Satisfaction, Validity, and State Updates

CS 536: Science of Programming, Fall 2018

8/29 solved, 9/20: p.5

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

C. Truth (Satisfaction and Validity) of Predicates

• Definition: A predicate \( p \) is satisfied in state \( \sigma \) if it evaluates to true in state \( \sigma \). (Note \( \sigma \) must be proper for \( p \).) We also say \( \sigma \) satisfies \( p \) or that \( p \) is true in \( \sigma \).

• Notation: \( \sigma \models p \) means \( \sigma \) satisfies \( p \). (The "\( \models \)" symbol is a "double turnstile".) Similarly, \( \sigma \not\models p \) means that \( \sigma \) does not satisfy \( p \). In both cases, \( \sigma \) has to be proper for \( p \).

• Example 1a: \( \{x = 0, y = 0\} \models x \geq y \land y \geq 0 \)
• Example 1b: \( \{x = 0, y = 2\} \not\models x \geq y \land y \geq 0 \)
• Example 1c: \( \{x = 5, y = 3\} \models x \geq y \land y \geq 0 \)
• Example 1d: \( \{x = 0, y = 0, z = 1\} \models x \geq y \land y \geq 0 \). (The extra binding for \( z \) doesn’t matter.)

• Satisfaction of \( \neg p \) versus nonsatisfaction of \( p \)

• One way to treat \( \neg p \) is that it means \( p \rightarrow F \) (if I assume \( p \), then I can get a contradiction). This way of understanding \( \neg p \) always works well.

• Right now, it's okay to think of \( \sigma \not\models p \) (\( p \) isn't true in \( \sigma \)) and \( \sigma \models \neg p \) (\( p \) is false in \( \sigma \)) as being the same. That's because the two possibilities (excluded middle: \( \sigma \models p \) or \( \sigma \models \neg p \)) pair with \( \sigma \not\models p \). Unfortunately, later, we'll have three possibilities: \( \sigma \models p \), \( \sigma \models \neg p \), or there's some problem (typically, \( \sigma \) isn’t an actual memory state because of a runtime error). We’ll still have \( \sigma \models \neg p \) implies \( \sigma \not\models p \), but the other direction turns into \( \sigma \not\models p \) implies \( \sigma \models \neg p \) or there's a problem with asking "\( \sigma \models p ? \)"). For \( \sigma \not\models p \) to imply \( \sigma \models \neg p \), we’ll need to know that there's no problem.

• We’re interested in satisfaction because (1) Our program specifications will only say what should happen if the state initially satisfies a given predicate; and (2) The state after executing the program should satisfy another given predicate. E.g., “if (we're in a memory state satisfying) \( x > 0 \), then after assigning \( x-1 \) to \( y \), we know (we'll be in a memory state satisfying) \( y \geq 0 \).”

• For non-quantified predicates, satisfaction is straightforward to check:

  • \( \sigma \models T \) and \( \sigma \not\models F \) always. (Note every state is proper for \( T \) and for \( F \), even \( \emptyset \), the empty memory state.)

  • \( \sigma \models e_1 \leq e_2 \) means \( \sigma(e_1) \) is less than \( \sigma(e_2) \). (Other comparisons are similar.)
From now on, let's treat:

- Updating States
- Validity
  - **Definition**: A predicate \( p \) is valid (written \( \models p \)) if it is satisfied in all (proper) states. I.e., for all \( \sigma \) (that are proper for \( p \)), we have \( \sigma \models p \).
  - **Example 2a**: \( \models x = x + 0 \) (assuming \( x \) ranges over integers).
  - **Example 2b**: \( \models \forall x. x < x + 1 \)
  - **Notation**: We write \( \not\models p \) if \( p \) is not valid (i.e., unsatisfiable in some (proper) state: \( \sigma \not\models p \) for some \( p \)).
  - **Example 3**: \( \not\models x^2 > x \) because \( \{ x = 0 \} \not\models x^2 > x \). Another counterexample is the state \( \{ x = 1 \} \).
  - **Improper states and \( \models \)**. If \( \sigma \) isn’t proper, then “Does \( \sigma \models p \)” isn’t a useful question. We certainly don’t have \( \sigma \models p \), but we're only interested in \( \sigma \models p \) or \( \sigma \not\models p \) when \( \sigma \) is proper.

- Tautologies, Contradictions, and Contingencies:
  - \( p \) is a tautology if it's valid (satisfied in every state), that is, \( \models p \).
  - \( p \) is a contradiction if \( \neg p \) is valid; that is, \( \models \neg p \).
  - \( p \) is a contingency if it’s neither a tautology nor a contradiction (for some \( \sigma \) and \( \tau \), \( \sigma \not\models p \) and \( \tau \not\models \neg p \)).

As with propositions, "\( p \) is not a tautology" (\( \not\models p \)) iff (if and only if) \( p \) is a contradiction or contingency. Similarly, "\( p \) is not a contradiction" (\( \not\models \neg p \)) iff \( p \) is a tautology or contingency. And "\( p \) is a contingency" iff \( \neg p \) is a contingency.

- **Example 4**: \( x^2 > x \) is a contingency
  - \( x = 0 \) is a counterexample for \( \not\models x^2 > x \), since \( \{ x = 0 \} \not\models x^2 > x \)
  - \( x = 2 \) is a counterexample for \( \not\models \neg(x^2 > x) \), since \( \{ x = 2 \} \not\models x^2 \leq x \).
  - (It might be easier to see this if we rewrite it as \( \{ x = 2 \} \not\models x^2 \leq x \).)

E. "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 5**: 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
  - \( \{ y = 1, x = 0 \} \models x^2 + 1 \geq y - 1 \)
  - \( \{ y = 1, x = 1 \} \models x^2 + 1 \geq y - 1 \)
  - \( \{ y = 1, x = 2 \} \models x^2 + 1 \geq y - 1 \)

* From now on, let's treat \( \neg(e_1 > e_2) \) and \( e_1 \leq e_2 \), etc. as being the same, at least on numbers.
• ....

- 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 \( \alpha (\alpha = 5 \text{ works nicely}) \).

- There is, however, 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 6:** 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 \( \{ z = 4, x = 5 \} \models x \geq z \) because once again, \( \{ z = 4, x = 5 \} \models x \geq z \).

- In **Example 6**, the name “\( x \)” 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 \( x \). 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

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

  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 \( x \):

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

- **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 \( \sigma[x \mapsto \alpha] \) 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 \( x \equiv y \), 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.)

\[†\] 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* \( \sigma \), ....
• **Example 7**: If \( \sigma = \{ x = 2, y = 6 \} \), then \( \sigma[x \rightarrow 0] = \{ x = 0, y = 6 \} \):
  - \( \sigma[x \rightarrow 0](x) = 0 \) (Even though \( \sigma(x) = 2 \))
  - \( \sigma[x \rightarrow 0](y) = \sigma(y) = 6 \) (Since we didn’t update \( y \))
  - \( \sigma[x \rightarrow 0](x+y) = 0+6 = 6 \) (Since the \( x \) in \( x+y \) gets evaluated to 0)
  - \( \sigma[x \rightarrow 0] = x^2 \leq 0 \) (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 \( \sigma[x \rightarrow x+1] \) 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, “\( \sigma[x \rightarrow \sigma(x+1)] \)” or “\( \sigma[x \rightarrow \alpha \) plus one] where \( \alpha = \sigma(x) \)” do make sense.

**Multiple Updates**

- We can do a sequence of updates on a state. E.g., \( \sigma[x \rightarrow 0][y \rightarrow 8] \) is a doubly updated state. Sequences of updates are read left-to-right, so this is \( (\sigma[x \rightarrow 0])[y \rightarrow 8] \).
  - **Example 8**: If \( \sigma = \{ x = 2, y = 6 \} \), then \( \sigma[x \rightarrow 0][y \rightarrow 8] = \{ x = 0, y = 6 \}[y \rightarrow 8] = \{ x = 0, y = 8 \} \).
  - The order of update doesn’t matter if you have two different variables.
  - **Example 9**: \( \sigma[x \rightarrow 0][y \rightarrow 8] = \sigma[y \rightarrow 8][x \rightarrow 0] \).
  - If you update the same variable twice, the second update supersedes the first.
  - **Example 10**: \( \sigma[x \rightarrow 0][x \rightarrow 17] = \sigma[x \rightarrow 17] = \sigma[x \rightarrow 0] \). Of course, if the second update is identical to the first, nothing happens: \( \sigma[x \rightarrow \alpha][x \rightarrow \alpha] = \sigma[x \rightarrow \alpha] \)
  - If you have to evaluate an expression, be sure to do it in the correct state.
    - Let \( \sigma(x) = 1 \) and let \( \tau = \sigma[x \rightarrow 2] \), then \( \tau[z \mapsto \sigma(x)+10] \) maps \( z \) to \( \sigma(x)+10 = 1+10 = 11 \). We can omit \( \tau \) and also write \( \sigma[x \rightarrow 2][z \mapsto \sigma(x)+10] \), 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 \( \sigma[x \rightarrow 2] \), then we can’t write \( \tau[z \mapsto \tau(x)+10] \) so we have to write \( \sigma[x \rightarrow 2][z \mapsto \sigma[x \rightarrow 2](x)+10] \). (This is pretty ugly, so giving \( \sigma[x \rightarrow 2] \) a name like \( \tau \) makes things more readable.)

**F. 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 \( \sigma[b[\alpha] \rightarrow \beta] \) means \( \sigma[b \mapsto \gamma[\alpha \mapsto \beta]] \) where \( \gamma \) is the function \( \sigma(b) \) [8/29]. In words, if \( \sigma \) includes the binding \( b = \) 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 11**: Say \( \sigma = \{ x = 3, b = (2, 4, 6) \} \), then \( \sigma[b[0] \rightarrow 8] = \{ x = 3, b = (8, 4, 6) \} \). Here, \( \sigma(b) \) is the function \( (2, 4, 6) \) (which means \( \{(0, 2), (1, 4), (2, 6)\} \)), so \( \sigma(b)[0 \rightarrow 8] \) (the update of function \( \sigma(b) \)) is the function \( (2, 4, 6)[0 \rightarrow 8] = (8, 4, 6) \).
G. 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**: $\sigma \vdash \exists x \in S. p$ 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 12a**: For any state $\sigma$, we can show $\sigma \vdash \exists x . x^2 \leq 0$ using 0 as the witness: $\sigma[x \mapsto 0] \models x^2 \leq 0$, since $\sigma[x \mapsto 0](x^2) = \sigma[x \mapsto 0](0) \leq \sigma[x \mapsto 0](0) = (0^2 \leq 0) = T$.

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

- **Example 12b**: If $\sigma(x)$ is, say 5, it’s still the case that $\sigma \vdash \exists x . x^2 \leq 0$ using 0 as the witness because $\sigma[x \mapsto 0] \models x^2 \leq 0$, 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 13**: If $\sigma(y) = 3$, then $\sigma \vdash \exists x . x^2 \leq y$ with $x = 0$ or 1 as possible witness values.

- **Definition**: $\sigma \vdash \forall x \in S. p$ 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 14**: To know $\sigma \vdash \forall x \in \mathbb{Z}. x^2 \geq x$, 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 $\alpha^2$ is $\geq \alpha$, this does hold. Recall that it doesn’t matter what $\sigma(x)$ is, since we’re interested in $\forall x \mapsto \alpha$.

- 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 15**: 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 $\sigma \vdash \forall x \in \mathbb{Z}. x^2 \geq x$. Even $\sigma = \emptyset$ works: $\emptyset \vdash \forall x \in \mathbb{Z}. x^2 \geq x$.

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

  - [9/20] (rewritten) **Example 16**: $\sigma \vdash \forall x \geq y^2 . \exists z . z \geq x + y^2$ iff (for every $\alpha \in \mathbb{Z}$, if $\alpha \geq \sigma(y)^2$, then there is some $\beta \in \mathbb{Z}$ such that $\beta > \alpha + \sigma(y)^2$).

    $\sigma \vdash \forall x \geq y^2 . \exists z . z \geq x + y^2$

    - if $\sigma \vdash \forall x . x > y^2 \to \exists z . z \geq x + y^2$

      - defn bounded $\forall$

    - if for every $\alpha \in \mathbb{Z}$, $\sigma[x \mapsto \alpha] \models x > y^2 \to \exists z . z \geq x + y^2$, defn $\vdash \forall$

  - Now, $\sigma[x \mapsto \alpha] \models x > y^2 \to \exists z . z \geq x + y^2$

    - if $\sigma[x \mapsto \alpha] \models x > y^2$ implies $\sigma[x \mapsto \alpha] \models \exists z . z \geq x + y^2$

      - defn $\vdash \exists$

    - if $\alpha > y^2$ implies $\sigma[x \mapsto \alpha] \models \exists z . z \geq x + y^2$ where $\gamma = \sigma(y)$

    - if $\alpha > y^2$ implies for some $\beta$, $\sigma[x \mapsto \alpha][z \mapsto \beta] \models z \geq x + y^2$

      - defn $\vdash \exists$

    - if $\alpha > y^2$ implies for some $\beta$, $\beta \geq \alpha + y^2$

      - defn $\vdash \exists$

  - Taking $\beta = 2\alpha$ for our witness value, we need $\alpha > y^2$ implies for some $2\alpha \geq \alpha + y^2$, which is true.

  - Note defining intermediate names like "let $\tau = \sigma[x \mapsto \alpha][z \mapsto \beta]"$ is allowed, if you prefer that style.

[9/20] end rewrite

**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 17: 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 $\sigma \models \forall x. \neg p$. [8/29] (This treats $\sigma \models \neg p$ as $\sigma \not\models p$; should probably try the $\sigma \models p \rightarrow \text{F}$ technique.)

- $\sigma \models \neg \exists x \in S. p$
  - iff $\sigma \not\models \exists x. p$ defn of $\sigma \models \neg$ predicate
  - iff for no $\alpha \in S$ do we have $\sigma[x \mapsto \alpha] \models p$ defn of $\sigma \models$ existential
  - iff for every $\alpha \in S$ we have $\sigma[x \mapsto \alpha] \not\models p$ equivalence of “no $\models$” vs “every $\not\models$”
  - iff for every $\alpha \in S$ we have $\sigma[x \mapsto \alpha] \models \neg p$ defn of $\sigma \models \neg$ predicate
  - iff $\sigma \models \forall x. \neg p$ 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 $\sigma \models \neg(\forall x. \neg\neg p)$ double $\neg$
    - iff $\sigma \models \neg(\neg\exists x. \neg p)$) DeMorgan law ($\neg \exists$ vs $\forall \neg$)
    - iff $\sigma \models \exists x. \neg p$ double $\neg$
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. Give a state that satisfies $2 \leq i+2 < \text{size}(b) \land 3 \leq b[i]^2 < b[i+1]^2 < b[i+2]^2$.

2. Let $\sigma = \{ x = 0, y = 5, b = (2, 4, 6, 8) \}$.
   
   a. Does $\sigma \models 0 \leq b[0] < \text{size}(b) \land x+y < b[b[0]]$?
   
   b. Does $\sigma \models \forall 0 \leq i < \text{size}(b). x < b[i]$? (I.e., does $\sigma \models \forall i. 0 \leq i < \text{size}(b) \rightarrow x < b[i]$?)
   
   c. Does $\sigma \models \exists 0 \leq i < \text{size}(b). b[i] > y$? (I.e., does $\sigma \models \exists i. 0 \leq i < \text{size}(b) \land b[i] > y$?) If so, with what witness?

3. Let $\sigma = \{ x = 3, y = 4, b = (7, 9) \}$.
   
   a. Does $\sigma \models x > \text{size}(b)$?
   
   b. Does $\sigma \models 0 \leq x < \text{size}(b) \land 0 \leq y < \text{size}(b) \land x < y \land b[x-2] < b[y-3]$?

4. Is $\vdash x = x+0$? If not, give a counterexample.

5. Let $\sigma = \{ x = 3, y = 4, b = (7, 9) \}$.
   
   a. Which of $\sigma(x)$, $\sigma(y)$, $\sigma(z)$, and $\sigma(b)$ are defined? What are their values?
   
   b. What is $\sigma[x \mapsto 9]$?
   
   c. What is $\sigma[x \mapsto 9][x \mapsto 4]$?
   
   d. What is $\sigma[b[1] \mapsto 6]$?
   
   e. What are $\sigma[x \mapsto 9][y \mapsto 10]$ and $\sigma[y \mapsto 10][x \mapsto 9]$? Are they equal?

6. 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 $\alpha = \beta$. 

---

CS 536: Science of Programming  
© James Sasaki, 2018
7. 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 \( \sigma \models \exists k . 0 \leq k - 1 \land k + 1 < \text{size}(b) \land b[k] < b[k + 1] \)? 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 \( \sigma \models \exists k . b[k] > 0 \)?

8. 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.
   f. \( \sigma \not\models (\forall x \in U . p) \) iff for _____ 1 state \( \sigma \) for _____ 2 \( \alpha \in U \), \( \sigma[x \mapsto \alpha] \) _____ 4.
   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. \( \models (\exists x \in U . (\forall y \in V . p)) \) iff for _____ 1 state \( \sigma \), for _____ 2 \( \alpha \in U \), and for _____ 2 \( \beta \in V \), we have \( \sigma[x \mapsto \alpha][y \mapsto \beta] \) _____ 4.
   k. \( \not\models (\exists x \in U . (\forall y \in V . p)) \) iff for _____ 1 state \( \sigma \), for _____ 2 \( \alpha \in U \), and for _____ 2 \( \beta \in V \), we have \( \sigma[x \mapsto \alpha][y \mapsto \beta] \models \models \not\models p \).
   l. \( \not\models (\forall x \in U . (\exists y \in V . p)) \) iff for _____ 1 state \( \sigma \), for _____ 2 \( \alpha \in U \), and for _____ 2 \( \beta \in V \), we have \( \sigma[x \mapsto \alpha][y \mapsto \beta] \models \models \not\models p \).
   m. \( \not\models (\forall x \in U . (\exists y \in V . p)) \) iff for _____ 1 state \( \sigma \), for _____ 2 \( \alpha \in U \), and for _____ 2 \( \beta \in V \), we have \( \sigma[x \mapsto \alpha][y \mapsto \beta] \) _____ 4.

9. 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 \( \not\models p_1 \).
7. (Quantified statements over arrays) Let $\sigma$: 

2. (Does $\sigma = \{x = 0, y = 5, b = (2, 4, 6, 8)\}$ satisfy ...?)
   a. Yes, $\sigma \models 0 \leq b[0] < \text{size}(b) \land x + y < b[0]$, since $0 \leq 2 < 3$ and $5 < 6$
   b. Yes, $\sigma \models \forall 0 \leq i < \text{size}(b).x < b[i]$, since $2, 4, 6,$ and $8$ are all $> 0$
   c. Yes, $\sigma \models \exists 0 \leq i < \text{size}(b).b[i] > y$, with witnesses $2$ or $3$, since $6$ and $8$ are both $> 5$

3. (Does $\sigma = \{x = 3, y = 4, b = (7, 9)\}$ satisfy ... ?
   a. Yes, $\sigma \models x > \text{size}(b)$, since $3 \geq 2$.
   b. No, $\sigma \not\models 0 \leq x < \text{size}(b) \land 0 \leq y < \text{size}(b) \land x < y \land b[x-2] < b[y-3]$ because $\sigma(x) = 3$ is not $< 2$
   and $\sigma(y) = 4$ is not $< 2$.

4. Yes, $\models x = x + 0$. (For all states $\sigma$, $\sigma \models x = x + 0$ because $\sigma(x) = \sigma(x) + 0$.)

5. a. $\sigma(x) = 3$, $\sigma(y) = 4$, and $\sigma(b) = (7, 9)$ are all defined (and $\sigma(z)$ is not).
   b. $\sigma[x \mapsto 9] = \{x = 9, y = 4, b = (7, 9)\}$
   c. $\sigma[x \mapsto 9][x \mapsto 4] = \{x = 9, y = 4, b = (7, 9)[x \mapsto 4] = \{x = 4, y = 4, b = (7, 9)\}$
   d. $\sigma[b[1] \mapsto 6] = \{x = 3, y = 4, b = (7, 9)[1 \mapsto 6] = \{x = 3, y = 4, b = (7, 6)\}$
   e. $\sigma[x \mapsto 9][y \mapsto 10]$ and $\sigma[y \mapsto 10][x \mapsto 9]$ both equal $\{x = 9, y = 10, b = (7, 9)\}$.

6. $\sigma[u \mapsto \alpha][v \mapsto \beta] = \sigma[v \mapsto \beta][u \mapsto \alpha]$ iff $u \neq v$ or ($u \equiv v$ and $\alpha = \beta$).

7. (Quantified statements over arrays) Let $\sigma(b) = (7, 5, 12, 16)$.
   a. Yes, $\sigma \models \exists k.0 \leq k \land k+1 < \text{size}(b) \land b[k] < b[k+1]$ with $1$ and $2$ as possible witnesses for $k$.
   b. Yes, $\sigma \models \exists k.0 \leq k-1 \land k+1 < \text{size}(b) \land b[k-1] < b[k] < b[k+1]$ with $2$ as the only witness that works.
   c. Yes, $\sigma \models \forall k.b[k] > 0$
   d. Yes, if $\sigma(k) = -5$, we still have $\sigma \models \exists k.b[k] > 0$, with witnesses $0, 1, 2, 3$. The key is that for $\sigma$ to satisfy the existential with witness call it $\alpha$, then we need $\sigma[k \mapsto \alpha] \equiv b[k] > 0$, 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):

   $\sigma[k \mapsto \alpha] \equiv b[k] > 0$
   
   if $\sigma[k \mapsto \alpha](b[k]) > \sigma[k \mapsto \alpha](0)$ defn state $\not\models$ 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 $b \not\equiv k$
   
   if $(\sigma[b])(\alpha) > 0$ $\sigma[k \mapsto \alpha](k)) = \alpha$
   
   if $7, 5, 12,$ or $16 > 0$ depending on $\alpha = 0, 1, 2,$ or $3$
8. (Validity/invalidity of quantified predicates)
   a. this σ, some α, ⊨ p
   b. this σ, every α, ⊨ p
   c. nothing of σ(x)
   d. nothing of σ(x)
   e. this σ, every α, ⊭ p
   f. this σ, some α, ⊭ p
   g. some σ, ⊭ ∀ x ∈ U. p
   h. some σ, every α, ⊭ p
   i. some σ, some α, ⊭ p
   j. every σ, some α, every β, ⊩ p
   k. some σ, every α, some β, ⊭ p
   l. every σ, every α, some β, ⊩ p
   m. some σ, some α, every β, ⊭ p

9. (∃∀ predicates versus ∀∃ predicates, specifically p₁ ≡ ∃ y . ∀ x . f(x) > y, and p₂ ≡ ∀ x . ∃ y . f(x) > y)
   a. The relation does hold: ⊩ p₁ implies ⊩ p₂. The short explanation is that for each value α for x, we need to find a value β for y that satisfies the body, but p₁ says that there’s a value that works for every α, so we can use that value for β. In more detail, assume p₁ is valid: for every state σ, there is some value β where for every value α, σ[y ↦ β][x ↦ α] ⊩ f(x) > y. To show that p₂ is valid, take an arbitrary state τ with value α for x. We need a witness value for the ∃ y; using p₁ with τ for σ, we get a β for the ∃ y of p₁ and use that as the witness for the ∃ y in p₂. So then we need τ[x ↦ α][y ↦ β] ⊩ f(x) > y. Substituting σ for τ and swapping the order of the updates, we need σ[y ↦ β][x ↦ α] ⊩ f(x) > y. But that’s exactly what p₁ provided.
   b. The relation does not hold: We can have ⊩ p₂ but ⊭ p₁. The easiest example is f(x) = x, then validity of p₁ would require us to find an integer value for y that is > every possible integer value of x, and no such value exists.