

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 $\sigma = \begin{bmatrix} a \mapsto 42 \\ b \mapsto 154 \\ x \mapsto 11 \end{bmatrix}$

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

(b) $x * (a - x) * b$

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

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

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

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

(a) $a + x * 14 - b \leq b + x \ \&\& \ \text{true}$

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

(c) $b - g * x + h \leq 0$

```

r ← 1;
while (1 ≤ n) {
  r ← n * r;
  n ← n - 1
}

```

Figure 1: A factorial program.

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

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

$$\sigma = \left[\begin{array}{l} n \mapsto 3 \\ a \mapsto 5 \end{array} \right]$$

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. $x \leftarrow y; (z \leftarrow x; y \leftarrow z)$, and
 2. $(x \leftarrow y; z \leftarrow x); y \leftarrow z$.
- (b) Show that, for any three statements s_1, s_2 and s_3 and any states σ and σ' , the following holds:

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

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: $MV(s)$ is the set of variables *modified* by statement s ; $FV(e)$ is the set of variables that are *free in expression* e ; and $UV(s)$ 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.

$MV(\epsilon) = \emptyset$	$FV(n) = \emptyset$
$MV(x \leftarrow a) = \{x\}$	$FV(x) = \{x\}$
$MV(s_1; s_2) = MV(s_1) \cup MV(s_2)$	$FV(a_1 + a_2) = FV(a_1) \cup FV(a_2)$
$MV(\text{if } b \text{ then } s_1 \text{ else } s_2) = MV(s_1) \cup MV(s_2)$	$FV(a_1 - a_2) = FV(a_1) \cup FV(a_2)$
$MV(\text{while } b \text{ } s) = MV(s)$	$FV(a_1 * a_2) = FV(a_1) \cup FV(a_2)$
	$FV(\text{true}) = \emptyset$
	$FV(\text{false}) = \emptyset$
$UV(\epsilon) = \emptyset$	$FV(!b) = FV(b)$
$UV(x \leftarrow a) = FV(a)$	$FV(b_1 \ \&\& \ b_2) = FV(b_1) \cup FV(b_2)$
$UV(s_1; s_2) = UV(s_1) \cup (UV(s_2) \setminus MV(s_1))$	$FV(b_2 \ \ b_2) = FV(b_1) \cup FV(b_2)$
$UV(\text{if } b \text{ then } s_1 \text{ else } s_2) = FV(b) \cup UV(s_1) \cup UV(s_2)$	$FV(a_1 = a_2) = FV(a_1) \cup FV(a_2)$
$UV(\text{while } b \text{ } s) = FV(b) \cup UV(s)$	$FV(a_1 \leq a_2) = FV(a_1) \cup FV(a_2)$

**** 8.

- (a) Show that, for any arithmetic expression a and any two states σ_1 and σ_2 , if for any $x \in FV(a)$ we have $\sigma_1(x) = \sigma_2(x)$, then we have $\llbracket a \rrbracket^{\mathcal{A}}(\sigma_1) = \llbracket a \rrbracket^{\mathcal{A}}(\sigma_2)$.
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 $x \in FV(b)$

we have $\sigma_1(x) = \sigma_2(x)$, then we have $\llbracket b \rrbracket^{\mathcal{B}}(\sigma_1) = \llbracket b \rrbracket^{\mathcal{B}}(\sigma_2)$.

- (c) Show that, for any statement s and any two state σ and σ' , if $\langle s, \sigma \rangle \Downarrow \sigma'$ then, for every $x \notin \text{MV}(s)$ we have $\sigma'(x) = \sigma(x)$.
- (d) Show that, for any statement s and any two states σ_1 and σ_2 , if for any $x \in \text{UV}(s)$ we have $\sigma_1(x) = \sigma_2(x)$, 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 $\sigma'_1(x) = \sigma'_2(x)$ for every $x \in \text{MV}(s)$.

This one requires creativity—as in, you'll need to generalise a few things to find an inductive property.