

CMSC 631, Fall 2006

Homework 2

Due Wednesday, Sept. 27, in class

1. For the language of commands that we saw in class (Figure 2), 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) $(x := 1; \mathbf{while} \ x \ \mathbf{do} \ (x := x + (-1)), s) \rightarrow^* s'$ where $s' = s[x \mapsto 0]$ 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 $y \neq x$.)

2. For the language of arithmetic expressions that we discussed in class (Figure 1), prove that for all e and s where $\text{dom}(s) \supseteq \text{vars}(e)$, either e is an integer i or else there exists an e' such that $(e, s) \rightarrow (e', s)$. We define $\text{vars}(e)$ as follows:

$$\begin{aligned} \text{vars}(x) &= \{x\} \\ \text{vars}(i) &= \emptyset \\ \text{vars}(e_1 + e_2) &= \text{vars}(e_1) \cup \text{vars}(e_2) \\ \text{vars}(e_1 * e_2) &= \text{vars}(e_1) \cup \text{vars}(e_2) \end{aligned}$$

You should prove this by induction on the height of the derivation $(e, s) \rightarrow (e', s)$.

3. Suppose we modify the language of commands and expressions in two ways. First, we add a new class of *boolean expressions* and modify the conditional operators **if** and **while** to use boolean guards. Second, we 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 3. Define small and big-step semantics for this new language. Here are a few more details about the language:

Boolean expressions These are largely straightforward; you should follow the same ideas as arithmetic expressions in defining their semantics. (Note that \vee is boolean “or”, \wedge is boolean “and”, and $!$ is boolean “not”.) You should assume that two expressions are equal if they are syntactically the same. That is, $v_1 = v_2$ if v_1 has exactly the same syntactic structure as v_2 . Note that you only define equality in terms of *values*, which are either integers i or integer lists l . In all cases, expressions will evaluate to values.

Lists A list l is a series of comma-separated integers, concluded by a *nil*. For example, $(1, \text{nil})$ is a list, and $(1, (2, (3, \text{nil})))$ 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 whole command reduces to c_1 . If e evaluates to some list (i, l) , then the whole thing evaluates to c_2 , and in the current store is modified to map x to i and y to l .

4. Can you prove the theorem in problem (2), above, for the new language defined in problem (3)? That is: for all c and s where $dom(s) \supseteq vars(c)$, there exists an e' and s' such that either $(c, s) \rightarrow s'$ or $(c, s) \rightarrow (c', s')$. (Here, $vars$ is extended to commands in the natural way.) If you can prove this, sketch the proof (you don't need to go into all the details). Otherwise, explain why it cannot be proved.

variables $x, y, z \in V$
integers $i, j, k \in \mathcal{Z}$
expressions $e ::= x \mid i \mid e_1 + e_2 \mid e_1 * e_2$

Figure 1: Language of arithmetic expressions

commands $c, d ::= \mathbf{skip} \mid x := e \mid \mathbf{if } e \mathbf{ then } c_1 \mathbf{ else } c_2 \mid \mathbf{while } e \mathbf{ do } c$

Figure 2: Language of commands (includes language of expressions)

variables $x, y, z \in V$
integers $i, j, k \in \mathcal{Z}$
lists $l, m ::= nil \mid i, l$
bool.expressions $a, b ::= true \mid false \mid b_1 \wedge b_2 \mid b_1 \vee b_2 \mid !b \mid e_1 = e_2$
expressions $e ::= x \mid i \mid e_1 + e_2 \mid e_1 * e_2 \mid e_1, e_2 \mid nil$
values $v ::= i \mid l$
commands $c, d ::= \mathbf{skip} \mid x := e \mid \mathbf{if } b \mathbf{ then } c_1 \mathbf{ else } c_2 \mid \mathbf{while } b \mathbf{ do } c$
 $\mid c_1; c_2 \mid \mathbf{case } e \mathbf{ of } nil \Rightarrow c_1 \mathbf{ or } (x :: y) \Rightarrow c_2$

Figure 3: Extended language of commands