Satisfaction, Validity, State Updates, Programs, Operational Semantics
CS 536: Science of Programming, Fall 2018
Due Wed Sep 19, 11:59 pm

9/7: p.2

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.
- Our simple programming language is a model for the kind of constructs seen in actual languages.
- Our programs stand for state transformers.

B. Objectives

At the end of this homework, you should be able to

- Design predicate functions for simple properties on values and arrays.
- Determine the satisfaction or validity of a predicate, using hypothetical updated states if necessary.
- Take a state and “update” it by adding or replacing a binding.
- Translate simple programs into our language.
- Use operational semantics to describe step-by-step execution of programs in our language.

C. Problems [100 points total]

If not otherwise specified, assume our quantifiers range over \( \mathbb{Z} \). (So \( \exists x \) means \( \exists x \in \mathbb{Z} \) and similarly for \( \forall x \).)

Part I: Satisfaction, Validity, and State Updates

1. [8 = 4 * 2 points] Let \( \sigma = \{ x = 2, y = 4, b = (11, 21, 31, 41) \} \).
   a. What is \( \sigma[y \mapsto 8][x \mapsto 5] \)?
   b. What is \( \sigma[y \mapsto 5](x) \)?
   c. What is \( \sigma[b[1] \mapsto 13][y \mapsto \sigma(b[1]/2)] \)? (Assumes integer division truncates, as in C.)
   d. When does \( \sigma[u \mapsto \alpha][v \mapsto \beta] = \sigma[v \mapsto \beta][u \mapsto \alpha] \)? (Note \( u \equiv \text{ or } \not\equiv \) and \( \alpha = \text{ or } \not= \beta \).

2. [12 = 4 * 3 points] Let state \( \sigma = \{ x = 2 \} \). Which of the statements below are correct and if not, why not? (Is the state illegal or improper? Or is the state proper but does not satisfy the predicate?)
   a. \( \sigma[y \mapsto 5] = \exists x . x > y \)
   b. \( \sigma[x \mapsto 3] = \forall y . y \geq 2 \rightarrow x < y^2 \)
   c. \( \sigma \models x * 0 = x * 0 \)
   d. \( \sigma[z \mapsto 5][y \mapsto 3][z \mapsto 6] = z = x * y \)

3. [9 = 3 * 3 points] Let \( \sigma = \{ x = 2, \ldots \} \) (so \( \sigma \) may or may not have bindings for variables \( \not\equiv x \).) For each the statements below, say whether or not the state is legal, and if legal, whether or not the claim holds or not.
   a. \( \sigma[z \mapsto 4] = \sigma \cup \{ z = 4 \} \)
   b. \( \sigma[y \mapsto 0][z \mapsto (1, 3, x+y)][b[0]] = 1. \)
   c. \( \sigma[v \mapsto 5](x) \) is undefined
4. [15 = 5 * 3 points] Complete each statement below by describing in English the needed property. Use the style of the activity in Lecture 4. E.g.,

\[ \sigma \vdash (\forall x \in U. \ p) \text{ iff for (this state } \sigma \text{ and) every } \alpha \in U, \sigma[x \mapsto \alpha] \vdash p \]

(You can leave out the phrase “this state \( \sigma \)” if you like, but you aren’t required to.)

a. \( \sigma \nvdash (\exists x \in U. \ p) \text{ iff } \ldots \)

b. \( \nvdash (\exists x \in U. \ p) \text{ iff } \ldots \)

c. \( \vdash (\exists x \in U. (\forall y \in V. \ p)) \text{ iff } \ldots \)

d. \( \nvdash (\exists x \in U. (\forall y \in V. \ p)) \text{ iff } \ldots \)

e. \( \nvdash (\forall x \in U. (\exists y \in V. \ p)) \text{ iff } \ldots \)

5. [6 = 3 * 2 points] In general, what is the logical equivalence relationship between the following pairs of predicates? (You can just say “\( \Rightarrow \)” or “\( \Leftarrow \)”. You should know why the relationship holds; you just do not have to write it down.)

a. \( \exists x. \exists y. p(x, y) \text{ and } \exists y. \exists x. p(x, y) \)

b. \( \forall x. \forall y. p(x, y) \text{ and } \forall y. \forall x. p(x, y) \)

c. \( \forall x. \exists y. p(x, y) \text{ and } \exists y. \forall x. p(x, y) \)

Part II: Program Syntax and Operational Semantics

6. [18 = 3 * 6 points] Translate the given C-like programs into our programming language.

a. [6 points] \( \text{for } (j = m, x = 1; j > 1; ) \) \( x *= y[-j]; \)

b. [6 points] \( j = m; x = 1; \text{ while } (--j >= 0) \) \( x *= y[j]; \)

c. [6 points] \( x = 1; j = 0; \text{ while } (j++ < m) \) \( x *= y[j]; \)

7. [12 = 2 * 6 points] Let \( \sigma = \{(x, \alpha), (y, \beta)\} \). Evaluate each \( (S, \sigma) \) below to completion.

a. \( S \equiv t := x ; x := y ; y := t \)

b. \( S \equiv \text{if } x < 0 \) \( \text{then } y := -x \) \( \text{else } y := x \) \( \text{fi} \)

8. [13 points] Let \( S \equiv s := 0 ; \text{ while } n \geq 0 \) \( S_1 \) \( \text{od} \) where the loop body \( S_1 \equiv s := s+n; n := n-1. \)

a. [3 points] If \( \sigma(n) < 0 \), what are the operational semantics of \( S \) in \( \sigma \)?

b. [4 points] Give the operational semantics of the loop body \( S_1 \) in an arbitrary state \( \tau \).

c. [6 points] Let \( \sigma(n) = 3 \). Write out the complete execution sequence for \( (S, \sigma) \). Use multi-step execution as in Example 10 in Lecture 5.

Part III: Predicate Function Design [9/7]: Changed the problem

9. [7 points] Let’s take the \( \mathcal{GT} \) predicate from the previous homework and extend it. **Notation:** \( b[i..j] > x \) means that every element in the (up-going) sequence \( b[i], b[i+1], \ldots, b[j-1] \) is \( > x \). If \( i \geq j \), the sequence is empty and \( b[i..j] > x \) is true. Also, let’s generalize from \( > \) to any of the comparison operators \( <, \leq, >, \geq, =, \) and \( \neq \).

a. Write a definition of \( b[i..j] > x \). In addition, ensure that \( i \) is a legal index and that \( j-1 \) either \( = -1 \) or is a legal index.
b. Write out a definition of a predicate function \( \text{split}(b, m, p, n) \) that is true if the \( 0..m-1 \) segment of \( b \) is \( \leq b[m] \) and \( b[m] \) is \( \leq \) every element in the \( p..n-1 \) segment of \( b \). Include limitations to ensure that indexes \( m \) and \( n-1 \) are always within range. Also ensure that the two segments don't overlap. (But either or both of the segments can be empty.) Remember, this has to be a predicate function, not a program that calculates a boolean value.