

CMSC 631, Fall 2007 Homework 2

Due Wednesday, Oct. 3, in class

1. For the language of commands that we saw in class (Figure 2, not including the boxed portions), show derivations that establish the following as legal evaluations:

(a) $((1 * 2) + (3 * x), s) \rightarrow^* (8, s)$ and likewise $((1 * 2) + (3 * x), s) \Downarrow (8, s)$ where $s(x) = 2$.

(b) **while** x **do** $(x := x + (-1)), s) \rightarrow^* s'$ where $s(x) = 1$ and $s' = s[x \mapsto 0]$, and likewise for \Downarrow .

(Recall that \rightarrow is the small-step relation, \Downarrow is the big-step relation. Also, the syntax $s[x \mapsto i]$ defines a new map s' such that $s'(x) = i$ and $s'(y) = s(y)$ for all $y \neq x$.) *Idea: modify your interpreter from project 1 to produce these derivations for you!*

2. For the language of arithmetic expressions that we discussed in class (Figure 1, the syntactic class e , not including the boxed portions), flesh out the proof of *equivalence* of the big-step and small-step semantics, given on slide 31 of the operational semantics lecture notes. In particular, prove that $eval((e, s), i) \text{ iff } (e, s) \Downarrow i$ where $eval((e, s), i)$ holds if $(e, s) \rightarrow^* (i, s)$. You don't have to consider the case for $e_1 * e_2$ since it's structurally identical to the $e_1 + e_2$ case.

3. Suppose we modify the language of commands and expressions to add *lists of integers* as a legal data structure in our language, along with a *pattern matching* operation for accessing the elements of a list. The new language is shown in Figure 2, with the new constructs in boxes. Define small and big-step semantics for this new language. Here are a few more details:

A list l is a series of comma-separated integers, concluded by a *nil*. For example, $(1, nil)$ is a list, and $(1, (2, (3, nil)))$ is a list. We've extended expressions e with the empty list *nil*, and the form e_1, e_2 which will evaluate to a list assuming that e_1 evaluates to an integer and e_2 evaluates to a list. To access the elements of a list, we use the **case** expression to do pattern matching. In the command **case** e **of** $nil \Rightarrow c_1$ **or** $(x :: y) \Rightarrow c_2$, if e evaluates to *nil*, then the next step is to evaluate c_1 . If e evaluates to some list (i, l) , then the next step is to evaluate c_2 in the current store modified to map x to i and y to l .

<i>variables</i>	x, y, z	\in	V
<i>integers</i>	i, j, k	\in	\mathcal{Z}
<i>lists</i>	l, m	$::=$	$\boxed{nil \mid i, l}$
<i>expressions</i>	e	$::=$	$x \mid i \mid e_1 + e_2 \mid e_1 * e_2 \mid \boxed{e_1, e_2 \mid nil}$
<i>values</i>	v	$::=$	$i \mid \boxed{l}$
<i>commands</i>	c, d	$::=$	$\mathbf{skip} \mid x := e \mid \mathbf{if}_{not0} i \mathbf{then} c_1 \mathbf{else} c_2 \mid \mathbf{while}_{not0} i \mathbf{do} c$ $\mid c_1; c_2 \mid \boxed{\mathbf{case} e \mathbf{of} nil \Rightarrow c_1 \mathbf{or} (x :: y) \Rightarrow c_2}$

Figure 1: Extended language of commands