CIS 500 — Software Foundations

Homework Assignment 10

Objects in the lambda-calculus

Due: Wednesday, November 29, 2006, by noon

Instructions: Use the `fullref` typechecker implementation to check your solutions to the exercises below. On Eniac, typing

```
~cis500/bin/fullref  <filename>
```

will run the `fullref` checker on your file and print the types and final values of the expressions in it.

Submit your solutions either in PDF or as one big ascii file.

1 Exercise TAPL exercise 18.6.1.

2 Exercise TAPL exercise 18.7.1.

3 Exercise This exercise and the two that follow consider an implementation of a class representing a set of integers, in the style of TAPL Section 18.11. With this encoding, the type of a set of integers is:

```
Set = {
  contains : Nat -> Bool,
  add : Nat -> Unit,
  add2 : Nat -> Nat -> Unit,
  remove : Nat -> Unit
}
```

This class contains a method for checking whether or not an element is in the set (`contains`) and methods for adding and removing elements (`add` and `remove`). The method `add2` simply adds both of its arguments to the set, naturally implemented by calling the `add` method twice. The type for the internal representation of sets will be:

```
SetRep = {x : Ref (Nat -> Bool)}
```

Following TAPL 18.11, we can implement the set class as below:

```
setClass =
  λr : SetRep.
    λthis : Unit -> Set.
      λ_ : Unit.
        { contains = λn:Nat. (!(r.x)) n,
          add = λn:Nat.
            let old = (!(r.x)) in
            r.x := λm:Nat. if m = n then true else old m,
          add2 =
            λm:Nat. λn:Nat.
            (this unit).add m;
            (this unit).add n,
          remove = λn:Nat.
            let old = (!(r.x)) in
            r.x := λm:Nat. if m = n then false else old m,
        }
```

Finally, a function to construct a new set might look like:

```
newEmptySet =
  λ_ : Unit. fix (setClass {x = ref (λn:Nat. false)}) unit
```

4 Exercise Now consider a subclass of `Set` that has a method returning the size of the set. That is, we would like an object of type:

```
SizeSet = {
  contains : Nat -> Bool,
  add : Nat -> Unit,
  add2 : Nat -> Nat -> Unit,
  remove : Nat -> Unit,
  size : Unit -> Nat
}
```

- Define a type `SizeSetRep` for the internal representation used by this class. (It should be a subtype of `SetRep`, of course.)
- Using this representation, write an implementation of a `sizeSetClass` as a subclass of the `setClass`. In particular, you must define the behavior of the method `add2` with inheritance.
- Write a function to construct a new, empty `SizeSet`.

5 Exercise

Consider another definition for the constructor of an empty set:

```
newEmptySet' = fix (setClass {x = ref (λn:Nat. false)})
```

- Does `newEmptySet'` have the same type as `newEmptySet`?
- Does `newEmptySet'` have the same behavior as `newEmptySet` (think about the issue of divergence)?

6 Exercise Consider a subclass of `Set` that adds a union operation:

```
UnionSet = {
  contains : Nat -> Bool,
  add : Nat -> Unit,
  add2 : Nat -> Nat -> Unit,
  remove : Nat -> Unit,
  union : Set -> Unit
}
```

The `union` operation will update the object by adding all of the elements in the set given as an argument. Are there any problems in implementing an object of this type (with the desired behavior for `union`)? You may assume that the internal representation could be entirely changed and that the language contains other data structures such as list and trees. Are there any problems implementing such an object in Java?

7 Exercise [Required for PhD groups; optional for others.] TAPL exercise 18.6.2.

8 Debriefing

1. Approximately how many hours (per person, on average) did you spend on this assignment?
2. Would you rate it as easy, moderate, or difficult?
3. How deeply do you feel you understand the material it covers (0%–100%)?
4. Any other comments?