State Updates, Satisfaction of Quantified Predicates

CS 536: Science of Programming, Fall 2021

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 class, 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 \(\{y = 1\} \models \forall x \in \mathbb{Z}. x^2 + 1 \geq y - 1\), we need to know that \(\{y = 1, x = \alpha\} \models x^2 + 1 \geq y - 1\) for every \(\alpha \in \mathbb{Z}\). I.e., we need
  - ....
  - \(\{y = 1, x = -1\} \models x^2 + 1 \geq y - 1\)
  - \(\{y = 1, x = 0\} \models x^2 + 1 \geq y - 1\)
  - \(\{y = 1, x = 1\} \models x^2 + 1 \geq y - 1\)
  - \(\{y = 1, x = 2\} \models x^2 + 1 \geq y - 1\)
  - ....
- Similarly, for \(\{z = 4\} \models \exists x \in \mathbb{Z}. x \geq z\), we need \(\{z = 4, x = \alpha\} \models x \geq z\) for some particular integer \(\alpha\) (\(\alpha = 5\) 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 \(\{z = 4\} \models \exists x \in \mathbb{Z}. x \geq z\) 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 \(\models \exists x \in \mathbb{Z}. x \geq z\) 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 \(\exists x \in \mathbb{Z}. x \geq z\). However, the two \(x\)’s in “\(\{z = 4, x = 5\} \models x \geq z\)” are the same \(x\). 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
\[ \{ z = 4, xo = -15 \} \models \exists xi \in \mathbb{Z} . xi \geq z \]
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:
\[ \{ z = 4, \text{(outer) } x = -15 \} \models \exists x \in \mathbb{Z} . x \geq z \] because \[ \{ z = 4, \text{(inner) } x = 5 \} \models x \geq z \]

- **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 \( y = x \), 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 \( \sigma[x \mapsto \alpha] \) is \( \sigma_0 \cup \{ x = \alpha \} \).
      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 \( x = v \) where \( v \) stands for a variable (not literally the variable \( v \)) then if \( v = x \), then \( \sigma[x \mapsto \alpha](v) = \sigma[x \mapsto \alpha](x) = \alpha \), otherwise (if \( v = v \), then \( \sigma[x \mapsto \alpha](v) = \sigma(v) \).
    - (You have to read \( \sigma[x \mapsto \alpha](v) \) left-to-right — we’re taking the function \( \sigma[x \mapsto \alpha] \) and applying it to \( v \). I.e., \( \sigma[x \mapsto \alpha](v) = (\sigma[x \mapsto \alpha])(v) \), where the left pair of parentheses are for grouping and the ones around \( v \) 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 \( 0 \))
    - \( \sigma[x \mapsto 0] = x^2 \leq 0 \) (Even though our starting \( \sigma \neq x^2 \leq 0 \))
  - 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.

* 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 \( \sigma \),....
• On the other hand, "σ[x ↦ σ(x+1)]" or "σ[x ↦ α plus one] where α = σ(x)" do make sense.

**Multiple Updates**

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

• **Example 4**: If σ = {x = 2, y = 6}, then σ[x ↦ 0][y ↦ 8] = {x = 0, y = 6}[y ↦ 8] = {x = 0, y = 8}.

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

• **Example 5**: σ[x ↦ 0][y ↦ 8] = σ[y ↦ 8][x ↦ 0].

• If you update the same variable twice, the second update supersedes the first.

• **Example 6**: σ[x ↦ 0][x ↦ 17] = σ[x ↦ 17] ≠ σ[x ↦ 17][x ↦ 0] = σ[x ↦ 0]

• Of course, if the second update is identical to the first, nothing happens: σ[x ↦ α][x ↦ α] = σ[x ↦ α]

• If you have to evaluate an expression, be sure to do it in the correct state.
  - Let σ(x) = 1 and let τ = σ[x ↦ 2], then τ[z ↦ σ(x)+10] maps z to σ(x)+10 = 1+10 = 11. We can omit τ and also write σ[x ↦ 2][z ↦ σ(x)+10], which gives the same state as τ.
  - On the other hand, look at τ[z ↦ τ(x)+10]. Since τ = σ[x ↦ 2], the value of τ(x)+10 = 12, so τ[z ↦ τ(x)+10] = τ[z ↦ 12].
  - If we hadn’t given the name τ = σ[x ↦ 2], then we would had to write σ[x ↦ 2][z ↦ σ[x ↦ 2](x) +10]. (This is pretty ugly, so giving σ[x ↦ 2] a name like τ 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 δ is a function on one argument and α and β are valid members of the domain and range of δ respectively, then the update of δ at a with β, written δ[α ↦ β], is the function defined by δ[α ↦ β](y) = β if y = α and δ[α ↦ β](y) = δ(y) if y ≠ α.

• **Definition**: If σ is a (proper) state for an array b and α is a valid index value for b, then σ[b[α] ↦ β] means σ[b ↦ η[α ↦ β]] where η = the function σ(b). In words, if σ includes the binding b = function η, then the updating σ at b[α] with β is just like updating σ at b with an updated version of η, namely η[α ↦ β].

• **Example 7**: Say σ = {x = 3, b = (2, 4, 6)}, then σ[b[0] ↦ 8] = {x = 3, b = (8, 4, 6)}. Here, σ(b) is the function (2, 4, 6) (which means {(0, 2), (1, 4), (2, 6)}), so σ(b)[0 ↦ 8] (the update of function σ(b)) is the function (2, 4, 6)[0 ↦ 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:** \( \sigma \models \exists x \in S. p \) if for one or more **witness** values \( \alpha \in S \), it's the case that \( \sigma[x \mapsto \alpha] \models p \). Note we're asking a hypothetical question: “If we were to calculate \( \sigma[x \mapsto \alpha] \), would we find that it satisfies \( p \)?”

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

  - **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 \( \sigma(x) \) being 5.

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

  - **Example 9:** If \( \sigma(y) = 3 \), then \( \sigma \models \exists x \cdot x^2 \leq y \) 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 \( \forall x \in S. p \) 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:** \( \sigma \models \forall x > y^2. \exists z. z \geq x+y^2 \) iff (for every \( \alpha \in \mathbb{Z} \), if \( \alpha > \sigma(y)^2 \), then there is some \( \beta \in \mathbb{Z} \) such that \( \beta \geq \alpha + \sigma(y)^2 \).

\[
\begin{align*}
\sigma &\models \forall x > y^2. \exists z. z \geq x+y^2 \\
&\text{iff } \sigma \models \forall x, x > y^2 \rightarrow \exists z. z \geq x+y^2 & \text{defn bounded } \forall \\
&\text{iff for every } \alpha \in \mathbb{Z}, \sigma[x \mapsto \alpha] \models x > y^2 \rightarrow \exists z. z \geq x+y^2, & \text{defn } \models \rightarrow \\
&\text{Now, } \sigma[x \mapsto \alpha] \models x > y^2 \rightarrow \exists z. z \geq x+y^2 \\
&\text{iff } \sigma[x \mapsto \alpha] \models x > y^2 \text{ implies } \sigma[x \mapsto \alpha] \models \exists z. z \geq x+y^2 & \text{defn } \models \rightarrow \\
&\text{iff } \alpha > y^2 \text{ implies } \sigma[x \mapsto \alpha] \models \exists z. z \geq x+y^2 & \text{where } y = \sigma(y) \\
&\text{iff } \alpha > y^2 \text{ implies for some } \beta, \sigma[x \mapsto \alpha][z \mapsto \beta] \models z \geq x+y^2 & \text{defn } \models \exists \\
&\text{iff } \alpha > y^2 \text{ implies for some } \beta, \beta \geq \alpha+y^2 & \text{defn } \models \geq \\
&\text{Taking } \beta = 2\alpha \text{ for our witness value, we need } \alpha > y^2 \text{ implies for some } 2\alpha \geq \alpha+y^2, \text{ which is true.} \\
&\text{Note defining intermediate names like "let } \tau = \sigma[x \mapsto \alpha][z \mapsto \beta] \text{" is allowed, if you prefer that style.}
\end{align*}
\]
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 $\neg \exists x. p \leftrightarrow \forall x. \neg p$.
  
  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 $\sigma \not\models \exists x. p$                                      \hspace{1cm} defn of $\sigma \models \neg \text{predicate}$
  
  iff for no $\alpha \in S$ do we have $\sigma[x \mapsto \alpha] \models p$        \hspace{1cm} defn of $\sigma \models \exists \text{existential}$
  
  iff for every $\alpha \in S$ we have $\sigma[x \mapsto \alpha] \not\models p$                        \hspace{1cm} equivalence of “no $\models$” vs “every $\not\models$”
  
  iff for every $\alpha \in S$ we have $\sigma[x \mapsto \alpha] \models \neg p$       \hspace{1cm} defn of $\sigma \models \neg \text{predicate}$
  
  iff $\sigma \models \forall x. \neg p$                                         \hspace{1cm} defn of $\sigma \models \forall \text{universal}$. 

- Showing the semantic property that $\models \neg \exists x. p \leftrightarrow \forall x. \neg p$ gives us a justification for adding $\neg \exists x. p \leftrightarrow \forall x. \neg p$ as a proof rule.