Final Exam - Solution

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 wish to augment the language with reference cells and propose the following subtyping rule:

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

(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.

Solution:
First example:

```
(λr: Ref {a: Nat, b: Nat}. !r.b) ref {a: Nat = zero}

⟶ [ref1 ↦ {a: Nat = zero}]
    (λr: Ref {a: Nat, b: Nat}. !r.b) ref1
⟶ [ref1 ↦ {a: Nat = zero}]
    !ref1.b
⟶ [ref1 ↦ {a: Nat = zero}]
    {a: Nat = zero}.b
```

And the subtyping derivation is:

$$\text{(S-Ref)}\ \frac{\text{(S-RcdWidth)}\ \dfrac{}{\{a : Nat, b : Nat\} <: \{a : Nat\}}}{Ref\ \{a : Nat\} <: Ref\{a : Nat, b : Nat\}}$$

Second example:

```
let r = ref {a: Nat = zero, b: Nat = zero}
in
  ((λu: Ref {a: Nat}. u := {a: Nat = zero}) r); !r.b

⟶ [ref1 ↦ {a: Nat = zero, b: Nat = zero}]
    ((λu: Ref {a: Nat}. u := {a: Nat = zero}) ref1); !ref1.b
⟶ [ref1 ↦ {a: Nat = zero, b: Nat = zero}]
    ref1 := {a: Nat = zero}; !ref1.b
⟶ [ref1 ↦ {a: Nat = zero}]
    !ref1.b
⟶ [ref1 ↦ {a: Nat = zero}]
    {a: Nat = zero}.b
```

And the subtyping derivation is:

$$\text{(S-Ref)}\ \frac{\text{(S-RcdWidth)}\ \dfrac{}{\{a : Nat, b : Nat\} <: \{a : Nat\}}}{Ref\ \{a : Nat, b : Nat\} <: Ref\{a : Nat\}}$$

Exercise 2 : Type Reconstruction (10 points)

Given the following lambda term t in the language used in Chapter 22 (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.

Solution:

(1) $(\tau, t) = ([(Nat \to Bool \to Bool) \to (Nat \to Bool) \to Nat, ..], t)$

(2) $C = \{X = Z \to A, Y = Z \to B, A = B \to C\}$

(3) $(\tau, t) = ([(A \to B \to C) \to (A \to B) \to A], t)$

Exercise 3 : Object encodings (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 encodings 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).

(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`. Write the first *four* steps of the evaluation.

Solution:

```
Object = { }
A      = { m: Unit → Object, n: Unit → Object }
ARep   = { a: Object }

classA = λrep: ARep.
           λthis: Unit → A.
             λ_: Unit.
                { m = λ_: Unit.(this unit).n unit,
                  n = λ_: Unit.rep.a }
newA   = λ_a: Object.
           let r = { a = _a }
           in
             fix (classA r) unit
```

```
  B       = { m: Unit → Object , n: Unit → Object }
  BRep    = { a: Object , b: Object }

  classB = λrep: BRep.
             λthis: Unit → B.
               λ_: Unit.
                 let super = classA rep this unit
                 in
                   { m = super.m,
                     n = λ_: Unit.rep.b }
  newB = λ_a: Object.λ_b: Object.
           let r = { a = _a, b = _b }
           in
             fix (classB r) unit

    (newB v1 v2 unit).m unit
⟶ ((let r = {a = v1, b = v2}
      in
        fix (classB r) unit) unit).m unit
⟶ (fix (classB {a = v1, b = v2}) unit).m unit
⟶ ((fix λthis: Unit → B.
           λ_: Unit.
             let super = classA {a = v1, b = v2} this unit
             in
               { m = super.m,
                 n = λ_: Unit.{a = v1, b = v2}.b }) unit).m unit
⟶ ((λ_: Unit.
       let super =
         classA {a = v1, b = v2}
           (fix λthis: Unit → B.
                  λ_: Unit.
                    let super = classA {a = v1, b = v2} this unit
                    in { m = super.m, n = λ_:Unit.{a = v1, b = v2}.b}) unit
       in
         { m = super.m,
           n = λ_: Unit.{a = v1, b = v2}.b }) unit).m unit

// and now it becomes really hairy

⟶ (let super = classA {a = v1, b = v2}
                 (λ_: Unit.
                    let super = classA {a = v1, b = v2} (fix f) unit
                    in { m = super.m, n = λ_:Unit.{a = v1, b = v2}.b}) unit
     in { m = super.m,
          n = λ_: Unit.{a = v1, b = v2}.b }).m unit
⟶ (let super = (λthis: Unit → A.
                  λ_: Unit.
                    { m = λ_: Unit.(this unit).n unit ,
```

```
                        n = λ_: Unit.{a = v1, b = v2}.a }
                ) (λ_: Unit.
                    let super = classA {a = v1, b = v2} (fix f) unit
                    in { m = super.m n = λ_: Unit.{a = v1, b = v2}.b}) unit
    in { m = super.m,
         n = λ_: Unit.rep.b }).m unit
⟶  (λ_: Unit.((
      (λ_: Unit.
         let super = classA {a = v1, b = v2} (fix f) unit
         in { m = super.m n = λ_: Unit.{a = v1, b = v2}.b}) unit) unit).n) unit
⟶ (((λ_: Unit.
       let super = classA {a = v1, b = v2} (fix f) unit
       in { m = super.m n = λ_: Unit.{a = v1, b = v2}.b}) unit) unit).n) unit
⟶ let super = classA {a = v1, b = v2} (fix f) unit
   in { m = super.m n = λ_: Unit.{a = v1, b = v2}.b}.n) unit
⟶ λ_: Unit.{a = v1, b = v2}.b unit
⟶ {a = v1, b = v2}.b
⟶ v2
```