Exercise Sheet 2: (Big Step) Operational Semantics

Due date: April 28th. You can hand in your solutions at the start of the exercise class.

Exercise 1 (Example of Derivation Tree) — 10%

Consider program

$$c: \quad z := 0;\ \mathsf{while}\ y \leq x\ \mathsf{do}\ \{z := z+1;\ x := x-y\}\ .$$

(a) [5%] Describe colloquially the effect of the program.

(b) [5%] Depict the derivation tree for $\langle c, \sigma\rangle \to \sigma'$, where σ is an initial state with $\sigma(x)=13$ and $\sigma(y)=5$. (You need not write derivations for the evaluation relation of arithmetic or Boolean expressions that show up as side conditions in the rules of the execution relation.)

Exercise 2 (Language Extension) — 10%

Extend the set of rules defining the execution relation $\to \subseteq (\mathsf{Cmd} \times \Sigma) \times \Sigma$ for incorporating statement $\mathsf{repeat}\ c\ \mathsf{until}\ b$. (The semantics of the repeat–until construct is not allowed to depend on the existence of the while–do construct in the language.)

Exercise 3 (Operational Equivalence) — 35%

In this exercise, we add some common statements of imperative programming languages to `WHILE` and provide their operational semantics. All in all, the new syntax of statements is given by the following grammar:

$$c' ::= c \mid x{+}{+} \mid \mathsf{for}\ c, b, c\ \mathsf{do}\ c \mid \mathsf{case}\ x\ \mathsf{of}\ s$$
$$s ::= a : c; s \mid \varepsilon$$

Here a, b, c, x are as in the lecture and ε represents the empty word.

(a) [2.5%] Extend the execution relation $\to$ developed in the lecture by an SOS rule for the increment operator $x{+}{+}$.

(b) [7.5%] Extend the execution relation $\to$ developed in the lecture by an SOS rule for the new construct $\mathsf{for}\ c_1, b, c_2\ \mathsf{do}\ c_3$. For example,

- $\langle \mathsf{for}\ x := 0, x * x < y, x := x{+}{+}\ \mathsf{do}\ y := y + 2, \{x \mapsto 2, y \mapsto 42\}\rangle \to \{x \mapsto 8, y \mapsto 58\}$ and
- $\langle \mathsf{for}\ x := 1, x < y, x := x * y\ \mathsf{do}\ y := y + y, \{x \mapsto 0, y \mapsto 1\}\rangle \to \{x \mapsto 1, y \mapsto 1\}$.

(c) [10%] Let c_1 be the program $y := 2;\ x := 1; \mathsf{while}\ x < y\ \mathsf{do}\ \{x := x * 4; y := y + 1\}$ and c_2 be the program $\mathsf{for}\, x := 1; y := 2, x < y, y{+}{+}\, \mathsf{do}\, x := x * 4$. Show that c_1 and c_2 are operationally equivalent, i.e. for each $\sigma \in \Sigma$, $\langle c_1, \sigma\rangle \to \sigma' \Leftrightarrow \langle c_2, \sigma\rangle \to \sigma'$.

(d) [5%] Extend the execution relation $\to$ developed in the lecture by an SOS rule for the new construct $\mathsf{case}\ x\ \mathsf{of}\ s$. Informally, this construct executes command c where c is the first occurence of a pair $a : c$ in s such that x equals the value of a.

(e) [10%] Provide a recursive function trans_x that takes a sequence s and transforms it into a statement c in the original `WHILE` language such that for all $\sigma \in \Sigma$, $\langle \mathsf{case}\ x\ \mathsf{of}\ s, \sigma\rangle \to \sigma'$ if and only if $\langle \mathrm{trans}_x(s), \sigma\rangle \to \sigma'$. Prove the correctness of your provided function.

Exercise 4 (Terminating Loops) **10%**

Show that if there exists state σ' such that $\langle \mathsf{while}\ b\ \mathsf{do}\ c, \sigma\rangle \to \sigma'$, then $\langle b, \sigma'\rangle \to \mathsf{false}$. Proof should proceed by induction over the structure of the derivation tree.

Exercise 5 (Program Behaviour) **35%**

For this exercise we consider a variant of the WHILE language without if–then–else statements and that contains repeat–until loops rather than while–do loops. Items (a) through (c) should adopt this language and item (c) should rely on the semantics of repeat–until loops provided in Exercise 2.

(a) [5%] Define a recursive function $\mathtt{mod}\colon \mathsf{Cmd} \to \mathcal{P}(\mathsf{Var})$ that computes the set of variables written by a program (i.e. those variables in the left hand side of assignments).

(b) [5%] Define a recursive function $\mathtt{dep}\colon \mathsf{Cmd} \to \mathcal{P}(\mathsf{Var})$ that computes the set of variables read by a program (i.e. those variables in the right hand side of assignments or in the guard of loops). You can assume that function $\mathtt{fv}\colon \mathsf{AExp} \cup \mathsf{BExp} \to \mathcal{P}(\mathsf{Var})$ (which computes the set of variables in arithmetic and Boolean expressions) is already given.

(c) [20%] Prove that the behaviour of a program is determined by the set of variables it reads. To formally state this claim, we require the auxiliary relation $=_R$ between program states given by clause

$$\sigma_1 =_R \sigma_2 \quad \triangleq \quad \forall x \in R \bullet\ \sigma_1(x) = \sigma_2(x)\ .$$

Said otherwise, two states are related by $=_R$ iff they assign the same value to variables in R. Now our main claim (to prove) can be stated as follows:
Consider program c and a pair of initial states σ_1 and σ_2 with $\sigma_1 =_{\mathtt{dep}(c)} \sigma_2$. Then for every pair of final states σ_1' and σ_2' such that $\langle c, \sigma_1\rangle \to \sigma_1'$ and $\langle c, \sigma_2\rangle \to \sigma_2'$, it holds $\sigma_1' =_{\mathtt{mod}(c)} \sigma_2'$.
Proof should proceed by induction over the structure of the derivation tree. For the proof you can rely, if necessary, on the following two auxiliary results (you need not prove them):

1. $\langle c, \sigma\rangle \to \sigma' \wedge x \notin \mathtt{mod}(c) \implies \sigma'(x) = \sigma(x)$.
2. $\sigma_1 =_{\mathtt{dep}(c)} \sigma_2 \implies (\exists \sigma_1' \bullet \langle c, \sigma_1\rangle \to \sigma_1' \Leftrightarrow \exists \sigma_2' \bullet \langle c, \sigma_2\rangle \to \sigma_2')$.

(d) [5%] Does the result in (c) remain valid if we include if–then–else statements in the language? Justify your answer.