Types & Programming Languages
Exercises 4

These exercises are based on the material in Lectures 6, 7, 8 and 9.

1. The typing rule for the equality test operator is

$$\frac{\Gamma \triangleright e : \mathsf{int} \quad \Gamma \triangleright f : \mathsf{int}}{\Gamma \triangleright e{=}{=}f : \mathsf{bool}}$$

   It also makes sense to allow equality tests on boolean values, and we might try the general typing rule

$$\frac{\Gamma \triangleright e : T \quad \Gamma \triangleright f : T}{\Gamma \triangleright e{=}{=}f : \mathsf{bool}}$$

   What problems are likely to be caused by this general form of equality test, if we have function types in the language? (Hint: think about how you would define reduction rules for the general equality test.)

2. Work out the type of each of the following expressions, in the simply typed lambda calculus combined with SEL. (Using the notation `fn x:T => e` for $\lambda x : T.e$).

   (a) `fn x:int => x+1`

   (b) `fn x:int => (fn y:bool => (x == 1) & y)`

   (c) `fn x:int->int => x(x(2))`

   (d) `fn x:int->int => (fn y:int => xy)`

3. Assuming that we have record types, what is the type of this expression?

   `{ a = 1+2, b = 1==2, c = { x = 3 } }`

   Show how this expression reduces to a value.

4. Assume that we have sum types or variant types. A very useful idea is the *option type.* Given a type `T` we construct the type

   `option T = < none:unit, some:T >.`

   An expression of type `option T` is either the value `none()` or an expression `some(e)` where `e` is an expression of type `T`.

   For example, `option int` has values `none()`, `some(1)`, `some(2)` etc. and we could also construct expressions such as `some(1+2)` and so on.

   Option types can be used to represent the results of computations which might return an error condition instead of a useful value: for example, a square root function might be given the type

   `sqrt :  float -> option float`

   to allow for the error case when the argument is negative.

   (a) Give an example of a `case` expression which would be used to analyse an expression of type `option int`, and show how it reduces.

   (b) Write down the typing rule for `case` expressions associated with the type `option int`.

5. We have seen how reference types allow us to describe the assignable variables of an imperative language: for example, the Java code

```
{ int x = 1;
  x = x + 1;
}
```

corresponds to

```
let val x = ref 1
in  x := (!x) + 1
end
```

When we use objects in Java there is slightly more going on. A variable of some object type `MyClass` stores either `null` or a reference to an object of type `MyClass`. So a Java variable `MyClass x` corresponds to a value of type `option ref MyClass`, with the value `none()` representing `null`.

Assume that we have a Java function

`MyClass f(int x)`.

A typical use of `f` might look like this:

```
MyClass x = f(1);
if (x == null)
   ... code 1 ...
else
   ... code 2 ...
```

Show how this style of coding would appear in our language.