Programming Languages and Computation

Week 5: Denotational Semantics and Induction

1 Denotational Semantics

* 1. Compute the value of the following arithmetic expressions in the state $\sigma = [x \mapsto -1, y \mapsto 11, z \mapsto 10]$. Your answer should be explicit about the steps you take and make reference to the definition of the denotation function.

(a) $(x + y) - z$

(b) $x * (12 - z)$

(c) $x - (z - 1)$

* 2. Compute the value of the following arithmetic expressions in the state $\sigma = [x \mapsto 11, y \mapsto 12]$. Your answer should be explicit about the steps you take and make reference to the definition of the denotation function.

(a) $(x + y) - z$

(b) $x * (12 - z)$

(c) $x - (z - 1)$

* 3. Compute the value of the following Boolean expressions in the state $\sigma = [x \mapsto 11, y \mapsto 12]$. Your answer should be explicit about the steps you take and make reference to the definition of the denotation function.

(a) $(x + y \leq z)$ && true

(b) $!(x * y = z) \,||\, x = 11$

(c) $x \leq y$ && $y \leq x$

The next questions relate to when arithmetic and Boolean expressions are *syntactically equivalent* or *semantically equivalent*. Two arithmetic or Boolean expressions are syntactically equivalent if they have exactly the same abstract syntax tree and semantically equivalent is they denote the same function. Remember that two functions are equal if, and only if, they have the same value on every input.

* 4. Which of the following arithmetic expressions are syntactically equivalent and which are semantically equivalent?

(a) $x * 3$

(b) $x * 1$

(c) $x + (x + x)$

(d) $(x + (x)) + x$

(e) $x + ((x) + x)$

(f) $x + (y * 0)$

* 5. Consider the Boolean expression $x \leq 2$ && $y \leq 3$. Find a semantically equivalent expression that does not use the && operator.

* 6. Find a state in which the arithmetic expressions $x * 2$ and $x + 3$ evaluate to the same value. Explain why they are *not* semantically equivalent.

** 7. Suppose that $e_1 \in \mathcal{A}$ and $e_2 \in \mathcal{A}$ are semantically equivalent arithmetic expressions. Prove that $e_1 + e_2$ is semantically equivalent to $e_1 * 2$. Your answer should make explicit reference to the denotation function.

** 8. Suppose that $e_1 + 1$ and $e_2 + 1$ are two arithmetic expressions that are semantically equivalent for some $e_1, e_2 \in \mathcal{A}$. Prove that e_1 and e_2 are also semantically equivalent.

** 9. Suppose that $e_1 * e_2$ and $e_1 * 2$ are two arithmetic expressions that are *not* semantically equivalent for some arithmetic expressions $e_1, e_2 \in \mathcal{A}$.

(a) Prove that e_2 cannot be semantically equivalent to 2.

(b) Find concrete examples of expressions $e_1, e_2 \in \mathcal{A}$ such that $e_1 * e_2$ and $e_1 * 2$ are not semantically equivalent but where there exists a state $\sigma \in \mathsf{State}$ such that $[\![e_2]\!]_{\mathcal{A}}(\sigma) = 2$.

** 10. Let us suppose we want to add a new construct to the language of arithmetic expressions:

$$A \to x \mid n \mid \cdots \mid \mathsf{let}\ x = A\ \mathsf{in}\ A$$

An expression let $x = e_1$ in e_2 using this construct should evaluate the sub-expression e_2 in a state where the variable x is mapped to the value of e_1. For example, the expression let $x = 2$ in $x + y$ when evaluated in the state $[x \mapsto 3, y \mapsto 2]$ should be 4.

Extend the definition of the denotation function $[\![\cdot]\!]_{\mathcal{A}}$ with an equation for this construct. You may find it useful to use the notation $\sigma[x \mapsto n]$ to represent the state σ updated such that $(\sigma[x \mapsto n])(x) = n$ and $(\sigma[x \mapsto n])(y) = \sigma(y)$ for all $y \neq x$. You do not need to change any other equations.

** 11. Now let us suppose we extend the language of arithmetic expressions with a different operator:

$$A \to x \mid n \mid \cdots \mid B\ ?\ A : A$$

An instance of this *ternary operator* $e_1 ? e_2 : e_3$ for some Boolean expression $e_1 \in \mathcal{B}$ and arithmetic expressions $e_2, e_3 \in \mathcal{A}$ behaves as e_2 in states where e_1 is true and behaves as e_3 otherwise.

Extend the definition of the denotation function $[\![\cdot]\!]_{\mathcal{A}}$ with an equation for this construct. Your answer may make reference to the denotation function for Boolean expressions.

2 Proof by Induction

* 12. Consider the exponential function for natural numbers with the following recursive definition:

$$x^0 = 1$$
$$x^{n+1} = x \cdot x^n$$

Prove by induction that $(x \cdot y)^z = x^z \cdot y^z$ for any $x, y, z \in \mathbb{N}$. You may assume that multiplication satisfies the usual laws of associativity and commutativity.

** 13. The *height* of an arithmetic expression is defined recursively as follows:

$$\begin{aligned}
\mathsf{height}(n) &= 1 \\
\mathsf{height}(x) &= 1 \\
\mathsf{height}(e_1 + e_2) &= 1 + \max\{\mathsf{height}(e_1), \mathsf{height}(e_2)\} \\
\mathsf{height}(e_1 - e_2) &= 1 + \max\{\mathsf{height}(e_1), \mathsf{height}(e_2)\} \\
\mathsf{height}(e_1 * e_2) &= 1 + \max\{\mathsf{height}(e_1), \mathsf{height}(e_2)\}
\end{aligned}$$

(a) Prove by structural induction over arithmetic expressions that $\mathsf{height}(e) > 0$ for all arithmetic expressions $e \in \mathcal{A}$.

(b) Prove by structural induction over arithmetic expressions that $2^{\mathsf{height}(e)-1} \geq \#\mathsf{FV}(e)$ for all arithmetic expressions $e \in \mathcal{A}$ where $\#\mathsf{FV}(e)$ is the number of free variables appearing in that expression.

** 14. If x is a variable and e_1 and e_2 are arithmetic expressions, then we write $e_1[e_2/x]$ for the expression that results from *substituting* e_2 for x in the expression e_1. Formally, this operation it is defined by recursion over the expression e_1 as follows:

$$\begin{aligned} n[e/x] &= n \\ y[e/x] &= \begin{cases} e & \text{if } x = y \\ y & \text{otherwise} \end{cases} \\ (e_1 + e_2)[e/x] &= e_1[e/x] + e_2[e/x] \\ (e_1 - e_2)[e/x] &= e_1[e/x] - e_2[e/x] \\ (e_1 * e_2)[e/x] &= e_1[e/x] * e_2[e/x] \end{aligned}$$

(a) Compute the value of the expression $(y - x)[z/x]$ in the state $[x \mapsto 1, y \mapsto 2, z \mapsto 3]$.

(b) Find a state σ such that $[\![y - x]\!]_{\mathcal{A}}(\sigma)$ evaluates to the same answer you got in part (a). What is the relationship between this state and the state $[x \mapsto 1, y \mapsto 2, z \mapsto 3]$?

(c) Prove by structural induction over expressions, for any state $\sigma \in \mathsf{State}$, any pair of arithmetic expressions e_1, $e_2 \in A$ and any variable $x \in \mathsf{Var}$, we have that:

$$[\![e_1[e_2/x]]\!]_{\mathcal{A}}(\sigma) = [\![e_1]\!]_{\mathcal{A}}(\sigma[x \mapsto [\![e_2]\!]_{\mathcal{A}}(\sigma)]).$$

Remember that e_1 may be an *arbitrary* variable.

** 15. Write down the induction principle for Boolean expressions. Try to generalise from the induction principle for arithmetic expressions as it appears in the reference notes (https://uob-coms20007.github.io/notes/semantics/induction.html).
Hint: the cases for Boolean expressions of the form $e_1 \leq e_2$ and $e_1 = e_2$ are not inductive cases as the sub-expressions are not actually Boolean expressions.

*** 16. We extend the notion of *free variables* of an arithmetic expression to Boolean expressions. Formally, we define a function $\mathsf{FV} : \mathcal{B} \to \mathcal{P}(\mathsf{Var})$ from Boolean expressions to sets of variables by recursion over the structure of expressions as follows:

$$\begin{aligned} \mathsf{FV}(\mathsf{true}) &= \emptyset \\ \mathsf{FV}(\mathsf{false}) &= \emptyset \\ \mathsf{FV}(e_1 \leq e_2) &= \mathsf{FV}(e_1) \cup \mathsf{FV}(e_2) \\ \mathsf{FV}(e_1 = e_2) &= \mathsf{FV}(e_1) \cup \mathsf{FV}(e_2) \\ \mathsf{FV}(!e) &= \mathsf{FV}(e) \\ \mathsf{FV}(e_1 \;\&\&\; e_2) &= \mathsf{FV}(e_1) \cup \mathsf{FV}(e_2) \\ \mathsf{FV}(e_1 \parallel e_2) &= \mathsf{FV}(e_1) \cup \mathsf{FV}(e_2) \end{aligned}$$

(a) Find two Boolean expressions e_1, $e_2 \in \mathcal{B}$ that are semantically equivalent, i.e. they evaluate to the same value on all states, but for which $\mathsf{FV}(e_1) \neq \mathsf{FV}(e_2)$.

(b) Prove by induction that for *all* Boolean expressions $e \in \mathcal{B}$ and pair of states $\sigma, \sigma' \in \mathsf{State}$ that:

$$[\![e]\!]_{\mathcal{B}}(\sigma) = [\![e]\!]_{\mathcal{B}}(\sigma')$$

where $\forall x \in \mathsf{FV}(e).\, \sigma(x) = \sigma'(x)$.

You may assume the fact that the analogous result holds for arithmetic expressions in your answer.

** 17. Define a recursive function for substitution acting on Boolean expressions, you may wish to model your answer on substitution for arithmetic expressions from Question 14.
Prove that your definition satisfies the property:

$$[\![e_1[e_2/x]]\!]_{\mathcal{B}}(\sigma) = [\![e_1]\!]_{\mathcal{B}}(\sigma[x \mapsto [\![e_2]\!]_{\mathcal{A}}(\sigma)]).$$

for any Boolean expression $e_1 \in \mathcal{B}$, arithmetic expression $e_2 \in \mathcal{A}$, and any variable $x \in \mathsf{Var}$. You may assume the analogous property shown in Question 14.

*** 18. The set of *contexts* is defined by the following grammar:

$$C \to \varepsilon \mid A + C \mid C + A \mid A - C \mid C - A \mid A * C \mid C * A$$

where A is an arbitrary arithmetic expression. We write $\mathcal{C}$ for the set of contexts.

Given a context $C \in \mathcal{C}$ and an arithmetic expression $e \in \mathcal{A}$, we write $C[e] \in \mathcal{A}$ for the arithmetic expression that is derived by replacing the "ε" in C with the expression e. For example, $(x + \varepsilon)[y]$ is the expression $x + y$. Formally, this operation is defined by recursion over contexts:

$$\begin{aligned}
\varepsilon[e_1] &= e_1 \\
(e_2 + C)[e_1] &= e_2 + C[e_1] \\
(C + e_2)[e_1] &= C[e_1] + e_2 \\
(e_2 - C)[e_1] &= e_2 - C[e_1] \\
(C - e_2)[e_1] &= C[e_1] - e_2 \\
(e_2 * C)[e_1] &= e_2 * C[e_1] \\
(C * e_2)[e_1] &= C[e_1] * e_2
\end{aligned}$$

(a) Consider the arithmetic expressions $x + x$ and $x * 2$ and the context $y + \varepsilon$. Show that $(y + \varepsilon)[x + x]$ and $(y + \varepsilon)[x * 2]$ are semantically equivalent.

(b) Now suppose e_1 and e_2 are arbitrary arithmetic expressions that are semantically equivalent. Show that $(y + \varepsilon)[e_1]$ and $(y + \varepsilon)[e_2]$ are semantically equivalent as well.

(c) Prove by structural induction that, for any context $C \in \mathcal{C}$, and any two semantically equivalent arithmetic expressions $e_1 \in \mathcal{A}$ and $e_2 \in \mathcal{A}$, that $C[e_1]$ and $C[e_2]$ are semantically equivalent.