3 Semantics of Programming Languages (pes20)

Consider the following syntax of an assembly language. It is similar to RV32 except that each memory location holds a 32-bit word, from a set $\texttt{u32} = \{0, \ldots, 2^{32} - 1\}$. It has instructions to load, store, add, branch-on-equal, and jump-and-link.

$$\begin{array}{l} \textit{register\_name},\ r,\ rd,\ rs,\ rs_a,\ rs_d ::= \ \texttt{x1} \mid \texttt{x2} \mid \texttt{x3} \\ \textit{instruction},\ i ::= \ \texttt{lw}\, rd, rs_1, imm \mid \texttt{sw}\, rs_a, rs_d, imm \mid \texttt{addi}\, rd, rs_1, imm \mid \texttt{beq}\, rs_1, rs_2, imm \\ \mid \texttt{jalr}\, rd, rs_1, imm \end{array}$$

A machine state is a tuple $\langle pc, R, M\rangle$ of a PC value $pc \in \texttt{u32}$, a register state $R : \textit{register\_name} \to \texttt{u32}$, and a partial memory state $M : \texttt{u32} \rightharpoonup \texttt{u32}$, not necessarily defined for the whole address space. Assume there is a partial function $\texttt{decode} : \texttt{u32} \rightharpoonup \textit{instruction}$, then the semantics of the load instruction can be defined by:

$$\frac{\begin{array}{l} M(pc)\,\texttt{defined} \\ \texttt{decode}\,(M(pc)) = \texttt{lw}\, rd, rs_1, imm \\ n = R(rs_1) + imm \\ M(n)\,\texttt{defined} \\ M(n) = n' \end{array}}{\langle pc, R, M\rangle \to \langle pc + 1, R + \{rd \mapsto n'\}, M\rangle}\ \textsc{lw}$$

(*a*) Give operational semantics rules for the other instructions. Comment briefly on any choices you had to make. [8 marks]

From now on, consider just the load, store, and add instructions. We want to impose a type discipline that distinguishes between integer and pointer values, both in registers and in memory, with types T that are either `uint` for integer values, or T`*` for pointers to values of type T. Suppose that no pointer arithmetic is allowed. Let Γ range over finite partial functions from register names and addresses to types.

(*b*) Define a judgement $\Gamma \vdash \langle R, M\rangle$ that checks that the register state R and memory state M are consistent with Γ, with every pointer-typed value being dereferenceable with a value of the appropriate type. Explain your definition briefly. [4 marks]

(*c*) Define a typing judgement $\Gamma \vdash i \dashv \Gamma'$ for instructions i where Γ is the type environment before i executes and Γ' is the type environment that can be assumed by the following instruction. Explain your definition briefly.

Your definition should be sound with respect to the operational semantics: if $\Gamma \vdash i \dashv \Gamma'$, assuming $\Gamma \vdash \langle R, M\rangle$ for some Γ before the instruction executes, with $M(pc)\,\texttt{defined}$ and $\texttt{decode}\,(M(pc)) = i$, then (1) there should exist some transition $\langle pc, R, M\rangle \to \langle pc', R', M'\rangle$, and (2) for any such transition, $\Gamma' \vdash \langle R', M'\rangle$. You should **not** prove this. [8 marks]