CIS 500 Software Foundations
Homework Assignment 10

Object Encodings and Featherweight Java

Due: Wednesday, December 8 2004, by noon

Submission instructions: Same as last time.

1 Exercise Show how to encode the following Java code into the lambda calculus with subtyping, records, references, and fixed points, in the style of TAPL chapter 18.

```
class E {
  protected int x = 1;
  int m() { x = x+1; return x; }
  int n() { x = x-1; return this.m(); }
}

class G extends E {
   int m() { x = x+2; return super.m(); }
}

G g = new G();
g.n();
```

You can test your solution using the lambda calculus interpreter installed on `eniac-l.seas.upenn.edu`. The interpreter may be run with: `/usr/local/tapl/fullsub/f filename.f` where `filename.f` is a file containing lambda calculus expressions. The output is the value of evaluating those expressions (if any). The syntax can been seen by looking at the examples in the file `test.f` in the same directory.

2 Exercise TAPL 18.11.1

3 Exercise Consider the five following class definitions, in the style of TAPL Chapter 18:

```
R = {};
O = {m:Unit->Unit, n:Unit->Unit, o:Unit->Unit};

classA = lambda r:R. lambda self: Unit->O. lambda _:Unit.
           {m = lambda _:Unit. (self unit).n unit,
            n = lambda _:Unit. (self unit).m unit,
            o = lambda _:Unit. unit };

classB = lambda r:R. lambda self: Unit->O. lambda _:Unit.
           let super = classA r self unit in
           {m = lambda _:Unit. unit,
            n = super.n,
            o = super.o };

classC = lambda r:R. lambda self: Unit->O. lambda _:Unit.
           let super = classA r self unit in
           {m = super.m,
            n = lambda _:Unit. super.n unit,
            o = super.o };
```

```
classD = lambda r:R. lambda self: Unit->O. lambda _:Unit.
           let super = classA r self unit in
           {m = super.m,
            n = lambda _:Unit. super.o unit,
            o = super.o };

classE = lambda r:R. lambda self: Unit->O. lambda _:Unit.
           let super = classA r self unit in
           {m = (self unit).m,
            n = super.n,
            o = super.o };
```

Each of these classes produces objects with the same type, `O`, which offers three methods. Their state representations are all the trivial record type, `R`. The methods `m`, `n`, and `o` all take `Unit` arguments and return`Unit` results—that is, the only interesting thing about them is whether they diverge or converge. The class`classA` is the "root class" from which the other four are derived.

Given these definitions, does evaluation of the following expressions converge (yielding `unit`) or diverge?

```
((fix (classA {})) unit).n unit

((fix (classA {})) unit).o unit

((fix (classB {})) unit).n unit

((fix (classB {})) unit).o unit

((fix (classC {})) unit).n unit

((fix (classC {})) unit).o unit

((fix (classD {})) unit).n unit

((fix (classD {})) unit).o unit

((fix (classE {})) unit).n unit

((fix (classE {})) unit).o unit
```

4 Exercise Declarative typing for FJ

1. FJ, like full Java, presents the typing relation in algorithmic form. There is no subsumption rule; instead several other rules include subtyping checks among their premises. Reformulate the typing rules for FJ as a declarative system, dropping these premises in favor of a subsumption rule.

   Your declarative system should not to be exactly the same as FJ with respect to casts. In your declarative system, replace the three rules for cast with the following rule:

$$\frac{\Gamma \vdash t_0 : D}{\Gamma \vdash (C)t_0 : C} \qquad \text{(T-C\textsc{ast})}$$

2. Prove the following preservation theorem for your declarative presentation of Featherweight Java, by induction on $\Gamma \vdash t : C$.

   If $\Gamma \vdash t : C$ and $t \longrightarrow t'$ then $\Gamma \vdash t' : C$.

   You may assume any lemmas (such as inversion,substitution, field and method typing) that you need.

5 Exercise Suppose we state the operational semantics of FJ with error using the evaluation contexts defined in 19.5.3. As in homework 8, we specify small-step evaluation by the following two rules:

$$\frac{t \longrightarrow_\beta t'}{E[t] \longrightarrow E[t']} \qquad \text{(E-Ctx)}$$

$$E[error] \longrightarrow error \qquad \text{(E-Error)}$$

The computation rules ($t \longrightarrow_\beta t'$) are the FJ rules E-ProjNew, E-InvkNew, E-CastNew, plus a new one called E-CastFail. This last rule is closer to Java by making a failed cast step to error.

$$\frac{C \not<: D}{(D)(new\ C(v)) \longrightarrow_\beta error} \qquad \text{(E-CastFail)}$$

Prove the progress lemma for this system: Suppose t is a closed, well-typed normal form. Then either t is a value or t is *error*.

6 Debriefing

1. How many hours (per person) did you spend on this assignment?
2. Would you rate it as easy, moderate, or difficult?
3. Did everyone in your study group participate?
4. How deeply do you feel you understand the material it covers (0%–100%)?

If you have any other comments, we would like to hear them; please send them to `cis500@cis.upenn.edu`.