Programming Languages and Computation

Week 7: Semantics of the While Language

This problem sheet considers the semantics of the While language and how to reason about it. Except where indicated otherwise, you should fully justify your answers from the definitions of semantics given in the reference material.

Expressions and their semantics

* 1. Evaluate the following arithmetic expressions in state $\sigma = \begin{bmatrix} \texttt{a} \mapsto 42 \\ \texttt{b} \mapsto 154 \\ \texttt{x} \mapsto 11 \end{bmatrix}$

(a) `a + x * 14 - b`

(b) `x * (a - x) * b`

Solution

The results can be obtained easily by doing a simple substitution and evaluating. The interest of asking for a derivation based on the formal definitions is to force thinking about associativity and computation defined by structural induction.

I only expand the first. The second has similar associativity.

(a)

$$\begin{aligned} [\![\texttt{a + x * 14 - b}]\!]^{\mathscr{A}}(\sigma) &= [\![\texttt{a + x * 14}]\!]^{\mathscr{A}}(\sigma) - [\![\texttt{b}]\!]^{\mathscr{A}}(\sigma) \\ &= [\![\texttt{a}]\!]^{\mathscr{A}}(\sigma) + [\![\texttt{x * 14}]\!]^{\mathscr{A}}(\sigma) - [\![\texttt{b}]\!]^{\mathscr{A}}(\sigma) \\ &= [\![\texttt{a}]\!]^{\mathscr{A}}(\sigma) + [\![\texttt{x}]\!]^{\mathscr{A}}(\sigma) * 14 - [\![\texttt{b}]\!]^{\mathscr{A}}(\sigma) \\ &= \sigma(a) + \sigma(x) * 14 - \sigma(b) \\ &= 42 + 11 * 14 - 154 \\ &= 42 \end{aligned}$$

(b)

$$[\![\texttt{x * (a - x) * b}]\!]^{\mathscr{A}}(\sigma) = 52514$$

* 2. Evaluate the following arithmetic expressions in state $\sigma = \begin{bmatrix} \texttt{x} \mapsto 11 \\ \texttt{b} \mapsto 12 \\ \texttt{h} \mapsto 84 \end{bmatrix}$

(a) `a + x * 14 - b`

(b) `b - g * x + h`

Solution

The only additional difficulty is that you need to remember that looking up a variable we have not explicitly defined yields value 0.

(a)

$$[\![\texttt{a + x * 14 - b}]\!]^{\mathscr{A}}(\sigma) = 142$$

(b)

$$[\![\texttt{b - g * x + h}]\!]^{\mathscr{A}}(\sigma) = 96$$

* 3. Evaluate the following boolean expressions in state $\sigma = \begin{bmatrix} \texttt{a} \mapsto 42 \\ \texttt{b} \mapsto 154 \\ \texttt{x} \mapsto 11 \end{bmatrix}$

(a) `a + x * 14 - b ≤ b + x && true`

(b) `!x * (a - x) * b = b + x && x = 11`

(c) `b - g * x + h ≤ 0`

Solution

No additional difficulty here again. Recall that `!` binds strongest of all.

(a)

$$[\![\texttt{a + x * 14 - b} \leq \texttt{b + x \&\& true}]\!]^{\mathscr{B}}(\sigma) = \top$$

(b)

$$[\![\texttt{!x * (a - x) * b = b + x \&\& x = 11}]\!]^{\mathscr{B}}(\sigma) = \top$$

(c)

$$[\![\texttt{b - g * x + h} \leq 0]\!]^{\mathscr{B}}(\sigma) = \bot$$

Writing While programs

* 4. Write the following While programs. You may find it useful to refer to the example While program shown in Figure 1, which computes a factorial, taking its input from variable `n`, and placing its

```
r ← 1;
while (1 <= n) {
 r ← n * r;
 n ← n - 1
}
```

Figure 1: A `factorial` program.

output in variable `r`.

(a) A program that increments the value stored in variable `n` by 2.

(b) A program that stores in variable `r` the absolute value of the variable initially stored in variable `n`.

(c) A program that swaps the values stored in variables `a` and `b`. (While programs operate over arbitrary integers in $\mathbb{Z}$ so you do not need to use a temporary variable, but you may do so.)

(d) A program that stores in variable `r` the remainder of the initial value of variable `n` in the division by 2. (If I were to write this as a C program: `r = n % 2`.) Assume that `n` is initially non-negative.

(e) A program that loops forever, constantly increasing by 1 the value of variable `n`.

Solution

(a) `n ← n + 2`

(b) `if (1 <= n) then r ← n else r ← 0 - n`

(c)
```
b ← a + b;
a ← b - a;
b ← b - a
```

(d)
```
while (1 <= n) {
 r ← n;
 n ← n - 2
}
```

(e)
```
 while (true) {
 n ← n + 1
}
```

** 5. Modify the program from Figure 1 so that, if the value n of variable `n` is initially negative, the

program terminates with the value of $-(|n|!)$ in variable `r`. (That is, the opposite of the factorial of the opposite of n.) Its behaviour on non-negative values of n should be unchanged.

Solution

```
if (n <= -1)
then m ← 0 - n
else m ← n;
r ← 1;
while (1 <= m) {
 r ← r * m;
 m ← m - 1
};
if (n <= -1)
then r ← 0 - r
else
```

Execution Traces

** 6. Consider the While programs you wrote in Question 4. For c ranging over them, give a derivation for $\langle c, \sigma\rangle \Downarrow \sigma'$ for an appropriate σ' (if such a σ' exists!) and

$$\sigma = \begin{bmatrix} \mathtt{n} \mapsto 3 \\ \mathtt{a} \mapsto 5 \end{bmatrix}$$

Solution

Derivations are hard to type. The important thing here is to make sure the derivations you give are for the programmes (as trees!) you actually wrote. In particular, if your programme includes `skip`s because of errant semicolons, then you should include those in your derivation.

(a) $$(\mathsf{BAss})\ \frac{}{\langle \mathtt{n} \leftarrow \mathtt{n} + 2, \sigma\rangle \Downarrow \sigma[\mathtt{n} \mapsto 5]}$$

(b) $$(\mathsf{BIf}_{\top})\ \frac{[\![\mathtt{1 <= n}]\!]^{\mathscr{B}}(\sigma) = \top \qquad (\mathsf{BAss})\ \dfrac{}{\langle \mathtt{r} \leftarrow \mathtt{n}, \sigma\rangle \Downarrow \sigma[\mathtt{r} \mapsto 3]}}{\langle \mathtt{if\ (1 <= n)\ then\ r \leftarrow n\ else\ r \leftarrow 0 - n}, \sigma\rangle \Downarrow \sigma[\mathtt{r} \mapsto 3]}$$

(c) $$\dfrac{(\mathsf{BAss})\ \dfrac{}{\langle \mathtt{b} \leftarrow \mathtt{a + b}, \sigma\rangle \Downarrow \sigma[\mathtt{b} \mapsto 5} \quad (\mathsf{BSeq})\ \dfrac{(\mathsf{BAss})\ \dfrac{}{\langle \mathtt{a} \leftarrow \mathtt{b - a}, \sigma[\mathtt{b} \mapsto 5]\rangle \Downarrow \sigma[\mathtt{b} \mapsto 5; \mathtt{a} \mapsto 0]} \quad (\mathsf{BAss})\ \dfrac{}{\langle \mathtt{b} \leftarrow \mathtt{b - a}, \sigma[\mathtt{b} \mapsto 5; \mathtt{a} \mapsto 0]\rangle \Downarrow \sigma[\mathtt{b} \mapsto 5; \mathtt{a} \mapsto 0]}}{\langle \mathtt{a} \leftarrow \mathtt{b - a;\ b} \leftarrow \mathtt{b - a}, \sigma[\mathtt{b} \mapsto 5]\rangle \Downarrow \sigma[\mathtt{b} \mapsto 5; \mathtt{a} \mapsto 0]}}{\langle \mathtt{b} \leftarrow \mathtt{a + b;\ a} \leftarrow \mathtt{b - a;\ b} \leftarrow \mathtt{b - a}, \sigma\rangle \Downarrow \sigma[\mathtt{b} \mapsto 5; \mathtt{a} \mapsto 0]}$$

(d) $$\dfrac{[\![\mathtt{1 <= n}]\!]^{\mathscr{B}}(\sigma) \qquad \dfrac{\dfrac{}{\langle \mathtt{r} \leftarrow \mathtt{n}, \sigma\rangle \Downarrow \sigma[\mathtt{r} \mapsto 3]} \quad \dfrac{}{\langle \mathtt{n} \leftarrow \mathtt{n - 2}, \sigma[\mathtt{r} \mapsto 3]\rangle \Downarrow \sigma[\mathtt{r} \mapsto 3; \mathtt{n} \mapsto 1]}}{\langle \mathtt{r} \leftarrow \mathtt{n;\ n} \leftarrow \mathtt{n - 2}, \sigma\rangle \Downarrow \sigma[\mathtt{r} \mapsto 3; \mathtt{n} \mapsto 1]} \quad \dfrac{\vdots}{\langle \mathtt{while\ (1 <= n)\ r} \leftarrow \mathtt{n;\ n} \leftarrow \mathtt{n - 2}, \sigma[\mathtt{r} \mapsto 3; \mathtt{n} \mapsto 1]\rangle \Downarrow \sigma[\mathtt{r} \mapsto 1; \mathtt{n} \mapsto -1]}}{\langle \mathtt{while\ (1 <= n)\ r} \leftarrow \mathtt{n;\ n} \leftarrow \mathtt{n - 2}, \sigma\rangle \Downarrow \sigma[\mathtt{r} \mapsto 1; \mathtt{n} \mapsto -1]}$$

(e) The last program does not terminate.

Properties of While Because we have defined the While language and its semantics precisely, we can now prove very general properties about them, and about *all* programmes we can write in While.

*** 7. On syntax, I mentioned that the associativity of sequential composition did not in fact matter, and chose to make it right associative. We'll first explore the choice, then the claim.

(a) Construct a derivation for the following two programs in some abstract state σ, justifying it fully using the rules of the semantics for While:

1. `x ← y; (z ← x; y ← z)`, and
2. `(x ← y; z ← x); y ← z`.

(b) Show that, for any three statements s_1, s_2 and s_3 and any states σ and σ', the following holds:

$$\langle s_1 \,;\, s_2 \,;\, s_3, \sigma\rangle \Downarrow \sigma' \iff \langle (s_1 \,;\, s_2) \,;\, s_3, \sigma\rangle \Downarrow \sigma'$$

Solution

(a) The observation to make here is that the leaf derivations are identical in both trees.

1. $$\dfrac{\dfrac{\dfrac{}{\langle \texttt{x} \leftarrow \texttt{y}, \sigma\rangle \Downarrow \sigma_1}\,(\text{BAss}) \quad \dfrac{\dfrac{}{\langle \texttt{z} \leftarrow \texttt{x}, \sigma_1\rangle \Downarrow \sigma_2}\,(\text{BAss}) \quad \dfrac{}{\langle \texttt{y} \leftarrow \texttt{z}, \sigma_2\rangle \Downarrow \sigma_3}\,(\text{BAss})}{\langle \texttt{z} \leftarrow \texttt{x};\ \texttt{y} \leftarrow \texttt{z}, \sigma_1\rangle \Downarrow \sigma_3}\,(\text{BSeq})}{\langle \texttt{x} \leftarrow \texttt{y};\ \texttt{z} \leftarrow \texttt{x};\ \texttt{y} \leftarrow \texttt{z}, \sigma\rangle \Downarrow \sigma_3}\,(\text{BSeq})$$

2. $$\dfrac{\dfrac{\dfrac{}{\langle \texttt{x} \leftarrow \texttt{y}, \sigma\rangle \Downarrow \sigma_1}\,(\text{BAss}) \quad \dfrac{}{\langle \texttt{z} \leftarrow \texttt{x}, \sigma_1\rangle \Downarrow \sigma_2}\,(\text{BAss})}{\langle \texttt{x} \leftarrow \texttt{y};\ \texttt{z} \leftarrow \texttt{x}, \sigma\rangle \Downarrow \sigma_2}\,(\text{BSeq}) \quad \dfrac{}{\langle \texttt{y} \leftarrow \texttt{z}, \sigma_2\rangle \Downarrow \sigma_3}\,(\text{BAss})}{\langle \texttt{x} \leftarrow \texttt{y};\ \texttt{z} \leftarrow \texttt{x};\ \texttt{y} \leftarrow \texttt{z}, \sigma\rangle \Downarrow \sigma_3}\,(\text{BSeq})$$

(b) Deconstruct one derivation and judiciously use its sub-derivations to reconstruct another.

The While language (and its expressions and boolean expressions) are also defined inductively. So are their semantics and the notions of derivation we have seen! This means we can reason about all those objects inductively.

Let us show some kind of *frame rule* for While programmes: the meaning of a While programme s in state σ only depends on the value given by σ to variables that are used in s before being modified by s. When a final state σ' exists, the value it gives to variables not modified by s is the value σ gives to those variables.

We must first define those sets of variables: $\mathsf{MV}(s)$ is the set of variables *modified by* statement s; $\mathsf{FV}(e)$ is the set of variables that are *free in expression* e; and $\mathsf{UV}(s)$ is the set of variables that are *used in statement* s (meaning, used before they are modified).

Note that the set of free variables in a statement will be an overapproximation of the true set. Alex might explain why we are content with this.

$$\begin{aligned}
\mathsf{MV}(\epsilon) &= \emptyset \\
\mathsf{MV}(\mathtt{x} \leftarrow a) &= \{\mathtt{x}\} \\
\mathsf{MV}(s_1\,;\,s_2) &= \mathsf{MV}(s_1) \cup \mathsf{MV}(s_2) \\
\mathsf{MV}(\mathtt{if}\ b\ \mathtt{then}\ s_1\ \mathtt{else}\ s_2) &= \mathsf{MV}(s_1) \cup \mathsf{MV}(s_2) \\
\mathsf{MV}(\mathtt{while}\ b\ s) &= \mathsf{MV}(s)
\end{aligned}$$

$$\begin{aligned}
\mathsf{FV}(\mathtt{n}) &= \emptyset \\
\mathsf{FV}(\mathtt{x}) &= \{\mathtt{x}\} \\
\mathsf{FV}(a_1 + a_2) &= \mathsf{FV}(a_1) \cup \mathsf{FV}(a_2) \\
\mathsf{FV}(a_1 - a_2) &= \mathsf{FV}(a_1) \cup \mathsf{FV}(a_2) \\
\mathsf{FV}(a_1 * a_2) &= \mathsf{FV}(a_1) \cup \mathsf{FV}(a_2)
\end{aligned}$$

$$\begin{aligned}
\mathsf{FV}(\mathtt{true}) &= \emptyset \\
\mathsf{FV}(\mathtt{false}) &= \emptyset \\
\mathsf{FV}(\mathtt{!}b) &= \mathsf{FV}(b) \\
\mathsf{FV}(b_1\ \&\&\ b_2) &= \mathsf{FV}(b_1) \cup \mathsf{FV}(b_2) \\
\mathsf{FV}(b_2\ \mid\mid\ b_2) &= \mathsf{FV}(b_1) \cup \mathsf{FV}(b_2) \\
\mathsf{FV}(a_1 = a_2) &= \mathsf{FV}(a_1) \cup \mathsf{FV}(a_2) \\
\mathsf{FV}(a_1 <= a_2) &= \mathsf{FV}(a_1) \cup \mathsf{FV}(a_2)
\end{aligned}$$

$$\begin{aligned}
\mathsf{UV}(\epsilon) &= \emptyset \\
\mathsf{UV}(\mathtt{x} \leftarrow a) &= \mathsf{FV}(a) \\
\mathsf{UV}(s_1\,;\,s_2) &= \mathsf{UV}(s_1) \cup (\mathsf{UV}(s_2) \setminus \mathsf{MV}(s_1)) \\
\mathsf{UV}(\mathtt{if}\ b\ \mathtt{then}\ s_1\ \mathtt{else}\ s_2) &= \mathsf{FV}(b) \cup \mathsf{UV}(s_1) \cup \mathsf{UV}(s_2) \\
\mathsf{UV}(\mathtt{while}\ b\ s) &= \mathsf{FV}(b) \cup \mathsf{UV}(s)
\end{aligned}$$

**** 8.

(a) Show that, for any arithmetic expression a and any two states σ_1 and σ_2, if for any $\mathtt{x} \in \mathsf{FV}(a)$ we have $\sigma_1(\mathtt{x}) = \sigma_2(\mathtt{x})$, then we have $[\![\mathtt{a}]\!]^{\mathscr{A}}(\sigma_1) = [\![\mathtt{a}]\!]^{\mathscr{A}}(\sigma_2)$.
Hint: By structural induction over a.

(b) Show that, for any boolean expression b and any two states σ_1 and σ_2, if for any $\mathtt{x} \in \mathsf{FV}(b)$ we have $\sigma_1(\mathtt{x}) = \sigma_2(\mathtt{x})$, then we have $[\![\mathtt{b}]\!]^{\mathscr{B}}(\sigma_1) = [\![\mathtt{b}]\!]^{\mathscr{B}}(\sigma_2)$.

(c) Show that, for any statement s and any two state σ and σ', if $\langle s, \sigma\rangle \Downarrow \sigma'$ then, for every $\mathtt{x} \notin \mathsf{MV}(s)$ we have $\sigma'(\mathtt{x}) = \sigma(\mathtt{x})$.
Hint: An induction over the statement s will fail. Can you tell why? Hint: You will need to reason by induction over the derivation of $\langle s, \sigma\rangle \Downarrow \sigma'$.

(d) Show that, for any statement s and any two states σ_1 and σ_2, if for any $\mathtt{x} \in \mathsf{UV}(s)$ we have $\sigma_1(\mathtt{x}) = \sigma_2(\mathtt{x})$, then for every σ_1' such that $\langle s, \sigma_1\rangle \Downarrow \sigma_1'$, there exists σ_2' such that $\langle s, \sigma_2\rangle \Downarrow \sigma_2'$ and $\sigma_1'(\mathtt{x}) = \sigma_2'(\mathtt{x})$ for every $\mathtt{x} \in \mathsf{MV}(s)$.
This one requires creativity—as in, you'll need to generalise a few things to find a property that is inductive.

Solution

(a) By structural induction over a, also noting that agreement over the free variables of a implies agreement over the free variables of its subexpressions.

(b) By structural induction over b, also noting that agreement over the free variables of b implies agreement over the free variables of its subexpressions.

(c) This *has* to be done by induction over the derivation, because derivation over the statement is not well-founded—`while` is not well-behaved—and finding a general decreasing measure over configurations would solve Halt. Once this is understood, and induction over derivations clicks, the proof is straightforward.

(d) This one is wide open. See François for feedback and hints.