Week 7 Recitation Solutions

See 8 Semantics for function rules

1. (*LetRec*) Write the abstract syntax for the following lettuce program and again for the same program if we used `let` instead of `let rec`:

```
let rec f = function(x) if x == 0 then 0 else f(x - 1) in
  f(2)
```

Solution:

AST for original code is:

```
LetRec(
  "f",
  "x",
  IfThenElse(
    Eq(Ident("x"), Num(Zero)),
    Num(Zero),
    FuncCall(Ident("f"), Minus(Ident("x"), Num(one)))
  ),
  FuncCall(Ident("f"), Num(two))
)
```

AST if we used `let` instead of `let rec` (note that the code would error when run because `f` is not available in the function body, thus the need for `let rec`):

```
Let(
  "f",
  FunDef(
    "x",
    IfThenElse(
      Eq(Ident("x"), Num(Zero)),
      Num(Zero),
      FuncCall(Ident("f"), Minus(Ident("x"), Num(one)))
    )
  ),
  FuncCall(Ident("f"), Num(two))
)
```

2. (*Closures*) What does the following program evaluate to under the current rules?

```
let y = 3 in
  let f = function(x) x + y in
    let y = 4 in
      f(2)
```

Solution:

`5`. The closed environment (stored in the closure) remembers what the value of each variable was when the function was *defined*, not when it was used.

3. (*Closures*) You might be wondering why we're going to all this trouble with Closures, why we don't just leave out the environment capturing all together. To see the reason, what does the example from 2 evaluate to under this updated (in blue) rule which ignores the closed environment?

$$\frac{\sigma \vdash e_f \Downarrow \texttt{Closure}(p, e_b, \pi) \quad \sigma \vdash e_a \Downarrow v_a \quad \sigma[p \mapsto e_a] \vdash e_b \Downarrow v \quad v_{\not} = \texttt{Error}}{\sigma \vdash \texttt{FunCall}(e_f, e_a) \Downarrow v} \; (\Downarrow\text{-FunCall-ok})$$

Solution:

6. We lose the stored variable since it was overwritten, and if this happens in a real world program with hundreds or thousands of lines between the definition of the function and the overwrite of the variable, it could cause lots of headaches.

4. (*Functions and derivations*) Write out the derivation for the following program:

```
(function(x) x)(3)
```

Solution:

The above construct is sometimes called an IIFE (Immediately Invoked Function Expression). The abstract syntax for the expression is as follows:

```
FunCall(FunDef("x", Ident("x")), Num(three)))
```

And the derivation is shown bellow (error checks are omitted)

$$\frac{\{\} \vdash \texttt{FunDef("x",Ident("x"))} \Downarrow \texttt{Closure("x"),Ident("x"),\{\})} \quad \{\} \vdash \texttt{Num(3)} \Downarrow \texttt{NumVal(3)} \quad \{\texttt{"x"} \mapsto \texttt{NumVal(3)}\} \vdash \texttt{Ident("x")} \Downarrow \texttt{NumVal(3)}}{\{\} \vdash \texttt{FunCall}(\texttt{FunDef}(\texttt{"x"}, \texttt{Ident}(\texttt{"x"})), \texttt{Num}(3)) \Downarrow \texttt{NumVal}(3)}$$