

**G22.3110 Honors Programming Languages  
Type Theory Homework**

**Due Monday, December 20**

1. In the simply typed lambda calculus,  $\lambda^{\rightarrow}$ , prove the following typing assertions using the typing axioms and inference rules.
  - a)  $x:\sigma, y:\sigma \rightarrow \tau \triangleright yx: \tau$
  - b)  $x:\sigma, y:\sigma \triangleright \lambda y:\sigma \rightarrow \tau. yx: (\sigma \rightarrow \tau) \rightarrow \tau$
  - c)  $\emptyset \triangleright (\lambda x:\sigma \rightarrow \sigma. \lambda y:\tau. x) (\lambda x:\sigma. x) : \tau \rightarrow \sigma \rightarrow \sigma$
2. In the predicative polymorphic calculus,  $\lambda^{\rightarrow, \Pi}$ , sketch the derivations of the following typing judgements.
  - a)  $\emptyset \triangleright \lambda s:U_1. \lambda t:U_1. \lambda x:s \rightarrow t. \lambda y:s. xy: \Pi s:U_1. \Pi t:U_1. (s \rightarrow t) \rightarrow s \rightarrow t$
  - b)  $s:U_1, x: s, y:s \triangleright \lambda z:s \rightarrow s \rightarrow s. zxy: (s \rightarrow s \rightarrow s) \rightarrow s$
  - c)  $s:U_1, x: s \rightarrow s \triangleright (\lambda y:s \rightarrow s. x) (\lambda x:s. x): s \rightarrow s$

3. In  $\lambda^{\rightarrow, \Pi, \text{let}}$ , the predicative polymorphic calculus extended with the ML-style let construct, show the derivation for

$$\emptyset \triangleright \text{let } f: \Pi t:U_1. t \rightarrow t = \lambda t:U_1. \lambda x:t. x \text{ in if } f \text{ true then } f \text{ else } 4: \text{nat}$$

Be sure to specify the relevant portions of the signature  $\Sigma$  of the language.

4. Write the full term (i.e. including the types) for the pre-term  $(\lambda x. x x)$  in the impredicative calculus.
5. Write the full term for a very simple abstract data type (using **abstype**). Be sure to include the type information.
6. What does the term *contravariance* refer to in the context of  $\lambda_{<}^{\rightarrow}$ , the simply typed lambda calculus with subtyping? Give the typing rule that induces the contravariance.
7. Given the signature of a language in  $\lambda_{<}^{\rightarrow}$  as  $\Sigma = \langle \{nat, int, real\}, \{nat <: int, int <: real\}, \dots \rangle$ , show the subtyping hierarchy of types of the form  $(\sigma \rightarrow \tau) \rightarrow \rho$ , where each of  $\sigma, \tau$ , and  $\rho$  are either *nat*, *int*, or *real*.
8. Given the type *point* as defined in the lecture notes, show that the type *movable* defined by

$$\text{type movable} = \langle \text{move: int} \rightarrow \text{int} \rightarrow \text{movable} \rangle$$

is a supertype of *point*.

9. Write, using a lambda calculus with F-bounded subtype polymorphism, a polymorphic *sort* procedure for sorting lists of elements of any record type  $\sigma$  providing the *lessthan* method of type  $\sigma \rightarrow \sigma \rightarrow \text{bool}$ . However, don't write the actual code for the sort function, just write its signature, along with any necessary type declarations. Assume  $\sigma \text{ list}$  is a valid type, for any type  $\sigma$ .