<u>Syntactic Substitution</u>

- In backward assignment rule, we replace $P(v)$ by $P(e)$ to get the wlp for statement $v := e$ and postcondition $P(v)$. The operation of going from $P(v)$ to $P(e)$ is called **syntactic substitution**.
    - o This substitution procedure is totally textual.
    - o We have learned how to update a state, we substitute the value that's bound with variable with another value; syntactic substitution is similar, but we substitute something syntactic with some other thing that's syntactic.

(Substitution in an Expression)

- **Substitution in an expression** is the following operation: take an expression e and replace its occurrence of variable v with expression e', written as $e[e'/v]$ and pronounced "e with e' for v".
    - o Like state updating, the squared parenthesis has very high hierarchy and when there are multiple substitutions, we calculate from left to right.

1. Finish the following substitution in expressions.
    a. $(x + y)[5 / x] \equiv 5 + y$
    b. $x + y\,[5 / x] \equiv x + y$
    o Substitution has very high hierarchy.
    o If there is no variable v in e, then $e[e'/v] \equiv e$.

    c. $(x + x)[2 / x] \equiv 2 + 2$
    o The substitution operation is done; and $(x + x)[2/x] \not\equiv 4$.

    d. $\big(x * (x + 1)\big)\,[b - c/x\,] \equiv (b - c) * \big((b - c) + 1\big)$
    e. $(b\,[x * y])\,[x + 3/x] \equiv b[(x + 3) * y]$
    f. $(\textbf{if } x > 0 \textbf{ then } -x \textbf{ else } 0 \textbf{ fi})[z + 2 / x] \equiv \textbf{if } z + 2 > 0 \textbf{ then } -(z + 2) \textbf{ else } 0 \textbf{ fi}$
    g. $(b[x * (x + 1) \div 2\,])\,[y + 4 / x] \equiv b[(y + 4) * \big((y + 4) + 1\big) \div 2\,]$

- In general, $e[e'/v]$ can be calculated using the following recursive algorithm:

$$e[e'/v] \equiv \begin{cases} c, & \text{if } e \text{ is a constant } c \\ e', & \text{if } e \equiv v \\ e, & \text{if } e \text{ is a variable and } e \not\equiv v \\ op\ (e_1[e'/v], e_2[e'/v], \dots), & \text{if } e \equiv op(e_1, e_2, \dots) \\ f(e_1[e'/v], e_2[e'/v], \dots), & \text{if } e \equiv f(e_1, e_2, \dots) \\ b[op\ (e_1[e'/v], e_2[e'/v], \dots)], & \text{if } e \equiv b[op(e_1, e_2, \dots)] \end{cases}$$

(Substitution in a Predicate)

- **Substitution in a predicate** is the following operation: take a predicate p and replace each (*free*) occurrence of variable v with expression e', written as $p[e'/v]$ and pronounced "p with e' for v".
    - o We don't need to worry about the word "free" if the predicate is not quantified.
    - o Again, the substitution is textual, so $(x > 0)[1 / x] \equiv 1 > 0$, but $(x > 0)[1 / x] \not\equiv T$.

- For a predicate that is not quantified, we substitute recursively in its sub-predicates or expressions. Note that, a non-quantified predicate might contain a quantified sub-predicate, such as $y < 0 \rightarrow \exists x. x \geq y$.

2. $(x > 0 \rightarrow y \geq x \mathbin{/} 2)[z + 1 \mathbin{/} x] \equiv (x > 0)[z + 1 \mathbin{/} x] \rightarrow (y \geq x \mathbin{/} 2)[z + 1 \mathbin{/} x]$
$$\equiv z + 1 > 0 \rightarrow y \geq (z + 1)/2$$

- Let Q stand for a quantifier $\forall$ or $\exists$. In our intuition, $(Qx. q)\, [e \mathbin{/} v]$ should be syntactically equivalent to $Qx. (q[e \mathbin{/} v])$, but in fact this is not always true. We need to consider whether a variable is free or bound.

- If an occurrence of a variable v in a predicate is within the scope of a quantifier over v, then it is a **bound occurrence**, else it is a **free occurrence**.
  - A variable v **is free in** p iff it has a free occurrence in p. Similarly, v **is bound in** p iff it has a bound occurrence in p.

3. Decide for each of the following variables, whether it is free or bound in p, where $p \equiv x > z \wedge \exists x. \exists y. y \leq f(x, y)$.
   a. Variable x is free and bound in p.
   b. Variable y is bound in p.
   c. Variable z is free in p.
   d. Variable u is neither free nor bound in p.

   - From the above example we can see that each occurrence of some variable can be either free or bound. But for a variable, it can be both/either/neither free and/or/nor bound in a predicate.

- To syntactic substitute in a quantified predicate, we **only substitute for free occurrences**.
  - But there still are other problems. For example, if we have predicate $\exists x. x = v^2$ and we want to replace v by $y + 1$, then $(\exists x. x = v^2)[y + 1 \mathbin{/} v] \equiv \exists x. x = (y + 1)^2$, and it looks correct to us. However, if we let $(\exists x. x = v^2)[x + 1 \mathbin{/} v] \equiv \exists x. x = (x + 1)^2$, it totally changes the semantic of this predicate; we don't want to have such a substitution.

- To $(Qx. q)[e \mathbin{/} v]$, there are three different cases:
  - Case 1: $x \equiv v$.
  - Case 2: $x \not\equiv v$, and x does not appear in e.
  - Case 3: $x \not\equiv v$, and x appears in e.

- For Case 1, we don't need to do anything, since we don't substitute for a bound occurrence: $(Qv. q)[e \mathbin{/} v] \equiv Qv. q$.
- For Case 2, we just need to need substitute the free occurrences of v in q: $(Qx. q)[e \mathbin{/} v] \equiv Qx. (q[e \mathbin{/} v])$

4. Finish the following substitution in predicates.
   a. $(x > 0 \wedge \exists x. x \leq f(y))[17 \mathbin{/} x] \equiv 17 > 0 \wedge \exists x. x \leq f(y)$.

   b. $(y \geq 0 \rightarrow \forall x. x > y \rightarrow x * x > y \wedge \exists y. f(y) > x)\, [17 \mathbin{/} y]$
   $\equiv 17 \geq 0 \rightarrow \forall x. (x > y \rightarrow x * x > y \wedge \exists y. f(y) > x)[17 \mathbin{/} y]$
   $\equiv 17 \geq 0 \rightarrow \forall x. (x > 17 \rightarrow x * x > 17 \wedge \exists y. f(y) > x)$

- For Case 3, we need to rename the quantified variable from x to something **fresh variable** (such as z in this example): $(Qx. q)[e \mathbin{/} v] \equiv (Qz. q[z \mathbin{/} x])\, [e \mathbin{/} v] \equiv Qz. q[z \mathbin{/} x][e \mathbin{/} v]$.

- o A fresh variable is defined to be a variable that is not in e or q. However, to avoid unexpected problems, if possible, simply choose some variable that is not used anywhere in the whole predicate.
- o For example: $(\exists x. x > 0 \rightarrow (\forall y. y + z < v^2) \wedge x > y)[y + 1 \,/\, v]$. When we only look at $(\forall y. y + z < v^2)[y + 1 \,/\, v]$, we are in case 3, and we can choose x as a fresh variable here to replace y; but x is used in other places in the predicate. To avoid future ambiguity, it is better to use variable w here.

5. $(g(x, v) < 0 \wedge (\exists x. x = v^2) \wedge h\,(y, v) > 0)\,[x + 1 \,/\, v]$
$$\equiv g(x, x + 1) < 0 \wedge (\exists z. (x = v^2)[z \,/\, x])\,[x + 1 \,/\, v] \wedge h\,(y, x + 1) > 0$$
$$\equiv g(x, x + 1) < 0 \wedge \exists z. (z = (x + 1)^2) \wedge h\,(y, x + 1) > 0$$

6. Let $member(x, b) \equiv \exists 0 \leq k < size(b). x = b[k]$.
   a. What is $member(12, b_1)$?
      In fact, $member(12, b_1) \equiv member(x, b)[12 \,/\, x][b_1 \,/\, b]$
$$\equiv (\exists 0 \leq k < size(b). x = b[k])[12 \,/\, x][b_1 \,/\, b]$$
$$\equiv (\exists 0 \leq k < size(b). 12 = b[k])[b_1 \,/\, b]$$
$$\equiv (\exists k. 0 \leq k < size(b) \wedge 12 = b[k])[b_1 \,/\, b]$$
$$\equiv \exists k. 0 \leq k < size(b_1) \wedge 12 = b_1[k]$$

   b. What is $member(k * c, b_2)$?
$$member(k * c, b_2) \equiv member(x, b)[k * c \,/\, x][b_2 \,/\, b]$$
$$\equiv (\exists 0 \leq k < size(b). x = b[k])[k * c \,/\, x][b_2 \,/\, b] \quad \text{\# case 3}$$
$$\equiv (\exists 0 \leq k < size(b). x = b[k])[k_0 \,/\, k][k * c \,/\, x][b_2 \,/\, b]$$
$$\equiv (\exists 0 \leq k_0 < size(b). x = b[k_0])[k * c \,/\, x][b_2 \,/\, b]$$
$$\equiv (\exists 0 \leq k_0 < size(b). k * c = b[k_0])[b_2 \,/\, b]$$
$$\equiv (\exists 0 \leq k_0 < size(b_2). k * c = b_2[k_0])$$

Forward Assignment Rule
- Quick review, with partially valid triple $\{Q(e)\}\ v := e\ \{Q(v)\}$ we got the backward assignment rule: $wlp(v := e, Q(v)) \equiv Q(e)$.
  - o With the knowledge of syntactical substitution, we can generalize the **backward assignment rule** as follows: $wlp(v := e, q) \equiv q[e \,/\, v]$.

- Now, we have another question. What can be used as a post condition for $\{p\}\ v := e\ \{?\}$ and what can be the strongest postcondition?
  - o Intuition tells us $\{p\}\ v := e\ \{p \wedge v = e\}$ looks true, and it does work in some situations; but in general, it is not correct.

7. Are the following triples valid (under either correctness)?
   a. $\{x > y\}\ z := 2\ \{x > y \wedge z = 2\}$.
      Yes. To justify it is valid, we can apply the backward assignment rule:
$$wlp(z := 2, x > y \wedge z = 2) \equiv (x > y \wedge z = 2)[2 \,/\, z] \equiv x > y \wedge (2 = 2) \Leftrightarrow x > y$$

   b. $\{x > 0\}\ x := x - 2\ \{x > 0 \wedge x = x - 2\}$.
      Clearly this is not valid, since $(x = x - 2) \Leftrightarrow F$.

   o Why $\{p\}\ v := e\ \{p \wedge v = e\}$ works in example 7.a but not 7.b? The difference is whether v has appears in e or p.

- We met situations like in 7.b before, what we did was to "record" the old value of v in precondition with a logical constant: $\{x > 0 \land x = x_0\}\ x := x - 2\ \{x_0 > 0 \land x = x_0 - 2\}$.

- **Aging** a variable x in the precondition is to introduce a logical constant to name the value of x before executing the statement.

- **[Forward Assignment Rule]** $p[v_0\ /\ v] \land v = e[v_0\ /\ v]$ is a postcondition for valid triple $\{p \land v = v_0\}\ v := e\ \{q\}$. Note that, even if we omit the $v = v_0$ in the precondition, it is understood. We can use an example to explain this.

8. Create a valid triple $\{x > 0 \land x = x_0\}\ x := x - 1\ \{q\}$.
    - With the forward assignment rule, we have $q \equiv x_0 > 0 \land x = x_0 - 1$; then we have:
    $$\{x > 0 \land x = x_0\}\ x := x - 1\ \{x_0 > 0 \land x = x_0 - 1\}$$

    - We can use existential to represent the postcondition to remove the aging of x in the precondition:
    $$\{x > 0\}\ x := x - 1\ \{\exists x_0.\ x_0 > 0 \land x = x_0 - 1\}$$

    - x_0 is fresh, it only appears in the postcondition; so, dropping of the quantifier and only keeping the body already implies the existence of such x_0. Then we have:
    $$\{x > 0\}\ x := x - 1\ \{x_0 > 0 \land x = x_0 - 1\}$$

- It is okay to omit the aging part in the precondition and use fresh logic variable x_0 for x in the postcondition directly to represent the old value of variable x.