Programming Languages and Computation

Week 7: Proof by Induction & Operational Semantics

1 Proof by Induction

* 1. 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.

Solution

We shall prove that $(x \cdot y)^z = x^z \cdot y^z$ for any $x, y, z \in \mathbb{N}$ by induction over $z \in \mathbb{N}$.

- In the base case, we have that $(x \cdot y)^0 = 1$ and $x^0 \cdot y^0 = 1 \cdot 1 = 1$. Therefore, $(x \cdot y)^0 = x^0 \cdot y^0$ as required.
- Let us suppose $(x \cdot y)^z = x^z \cdot y^z$ holds for some $z \in \mathbb{N}$. We must show that $(x \cdot y)^{z+1} = x^{z+1} \cdot y^{z+1}$ holds. By definition, $(x \cdot y)^{z+1} = (x \cdot y) \cdot (x \cdot y)^z$. It then follows from our induction hypothesis that $(x \cdot y)^{z+1} = (x \cdot y) \cdot x^z \cdot y^z$. Therefore, $(x \cdot y)^{z+1} = x^{z+1} \cdot y^{z+1}$ as required.

** 2. 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.

Hint: Try using the facts that, if $x \geq 2y, 2z$, then $x \geq y + z$, and that $\#A + \#B \geq \#(A \cup B)$.

Solution

(a) We shall prove that $\mathsf{height}(e) > 0$ for all arithmetic expressions $e \in \mathcal{A}$ by induction as follows:

- In the case of a variable or a numeric literal, the height is clearly greater than 0.
- In the inductive cases, we have that induction hypotheses $\mathsf{height}(e_1)$, $\mathsf{height}(e_2) > 0$. It then follows that $1 + \max\{\mathsf{height}(e_1), \mathsf{height}(e_2)\} > 0$ as required.

(b) We shall prove that $2^{\mathsf{height}(e)-1} \geq \#\mathsf{FV}(e)$ for all arithmetic expressions $e \in \mathcal{A}$ by induction as follows:

- In the case of a variable $x \in \mathsf{Var}$, the height is 1 and $\#\mathsf{FV}(x) = 1$. Therefore, $2^{\mathsf{height}(x)-1} = 2^0 = 1$ and thus $2^{\mathsf{height}(x)-1} \geq 1$ as required.
- Similarly, in the case of a numeric literal $n \in \mathbb{Z}$, the height is 1 and $\#\mathsf{FV}(x) = 0$. Therefore, $2^{\mathsf{height}(n)-1} = 2^0 = 1$ and thus $2^{\mathsf{height}(n)-1} \geq 0$ as required.
- Now consider an expression of the form $e_1 + e_2$ where, inductively, we know that $2^{\mathsf{height}(e_1)-1} \geq \#\mathsf{FV}(e_1)$ and $2^{\mathsf{height}(e_2)-1} \geq \#\mathsf{FV}(e_2)$. By definition, $\mathsf{height}(e_1 + e_2)$ is equal to $1 + \max\{\mathsf{height}(e_1), \mathsf{height}(e_2)\}$. Therefore, $2^{\mathsf{height}(e_1+e_2)-1}$ is at least as large as both $2^{\mathsf{height}(e_1)}$ and $2^{\mathsf{height}(e_2)}$. It then follows from the induction hypotheses, that $2^{\mathsf{height}(e_1+e_2)-1}$ is at least as large as both $2 \cdot \#\mathsf{FV}(e_1)$ and $2 \cdot \#\mathsf{FV}(e_2)$. Finally, as $\#\mathsf{FV}(e_1) + \#\mathsf{FV}(e_2) \geq \#\mathsf{FV}(e_1 + e_2)$, we have that $2^{\mathsf{height}(e_1+e_2)-1} \geq \#\mathsf{FV}(e_1 + e_2)$ as required.
- The cases for subtraction and multiplication are analogous to that of addition.

** 3. If x is a variable and e_1 and e_2 are arithmetic expressions, then we write $e_1[x \mapsto e_2]$ 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:

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

(a) Compute the value of the expression $(y - x)[x \mapsto z]$ 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:

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

Remember that e_1 may be an *arbitrary* variable.

Solution

(a) The expression $(y-x)[x \mapsto z]$ is by definition equal to $y-z$. Therefore, evaluating in the state $[x \mapsto 1, y \mapsto 2, z \mapsto 3]$ leads to the value -1.

(b) Under the state $[x \mapsto 3, y \mapsto 2, z \mapsto 3]$, the expression $y-x$ evaluates to -1. This state is derived by replacing the value for x with the value for z, thus capturing the behaviour of the substitution $[x \mapsto z]$.

(c) We shall prove by structural induction that:

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

for any state $\sigma \in \mathsf{State}$, any pair of arithmetic expressions $e_1, e_2 \in A$ and any variable $x \in \mathsf{Var}$ by structural induction over e_1.

- Suppose e_1 is a variable $y \in \mathsf{Var}$. In order to correctly apply the substitution, we need to know whether the variable y is equal to the variable x. Therefore, there are two subcases to consider:
  - If $y = x$, then we have that $[\![y[x \mapsto e_2]]\!]_{\mathcal{A}}(\sigma) = [\![e_2]\!]_{\mathcal{A}}(\sigma)$. On the other hand, $[\![y]\!]_{\mathcal{A}}(\sigma[x \mapsto [\![e_2]\!]_{\mathcal{A}}(\sigma)])$ is by definition $[\![e_2]\!]_{\mathcal{A}}(\sigma)$. Therefore, $[\![y[x \mapsto e_2]]\!]_{\mathcal{A}}(\sigma) = [\![y]\!]_{\mathcal{A}}(\sigma[x \mapsto [\![e_2]\!]_{\mathcal{A}}(\sigma)])$ as required.
  - Otherwise, let us suppose that $y \neq x$. In this case, $[\![y[x \mapsto e_2]]\!]_{\mathcal{A}}(\sigma) = \sigma(y)$ as the expression y is unaffected by the substitution Likewise, $[\![y]\!]_{\mathcal{A}}(\sigma[x \mapsto [\![e_2]\!]_{\mathcal{A}}(\sigma)])$ is equal to $\sigma(y)$ as both the states σ and $\sigma[x \mapsto [\![e_2]\!]_{\mathcal{A}}(\sigma)]$ assign the same value to the variable y. Therefore, we have that $[\![y[x \mapsto e_2]]\!]_{\mathcal{A}}(\sigma) = [\![y]\!]_{\mathcal{A}}(\sigma[x \mapsto [\![e_2]\!]_{\mathcal{A}}(\sigma)])$ as required.
- Now let us suppose e_1 is a numeric literal $n \in \mathbb{Z}$. In this case, $[\![n[x \mapsto e_2]]\!]_{\mathcal{A}}(\sigma) = n$ and, likewise, $[\![n]\!]_{\mathcal{A}}(\sigma[x \mapsto [\![e_2]\!]]) = n$. Therefore, $[\![n[x \mapsto e_2]]\!]_{\mathcal{A}}(\sigma) = [\![n]\!]_{\mathcal{A}}(\sigma[x \mapsto [\![e_2]\!]])$ as required.
- Now let us suppose e_1 is of the form $e_3 + e_4$ for some arithmetic expressions $e_3, e_4 \in \mathcal{A}$. Then $[\![(e_3+e_4)[x \mapsto e_2]]\!]_{\mathcal{A}}(\sigma) = [\![e_3[x \mapsto e_2]]\!]_{\mathcal{A}}(\sigma) + [\![e_4[x \mapsto e_2]]\!]_{\mathcal{A}}(\sigma)$ by definition. Our induction hypotheses tell us that $[\![e_3[x \mapsto e_2]]\!]_{\mathcal{A}}(\sigma) = [\![e_3]\!]_{\mathcal{A}}(\sigma[x \mapsto [\![e_2]\!]])$ and likewise for e_4. Therefore, we have that $[\![(e_3+e_4)[x \mapsto e_2]]\!]_{\mathcal{A}}(\sigma) = [\![e_3]\!]_{\mathcal{A}}(\sigma[x \mapsto [\![e_2]\!]_{\mathcal{A}}(\sigma)]) + [\![e_4]\!]_{\mathcal{A}}(\sigma[x \mapsto [\![e_2]\!]_{\mathcal{A}}(\sigma)])$. It follows then that $[\![(e_3+e_4)[x \mapsto e_2]]\!]_{\mathcal{A}}(\sigma) = [\![e_3+e_4]\!]_{\mathcal{A}}(\sigma[x \mapsto [\![e_2]\!]_{\mathcal{A}}(\sigma)])$ as required.
- The cases of subtraction and multiplication are analogous to that of addition.

** 4. 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/reference/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.

Solution

The induction principle for Boolean expressions states that $P(e)$ is true of all $e \in \mathcal{B}$ whenever:

- $P(\mathsf{true})$ is true;
- $P(\mathsf{false})$ is true;

- $P(e_1 \mathbin{\&\&} e_2)$ is true for any $e_1, e_2 \in \mathcal{B}$ such that $P(e_1)$ and $P(e_2)$ is true;
- $P(e_1 \mathbin{||} e_2)$ is true for any $e_1, e_2 \in \mathcal{B}$ such that $P(e_1)$ and $P(e_2)$ is true;
- $P(!e)$ is true for any $e \in \mathcal{B}$ such that $P(e)$ is true;
- $P(e_1 = e_2)$ is true for any $e_1, e_2 \in \mathcal{A}$;
- And, $P(e_1 \leq e_2)$ is true for any $e_1, e_2 \in \mathcal{A}$.

** 5. 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:

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

(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.

Solution

(a) The Boolean expressions true and $\mathsf{true} \mathbin{||} x \leq y$ are semantically equivalent but have a different set of free variables.

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

- In the case of the constant true, we have that $[\![\mathsf{true}]\!]_{\mathcal{B}}(\sigma) = \top$ regardless of σ. In particular, $[\![\mathsf{true}]\!]_{\mathcal{B}}(\sigma) = [\![\mathsf{true}]\!]_{\mathcal{B}}(\sigma')$ for any two states $\sigma, \sigma' \in \mathsf{State}$.
- The case of the constant false is analogous to that of true.

- Now consider a Boolean expression of the form $e_1 \leq e_2$ where $e_1 \in \mathcal{A}$ and $e_2 \in \mathcal{A}$ are arithmetic expressions. Let $\sigma, \sigma' \in \mathsf{State}$ be states such that $\forall x \in \mathsf{FV}(e_1 \leq e_2).\, \sigma(x) = \sigma'(x)$. By definition, $\mathsf{FV}(e_1 \leq e_2) = \mathsf{FV}(e_1) \cup \mathsf{FV}(e_2)$. Therefore, we also know that $\forall x \in \mathsf{FV}(e_1).\, \sigma(x) = \sigma'(x)$ and likewise for e_2. It then follows that $[\![e_1]\!]_{\mathcal{A}}(\sigma) = [\![e_1]\!]_{\mathcal{A}}(\sigma')$ and likewise for e_2. Thus, $[\![e_1 \leq e_2]\!]_{\mathcal{B}}(\sigma) = [\![e_1 \leq e_2]\!]_{\mathcal{B}}(\sigma')$ as required.
- The case of Boolean expressions of the form $e_1 = e_2$ is analogous to the preceding case.
- Now consider a Boolean expression of the form $!e$ where, inductively, we know that $[\![e]\!]_{\mathcal{B}}(\sigma) = [\![e]\!]_{\mathcal{B}}(\sigma')$ whenever $\forall x \in \mathsf{FV}(e).\, \sigma(x) = \sigma'(x)$. Let $\sigma, \sigma' \in \mathsf{State}$ be states such that $\forall x \in \mathsf{FV}(!e).\, \sigma(x) = \sigma'(x)$. As $\mathsf{FV}(!e) = \mathsf{FV}(e)$, we have that $\forall x \in \mathsf{FV}(e).\, \sigma(x) = \sigma'(x)$. Therefore, the induction hypothesis applies. It then follows that:

$$\begin{aligned} [\![!e]\!]_{\mathcal{B}}(\sigma) &= \neg[\![e]\!]_{\mathcal{B}}(\sigma) \\ &= \neg[\![e]\!]_{\mathcal{B}}(\sigma') \\ &= [\![!e]\!]_{\mathcal{B}}(\sigma') \end{aligned}$$

  as required.
- Now consider a Boolean expression of the form e_1 && e_2 where, inductively, we know that $[\![e_1]\!]_{\mathcal{B}}(\sigma) = [\![e_1]\!]_{\mathcal{B}}(\sigma')$ whenever $\forall x \in \mathsf{FV}(e_1).\, \sigma(x) = \sigma'(x)$ and likewise for e_2. Let $\sigma, \sigma' \in \mathsf{State}$ be states such that $\forall x \in \mathsf{FV}(e_1 \mathbin{\&\&} e_2).\, \sigma(x) = \sigma'(x)$. As $\mathsf{FV}(e_1 \mathbin{\&\&} e_2) \supseteq \mathsf{FV}(e_1), \mathsf{FV}(e_2)$, we have that $\forall x \in \mathsf{FV}(e_1).\, \sigma(x) = \sigma'(x)$ and likewise for e_2. Therefore, the induction hypotheses apply. It then follows that:

$$\begin{aligned} [\![e_1 \mathbin{\&\&} e_2]\!]_{\mathcal{B}}(\sigma) &= [\![e_1]\!]_{\mathcal{B}}(\sigma) \wedge [\![e_2]\!]_{\mathcal{B}}(\sigma) \\ &= [\![e_1]\!]_{\mathcal{B}}(\sigma') \wedge [\![e_2]\!]_{\mathcal{B}}(\sigma') \\ &= [\![e_1 \mathbin{\&\&} e_2]\!]_{\mathcal{B}}(\sigma') \end{aligned}$$

  as required.
- The case of Boolean expressions of the form $e_1 \mathbin{||} e_2$ is analogous to the preceding case.

** 6. 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.

Solution

(a) The expression $(y + \varepsilon)[x + x]$ is equal to $y + x + x$ and the expression $(y + \varepsilon)[x * 2]$ is equal to the expression $y + (x * 2)$. Both these expressions denote the function that maps a state σ to the integer $\sigma(y) + 2\sigma(x)$. Therefore, they are semantically equivalent.

(b) Let e_1 and e_2 be semantically equivalent arithmetic expressions. The expression $(y + \varepsilon)[e_1]$ maps a state σ to the value $\sigma(y) + [\![e_1]\!]_{\mathcal{A}}(\sigma)$ and, likewise the expression $(y + \varepsilon)[e_2]$ maps a state σ to the value $\sigma(y) + [\![e_2]\!]_{\mathcal{A}}(\sigma)$. As e_1 and e_2 are semantically equivalent, we know that $[\![e_1]\!]_{\mathcal{A}}(\sigma) = [\![e_2]\!]_{\mathcal{A}}(\sigma)$ for all states. It then follows that $(y + \varepsilon)[e_1]$ and $(y + \varepsilon)[e_2]$ are semantically equivalent as required.

(c) We shall prove 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 by induction on the context:

   - In the base case with a context ε, we must show that $\varepsilon[e_1]$ is semantically equivalent to $\varepsilon[e_2]$ given that e_1 and e_2 are semantically equivalent. As the expression $\varepsilon[e_1]$ is equal to e_1 and likewise for e_2, this case is trivial.
   - Now consider a context of the form $e_3 + C$. We must show that $(e_3 + C)[e_1]$ is semantically equivalent to $(e_3 + C)[e_2]$ given that e_1 and e_2 are semantically equivalent. By definition, $(e_3 + C)[e_1] = e_3 + C[e_1]$ and likewise $(e_3 + C)[e_2] = e_3 + C[e_2]$. By induction, we know that $C[e_1]$ is semantically equivalent to $C[e_2]$. It then follows that $e_3 + C[e_1]$ is semantically equivalent to $e_3 + C[e_2]$ and, therefore, that $(e_3 + C)[e_1]$ is semantically equivalent to $(e_3 + C)[e_2]$ as required.
   - The other cases are analogous to the preceding case.

*** 7. *Strong induction* is a variation on proof by induction where the induction hypothesis applies to *all* smaller values rather than just the predecessor or the subtrees. Formally, it can be stated as following principle:

> In order to prove $\forall n \in \mathbb{N}. P(n)$, prove:
>
> 1. $P(0)$;
> 2. And, $P(n+1)$ under the assumption that $P(m)$ holds for all $m \leq n$.

Prove that strong induction follows from standard induction. That is, given a property P of the natural numbers that satisfies the requirements (1) and (2), prove that $P(n)$ is true for all $n \in \mathbb{N}$.

Hint: Try proving the property $P'(n) = \forall m \leq n. P(n)$ by induction.

Solution

Suppose P were a property of the natural numbers that satisfies (1) and (2). Let $P'(n)$ be the property $\forall m \leq n. P(m)$. We shall prove that $P'(n)$ holds for all $n \in \mathbb{N}$ by induction:

- In the base case, we must show that $\forall m \leq 0. P(m)$ holds. This is equivalent to the requirement that $P(0)$ holds, which we know to be true by (1).
- Now suppose $P'(n)$ holds, and we must show that $P'(n+1)$ holds. Therefore, consider some $m \in \mathbb{N}$ such that $m \leq n+1$. If $m \leq n$, then $P'(n)$ already implies that $P(m)$ holds as required. Otherwise, $m = n+1$. In this case, we may apply the fact that $P(n+1)$ holds when $P(m)$ holds for all $m \leq n+1$ as given by (2). In other words, $P'(n)$ implies $P(n+1)$. Therefore, $P(n+1)$ as required.

We have shown that $\forall n \in \mathbb{N}. P'(n)$. To conclude that $P(n)$ holds for all $n \in \mathbb{N}$ consider some such $n \in \mathbb{N}$. We have that $P'(n)$ holds, i.e. $\forall m \leq n. P(m)$. In particular, $P(n)$ holds as required. Thus, completing our proof.

2 Operational Semantics

This section is about the big-step operational semantics of While programs as given by the relation $\Downarrow \subseteq \mathcal{S} \times \mathsf{State} \times \mathsf{State}$, which is defined inductively by these inference rules:

$$\frac{}{\mathsf{skip},\, \sigma \Downarrow \sigma} \qquad \frac{}{x \leftarrow e,\, \sigma \Downarrow \sigma[x \mapsto [\![e]\!]_{\mathcal{A}}(\sigma)]}$$

$$\frac{S_1,\, \sigma_1 \Downarrow \sigma_2 \quad S_2,\, \sigma_2 \Downarrow \sigma_3}{S_1;\, S_2,\, \sigma_1 \Downarrow \sigma_3} \qquad \frac{S_1,\, \sigma_1 \Downarrow \sigma_2}{\mathsf{if}\ e\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2,\, \sigma_1 \Downarrow \sigma_2}\ [\![e]\!]_{\mathcal{B}}(\sigma_1) = \top$$

$$\frac{S_2,\, \sigma_1 \Downarrow \sigma_2}{\mathsf{if}\ e\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2,\, \sigma_1 \Downarrow \sigma_2}\ [\![e]\!]_{\mathcal{B}}(\sigma_1) = \bot \qquad \frac{}{\mathsf{while}\ e\ \mathsf{do}\ S,\, \sigma \Downarrow \sigma}\ [\![e]\!]_{\mathcal{B}}(\sigma) = \bot$$

$$\frac{S,\, \sigma_1 \Downarrow \sigma_2 \quad \mathsf{while}\ e\ \mathsf{do}\ S,\, \sigma_2 \Downarrow \sigma_3}{\mathsf{while}\ e\ \mathsf{do}\ S,\, \sigma_1 \Downarrow \sigma_3}\ [\![e]\!]_{\mathcal{B}}(\sigma_1) = \top$$

Figure 1: Inference rules for operational semantics.

* 8. Write down a derivation for the judgement $x \leftarrow 1; \{x \leftarrow 2; x \leftarrow 3\}, [] \Downarrow [x \mapsto 3]$ using the inference rules in Figure 1.

Solution

$$\dfrac{\dfrac{}{x \leftarrow 1, [] \Downarrow [x \mapsto 1]} \quad \dfrac{\dfrac{}{x \leftarrow 2, [x \mapsto 1] \Downarrow [x \mapsto 2]} \quad \dfrac{}{x \leftarrow 3, [x \mapsto 2] \Downarrow [x \mapsto 3]}}{x \leftarrow 2; x \leftarrow 3, [x \mapsto 1] \Downarrow [x \mapsto 3]}}{x \leftarrow 1; \{x \leftarrow 2; x \leftarrow 3\}, [] \Downarrow [x \mapsto 3]}$$

* 9. Write down a derivation for the judgement $\{x \leftarrow 1; x \leftarrow 2\}; x \leftarrow 3, [] \Downarrow [x \mapsto 3]$ using the inference rules in Figure 1.

Solution

$$\dfrac{\dfrac{\dfrac{}{x \leftarrow 1, [] \Downarrow [x \mapsto 1]} \quad \dfrac{}{x \leftarrow 2, [x \mapsto 1] \Downarrow [x \mapsto 2]}}{x \leftarrow 1; x \leftarrow 2, [] \Downarrow [x \mapsto 2]} \quad \dfrac{}{x \leftarrow 3, [x \mapsto 2] \Downarrow [x \mapsto 3]}}{\{x \leftarrow 1; x \leftarrow 2\}; x \leftarrow 3, [] \Downarrow [x \mapsto 3]}$$

* 10. Write down a derivation for the judgement $\{x \leftarrow 1; x \leftarrow 2\}; x \leftarrow 3, [] \Downarrow [x \mapsto 3]$ using the inference rules in Figure 1.

Solution

$$\dfrac{\dfrac{\dfrac{}{x \leftarrow 1, [] \Downarrow [x \mapsto 1]} \quad \dfrac{}{x \leftarrow 2, [x \mapsto 1] \Downarrow [x \mapsto 2]}}{x \leftarrow 1; x \leftarrow 2, [] \Downarrow [x \mapsto 2]} \quad \dfrac{}{x \leftarrow 3, [x \mapsto 2] \Downarrow [x \mapsto 3]}}{\{x \leftarrow 1; x \leftarrow 2\}; x \leftarrow 3, [] \Downarrow [x \mapsto 3]}$$

* 11. Compute the final state for the program if $x \leq y$ then $x \leftarrow y$ else $y \leftarrow x$ when executed in each of the following states:

- $[]$
- $[x \mapsto 2, y \mapsto 3]$
- $[x \mapsto 4, y \mapsto 2]$

Solution

- $[]$
- $[x \mapsto 3, y \mapsto 3]$
- $[x \mapsto 4, y \mapsto 4]$

* 12. Find a state σ such that $x \leftarrow 1; y \leftarrow x * 2, [] \Downarrow \sigma$. You must also write down the derivation of the statement.

Solution

The state $[x \mapsto 1, y \mapsto 2]$ satisfies the requirement with the associated derivation:

$$\dfrac{\overline{x \leftarrow 1, [\,] \Downarrow [x \mapsto 1]} \quad \overline{y \leftarrow x * 2, [x \mapsto 1] \Downarrow [x \mapsto 1, y \mapsto 2]}}{x \leftarrow 1; y \leftarrow x * 2, [\,] \Downarrow [x \mapsto 1, y \mapsto 2]}$$

* 13. Find a state $\sigma \in \mathsf{State}$ for which there exists a derivation of the judgement $\mathsf{while}\ !(x \leq -1)\ \mathsf{do}\ x \leftarrow x + d, [d \mapsto -1] \Downarrow \sigma$. You should provide the derivation.

Solution

The state $[d \mapsto -1, x \mapsto -1]$ will satisfy the requirements. The associated derivation is:

$$\dfrac{\overline{x \leftarrow x + d, [d \mapsto -1] \Downarrow [d \mapsto -1, x \mapsto -1]} \quad \overline{\mathsf{while}\ !(x \leq -1)\ \mathsf{do}\ x \leftarrow x + d, [d \mapsto -1, x \mapsto -1] \Downarrow [d \mapsto -1, x \mapsto -1]}}{\mathsf{while}\ !(x \leq -1)\ \mathsf{do}\ x \leftarrow x + d, [d \mapsto -1] \Downarrow [d \mapsto -1, x \mapsto -1]}$$

** 14. Find a state σ such that $x \leftarrow 2; y \leftarrow x * y, \sigma \Downarrow [x \mapsto 2, y \mapsto 4]$. You must write down the derivation of the statement.

Solution

The state $[y \mapsto 2]$ will satisfy the requirements. The associated derivation is:

$$\dfrac{\overline{x \leftarrow 2, [y \mapsto 2] \Downarrow [x \mapsto 2, y \mapsto 2]} \quad \overline{y \leftarrow x * y, [x \mapsto 2, y \mapsto 2] \Downarrow [x \mapsto 2, y \mapsto 4]}}{x \leftarrow 1; y \leftarrow x * y, [y \mapsto 2] \Downarrow [x \mapsto 1, y \mapsto 4]}$$

* 15. Suppose $e \in \mathcal{B}$ is a Boolean expression that is semantically equivalent to false. Prove that $\mathsf{while}\ e\ \mathsf{do}\ S, \sigma \Downarrow \sigma$ for any state $\sigma \in \mathsf{State}$.

Solution

As $e \in \mathcal{B}$ is semantically equivalent to false. We have that $[\![e]\!]_{\mathcal{B}}(\sigma) = \bot$ for any state $\sigma \in \mathsf{State}$. Therefore, we have that $\mathsf{while}\ e\ \mathsf{do}\ S, \sigma \Downarrow \sigma$ for any state $\sigma \in \mathsf{State}$ as required.

** 16. Suppose $S_1, S_2 \in \mathcal{S}$ are two statements such that $S_1, \sigma \Downarrow \sigma'$ and $S_2, \sigma \Downarrow \sigma'$ for some states $\sigma, \sigma' \in \mathsf{State}$. Prove that $\mathsf{if}\ e\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2, \sigma \Downarrow \sigma'$ for any Boolean expression $e \in \mathcal{B}$.

Solution

Let $e \in \mathcal{B}$ be some Boolean expression and $\sigma, \sigma' \in \mathsf{State}$ be two states. Let us consider two cases:

- Suppose $[\![e]\!]_{\mathcal{B}}(\sigma)$ is true. Then we have the following derivation of the form:

$$\dfrac{S_1, \sigma \Downarrow \sigma'}{\mathsf{if}\ e\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2, \sigma \Downarrow \sigma'}$$

- Suppose, otherwise, that $[\![e]\!]_{\mathcal{B}}(\sigma)$ is false. Then equally we have the following derivation of the form:

$$\dfrac{S_2, \sigma \Downarrow \sigma'}{\mathsf{if}\ e\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2, \sigma \Downarrow \sigma'}$$

Warning: the following question has been changed due to typos (and very large derivations). It may not align your previous work.

** 17. Suppose we introduce a new language construct do S while e where $S \in \mathcal{S}$ is a statement and $e \in \mathcal{B}$ is a Boolean expression. The operational semantics for this construct is given by the following inference rules:

$$\frac{S,\, \sigma_1 \Downarrow \sigma_2}{\textsf{do } S \textsf{ while } e,\, \sigma_1 \Downarrow \sigma_2}\ [\![e]\!]_{\mathcal{B}}(\sigma_2) = \bot \qquad \frac{S,\, \sigma_1 \Downarrow \sigma_2 \quad \textsf{do } S \textsf{ while } e,\, \sigma_2 \Downarrow \sigma_3}{\textsf{do } S \textsf{ while } e,\, \sigma_1 \Downarrow \sigma_3}\ [\![e]\!]_{\mathcal{B}}(\sigma_2) = \top$$

(a) Find a state $\sigma \in \textsf{State}$ such that do $x \leftarrow x+1$ while $x \leq 1$, $[\,] \Downarrow \sigma$ and give the associated derivation.

(b) For a given statement $S \in \mathcal{S}$ and a Boolean expression $e \in \mathcal{B}$ find a While program that is equivalent to the program do S while e but does not use the new construct. That is, find a statement $S' \in \mathcal{S}$ such that:

$$S',\, \sigma \Downarrow \sigma' \iff \textsf{do } S \textsf{ while } e,\, \sigma \Downarrow \sigma'$$

You do not need to prove that your answer is correct but should provide a derivation of judgement S', $[\,] \Downarrow \sigma$ where S is given to be the statement $x \leftarrow x+1$, e is given to be the expression $x \leq 1$ and σ is the state from part (a).

Solution

(a) The state is $[x \mapsto 2]$ and the derivation is:

$$\dfrac{\dfrac{}{x \leftarrow x+1,\, [\,] \Downarrow [x \mapsto 1]} \quad \dfrac{\dfrac{}{x \leftarrow x+1,\, [x \mapsto 1] \Downarrow [x \mapsto 2]}}{\textsf{do } x \leftarrow x+1 \textsf{ while } x \leq 1,\, [x \mapsto 1] \Downarrow [x \mapsto 2]}}{\textsf{do } x \leftarrow x+1 \textsf{ while } x \leq 1,\, [\,] \Downarrow [x \mapsto 2]}$$

(b) The statement S; while e do S is equivalent.

$$\dfrac{\dfrac{}{x \leftarrow x+1,\, [\,] \Downarrow [x \mapsto 1]} \quad \dfrac{\dfrac{}{x \leftarrow x+1,\, [x \mapsto 1] \Downarrow [x \mapsto 2]} \quad \dfrac{}{\textsf{while } x \leq 1 \textsf{ do } x \leftarrow x+1,\, [x \mapsto 2] \Downarrow [x \mapsto 2]}}{\textsf{while } x \leq 1 \textsf{ do } x \leftarrow x+1,\, [x \mapsto 1] \Downarrow [x \mapsto 2]}}{x \leftarrow x+1;\ \textsf{while } x \leq 1 \textsf{ do } x \leftarrow x+1,\, [\,] \Downarrow [x \mapsto 2]}$$

*** 18. Suppose we introduce a new language construct for x do S where $S \in \mathcal{S}$ is a statement and $x \in \textsf{Var}$ is a variable The operational semantics for this construct is given by the following inference rules:

$$\frac{}{\textsf{for } x \textsf{ do } S,\, \sigma \Downarrow \sigma}\ \sigma(x) \leq 0 \qquad \frac{S,\, \sigma_1 \Downarrow \sigma_2 \quad \textsf{for } x \textsf{ do } S,\, \sigma_2[x \mapsto \sigma_2(x) - 1] \Downarrow \sigma_3}{\textsf{for } x \textsf{ do } S,\, \sigma_1 \Downarrow \sigma_3}\ \sigma_1(x) > 0$$

(a) Find a state $\sigma \in \textsf{State}$ such that for x do $y \leftarrow y + x$; $x \leftarrow x - 2$, $[x \mapsto 3] \Downarrow \sigma$ and give the associated derivation.

(b) For a given statement $S \in \mathcal{S}$ and a variable $x \in \textsf{Var}$ find a While program that is equivalent to the program for x do S but does not use the new construct. That is, find a statement $S' \in \mathcal{S}$ such that:

$$S',\, \sigma \Downarrow \sigma' \iff \textsf{for } x \textsf{ do } S$$

You do not need to prove that your answer is correct but should provide a derivation of judgement $S', [x \mapsto 3] \Downarrow \sigma$ where S is given to be the statement $y \leftarrow y + x;\ x \leftarrow x - 2$, x is given to be the variable x and σ is the state from part (a).

Solution

In the answers to this question we have used † and †† to decompose large derivations. When appearing as premise, they are to be understood as standing in for the derivation with the corresponding symbol to the left.

(a)

$$\dagger\ \dfrac{\dfrac{}{\text{for } x \text{ do } y \leftarrow y + x;\ x \leftarrow x - 2, [x \mapsto 0, y \mapsto 3] \Downarrow [x \mapsto 0, y \mapsto 3]}}{\text{for } x \text{ do } y \leftarrow y + x;\ x \leftarrow x - 2, [x \mapsto 3] \Downarrow [x \mapsto 0, y \mapsto 3]}$$

$$\dagger\ \dfrac{\dfrac{}{y \leftarrow y + x, [x \mapsto 3] \Downarrow [x \mapsto 3, y \mapsto 3]} \quad \dfrac{}{x \leftarrow x - 2, [x \mapsto 3, y \mapsto 3] \Downarrow [x \mapsto 1, y \mapsto 3]}}{y \leftarrow y + x;\ x \leftarrow x - 2, [x \mapsto 3] \Downarrow [x \mapsto 1, y \mapsto 3]}$$

(b) The statement while $!(x \leq 0)$ do $S;\ x \leftarrow x - 1$ is equivalent.

$$\dagger\ \dfrac{\dfrac{}{\text{while } !(x \leq 0) \text{ do } y \leftarrow y + x;\ x \leftarrow x - 2;\ x \leftarrow x - 1, [x \mapsto 0, y \mapsto 3] \Downarrow [x \mapsto 0, y \mapsto 3]}}{\text{while } !(x \leq 0) \text{ do } y \leftarrow y + x;\ x \leftarrow x - 2;\ x \leftarrow x - 1, [x \mapsto 3] \Downarrow [x \mapsto 0, y \mapsto 3]}$$

$$\dagger\ \dfrac{\dfrac{}{y \leftarrow y + x, [x \mapsto 3] \Downarrow [x \mapsto 3, y \mapsto 3]} \quad \dagger\dagger}{y \leftarrow y + x;\ x \leftarrow x - 2;\ x \leftarrow x - 1, [x \mapsto 3] \Downarrow [x \mapsto 0, y \mapsto 3]}$$

$$\dagger\dagger\ \dfrac{\dfrac{}{x \leftarrow x - 2, [x \mapsto 3, y \mapsto 3] \Downarrow [x \mapsto 1, y \mapsto 3]} \quad \dfrac{}{x \leftarrow x - 1, [x \mapsto 1, y \mapsto 3] \Downarrow [x \mapsto 0, y \mapsto 3]}}{x \leftarrow x - 2;\ x \leftarrow x - 1, [x \mapsto 3, y \mapsto 3] \Downarrow [x \mapsto 0, y \mapsto 3]}$$