/2 \geq 2 \wedge \exists z: x/2 < z \wedge z < x/2 * (x/2)$$
 - Semantics of substituted expressions
 - There's a nice connection between the values of expressions s and $s[u := t]$:
 - **Substitution Lemma:** $\sigma(s[u := t]) = \sigma[u := \sigma(t)](s)$.
 - The value of s with t for u is the value of s if we update u to the value of t .
 - This can be proved by induction on the structure of s ; the text gives details.
 - Example: $\sigma((2 * x)[x := y])$ should $= \sigma[x := \sigma(y)](2 * x)$.
 - $\sigma((2 * x)[x := y]) = \sigma(2 * y) = 2 * \sigma(y)$
 - $\sigma[x := \sigma(y)](2 * x) = \sigma[x := \sigma(y)](2) * \sigma[x := \sigma(y)](x) = 2 * \sigma(y)$.
 - There's also a substitution lemma for predicates: $\sigma \models p[u := t]$ iff $\sigma[u := \sigma(t)] \models p$.
 - Again, one can prove this by induction on the structure of p ; the text gives details.

G. Formal Proofs and Proof Rules

- Now that we have a good idea of the semantics of correctness triples, we can develop proof rules for them.
- A formal proof is a syntactic object, with syntax rules.
- Various ways to write out proofs.

- Hilbert-style proof: List of judgments and justifications. (High-school geometry)
 1. Length of AB = length of XY Assumption
 2. Angle ABC = Angle XYZ Assumption
 3. Length of BC = length of YZ Assumption
 4. Triangles ABC, XYZ are congruent Side-angle-side, lines 1, 2, 3
- Each line's assertion is an assumption, an axiom, or follows by some rule that appeals to earlier lines in the proof.
- Notation: $\vdash \{p\} S \{q\}$ = "We can prove (that) $\{p\} S \{q\}$ (is valid)"
 - The \vdash symbol is a single turnstile.
- Formally, a proof system over a set of formulas is determined by a set of axioms and proof rules.

H. Proof System PD [Partial correctness of Deterministic programs]

Axiom 1 (Skip)

- $\vdash \{p\} \text{skip} \{p\}$ (for every assertion p)
- To argue very formally that this is reasonable, you'd prove that for all states σ , $\sigma \models \{p\} \text{skip} \{p\}$

Axiom 2 (Assignment)

- The assignment rule is complicated/subtle because it needs to connect the states before and after the assignment.
- Textbook presents the "backward" assignment rule.
 - $\vdash \{p[u := t]\} u := t \{p\}$
 - Looks weird but does work.
- Example 1:

$$\{b+1 \text{ odd}\} b := b+1 \{b \text{ odd}\}$$
- Example 2:

$$\{0 \leq i+1 \leq n, s = \text{sum}(0, i+1)\}$$

$$i := i+1$$

$$\{0 \leq i \leq n, s = \text{sum}(0, i)\}$$

Rule 6 (Consequence) [skipping ahead a bit]

$$\frac{p_1 \rightarrow p_2 \quad \{p_2\} S \{q_1\} \quad q_1 \rightarrow q_2}{\{p_1\} S \{q_2\}}$$

- The consequence rule is a helper rule (it's not connected with a particular kind of statement). It allows you to strengthen preconditions and weaken postconditions.
- **Definition:** p_1 is **stronger than** p_2 iff p_2 is **weaker than** p_1 iff $p_1 \rightarrow p_2$.
- The consequence rule lets you simultaneously strengthen the precondition and weaken the postcondition. Often, we want to do one of these but not both. We can derive two simpler rules to do this.

Weaken Postcondition

$$\frac{\{p\} S \{q_1\} \quad q_1 \rightarrow q_2}{\{p\} S \{q_2\}}$$

Strengthen Precondition

$$\frac{p_1 \rightarrow p_2 \quad \{p_2\} S \{q\}}{\{p_1\} S \{q\}}$$

- Whichever version of the rule you use, at least one of the antecedents (above the line) is a predicate calculus formula (an implication).
- These implications get listed in the proof; each one introduces an obligation for a proof using everyday predicate logic (which we'll almost always omit).
- Example:
 - From $\mathbf{true} \rightarrow 0 = 0$ and $\{0 = 0\} i := 0 \{i = 0\}$ we can get $\{\mathbf{true}\} i := 0 \{i = 0\}$ by strengthening the precondition

Rule 3 (Composition):

$$\frac{\{p\} S_1 \{r\} \quad \{r\} S_2 \{q\}}{\{p\} S_1 ; S_2 \{q\}}$$

- Antecedents above the line, conclusion below. (“If you can prove $\{p\} S_1 \{r\}$ and you can prove $\{r\} S_2 \{q\}$, then you can prove $\{p\} S_1 ; S_2 \{q\}$.”)
- Example

$$\frac{\{0 = 0 \wedge 0 = 0\} i := 0 \{i = 0 \wedge i = 0\} \quad \{i = 0 \wedge i = 0\} s := i \{i = 0 \wedge s = 0\}}{\{0 = 0 \wedge 0 = 0\} i := 0; s := i \{i = 0 \wedge s = 0\}}$$

- In a formal proof, the format would be

Line	Claim	Rule
1	$\{0 = 0 \wedge 0 = 0\} i := 0 \{i = 0\}$	Assignment
2	$\{i = 0 \wedge i = 0\} s := i \{i = 0 \wedge s = 0\}$	Assignment
3	$\{\mathbf{true}\} i := 0; s := 0 \{i = 0 \wedge s = 0\}$	Composition, lines 1, 2

Rule 4 (Conditional):

$$\frac{\{p \wedge B\} S_1 \{q\} \quad \{p \wedge \neg B\} S_2 \{q\}}{\{p\} \mathbf{if} B \mathbf{then} S_1 \mathbf{else} S_2 \mathbf{fi} \{q\}}$$

- If you know that:
 - Running the **true** branch S_1 in a state satisfying p and the **if** test establishes q
 - And running the **false** branch S_2 in a state satisfying p and the negation of the **if** test establishes q
- Then you know that running the **if-else** in a state satisfying p establishes q .
- Example:

$$\frac{\{\mathbf{true} \wedge x \geq 0\} y := x \{y \geq 0\} \quad \{\mathbf{true} \wedge x < 0\} y := -x \{y \geq 0\}}{\{\mathbf{true}\} \mathbf{if} x \geq 0 \mathbf{then} y := x \mathbf{else} y := -x \mathbf{fi} \{y \geq 0\}}$$

- There's no separate rule for an **if-then**: Treat it as an **if-else skip**.

Rule 5 (Loop)

- To understand the rule, we'll look at **loop invariants** after presenting the rule.

$$\frac{\{p \wedge B\} S_1 \{p\}}{\{p\} \mathbf{while} B \mathbf{do} S_1 \mathbf{od} \{p \wedge \neg B\}}$$

- Example: (where $p \equiv 0 \leq i \leq n \wedge s = \text{sum}(0, i)$)

$$\frac{\{p \wedge i < n\} i := i+1; s := s+i \{p\}}{\{p\} \mathbf{while} i < n \mathbf{do} i := i+1; s := s+i \mathbf{od} \{p \wedge i \geq n\}}$$

Next time: Continue with loop invariants, larger examples and proof outlines.