Final Exam

Type Systems

February 7, 2007

Last Name : ____________________

First Name : ____________________

Section : ____________________

Exercise	Points	Achieved Points
1	12	
2	10	
3	18	
Total	40	

Exercise 1 : References and Subtyping (12 points)

Let us consider the simply typed lambda calculus with subtyping and records, as defined in Chapter 15 of the book (TAPL, p.186 – 187). We want to augment the language with reference cells and propose the following subtyping rule:

$$\text{(S-REF)} \ \frac{S_1 <: T_1 \quad T_1 <: S_1}{\mathit{Ref}\ S_1 <: \mathit{Ref}\ T_1}$$

(1) Write a short program that will fail with a runtime type error (get stuck) if the *first* premise of the (S-REF) rule is dropped.

(2) Write another program that will fail if the *second* premise is dropped.

For both terms, write the evaluation steps until they get stuck.

You can assume your language provides `unit` and sequence.

Exercise 2 : Type Reconstruction (10 points)

Given the following lambda term t in the language used in Chapter 22 (TAPL, p.317):

$$\lambda x : X.\lambda y : Y.\lambda z : Z.(xz)(yz)$$

(1) Find *three* different solutions for the type of t in the empty context.

(2) Find the constraint set C for t.

(3) Find a principal type for t.

Exercise 3 : Object encoding (18 points)

Consider the following two classes written in Featherweight Java:

```
class A extends Object {
  Object a;
  A(Object a) { super(); this.a = a; }

  Object m() { return this.n(); }
  Object n() { return this.a;   }
}

class B extends  A {
  Object b;
  B(Object a, Object b) { super(a); this.b = b; }

  Object n() { return this.b; }
}
```

(1) Give an encoding in simply typed lambda calculus with records, `unit`, `let` and `fix`, along the lines you have seen in the lecture. You are allowed to define shortcut names for types in order to make terms more legible. Your encoding should work with a "call by value" evaluation strategy.

Hints:

- For each class, you will need to define a class type, an object state type, a function which generates the class, and another one which constructs objects of that class.
- In Featherweight Java constructors are fully determined by the class definitions, so there is no need to model them precisely with regard to delegation to the super constructor: the constructor function may initialize all fields (including inherited fields).

Here is a sketch to get you started:

```
Object = { }
A      = { ... }
ARep   = { ... }

classA = λrep: ARep.
             λthis: ...
newA   = λ_a: ...

...
```

(2) Encode the expression `new B(v1, v2).m()` using the encoding you developed previously, in a context where `v1` and `v2` are two values of type `Object` and write only the first *four* steps of the evaluation.