State Updates, Satisfaction of Quantified Predicates

CS 536: Science of Programming, Fall 2019

A. Why?

• A predicate is satisfied relative to a state; it is valid if it is satisfied in all states.
• State updates occur when we introduce new variables or change the values of existing variables.

B. Outcomes

At the end of this lecture, you should

• Know what it means to update a state.
• Know what it means for a quantified predicate to be valid or be satisfied in a state.

C. "Updating" States

• To check quantified predicates for satisfaction, we need to look at different states that are related to, but not identical to, our starting state.

• Example 1: For \( \{ y = 1 \} \models \forall x \in \mathbb{N}. x^2 + 1 \geq y - 1 \), we need to know that \( \{ y = 1, x = \alpha \} \models x^2 + 1 \geq y - 1 \) for every natural number \( \alpha \). I.e., we need
  
  • \( \{ y = 1, x = 0 \} \models x^2 + 1 \geq y - 1 \)
  • \( \{ y = 1, x = 1 \} \models x^2 + 1 \geq y - 1 \)
  • \( \{ y = 1, x = 2 \} \models x^2 + 1 \geq y - 1 \)
  • ….

  • Similarly, for \( \{ z = 4 \} \models \exists x \in \mathbb{N}. x \geq z \), we need \( \{ z = 4, x = \alpha \} \models x \geq z \) for some particular natural number \( \alpha \) (\( \alpha = 5 \) works nicely).

• There is a complicating factor. If the quantified variable already appears in the state, then we need to replace its binding with one that gives the value we’re interested in checking.

• Example 2: We already know \( \{ z = 4 \} \models \exists x \in \mathbb{N}. x \geq z \) because \( \{ z = 4, x = 5 \} \models x \geq z \). If we start with the state \( \{ z = 4, x = -15 \} \), which already has a binding for \( x \), we find that the new state \( \models \exists x \in \mathbb{N}. x \geq z \) because once again, \( \{ z = 4, x = 5 \} \models x \geq z \) holds.

• In Example 2, the \( x \) that appears in \( \{ z = 4, x = 5 \} \) is not the same \( x \) that appears within \( \exists x \in \mathbb{N}. x \geq z \). However, the two \( x \)’s in “\( \{ z = 4, x = 5 \} \models x \geq z \)” are the same. Giving the two \( x \)’s the same name causes the confusion. If we gave the \( x \)’s different names, there’d be no problem with understanding; let \( xo \) be the “outer” \( x \) and \( xi \) be the “inner” \( x \), then

\[
\{ z = 4, xo = -15 \} \models \exists xi \in \mathbb{N}. xi \geq z
\]

because

\[
\{ z = 4, xo = -15, xi = 5 \} \models xi \geq z
\]
• When we use the same name \( x \), the binding for the outer \( x \) becomes invisible, overridden by the binding for the inner \( x \):

\[
\{ z = 4, \ (outer) \ x = -15 \} \models \exists x \in \mathbb{N}. \ x \geq z \text{ because } \{ z = 4, \ x = 5 \} \models x \geq z
\]

• **Definition:** For any state \( \sigma \), variable \( x \), and value \( \alpha \), the **update of \( \sigma \) at \( x \) with \( \alpha \)** (written \( \sigma[x \mapsto \alpha] \)) is the state that is a copy of \( \sigma \) except that it binds variable \( x \) to value \( \alpha \).

• Let \( \tau = \sigma[x \mapsto \alpha] \), then \( \tau(x) = \alpha \); if variable \( y \not\equiv x \), then \( \tau(y) = \sigma(y) \).

• Note \( \tau(x) = \alpha \) regardless of whether \( \sigma(x) \) is defined or not. If \( \sigma(x) \) is defined, its type and exact value are irrelevant.

• Set theoretically,

  • If \( x \) has no binding in \( \sigma \), then \( \sigma[x \mapsto \alpha] \) is \( \sigma \cup \{ x = \alpha \} \): It’s like \( \sigma \) but has been extended with \( x = \alpha \).

  • If \( x \) has a binding in \( \sigma \), say \( \sigma = \{ x = \beta \} \cup \sigma_0 \) where \( \sigma_0 \) is the rest of \( \sigma \), then \( \sigma[x \mapsto \alpha] = \sigma_0 \cup \{ x = \alpha \} \).

  It’s like \( \sigma \) but has the binding \( x = \alpha \), not \( x = \beta \). (Having two bindings for \( x \) would be illegal.)

• **Important:** Calling it the “update” of \( \sigma \) is kind of misleading because we’re not modifying \( \sigma \).

• Taking \( \sigma[x \mapsto \alpha] \) **does not do** an update in place; if we define \( \tau = \sigma[x \mapsto \alpha] \), then \( \sigma \) is still \( \sigma \).

• Conceptually, we aren’t modifying \( \sigma \), we’re creating a new state.

• We’re not required to give \( \sigma[x \mapsto \alpha] \) a new name; we can write it out explicitly:

  • If \( x \equiv y \), then \( \sigma[x \mapsto \alpha](y) = \alpha \); otherwise (if \( x \not\equiv y \)), then \( \sigma[x \mapsto \alpha](y) = \sigma(y) \).

  • (You have to read \( \sigma[x \mapsto \alpha](y) \) left-to-right — we’re taking the function \( \sigma[x \mapsto \alpha] \) and applying it to \( y \).

  I.e., \( \sigma[x \mapsto \alpha](y) = (\sigma[x \mapsto \alpha])(y) \), where the left pair of parentheses are for grouping and the ones around \( y \) are for the function call.)

• **Example 3:** If \( \sigma = \{ x = 2, \ y = 6 \} \), then \( \sigma[x \mapsto 0] = \{ x = 0, \ y = 6 \} \):

  • \( \sigma[x \mapsto 0](x) = 0 \) (Even though \( \sigma(x) = 2 \))

  • \( \sigma[x \mapsto 0](y) = \sigma(y) = 6 \) (Since we didn’t update \( y \))

  • \( \sigma[x \mapsto 0](x+y) = 0+6 = 6 \) (Since the \( x \) in \( x+y \) gets evaluated to 0)

  • \( \sigma[x \mapsto 0] \models x^2 \leq 0 \) (Even though our starting \( \sigma \not\equiv x^2 \leq 0 \))

• The value part of an update has to be a semantic value, not a syntactic one, so \( \sigma[x \mapsto x+1] \) isn’t well-formed.

  • In these notes, it may help to remember that since \( x+1 \) is **in this font**, it’s syntactic.

  • On the other hand, “\( \sigma[x \mapsto \sigma(x+1)] \)” or “\( \sigma[x \mapsto \alpha \text{ plus one} \)” where \( \alpha = \sigma(x) \)” do make sense.

**Multiple Updates**

• We can do a sequence of updates on a state. E.g., \( \sigma[x \mapsto 0][y \mapsto 8] \) is a doubly updated state. Sequences of updates are read left-to-right, so this is \( (\sigma[x \mapsto 0])[y \mapsto 8] \).

• **Example 4:** If \( \sigma = \{ x = 2, \ y = 6 \} \), then \( \sigma[x \mapsto 0][y \mapsto 8] = \{ x = 0, \ y = 6 \}[y \mapsto 8] = \{ x = 0, \ y = 8 \} \).

• The order of update doesn’t matter if you have two different variables.

• **Example 5:** \( \sigma[x \mapsto 0][y \mapsto 8] = \sigma[y \mapsto 8][x \mapsto 0] \).

---

* Unfortunately, “update” is the traditional name, and for myself, I can't find any word that's exactly right. We're not always **extending** \( \sigma \), we're not always **superseding** \( \sigma \), ....
If you update the same variable twice, the second update supersedes the first.

**Example 6:** \[ \sigma[x \mapsto 0][x \mapsto 17] = \sigma[x \mapsto 17] \neq \sigma[x \mapsto 17][x \mapsto 0] = \sigma[x \mapsto 0] \]

Of course, if the second update is identical to the first, nothing happens: \[ \sigma[x \mapsto \alpha][x \mapsto \alpha] = \sigma[x \mapsto \alpha] \]

If you have to evaluate an expression, be sure to do it in the correct state.

- Let \( \sigma(x) = 1 \) and let \( \tau = \sigma[x \mapsto 2] \), then \( \tau[z \mapsto \sigma(x)+10] \) maps \( z \) to \( \sigma(x)+10 = 1+10 = 11 \). We can omit \( \tau \) and also write \( \sigma[x \mapsto 2][z \mapsto \sigma(x)+10] \), which gives the same state as \( \tau \).
- On the other hand, \( \tau[z \mapsto \tau(x)+10] \) maps \( z \) to \( \tau(x)+10 = 2+10 = 12 \). Here, if we don’t give a name to \( \sigma[x \mapsto 2] \), then we can’t write \( \tau[z \mapsto \tau(x)+10] \) so we have to write \( \sigma[x \mapsto 2][z \mapsto \sigma[x \mapsto 2](x)+10] \). (This is pretty ugly, so giving \( \sigma[x \mapsto 2] \) a name like \( \tau \) makes things more readable.)

### D. Updating Array Values

- Updating array elements like \( b[0] \) is a bit more complicated than updating simple variables like \( x \) and \( y \). First, let’s extend our notion of updating states to updating general functions.

**Definition:** If \( \varphi \) is a function on one argument and \( \alpha \) and \( \beta \) are valid members of the domain and range of \( \varphi \) respectively, then the **update of \( \varphi \) at \( \alpha \) with \( \beta \)**, written \( \varphi[\alpha \mapsto \beta] \), is the function defined by \( \varphi[\alpha \mapsto \beta](\gamma) = \beta \) if \( \gamma = \alpha \) and \( \varphi[\alpha \mapsto \beta](\gamma) = \varphi(\gamma) \) if \( \gamma \neq \alpha \).

**Definition:** If \( \sigma \) is a (proper) state for an array \( b \) and \( \alpha \) is a valid index value for \( b \), then \( \sigma[b[\alpha] \mapsto \beta] \) means \( \sigma[b \mapsto \gamma[\alpha \mapsto \beta]] \) where \( \gamma \) is the function \( \sigma(b) \). In words, if \( \sigma \) includes the binding \( b = \text{function} \gamma \), then the updating \( \sigma \) at \( b[\alpha] \) with \( \beta \) is just like updating \( \sigma \) at \( b \) with an updated version of \( \gamma \), namely \( \gamma[\alpha \mapsto \beta] \).

**Example 7:** Say \( \sigma = \{ x = 3, b = (2, 4, 6) \} \), then \( \sigma[b[0] \mapsto 8] = (x = 3, b = (8, 4, 6)) \). Here, \( \sigma(b) \) is the function \( (2, 4, 6) \) (which means \( \{(0, 2), (1, 4), (2, 6)\} \)), so \( \sigma(b)[0 \mapsto 8] \) (the update of function \( \sigma(b) \)) is the function \( (2, 4, 6)[0 \mapsto 8] = (8, 4, 6) \).

### E. Satisfaction of Quantified Predicates

- One use of updated states is for describing how assignment works. (We’ll see this later.) The other use for updated states is for defining when quantified predicates are satisfied.

**Definition:** \( \sigma \models \exists x \in S. \varphi \) if for one or more witness values \( \alpha \in S \), it’s the case that \( \sigma[x \mapsto \alpha] \models \varphi \). Note we’re asking a hypothetical question: “If we were to calculate \( \sigma[x \mapsto \alpha] \), would we find that it satisfies \( \varphi \)?”

**Example 8a:** For any state \( \sigma \), we can show \( \sigma \models \exists x. x^2 \leq 0 \) using 0 as the witness: \( \sigma[x \mapsto 0] \models x^2 \leq 0 \), since \( \sigma[x \mapsto 0](x^2) \leq \sigma[x \mapsto 0](0) = (0^2 \leq 0) = T \).

- Remember, \( \sigma(x) \) is irrelevant, since \( \sigma[x \mapsto \alpha] \) overrides any value for \( \sigma(x) \).

**Example 8b:** If \( \sigma(x) \) is, say 5, it’s still the case that \( \sigma \models \exists x. x^2 \leq 0 \) using 0 as the witness because we \( \sigma[x \mapsto 0] \models x^2 \leq 0 \), regardless of \( \sigma(x) = 5 \).

- If there are many successful witness values, we don’t have to specify all of them; we just need one.

**Example 12:** If \( \sigma(y) = 3 \), then \( \sigma \models \exists x. x^2 \leq y \) with \( x = 0 \) or 1 as possible witness values.
• **Definition**: \( \sigma \models \forall x \in S, p \) if for every value \( \alpha \in S \), we have \( \sigma[x \mapsto \alpha] \models p \). (Again, this is hypothetical: “If for every \( \alpha \), we were to calculate \( \sigma[x \mapsto \alpha] \), would we find that it satisfies \( p \)?”

• **Example 10**: To know \( \sigma \models \forall x \in \mathbb{Z}, x^2 \geq x \), we need to know \( \sigma[x \mapsto \alpha] \models x^2 \geq x \) for every \( \alpha \in \mathbb{Z} \).
Since for every integer \( \alpha \), indeed \( \alpha^2 \geq \alpha \), this does hold. Recall that it doesn’t matter what \( \sigma(x) \) is, since we’re interested in \( \sigma[x \mapsto \alpha] \).

• When asking if \( \sigma \) satisfies \( \forall x \in S, q \) or \( \exists x \in S, q \), we don’t care about \( \sigma(x) \). For a predicate \( p \) in general, for the question “Does \( \sigma \models p ? \)” only depends on how \( \sigma \) operates on the non-quantified variables of \( p \).

• **Example 11**: Since the body of \( \forall x \in \mathbb{Z}, x^2 \geq x \) uses only the quantified variable \( x \), it doesn’t matter what bindings \( \sigma \) has when checking \( \sigma \models \forall x \in \mathbb{Z}, x^2 \geq x. \) Even \( \sigma = \emptyset \) works: \( \emptyset \models \forall x \in \mathbb{Z}, x^2 \geq x \).

• Note with nested quantifiers, the notation does get more complicated.

• **Example 12**: \( \sigma \models \forall x \geq y^2, \exists z, z > x + y^2 \) iff (for every \( \alpha \in \mathbb{Z}, \) if \( \alpha \geq \sigma(y)^2 \), then there is some \( \beta \in \mathbb{Z} \) such that \( \beta > \alpha + \sigma(y)^2 \).

\[
\sigma \models \forall x \geq y^2, \exists z, z \geq x + y^2
\]
iff \( \sigma \models \forall x, x > y^2 \rightarrow \exists z, z \geq x + y^2 \) \quad defn bounded \( \forall \)
iff for every \( \alpha \in \mathbb{Z}, \sigma[x \mapsto \alpha] \models x > y^2 \rightarrow \exists z, z \geq x + y^2, \) \quad defn \( \models \forall \)

Now, \( \sigma[x \mapsto \alpha] \models x > y^2 \) implies \( \sigma[x \mapsto \alpha] \models \exists z, z \geq x + y^2 \)
iff \( \sigma[x \mapsto \alpha] \models x > y^2 \) implies \( \sigma[x \mapsto \alpha] \models \exists z, z \geq x + y^2 \) \quad defn \( \models \rightarrow \)
iff \( \sigma[x \mapsto \alpha] \models \exists z, z \geq x + y^2 \) \quad defn \( \models \exists \)
iff \( \alpha > \sigma(y)^2 \) implies for some \( \beta, \sigma[x \mapsto \alpha] [z \mapsto \beta] \models z \geq x + y^2 \)
iff \( \alpha > \sigma(y)^2 \) implies for some \( \beta, \beta \geq \alpha + \sigma(y)^2 \)
iff \( \alpha > \sigma(y)^2 \) \quad defn \( \models \geq \)

Taking \( \beta = 2\alpha \) for our witness value, we need \( \alpha > \sigma(y)^2 \) implies for some \( 2\alpha \geq \alpha + \sigma(y)^2 \), which is true.
Note defining intermediate names like ”let \( \tau = \sigma[x \mapsto \alpha] [z \mapsto \beta] \)” is allowed, if you prefer that style.

**Justifying DeMorgan’s Laws for Quantified Predicates**

• In general, we want our systems of reasoning to be **sound**: We want the textual transformations that make up logical equivalence to reflect truths about how our semantics work.

• **Example 15**: Here is a check of DeMorgan’s law for existentials, which says \( \neg \exists x, p \equiv \forall x, \neg p \).
Semantically, we want each of these to be valid if and only if the other is. So we need \( \sigma \models \neg \exists x, p \) if and only if \( \sigma \models \forall x, \neg p \).

\[
\sigma \models \neg \exists x \in S, p
\]
iff \( \sigma \not\models \exists x, p \) \quad defn of \( \sigma \models \neg \)-predicate
iff for no \( \alpha \in S \) do we have \( \sigma[x \mapsto \alpha] \models p \) \quad defn of \( \sigma \models \exists \)-existential
iff for every \( \alpha \in S \) we have \( \sigma[x \mapsto \alpha] \not\models p \) \quad equivalence of “no \( \models \)” vs “every \( \not\models \)”
iff for every \( \alpha \in S \) we have \( \sigma[x \mapsto \alpha] \models \neg p \) \quad defn of \( \sigma \models \neg \)-predicate
iff \( \sigma \models \forall x, \neg p \) \quad defn of \( \sigma \models \) universal.

• By using this property of \( \neg \exists \), we can get a short proof of soundness for the negation of a universal: For all \( \sigma \),
\[ \sigma \models \neg \forall x. p \]

iff \[
\sigma \models \neg (\forall x. \neg \neg p)
\]

iff \[
\sigma \models \neg (\neg \exists x. \neg p)
\]

iff \[
\sigma \models \exists x. \neg p
\]

double \(\neg\)

DeMorgan law (\(\neg \exists vs \forall \neg\))

double \(\neg\)
Satisfaction, Validity, and State Updates

CS 536: Science of Programming, Fall 2019

A. Why
• A predicate is satisfied or unsatisfied relative to a state.
• A predicate is valid if it is satisfied in all states.
• State updates occur when we introduce new variables or change the values of existing variables.

B. Outcomes
At the end of today, you should
• Know how to check a predicate for satisfaction in a state, how to check a predicate for validity, and know how to update a state.

C. Questions
1. Say $u$ and $v$ stand for variables (possibly the same variable) and $\alpha$ and $\beta$ are values (possibly equal). When is $\sigma[u \mapsto \alpha][v \mapsto \beta] = \sigma[v \mapsto \beta][u \mapsto \alpha]$? Hint: There are four cases because maybe $x \equiv y$, maybe $\alpha = \beta$.

2. Let $\sigma(b) = (7, 5, 12, 16)$.
   a. Does $\sigma \models \exists k. 0 \leq k \land k+1 < \text{size}(b) \land b[k] < b[k+1]$? If so, what was your witness values for $k$?
   b. Does $\sigma \models \exists k. 0 \leq k-1 \land k+1 < \text{size}(b) \land b[k-1] < b[k] < b[k+1]$? If so, what was your witness values for $k$?
   c. Does $\sigma \models \forall k. b[k] > 0$?
   d. If $\sigma(k) = -5$, then does $\sigma \models \exists k. b[k] > 0$?

3. For each of the situations below, fill in the blanks to describe when the situation holds.
   Fill in _____ 1 with “some”, “every”, or “this”
   Fill in _____ 2 with “some” or “every”,
   Fill in _____ 3 with “$\sigma(x)$ must be undefined”, “$\sigma(x)$ must be defined and $\sigma \models p$”, or “nothing of $\sigma(x)$”,
   Fill in _____ 4 with “$\models p$” or “$\not\models p$”.
   a. $\sigma \models (\exists x \in U. p)$ iff for _____ 1 state $\sigma$ and _____ 2 $\alpha \in U$, $\sigma[x \mapsto \alpha]$ _____ 4
   b. $\sigma \models (\forall x \in U. p)$ iff for _____ 1 state $\sigma$ and _____ 2 $\alpha \in U$, $\sigma[x \mapsto \alpha]$ _____ 4
   c. $\sigma \models (\exists x \in U. p)$ requires _____ 3.
   d. $\sigma \models (\forall x \in U. p)$ requires _____ 3.
   e. $\sigma \not\models (\exists x \in U. p)$ iff for _____ 1 state $\sigma$ for _____ 2 $\alpha \in U$, $\sigma[x \mapsto \alpha]$ _____ 4 $p$.
   f. $\sigma \not\models (\forall x \in U. p)$ iff for _____ 1 state $\sigma$ for _____ 2 $\alpha \in U$, $\sigma[x \mapsto \alpha]$ _____ 4 $p$.
   g. $\not\models (\forall x \in U. p)$ iff for _____ 2 state $\sigma$, we have $\sigma$ _____ 4 $(\forall x \in U. p)$.
   h. $\not\models (\exists x \in U. p)$ iff for _____ 2 state $\sigma$, we have $\sigma$ _____ 4 $(\exists x \in U. p)$.
   i. $\not\models (\forall x \in U. p)$ iff for _____ 2 state $\sigma$, and for _____ 2 $\alpha \in U$, we have $\sigma[x \mapsto \alpha]$ _____ 4.
j. $\forall (\exists x \in U . (\forall y \in V . p))$ iff for $\alpha \in U$, we have $\sigma[ x \mapsto \alpha][ y \mapsto \beta]$ ______ $4$. 

k. $\neg (\exists x \in U . (\forall y \in V . p))$ iff for $\alpha \in U$, we have $\sigma[ x \mapsto \alpha][ y \mapsto \beta] $ $\models | \models | \neg | p$.

l. $\forall (\forall x \in U . (\exists y \in V . p))$ iff for $\alpha \in U$, we have $\sigma[ x \mapsto \alpha][ y \mapsto \beta] $ $\models | \models | p$.

m. $\neg (\forall x \in U . (\exists y \in V . p))$ iff for $\alpha \in U$, we have $\sigma[ x \mapsto \alpha][ y \mapsto \beta]$ ______ $4$.

4. Let $p_1 \equiv \exists y . \forall x . f(x) > y$, and let $p_2 \equiv \forall x . \exists y . f(x) > y$. (As usual, assume a domain of $\mathbb{Z}$.)

a. Is it the case that (regardless of the definition of $f$), if $p_1$ is valid then so is $p_2$? If so, explain why. If not, give a definition of $f(x)$ and show $\models p_1$ but $\not\models p_2$.

b. (Repeat part a in the other direction.) Is it the case that (regardless of the definition of $f$), if $p_2$ is valid then so is $p_1$? If so, explain why. If not, give a definition of $f(x)$ and show $\models p_2$ but $\not\models p_1$. 

CS 536: Solution to Activity 4 (Satisfaction, Validity, and State Updates)

1. \( \sigma[u \mapsto \alpha][v \mapsto \beta] = \sigma[v \mapsto \beta][u \mapsto \alpha] \) iff \( u \neq v \) or \( (u \equiv v \text{ and } \alpha = \beta) \).

2. (Quantified statements over arrays) Let \( \sigma(b) = (7, 5, 12, 16) \).
   a. Yes, \( \sigma \models \exists k \cdot 0 \leq k \wedge k+1 < \text{size}(b) \wedge b[k] < b[k+1] \) with 1 and 2 as possible witnesses for \( k \).
   b. Yes, \( \sigma \models \exists k \cdot 0 \leq k-1 \wedge k+1 < \text{size}(b) \wedge b[k-1] < b[k] < b[k+1] \) with 2 as the only witness that works.
   c. Yes, \( \sigma \models \forall k \cdot b[k] > 0 \)
   d. Yes, if \( \sigma(k) = -5 \), we still have \( \sigma \models \exists k \cdot b[k] > 0 \), with witnesses 0, 1, 2, 3. The key is that for \( \sigma \) to satisfy the existential with witness call it \( \alpha \), then we need \( \sigma[k \mapsto \alpha] \models b[k] > 0 \), which doesn’t depend on \( \sigma(k) \) because the update of \( \sigma \) uses \( k = \alpha \), not \( k \) whatever \( \sigma(k) \) happens to be. Here’s a step-by-step explanation (this is way too much detail for a test):
   
   \[
   \sigma[k \mapsto \alpha] \models b[k] > 0
   \]
   
   if \( \sigma[k \mapsto \alpha](b[k]) > \sigma[k \mapsto \alpha](0) \) defn state \( \models \) relational test
   
   if \( (\sigma[k \mapsto \alpha](b))(\sigma[k \mapsto \alpha](k)) > 0 \) the value of 0 is zero
   
   if \( (\sigma(b))(\sigma[k \mapsto \alpha](k)) > 0 \) \( \sigma[k \mapsto \alpha](b) = \sigma(b) \) because \( b \neq k \)
   
   if \( (\sigma(b))(\alpha) > 0 \) \( \sigma[k \mapsto \alpha](k) = \alpha \)
   
   if \( 7, 5, 12, \text{ or } 16 > 0 \) depending on \( \alpha = 0, 1, 2, \text{ or } 3 \)

3. (Validity/invalidity of quantified predicates)
   a. this \( \sigma \), some \( \alpha \), \( \not\models p \)
   b. this \( \sigma \), every \( \alpha \), \( \models p \)
   c. nothing of \( \sigma(x) \)
   d. nothing of \( \sigma(x) \)
   e. this \( \sigma \), every \( \alpha \), \( \not\models p \)
   f. this \( \sigma \), some \( \alpha \), \( \not\models p \)
   g. some \( \sigma \), \( \not\models \forall x \in U. p \)
   h. some \( \sigma \), every \( \alpha \), \( \not\models p \)
   i. some \( \sigma \), some \( \alpha \), \( \not\models p \)
   j. every \( \sigma \), some \( \alpha \), every \( \beta \), \( \models p \)
   k. some \( \sigma \), every \( \alpha \), some \( \beta \), \( \not\models p \)
   l. every \( \sigma \), every \( \alpha \), some \( \beta \), \( \models p \)
   m. some \( \sigma \), some \( \alpha \), every \( \beta \), \( \not\models p \)

4. (\( \exists \forall \) predicates versus \( \forall \exists \) predicates, specifically \( p_1 \equiv \exists y. \forall x. \mathsf{f}(x) > y \), and \( p_2 \equiv \forall x. \exists y. \mathsf{f}(x) > y \))
   a. The relation does hold: \( \models p_1 \) implies \( \models p_2 \). The short explanation is that for each value \( \alpha \) for \( x \), we need to find a value \( \beta \) for \( y \) that satisfies the body, but \( p_1 \) says that there’s a value that works for every \( \alpha \), so we can use that value for \( \beta \). In more detail, assume \( p_1 \) is valid: for every state \( \sigma \), there is some value \( \beta \) where for every value \( \alpha \), \( \sigma[y \mapsto \beta][x \mapsto \alpha] \models \mathsf{f}(x) > y \). To show that \( p_2 \) is valid, take an arbitrary state \( \tau \)
with value $\alpha$ for $x$. We need a witness value for the $\exists y$; using $p_1$ with $\tau$ for $\sigma$, we get a $\beta$ for the $\exists y$ of $p_1$ and use that as the witness for the $\exists y$ in $p_2$. So then we need $\tau[x \mapsto \alpha][y \mapsto \beta] \models f(x) > y$.
Substituting $\sigma$ for $\tau$ and swapping the order of the updates, we need $\sigma[y \mapsto \beta][x \mapsto \alpha] \models f(x) > y$. But that’s exactly what $p_1$ provided.

b. The relation does not hold: We can have $\models p_2$ but $\not\models p_1$. The easiest example is $f(x) = x$, then validity of $p_1$ would require us to find an integer value for $y$ that is $>\,$ every possible integer value of $x$, and no such value exists.