Semantics of Programming Languages (PMS)

Consider the imperative language syntax below. Here \( n \) ranges over 32-bit numbers \( \mathbb{N}_{32} = [0, ..., 2^{32} - 1] \), with modular addition \( \oplus \), and \( x \) ranges over an infinite set of identifiers.

\[
e ::= n \mid \text{ref } e \mid !e \mid e ::= e' \mid \text{skip} \mid e; x \mid \text{let } x = e \text{ in } e'
\]

We give it two semantics. The first extends the syntax with abstract locations \( l \) (taken from some infinite set \( L \)) and has an abstract store \( s \), a finite partial function from abstract locations to values \( v ::= n \mid l \). The initial abstract store \( s_0 \) is the partial function with empty domain. The semantic rules are all standard; the most interesting are shown below for reference.

\[
\langle e_1, s_1 \rangle \rightarrow \langle e_2, s_2 \rangle
\]

\[
\begin{array}{ll}
\text{REF1} & l \notin \text{dom}(s) \\
\text{DEREF1} & l \in \text{dom}(s) \land s(l) = v \\
\text{ASSIGN1} & l \in \text{dom}(s)
\end{array}
\]

For the second semantics we have a concrete store \( M \), a total function from concrete addresses \( n \in \mathbb{N}_{32} \) to values which here are also just numbers \( n' \in \mathbb{N}_{32} \), together with a counter \( a \in \mathbb{N}_{32} \) that records the next unallocated address. This semantics uses the abstract syntax exactly as above, without abstract locations. The initial concrete store \( M_0 \) maps all addresses to 0; the initial \( a_0 = 0 \). The interesting rules are:

\[
\langle e_1, M_1, a_1 \rangle \Rightarrow \langle e_2, M_2, a_2 \rangle
\]

\[
\begin{array}{ll}
\text{REF1}' & M(n) = n' \\
\text{DEREF1}' & \langle \text{let } x = \text{ref } 3 \text{ in } e'; !x \rangle
\end{array}
\]

Consider expressions \( e \) of the form \( \text{let } x = \text{ref } 3 \text{ in } e'; !x \), where \( e' \) does not contain any free occurrences of \( x \) or any abstract locations \( l \).

(a) Can \( e \) (with the initial store) reduce to a value different from 3, (i) in the abstract semantics or (ii) in the concrete semantics? In each case, either give an example and explain it or give a careful informal argument why not. [8 marks]

(b) Define a large subset of the expressions that reduce to the same value in both semantics. Explain your answer. [8 marks]

(c) Discuss the advantages and disadvantages of the two semantics for a C-like systems programming language. [4 marks]