Satisfaction of a Quantified Predicates

- 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 $\sigma[x \mapsto \alpha] \vDash p$.

1. 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.

2. Which of the following state satisfies $x < 3 \wedge \exists x. b[x] > 5$?
   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 $\sigma[x \mapsto \alpha] \vDash p$.

3. 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"? Without consider the possible runtime error during the evaluation, we can use "$\sigma \nvDash p \Leftrightarrow \sigma \vDash \neg p$" then 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 of Predicates

- Let p be a proposition or predicate. $\vDash p$ means $\sigma \vDash p$ for all σ, and we say p is **valid**. In other words, we can say "It is always true that, p".
  - $\vDash p \Leftrightarrow \forall \sigma \in S. \sigma \vDash p$ (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**. In other words, we can say "p is not always true".
  - $\nvDash p \Leftrightarrow \exists \sigma \in S. \sigma \nvDash p$ (where S is the collection of all well-formed states are proper for p)

4. True or False.
   a. $\vDash x > 1 \rightarrow x^2 > x$ True, because all states satisfy $x > 1 \rightarrow x^2 > x$
   b. $\vDash x > 1 \wedge x^2 > x$ False, because $\{x = 0\} \nvDash x > 1 \wedge x^2 \geq x$

   c. $\vDash \forall x. x > 1 \rightarrow x^2 > x$ True
   We need to check that whether $\forall \sigma. \sigma \vDash \forall x. x > 1 \rightarrow x^2 > x$. While check whether some $\sigma \vDash \forall x. x > 1 \rightarrow x^2 > x$, we don't need any bindings in σ, since this universal quantified x and its body contains only variable x as well. Therefore, we only need to check whether any one $\sigma \vDash \forall x. x > 1 \rightarrow x^2 > x$, and we can simple pick $\sigma = \emptyset$ to process.

- If p is a predicate about x and x is not a variable being bounded in p, then $\vDash p \Leftrightarrow \vDash \forall x. p$.

   d. $\vDash \forall x. x > 1 \wedge x^2 > x$ False
   e. $\vDash \exists x. x > 1 \wedge x^2 > x$ True

5. Is the following predicate valid?

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

$\exists y. y \neq 0 \wedge x * y \neq 0$ is a predicate about x and x is not the variable being bounded here, so semantically its validity is equivalent to $\vDash \forall x. \exists y. y \neq 0 \wedge x * y \neq 0$; then it is easy to see this predicate is invalid.

To show it is invalid, 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>

- **Program**: A program is simply a statement, typically a sequence statement.
- In general, a statement is a standalone unit of execution whose purpose is not creating a value (opposite to expression) but creating changes in memory. 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: $v := e$ or $b[e_0][e_1] \dots [e_{n-1}] := e$
    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: $\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi}$
  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 $\textbf{if } B \textbf{ then } S_1 \textbf{ else skip fi}$ to $\textbf{if } B \textbf{ then } S_1 \textbf{ fi}$ so we don't need to formally define a $\textbf{if} - \textbf{then}$ statement.

- **Iterative** statement: $\textbf{while } B \textbf{ do } S \textbf{ od}$
  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 $\textbf{while} - \textbf{do}$. For example, if we need: $\textbf{for } x = e_1 \textbf{ to } e_2 \textbf{ do } S$, we turn it into: $x \coloneqq e_1; \textbf{while } x < e_2 \textbf{ do } S; x := x + 1 \textbf{ od}$

.

6. Translate the following Java statements into statements in our programming language.
   a. $x = (y == 2)?\ 5 + x :\ 6;$
      $x \coloneqq \textbf{if } y = 2 \textbf{ then } 5 + x \textbf{ else } 6 \textbf{ fi}$

   b. $\textbf{if } (y == 2)\ \{x = 5 + x;\}\ \textbf{else}\ \{x = 6;\}$
      $\textbf{if } y = 2 \textbf{ then } x \coloneqq 5 + x \textbf{ else } x \coloneqq 6 \textbf{ fi}$

   c. $\textbf{if } (y == 2)\ \{x = 5 + x;\}\ x = 6;$
      $\textbf{if } y = 2 \textbf{ then } x \coloneqq 5 + x \textbf{ fi}; x \coloneqq 6$ or $\textbf{if } y = 2 \textbf{ then } x \coloneqq 5 + x \textbf{ else skip fi};\ x \coloneqq 6$

7. 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.

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

   It is also acceptable to write it in one line:
   $\textbf{if } n < 0 \textbf{ then } y \coloneqq 0 \textbf{ else } x \coloneqq 0;\ y \coloneqq 1;\ \textbf{while } x < n \textbf{ do } x \coloneqq x + 1;\ y \coloneqq y + y \textbf{ od fi}$

8. Translate the following C statements into statements in our programming language.
   a. $x = a * ++z$
      $z \coloneqq z + 1;\ x \coloneqq a * z$

b. $x = a *\ z++$
$x := a * z; z := z + 1$

c. $\mathbf{while}\ (--x\ \ >= 0)\ \ z *= x;$
$x := x - 1; \mathbf{while}\ x \geq 0\ \mathbf{do}\ z := z * x; x := x - 1\ \mathbf{od}$

d. $\mathbf{while}\ (x-- >= 0)\ \ z *= x;$
$\mathbf{while}\ x \geq 0\ \mathbf{do}\ x := x - 1; z := z * x\ \mathbf{od}; x := x - 1$