

CMSC 631, Fall 2007

Homework 3

Due Friday, October 12, at my office by 2pm

- For each type, construct a simply-typed lambda calculus term (variables, functions, and function application only) whose *most general* type is that type, or argue that no term has that type. It should have the type you specify in the *empty* type environment; i.e., it has type τ such that $\vdash e : \tau$.

(Hint: You can double-check your answers in OCaml. *Extra credit:* for any type that has no simply-typed lambda calculus term, give an OCaml term that does have the type *without using the $:$ operator to assign a type.*)

- $\alpha \rightarrow \beta \rightarrow \beta$
- $(\alpha \rightarrow \beta \rightarrow \gamma) \rightarrow \beta \rightarrow \alpha \rightarrow \gamma$
- $\alpha \rightarrow \beta$
- $\alpha \rightarrow \alpha \rightarrow \alpha$

- Does the simply-typed lambda calculus with integers have a *subject expansion* property, meaning if $\Gamma \vdash e : \tau$ and $e' \rightarrow e$, does $\Gamma \vdash e' : \tau$? Here \rightarrow is reduction under call-by-value semantics. Either prove that subject expansion holds, or give a counterexample showing that it does not hold.
- Recall the extended language of commands we saw in the last homework (see Figure 1). Define a type system for this language, defining the syntax of types τ and type environments Γ , and inference rules for the judgments $\Gamma \vdash e : \tau$ (“in environment Γ expression e has type τ); $\Gamma \vdash s$ (“the types of variables in store s have the types assigned by Γ ”), and $\Gamma \vdash c$ (“command c is well-typed in environment Γ ”). Here is an example rule from the last judgment:

$$\text{TIf} \frac{\Gamma \vdash c_1 \quad \Gamma \vdash c_2 \quad \Gamma \vdash e : \text{int}}{\Gamma \vdash \text{if}_{\text{not0}} e \text{ then } c_1 \text{ else } c_2}$$

The goal of your type system is soundness, in other words: if $\Gamma \vdash s$ and $\Gamma \vdash c$ then either $(s, c) \rightarrow^* s'$ for some s' or else (s, c) diverges. *Extra credit:* prove this theorem.

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

Figure 1: Extended language of commands