Updating a State (Continue)

1. True or False
   a. If $\sigma(x)$ is not defined, then $\sigma[x \mapsto 0] = \sigma \cup \{x = 0\}$.
      True
   b. If $\sigma(x)$ is defined and $\sigma(x) \neq 0$, then $\sigma[x \mapsto 0] = \sigma \cup \{x = 0\}$.
      False, $\sigma \cup \{x = 0\}$ becomes ill-formed since x appears twice.
   c. Let $\sigma = \{x = 5\}$, then $\sigma[x \mapsto 0] \vDash x \geq x^2$.
      Ture
   d. Let $x \not\equiv y$ be both bind in σ, then $\sigma[x \mapsto 0](y) = \sigma(y)$
      True.
   e. Let $\sigma = \{x = 5\}$, then $\sigma[x \mapsto x + 1] = \{x = 6\}$
      False, we cannot bind a variable with an expression (something syntactic), it becomes ill-formed.
   f. Let $\sigma = \{x = 5\}$, then $\sigma[x \mapsto 2 + 1] = \{x = 3\}$
      True, $2 + 1$ is a semantic value. Remember that a function who returns a primitive type is also semantic.
   g. Let $\sigma = \{x = 5\}$, $\sigma[x \mapsto \sigma(x + 1)] = \{x = 6\}$
      True.

- We can do a sequence of updates on a state, such as $\sigma[x \mapsto 0]\ [y \mapsto 8]$. Here, we read it left-to-right, so it semantically equals to $(\sigma[x \mapsto 0])[y \mapsto 8]$.
  - For example, let $\sigma = \{x = 2, y = 6\}$, then $\sigma[x \mapsto 0][y \mapsto 8] = \{x = 0, y = 6\}[y \mapsto 8] = \{x = 0, y = 8\}$.

2. True or False
   a. Let $x \not\equiv y$, then $\sigma[x \mapsto 0][y \mapsto 8] = \sigma[y \mapsto 8][x \mapsto 0]$
      True. The order of update doesn't matter if we have two different variables.
   b. Let $x \not\equiv y$, then $\sigma[x \mapsto 0][y \mapsto 8] \equiv \sigma[y \mapsto 8][x \mapsto 0]$
      False. Although they give the same state, the updating procedures are different.
   c. $\sigma[x \mapsto 0][x \mapsto 8] = \sigma[x \mapsto 8]$
      True. The second update supersedes the first.
   d. $\sigma[x \mapsto 0][x \mapsto 8] \equiv \sigma[x \mapsto 8]$
      False. Although they give the same state, the updating procedures are different.

3. Let $\sigma = \{x = 1\}$, then what is $\sigma[x \mapsto 2][z \mapsto \sigma[x \mapsto 3](x) + 10]$?

$$
\begin{aligned}
\sigma[x \mapsto 2][z \mapsto \sigma[x \mapsto 3](x) + 10] \quad &= \{x = 2\}[z \mapsto \sigma[x \mapsto 3](x) + 10] \\
&= \{x = 2\}[z \mapsto \{x = 3\}(x) + 10] \\
&= \{x = 2\}[z \mapsto 13] \\
&= \{x = 2, z = 13\}
\end{aligned}
$$

- How to update a value in an array? What do we do if we want to update the value in $b[0]$? Since we handle array as a function from an index to the value stored, here let’s expand the notion of updating states to updating functions.
- If δ is a function and α and β are valid elements of the domain and range of δ respectively, then the update of δ at α with β, written $\delta\ [\alpha \mapsto \beta]$, is the function defined by $\delta[\alpha \mapsto \beta](\gamma) = \beta$ if $\gamma = \alpha$ and $\delta[\alpha \mapsto \beta](\gamma) = \delta\ (\gamma)$ if $\gamma \neq \alpha$.

- Note that, if we consider state as a function, then the definition of updating a state follows the above definition as well. The only difference is that the α and γ here are values.
  For example, let function $\delta = \{(4,6), (3,7), (2,5)\}$, then $\delta[2 \mapsto 3] = \{(4,6), (3,7), (2,3)\}$, $\delta[2 \mapsto 3](2) = 3$, $\delta[2 \mapsto 6](3) = 7$.

- Say σ is a (proper) state with an array b, with η = the function $\sigma(b)$. If α is a valid index value for b, then $\sigma[b[\alpha] \mapsto \beta]$ means $\sigma\left[b \mapsto \eta[\alpha \mapsto \beta]\right]$. So, updating σ at $b[\alpha]$ with β involves updating σ with an updated version of η , namely $\eta\ [\alpha \mapsto \beta]$, as the value of b.
  - For example, $\sigma = \{x = 3, b = (2, 4, 6)\}$, then $\sigma[b[0] \mapsto 8] = \{x = 3, b = (8, 4, 6)\}$. Here, $\sigma(b)$ is $(2, 4, 6)$ as a function (which can also be written $\{(0, 2), (1, 4), (2, 6)\}$, so $\sigma(b)[0 \mapsto 8]$ is the function $(2, 4, 6)[0 \mapsto 8] = (8, 4, 6)$.

<u>Satisfaction of a Quantified Predicate</u>

- We haven't discussed how to decide whether a state satisfies a quantified predicate. If the state does not contain the quantified variable, it is not hard to understand the problem: does $\{y = 1\}$ satisfies $\forall x. x^2 \geq y - 1$? But what if the quantified variable is in the state: does $\{z = 4, x = -5\} \vDash \exists\ x. x \geq z$?

- $\sigma \vDash \exists x \in S. p$ if for one or more **witness values** $\alpha \in S$, it's the case that $\sigma[x \mapsto \alpha] \vDash p$.

4. True or False?
   a. $\{z = 4, x = -5\} \vDash \exists\ x. x \geq z$?
      True. We can find $x = 5$ such that $\{z = 4, x = -5\}[x \mapsto 5] = \{z = 4, x = 5\}$ satisfies $x \geq z$.
   b. $\sigma \vDash \exists\ x. x^2 \leq 0$?
      True. x has domain $\mathbb{Z}$, and we can find $x = 0$ such that $\sigma[x \mapsto 0]$ satisfies $x^2 \leq 0$.

   - From these examples, we can see that if a variable is bounded in an existential quantifier, its current value in a state doesn't affect the satisfaction of the state.

5. Which of the following state satisfies $x < 3 \land \exists x. b[x] > 5$?
   a. $\{x = 0,\ b = (2, 4, 3, 1)\}$
   b. $\{x = 1,\ b = (1, 3, 5, 7)\}$
   c. $\{x = 2,\ b = (1, 3, 5, 4)\}$
   d. $\{x = 3,\ b = (6, 5, 3, 1)\}$

- $\sigma \vDash \forall x \in S. p$ if for every value $\alpha \in S$, we have $\sigma[x \mapsto \alpha] \vDash p$.

6. True or False.
   a. $\{y = 1\} \vDash \forall x \in \mathbb{Z}. x^2 \geq y - 1$?
      True. $\{y = 1\}(y - 1) = 0$, and we know that for all integer α, we have $\alpha^2 \geq 0$.
   b. $\{x = -1\} \vDash \forall x \in \mathbb{Z}. x^2 \geq x$?
      True. We know that for all integer α, we have $\alpha^2 \geq \alpha$.

   - From this example, we can see that if a variable is bounded in a universal quantifier, its current value in a state doesn't matter as well.

- How about "doesn't satisfy"? Let's use "$\sigma \nvDash p \Leftrightarrow \sigma \vDash \neg p$" for now and we can apply DeMorgan's Law here:
  - $\sigma \nvDash \exists x \in S. p \Leftrightarrow \sigma \vDash \neg\exists x \in S. p \Leftrightarrow \sigma \vDash \forall x \in S. \neg p$
  - $\sigma \nvDash \forall x \in S. p \Leftrightarrow \sigma \vDash \neg\forall x \in S. p \Leftrightarrow \sigma \vDash \exists x \in S. \neg p$

(Validity)

- Let p be a proposition or predicate. $\vDash p$ means $\sigma \vDash p$ for all σ, and we say p is **valid**.
  - $\vDash p \Leftrightarrow \forall \sigma \in S. \sigma \vDash p$ (where S is the collection of all well-formed states that are proper for p)
- $\nvDash p$ means $\sigma \nvDash p$ for **some** σ, and we say p is **invalid**.
  - $\nvDash p \Leftrightarrow \exists \sigma \in S. \sigma \nvDash p$ (where S is the collection of all well-formed states are proper for p)

7. Is the following predicate valid? Justify your answer.

$$\exists y. y \neq 0 \wedge x * y \neq 0$$

It is invalid. To show it is , we can argue that:

$\nvDash \exists y. y \neq 0 \wedge x * y \neq 0$

$\Leftrightarrow\ \exists \sigma. \sigma \nvDash \exists y. y \neq 0 \wedge x * y \neq 0$	definition of invalid
$\Leftrightarrow\ \exists \sigma. \sigma \vDash \neg \exists y. y \neq 0 \wedge x * y \neq 0$	definition of $\sigma \nvDash p$
$\Leftrightarrow\ \exists \sigma. \sigma \vDash \forall y. \neg(y \neq 0 \wedge x * y \neq 0)$	DeMorgan's Law
$\Leftrightarrow\ \exists \sigma. \sigma \vDash \forall y. y = 0 \vee x * y = 0$	DeMorgan's Law

We can find that $\sigma = \{x = 0\}$ is a witness, since for all possible values of y, we always have $y = 0$ or $x * y = 0$.

<u>Syntax of Statements in Our Programming Language</u>

- In general, a statement is a standalone unit of execution whose purpose is not creating a value (opposite to expression). We usually use letter S to represent a statement in our programming language.
- We initially introduce 5 types of statements here, and we will introduce more in future classes.
  - **No-op** statement: $\mathbf{skip}$
    It simply means do nothing.

  - **Assignment** statement: $v \coloneqq e$ or $b[e_0][e_1] \dots [e_{n-1}] \coloneqq e$
    Assigning expression e to variable v or assigning expression e to a certain index in an n-dimensional array b.

  - **Sequence** statement: $S; S'$
    Do S then do S'. Note that S' can be another sequence statement, then we have a longer sequence like: $S_1; S_2; S_3$.

  - **Conditional** statement: $\mathbf{if}\ B\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2\ \mathbf{fi}$
    Do S_1 if B is evaluated to $True$, do S_2 if B is evaluated to $False$.
    - A conditional statement and a conditional expression can look alike, we tell one another by context. Note that S_1 and S_2 both must be statements.
    - When S_2 is a no-op statement, then we can simply it from $\mathbf{if}\ B\ \mathbf{then}\ S_1\ \mathbf{else\ skip\ fi}$ to $\mathbf{if}\ B\ \mathbf{then}\ S_1\ \mathbf{fi}$ so we don't need to formally define a $\mathbf{if} - \mathbf{then}$ statement.

  - **Iterative** statement: $\mathbf{while}\ B\ \mathbf{do}\ S\ \mathbf{od}$
    A "while loop" with loop condition B and do S in each iteration.
    - We don't have "for loops" in our language but we can simulate it using $\mathbf{while} - \mathbf{do}$. For example, if we need: $\mathbf{for}\ x = e_1\ \mathbf{to}\ e_2\ \mathbf{do}\ S$, we turn it into: $x \coloneqq e_1; \mathbf{while}\ x < e_2\ \mathbf{do}\ S; x := x + 1\ \mathbf{od}$

- **Program**: A program is simply a statement, typically a sequence statement.

8. Create a program that calculates the power of 2. We run it with input integer n and returns $y = 2^n$; unless $n < 0$, in which case we return 0.

If we write it with indentation, then one way to write it is as follows.

$$
\begin{array}{l}
\textbf{if } n < 0 \textbf{ then} \\
\qquad y := 0 \\
\textbf{else} \\
\qquad x := 0\,; \\
\qquad y := 1\,; \\
\qquad \textbf{while } x < n \\
\qquad \textbf{do} \\
\qquad\qquad x := x + 1; \\
\qquad\qquad y := y + y \\
\qquad \textbf{od} \\
\textbf{fi}
\end{array}
$$

It is also acceptable to write it in one line:

$\textbf{if } n < 0 \textbf{ then } y \coloneqq 1 \textbf{ else } x \coloneqq 0;\ y \coloneqq 1;\ \textbf{while } x < n \textbf{ do } x \coloneqq x + 1;\ y \coloneqq y + y \textbf{ od fi}$