04/06/18

Liav Rabi, id: 205688328
Benny Peresetsky, id: 310727771

PPL – Assignment 4

Question 1

a. {T1 = [T3->T4], T5=[T3->T4], T2=Number}
b. There is no MGU because than T1=Number and T1=Symbol also.
c. {T1=T2}
d. {}

Question 2

The reason we can type check letrec expressions without specific problems related to recursion or recursive environment is because the type checker sees only program text and does not runs over actual data(evaluating) as the interpreter does. Therefore, the type check runs over the expression one time where it binds identifiers to types and compress sets of values into types. All the operations which mentioned above are the reason why there is no need for a recursion or anything related (all the expressions are already type checked in the first time, therefore there is no need to run over the expressions more than once).

Question 3

```
let program = A.parse("(let ((a #f))(let ((b a))(let ((c b)) (let ((d c)) d))))");
let type = I.typeofExp(program, E.makeEmptyTEnv());
assert.equal(T.isTVar(type), true);
console.log(type);
let t1 = T.isTVar(type) ? unbox(type.contents) : undefined;
assert.equal(T.isTVar(t1), true);
console.log(t1);
let t2 = T.isTVar(t1) ? unbox(t1.contents) : undefined;
assert.equal(T.isTVar(t2), true);
console.log(t2);
let t3 = T.isTVar(t2) ? unbox(t2.contents) : undefined;
assert.equal(T.isTVar(t3), true);
console.log(t3);
let t4 = T.isTVar(t3) ? unbox(t3.contents) : undefined;
assert.equal(T.isBoolTExp(t4), true);
console.log(t4);
```

```
{ tag: 'TVar',
  var: 'T_4',
  contents: [ { tag: 'TVar', var: 'T_3', contents: [Array] } ] }
{ tag: 'TVar',
  var: 'T_3',
  contents: [ { tag: 'TVar', var: 'T_2', contents: [Array] } ] }
{ tag: 'TVar',
  var: 'T_2',
  contents: [ { tag: 'TVar', var: 'T_1', contents: [Array] } ] }
{ tag: 'TVar', var: 'T_1', contents: [ { tag: 'BoolTExp' } ] }
{ tag: 'BoolTExp' }
```