

Unassessed Exercise Sheet 5+

Attempt as many questions as you can during the exercise class, and work on the remainder at home. The model solutions will be made available on Thursday, the 4th March. Hand in your solutions to the tutor in the Friday exercise class, on 5 Mar, 2010, to receive feedback.

Exercise 1: Type rules

letrec can be regarded as syntactic sugar in ML (which has a built-in **let** construct):

$$\mathbf{letrec} \ f = M \ \mathbf{in} \ N \quad \equiv \quad \mathbf{let} \ f = \mathbf{Y}(\lambda f. M) \ \mathbf{in} \ N$$

Show that **letrec** has the following derived type rule:

$$\frac{\Gamma, f : T \vdash M : T \quad \Gamma, f : \forall \vec{t}. T \vdash N : T'}{\Gamma \vdash \mathbf{letrec} \ f = M \ \mathbf{in} \ N : T'}$$

by giving a type derivation for the desugared form of **letrec**. (The type variables \vec{t} used in the quantification $\forall \vec{t}$ include all the free type variables of T that do not occur as free type variables in Γ .)

Exercise 2: Type inference

Given below is the definition of a function *take* which selects the first few elements of a list:

$$\mathbf{letrec} \ take = \lambda n. \lambda l. \ \text{if } (n = 0) \ nil \\ \quad (\text{cons } (\text{car } l) \ (take \ (n - 1) \ (\text{cdr } l)))$$

Calculate the principal type (most general type) of *take*, using type inference. (You don't need to show the type derivation.)

Exercise 3: Explicit polymorphism

- Rewrite the definition of *take* in the explicitly-typed polymorphic lambda calculus.
- Rewrite it again in Java with generics. (Assume that list objects have methods `null`, `car` and `cdr`, as in Handout 6, paragraph 11.)

Exercise 4: Higher-order functions

- Write the recursive definition of a higher-order function *filter*, which takes as arguments a boolean function p and a list l , and returns a list of all the elements in l for which p is true. For example:

$$filter \ even \ (\mathbf{list} \ 4 \ 5 \ 8 \ 13 \ 20)$$

should return a list containing 4, 8, 20.

- Calculate the principal type of *filter*.
- Rewrite *filter* in the explicitly-typed polymorphic lambda calculus.

Exercise 5: Higher-order functions

- a. Write the recursive definition of a higher-order function *map*, which takes as arguments a unary function *f* and a list *l*, and returns a list of values obtained by applying *f* to the elements of *l*. For example:

$$\text{filter } (\lambda x. x + 1) \text{ (list 4 5 8 13 20)}$$

should return a list containing 5, 7, 9, 14, 21.

- b. Calculate the principal type of *map*.