9 Semantics of Programming Languages (pes20)

Some programmers believe that C pointers behave just as machine-word addresses. That's not true for actual C, but it's interesting to explore what some of the consequences would be if it were.

(*a*) Give a semantics with machine-word (64-bit integer) pointers for the following vaguely C-ish language in which function arguments are mutable, allocated to fresh machine-word addresses on function entry, $\&x$ denotes the address allocated for x, and dereferencing and update are allowed on any memory location. The semantics should define a small-step transition relation $A, M, e \longrightarrow A', M', e'$, where memory $M : \mathbb{N}_{64} \to \mathbb{N}_{64}$ holds a 64-bit integer at each 64-bit integer address, and an allocation environment $A \subseteq \mathbb{N}_{64}$ records which addresses have been allocated so far.

$$e ::= \quad n \mid e + e' \mid \&x \mid *e \mid *e = e' \mid e; e' \mid \mathbf{fun}\,(x)\{e\} \mid e\; e'$$

In $\mathbf{fun}\,(x)\{e\}$ the x binds in e. [9 marks]

(*b*) Describe all the configurations in which your semantics gets stuck. [2 marks]

(*c*) Give two examples in which a function can observably modify the function argument of its caller, one syntactically obviously and one indirectly, and explain them briefly. [2 marks]

(*d*) Comment on situations in which optimisations that move code, such as common subexpression elimination, are invalid in your semantics, with an example. [1 mark]

(*e*) (*i*) Define a static type system for this language that prevents confusion of integer and pointer values, where the types of machine words are

$$T_{\mathrm{w}} ::= \quad \mathbf{int} \mid T_{\mathrm{w}}*$$

the types of expressions are

$$T ::= \quad T_{\mathrm{w}} \mid T_{\mathrm{w}} \to T'_{\mathrm{w}}$$

and functions are type-annotated $\mathbf{fun}\,(x : T_{\mathrm{w}})\{e : T'_{\mathrm{w}}\}$. [4 marks]

(*ii*) State which of your Part (*b*) configurations are ruled out by the type system. [2 marks]