Substitution continued and Strongest Precondition

CS 536: Science of Programming, Spring 2022

B. Refresher: Syntactic Substitution

- Recall that $wp(v := e, P(v)) \equiv P(e)$
- The operation of going from $P(v)$ to $P(e)$ is called syntactic substitution.
- A common notation is $[e/v]p$.

C. Refresher: Free and Bound Variables and Occurrences of Variables

- Notation: Q stands for a quantifier ($\forall$ or $\exists$).
- For the definition of $[e/v](Q\,x.q)$, our natural instinct is to think that $[e/v](Q\,x.q) \equiv (Q\,x.([e/v]q))$, but in fact this isn't always true because of a distinction between "free" and "bound" occurrences of variables.
- Definition: 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 (= occurs free in) p iff it has a free occurrence in p. Similarly, v is bound in (= occurs bound in) p iff it has a bound occurrence in p. (In computer science terms, local variables have bound occurrences, and non-local variables have free occurrences.)
- For any variable v and predicate p, there are four possibilities:
  - v is neither free nor bound in p (this case applies when v doesn't occur at all in p).
  - v is free but not bound in p: v occurs at least once in p, and all the occurrences of v are free.
  - v is not free but is bound in p: v occurs at least once in p, and all the occurrences of v are bound.
  - v is free and bound in p: v occurs at least twice in p with at least one occurrence being free and at least one occurrence being bound.
- Example 9: If $p \equiv x > z \wedge \exists x.\exists y.y \leq f(x,y)$, then
  - x is free and bound in p. (Its first occurrence is free; its second is bound.)
  - y is bound in p but not free in p.
  - z is free in p but not bound in p.
  - w is neither free nor bound in p.

- The reason we're interested in occurrences of variables being free or bound in a predicate is that we only substitute for free occurrences of a variable. In computer science terms, we're looking for non-local variables, not local variables.
- Taking polynomials as an example, $p(x) = x^2 + a\ x + y$. If we want to substitute 17 for y, that's fine: $p(x) = x^2 + a\ x + 17$; substituting expressions with variables that aren't bound in the definition is okay too: substituting $(z^3 + 1)$ for y gives us $p(x) = x^2 + a\ x + (z^3 + 1)$. But if we want to substitute something like $(x+3)$ for y (note: x is the defined parameter variable), we don't want $p(x) = x^2 + a\ x + (x+3)$. But if we had defined $p(w) = w^2 + a\ w + y$, then substituting $(x+3)$ for y gives us $p(w) = w^2 + a\ w + (x+3)$.

C. *Substitution Into A Quantified Predicate*

- In case 1 of the definition of substitution, the major operator of the predicate was not a quantifier, it was a conjunction or disjunction, etc.
- In the remaining cases, we substitute into a quantified predicate: $[e/v](Qx.q)$.

Substitution Case 2 (refresher): Quantified Variable ≡ Variable to Replace

- In the simplest quantifier case, the quantified variable matches the variable we're substituting for. I.e., we have $[e/v](Q\ v.q)$.
- Since all the occurrences in q of v are bound, there are no free occurrences of v in $Q\ v.q$, so there's nothing to replace: $[e/v](Qv.q) \equiv Q\ v\ .\ q$.
- Example 10: $[17/x]\ (x > 0\ \wedge\ \exists x.x \leq f(y)) \equiv 17 > 0\ \wedge\ \exists x.x \leq f(y)$. Here, the first occurrence of x (in $x > 0$) is free, so we replace it with 17, but the second occurrence of x is bound, so we don't do any replacement.

Substitution Case 3 (refresher): Quantified Variable Doesn't Occur in Replacement Expression

- If $x \not\equiv v$ and x does not occur in e, then $[e/v](Qx.q) \equiv (Q\ x.([e/v]q))$. Here, we go through q and replace its free occurrences of v with e.
- Example 11: $[17/y](\ y \geq 0 \rightarrow \forall x\ .\ x > y \rightarrow x * x > y\ \wedge\ \exists\ y.f(y) > x)\ )$
  $\equiv 17 \geq 0 \rightarrow \forall x\ .\ [17/y](x > y \rightarrow x * x > y\ \wedge\ \exists\ y.f(y) > x)$
  $\equiv 17 \geq 0 \rightarrow \forall x\ .\ x > 17 \rightarrow x * x > 17\ \wedge\ \exists\ y.f(y) > x$ (y in $f(y)$ is bound, so no substituting for it)
- In case 3, the restriction that the quantified variable not appear in e keeps us from having a "capture" problem, where occurrences of x in e are free, but when we we replace an occurrence of v by e in $Qx.[e/v]q$, the occurrences of x in e become bound, which changes their meaning.

- Example 11: $[x+1/v](\exists y.y = v^2) \equiv \exists y.y = (x+1)^2$. If we were to let $[x+1/v](\exists x.x = v^2)$ be $\exists x.x = (x+1)^2$, then the x in $x+1$ would become bound to the $\exists x$ (= the x would be "captured").
  - (Before the substitution, the x in $\ldots[x+1/v]$ was not quantified, so after the substitution, we also want x to not be quantified.)
- The way out of this problem is to rename the quantified variable from x to something not in e; that way the quantifier can't capture occurrences of x.

Substitution Case 4: Quantified Variable Does Occur in Replacement Expression

- This case is the most complicated one. If $x \not\equiv v$ and x occurs in e, then what we do is replace the quantified variable with one that doesn't appear in the quantifier's body. Then we proceed as in Case 3.
  - So, $[e/\mathrm{v}](\mathrm{Qx.q}) \equiv (\mathrm{Qz.}[e/\mathrm{v}][z/\mathrm{x}]\mathrm{q} \equiv (\mathrm{Qz.}([e/\mathrm{v}][z/\mathrm{x}]\mathrm{q}))$
    where z is a fresh variable (one not used in e or q).
    - Note here we're doing multiple substitutions: $[e/\mathrm{v}][z/\mathrm{x}]\mathrm{q}$ means "substitute z for x in q, and then substitute e for v in the result of the first substitution". We do substitutions from the inside out (right to left).
- Example 12: Using z as a fresh variable, we have

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

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

  // Pick fresh variable, quantify over it and then substitute for it in the body

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

  $\equiv g(x, x+1) < 0 \wedge (\exists z.z = (x+1)^2 \wedge h(y, x+1) > 0$
  - Note there's some ambiguity in the definition: Which "fresh" variable should we choose?

D. *The Strongest Postcondition (sp)*

- Definition: Given a precondition p and program S, the strongest postcondition of p and S is *(*the predicate that stands for*)* the set of states we can terminate in if we run S starting in a state that satisfies p. In symbols,
  - $sp(p, S) = \{\tau \mid \tau \in M(S, \sigma)/\perp$ for some σ where $\sigma \vDash p\}$.
  - Equivalently, $sp(p, S) = \bigcup (M(S, \sigma) - \perp)$ where $\sigma \vDash p$.
- I.e., $sp(p, S)$ is the image of $M(S, \ldots) - \perp$ (considered as a function) over the states that satisfy p.

- Figure 1 shows the relationship between p, S, and $sp(p, S)$:
  - If $\sigma \models p$, then every state in $M(S, \sigma) - \bot$ is by definition in $sp(p, S)$, so $\models \{p\}\ S\ \{sp(p, S)\}$.
    - This is only valid for partial correctness: Starting in a state that satisfies p might yield $\bot$.
    - (To get total correctness, we need termination: $\models [p]\ S\ [sp(p, S)]$ iff $\models [p]\ S\ [T]$.)

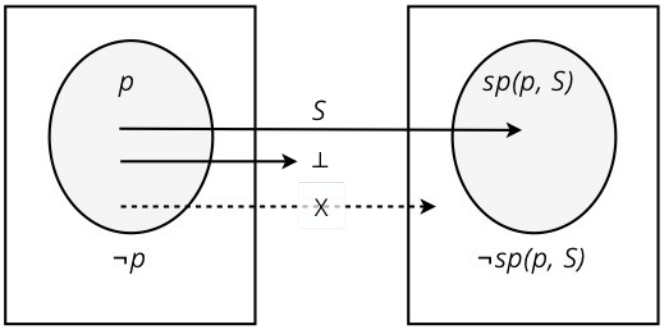

Figure 1: sp(p, S) is the set of states reachable via S from p

- Example 4: Let $W \equiv$ *while* $i \neq 0\ \{i := i\text{-}1\}$, then $sp(i \geq 0, W) \equiv i = 0$, and we can verify that $\{i \geq 0\}\ W\ \{sp(i \geq 0, W)\}$ is both partially and totally correct.
- Example 5: For the same W, weakening $i \geq 0$ produces the same sp. At the limit, $sp(T, W) \equiv i = 0$. Here, the sp is partially correct but not totally correct: $\models \{T\}\ W\ \{sp(T, W)\}$ but $\not\models [T]\ W\ [sp(T, W)]$. Of course, this is because W doesn't terminate starting with $i < 0$.
- The strongest postcondition: For partial correctness, $sp(p, S)$ is a postcondition. What makes it the *strongest* postcondition is that it implies any other postcondition: for general q, $\models \{p\}\ S\ \{q\}$ iff $sp(p, S) \Rightarrow q$.
- Lemma: $\models \{p\}\ S\ \{q\}$ iff $sp(p, S) \Rightarrow q$.
  - The $\Leftarrow$ direction holds by postcondition weakening: We have $\models \{p\}\ S\ \{sp(p, S)\}$ and $sp(p, S) \Rightarrow q$, therefore $\models \{p\}\ S\ \{q\}$.
  - For the $\Rightarrow$ direction, assume $\models \{p\}\ S\ \{q\}$ and let $\tau \models sp(p, S)$. Since $\tau \models sp(p, S)$, we have $\tau \in M(S, \sigma) - \bot$ for some $\sigma \models p$. But $\sigma \models \{p\}\ S\ \{q\}$ tells us that $M(S, \sigma) - \bot \models q$, so $\tau \models q$. So $\tau \models sp(p, S)$ implies $\tau \models q$, so we have $sp(p, S) \Rightarrow q$.

E. Forward Assignment Rules

- We already have a "backwards" assignment rule, $\{[e/x]q\}\ x := e\ \{q\}$
- But, for sp, we need to go the other direction, $\{p\}\ x := e\ \{???\}$ — what can we use for the postcondition?
  - Most people's first guess is $\{p\}\ x := e\ \{p \wedge x = e\}$, which can work under certain conditions.

New Variable Introduction

- If x is a new (fresh) variable (doesn't appear free in p and doesn't appear in e) then $\{p\}\ x := e\ \{p \wedge x = e\}$.
  - For example, $\{x > y\}\ z := 2\ \{x > y \wedge z = 2\}$
- To justify this, using *wlp*, we know $\{[e/v](p \wedge v = e)\}\ v := e\ \{p \wedge v = e\}$.
  - Expanding, $[e/v](p \wedge v = e) \equiv [e/v]p \wedge e = [e/v]e$.
  - Since v is fresh, it doesn't occur in p or e, so $[e/v]p \equiv p$ and $[e/v]e \equiv e$. So we need $\{p \wedge e = e\}\ v := e\ \{p \wedge v = e\}$, which definitely holds.

Forward Assignment (General Case)

- As an example of why $\{p\}\ v := e\ \{p \wedge v = e\}$ doesn't work in general, consider $\{x > 0\}\ x := x - 2\ \{???\}$.
  - We certainly don't have $\{x > 0\}\ x := x - 2\ \{x > 0 \wedge x = x - 2\}$. If we look more carefully, the relationship we're trying to capture with $x > 0 \wedge x = x - 2$ is:

(value of x before asgt) > 0 ∧ (the current value of x) = (value of x before asgt) – 2

  - Since subtraction is invertible, we can model this with $(x + 2 > 0 \wedge x = (x + 2) - 2)$.
  - But not all assignments are invertible: Consider $\{x > 0\}\ x := x/2\ \{???\}$. Because of truncating integer division, $(2*x > 0 \wedge x = (2*x/2))$ is only true for even values of x.
  - What we can do instead is to introduce a name for *(the value of x before the assignment)*. We've seen this before: it's a ghost variable !
  - If we use x_0 as this name, we can say $\{x_0 = x \wedge x > 0\}\ x := x/2\ \{x_0 > 0 \wedge x = x_0/2\}$.
- The general rule is $\{p \wedge v = v_0\}\ v := e\ \{[v_0/v]p \wedge v = [v_0/v]e\}$. In the precondition, we may be able to omit $v = v_0$ as being understood.
  - Example 1a: $\{x > 0 \wedge x = x_0\}\ x := x - 1\ \{x_0 > 0 \wedge x = x_0 - 1\}$.
  - Example 2a: $\{s = sum(0, i) \wedge s = s_0\}\ s := s + i + 1\ \{s_0 = sum(0, i) \wedge s = s_0 + i + 1\}$.

Correctness of The Forward Assignment Rule

- The forward assignment rule appears to be very different from our earlier "backward" assignment rule, but actually, we can derive the forward assignment rule using the backward assignment rule.
- Theorem (Correctness of Forward Assignment: $\models \{p \wedge v = v_0\}\ v := e\ \{[v_0/v]\ p \wedge v = [v_0/v]e\}$, where v_0 is a fresh logical constant.
- Proof:

- With backward assignment, we know $\{wlp(v := e, q)\}\ v := e\ \{q\}$ when $q \equiv [v_0/v]p \wedge v = [v_0/v]e\}$.
- If we show $(p \wedge v = v_0) \rightarrow wlp(v := e, q)$, then $\{p \wedge v = v_0\}\ v := e\ \{q\}$ will hold by precondition strengthening.
- Let's expand the *wlp* expression:

  $wlp(v := e, q) \equiv [e/v]q$
  $\equiv [e/v]([v_0/v]p \wedge v = [v_0/v]e)$
  $\equiv [e/v][v_0/v]\ p \wedge [e/v]v = [e/v][v_0/v]e$
- We can simplify this last predicate by looking at its three parts.
  - The easiest one is $[e/v]v \equiv e$; it follows immediately from the definition of substitution.
  - Since v doesn't appear in $[v_0/v]p$, we have $[e/v][v_0/v]p \equiv [v_0/v]p$, which is $\Leftrightarrow p \wedge v = v_0$.
  - Similarly, since v doesn't appear in $[v_0/v]e$, we have $[e/v][v_0/v]e \equiv [v_0/v]e$, so $[e/v]v = [e/v][v_0/v]e$ is $\equiv e = [v_0/v]e$ when $v = v_0$.
- So $([e/v][v_0/v]\ p \wedge [e/v]v = [e/v][v_0/v]e)$
  $\Leftrightarrow [v_0/v]\ p \wedge e = [v_0/v]e$
  $\Leftrightarrow p \wedge v = v_0 \wedge e = e$
  $\Leftrightarrow p \wedge v = v_0$
- So the forward assignment precondition implies the *wlp* of the assignment and post-condition, which completes the argument.

- For a particular example, with $\{x > 0 \wedge x = x_0\}\ x := x\text{-}1\ \{x_0 > 0 \wedge x = x_0\text{-}1\}$, we find $wlp(x := x\text{-}1, x_0 > 0 \wedge x = x_0\text{-}1) \equiv x_0 > 0 \wedge x\text{-}1 = x_0\text{-}1$, which is implied by $x > 0 \wedge x = x_0$.
- Note that the simpler rule for introducing a new variable is a special case of the general rule:
  - We want $\{p\}\ v := e\ \{p \wedge v = e\}$ if v doesn't occur in e and v is not free in p. The forward assignment rule says $\{p\}\ v := e\ \{[v_0/v]p \wedge v = [v_0/v]e\}$, where v_0 is a fresh logical constant. Since v does not occur in e, we know $[v_0/v]e \equiv e$. Similarly, since v isn't free in p, we know $[v_0/v]p \equiv p$. Substituting gives us $\{p\}\ v := e\ \{p \wedge v = e\}$.
- Not shown: So we've shown that the forward assignment rule can be derived if we have the backward assignment rule. Turns out the other direction is also true: You can derive the backward assignment rule from the forward assignment rule.

F. Strongest Postconditions for Loop-Free Programs

- It turns out (we won't prove this) that the forward assignment rule gives the strongest postcondition of the precondition and assignment, and we can use it as the base for calculating the strongest postcondition of a loop-free program:
  - $sp(p, skip) \equiv p$
  - $sp(p, v := e) \equiv [v_0/v]p \wedge v = [v_0/v]e$, where v_0 is a fresh constant.

- Note this isn't quite the same as our forward assignment rule: $v = v_0$ isn't in the precondition because we can't change the precondition.
- If the triple is {T} x:= x + 5 {$x = x_0 + 5$}, we now know that x is 5 more than *some* x_0 but we've lost the information that it's 5 more than what x was at the start of the program.
- Usually, that's not really information we care about. For example, the triple is more likely to be {$x > 0$} x:= x + 5 {$x_0 > 0 \wedge x = x_0 + 5$}, which tells us $x > 5$, which is more likely to be a fact we care about.
- Or, if we know even more about x, then, e.g., *sp(x = 1, x:= x + 5)*
  $= [x_0/x](x = 1) \wedge x = [x_0/x](x+5) = x_0 = 1 \wedge x = x_0 + 5 \Leftrightarrow x = 6$

- *sp(p, s_1; s_2) ≡ sp(sp(p, s_1),s_2)*
  - The most we can know after s_1 ; s_2 is the most we know after executing s_2 in the state that is the most we know after s_1.
- *sp(p, if e then {s_1} else {s_2}) ≡ sp(p ∧ e, s_1) ∨ sp(p ∧ ¬e, s_2)*
  - After executing an *if-else*, we know what we know after executing the true branch *(*after the test succeeded*)* or the false branch *(*after the test failed*)*, but we don't know which branch was taken.
  - For specific *p*, *e*, s_1, and s_2, we might be able to infer which branch was taken, but in the general case, we can't.

G. Examples of Strongest Postconditions

Example 6:

- Let's use *sp* to fill in the postcondition of $\{x > y\}\ x := x+k;\ y := y+k\ \{???\}$
- *(*Presumably, the postcondition will imply that *x* is still > *y.)*
- Let $p \equiv x > y$, then $sp(p, x := x + k;\ y := y + k) \equiv sp(sp(p, x := x + k), y := y + k)$.
- Let's calculate the inner *sp* first:

$$sp(p, x := x + k)$$
$$\equiv sp(x > y, x := x + k)$$
$$\equiv [x_0/x](x > y) \wedge x = [x_0/x]\,(x + k)$$
$$\equiv x_0 > y \wedge x = x_0 + k$$

- Then going back to our original problem and the outer *sp*, we get

$$sp(p, x := x + k;\ y := y + k)$$
$$\equiv sp(sp(p, x := x + k), y := y + k)$$
$$\equiv sp(x_0 > y \wedge x = x_0 + k, y := y + k)$$
$$\equiv [y_0/y]\,(x_0 > y \wedge x = x_0 + k) \wedge y = [y_0/y](y + k)$$
$$\equiv\ x_0 > y_0 \wedge x = x_0 + k \wedge y = y_0 + k$$

So our original triple can be filled in as

- $\{x > y\}\ \ x := x + k;\ y := y + k\ \ \{x_0 > y_0 \wedge x = x_0 + k \wedge y = y_0 + k\}$

Example 7:

- Let $p \equiv y = y_0$ and $S \equiv$ *if* $x \geq 0$ *then* $\{y := x\}$ *else* $\{y := -x\}$. Then
  - $sp(p \wedge x \geq 0, y := x) \equiv [y_0/y](x \geq 0) \wedge y = [y_0/y]x \equiv x \geq 0 \wedge y = x$
  - $sp(p \wedge x < 0, y := -x) \equiv [y_0/y](x < 0) \wedge y = [y_0/y]-x \equiv x < 0 \wedge y = -x$
- So $sp(y = y_0,$ *if* $x \geq 0$ *then* $\{y := x\}$ *else* $\{y := -x\})$

  $\equiv sp(p \wedge x \geq 0, y := x) \vee sp(p \wedge x < 0, y := -x)$

  $\equiv x \geq 0 \wedge y = x \ \vee \ x < 0 \wedge y = -x$
- Note that since the assignments to y don't depend on y_0. we don't find y_0 occurring in the result.

Example 9:

- Let $S \equiv$ *if* $even(x)$ *then* $\{x := x+1\}$ *else* $\{x := x+2\}$ and let $p \equiv x \geq 0$, then

  $sp(p, S) \equiv sp(p \wedge even(x), x := x+1) \vee sp(p \wedge odd(x), x := x+2)$
  - For the true branch,

    $sp(p \wedge even(x), x := x+1)$

    $\equiv [x_0/x](p \wedge even(x)) \wedge x = [x_0/x](x+1)$

    $\equiv [x_0/x](x \geq 0 \wedge even(x)) \wedge x = [x_0/x](x+1)$

    $\equiv x_0 \geq 0 \wedge even(x_0) \wedge x = x_0+1$
  - For the false branch,

    $sp(p \wedge odd(x), x := x+2)$

    $\equiv [x_0/x](p \wedge odd(x)) \wedge x = [x_0/x](x+2)$

    $\equiv [x_0/x](x \geq 0 \wedge odd(x)) \wedge x = x_0+2$

    $\equiv x_0 \geq 0 \wedge odd(x_0) \wedge x = x_0+2$
- So $sp(p, S)$

  $\equiv (x_0 \geq 0 \wedge even(x_0) \wedge x = x_0+1) \vee (x_0 \geq 0 \wedge odd(x_0) \wedge x = x_0+2)$
- If we want to logically simplify, we get $x_0 \geq 0 \wedge (even(x_0) \wedge x = x_0+1 \vee odd(x_0) \wedge x = x_0+2)$.
- That's about as far as we can simplify without losing information. For example, we *could* simplify to $(x_0 \geq 0 \wedge (x = x_0 + 1 \vee x = x_0 + 2))$ or to $(x \geq 1 \wedge odd(x))$, but whether we *want* to simplify or not depends on where we're using the *sp* in the overall proof of correctness for the program.

Example 10:

- Let $S \equiv$ *if* $x \geq y$ *then* $\{x := x-y\}$ *else* $\{y := y-x\}$
- Let $q_1 \equiv sp(x = x_0 \wedge y = y_0, S)$

  $\equiv sp(x = x_0 \wedge y = y_0 \wedge x \geq y, x := x-y) \vee sp(x = x_0 \wedge y = y_0 \wedge x < y, y := y-x)$

  $\equiv (y = y_0 \wedge x_0 \geq y \wedge x = x_0-y) \vee (x = x_0 \wedge x < y_0 \wedge y = y_0-x)$
- With q_1, if the first disjunct holds, then $x_0 \geq y_0$; if the second disjunct holds, then $x_0 < y_0$.

- It may seem weird that we introduced $x = x_0 \wedge y = y_0$ in the precondition, but if we didn't:
  - Let $q_2 \equiv sp(T, S)$
    $\equiv sp(x \geq y, x := x\text{-}y) \vee sp(x < y, y := y\text{-}x)$
    $\equiv (x_0 \geq y \wedge x = x_0\text{-}y) \vee (x < y_0 \wedge y = y_0\text{-}x)$
  - With q_2, if the first disjunct holds, we know $x_0 \geq y$, but not $x_0 \geq y_0$. Symmetrically, if the second disjunct holds, we know $x < y_0$, but not $x_0 < y_0$. (We certainly know $x_0 \geq y_0$ or $x_0 < y_0$, but we've lost the connections with them and the two disjuncts.)
  - To get this information, we'd need to "remember" $x = x_0$ in the precondition.

Example 11:

- If we have a sequence of assignments to one variable, then we introduce multiple logical variables to talk about its values at different times in the sequence.
- To complete $\{x > f(x, y)\}\ x := x+1;\ x := x*x\ \{???\}$, we'll calculate the strongest postcondition.
  - To talk about the original value of x, and the value of x between the two assignments, we need two different variables.
- We need $sp(x > f(x, y), S_1; S_2) \equiv sp(sp(x > f(x, y), S_1), S_2)$ where $S_1 \equiv x := x+1$ and $S_2 \equiv x := x*x$.

  $sp(x > f(x, y), S_1)$
  $\equiv sp(x > f(x, y), x := x+1)$
  $\equiv [x_0/x](x > f(x, y)) \wedge x = [x_0/x](x+1)$ (using x_0 as the fresh variable)
  $\equiv x_0 > f(x_0, y) \wedge x = x_0+1$

  $sp(sp(x > f(x, y), S_1), S_2)$
  $\equiv sp(x_0 > f(x_0, y) \wedge x = x_0+1, x := x*x)$
  $\equiv [x_1/x](x_0 > f(x_0, y) \wedge x = x_0+1) \wedge x = [x_1/x](x*x)$

  (using x_1 as the fresh variable)
  $\equiv x_0 > f(x_0, y) \wedge x_1 = x_0+1 \wedge x = x_1*x_1$
- We have to be careful with which variables we use where in the answer. In $x_0 > f(x_0, y)$, we need x_0 to talk about the value of x before the first assignment. Between the two assignments, we have the then-current $x = x_0+1$, but after the second assignment, we use x_1 to name the value of x between the two assignments because the now-current $x \neq x_1$. So after the second assignment we have $x_1 = x_0+1$ and $x = x_1*x_1$. But the $x_0 > f(x_0, y)$ clause remains the same because it talks about something that was true before the first assignment.