Problem 1  [10 points] Mitchell Exercise 9.5 pg 276

Problem 2  [15 points] Here is a fragment of MinML with unit, sums, functions, universal, and existential types. Write out a complete set of typing rules for this subset. Note that we provide first-class polymorphism for this subset.

\[\tau ::= \text{unit} \mid \tau_1 \to \tau_2 \mid (\tau_1 + \tau_2) \mid \forall t (\tau) \mid \exists t (\tau) \mid t\]

\[e ::= () \mid (\text{fn}(x : \tau) \Rightarrow e) \mid \text{apply}(e_1, e_2) \mid \text{inl}(\tau_1 + \tau_2)(e) \mid \text{inr}(\tau_1 + \tau_2)(e) \mid (\text{case } e \text{ of } \text{inl}(x_1: \tau_1) \Rightarrow e_1 | \text{inr}(x_2: \tau_2) \Rightarrow e_2) \mid (\text{Fun } t \text{ in } e) \mid \text{inst}(e, \tau) \mid (\text{pack } \tau_1 \text{ with } e \text{ as } \exists t (\tau_b)) \mid (\text{open } e_1 \text{ as } t \text{ with } x : \tau \text{ in } e_2)\]

(a)  [5 points] Define a relation \(\Delta \vdash \tau \text{ ok}\) similar to the one described by Harper on page 142 (rules 20.1 - 20.5).

(b)  [10 points] Define a relation \(\Gamma \vdash_{\Delta} e : \tau\) similar to the one described by Harper on page 142 (rules 20.7 - 20.8). Please write out all the rules for the language described above. Pay particular attention to the rules for \text{fn}, \text{inl}, \text{inr}, \text{and case}. See Problem 3 for more details.

Problem 3  [20 points] Harper defines the following judgment \(\Delta \vdash \Gamma \text{ ok}\) as follows

\[\frac{\Delta \vdash \Gamma(x) \text{ ok}}{(\forall x \in \text{dom}(\Gamma)) \Delta \vdash \Gamma \text{ ok}}\]

Typically when dealing with typing judgments that keep track of free type-variables we have the following Lemma.

Lemma  If \(\Delta \vdash \Gamma \text{ ok}\) and \(\Gamma \vdash_{\Delta} e : \tau\) then \(\Delta \vdash \tau \text{ ok}\)

Proving this theorem is done via induction over the typing rules. Prove this theorem using the definitions you wrote for Problem 2. You may need to add extra conditions to your typing relation to make your proof go through. If you do this be sure to update your answer to Problem 2(b).

Problem 4  [15 points] The following source program

```plaintext
type point = (real * real)
type shape = point -> bool
fun half_plane(p1:point,p2:point) = let
```


val k1 = (x1-x0)
val k2 = (y1-y0)
val k3 = k1*y0 - k2*x0
in (fn (x2,y2) => (k1*y2) >= ((k2*x2) + k3):shape end

fun intersect(f1:shape,f2:shape) = ((fn p => (f1 p) andalso (f2 p)):shape)

is closure converted into the following program

type point = (real * real)
type shape = exists 'env in ('env * (('env * point) -> bool))

fun half_plane(p1,p2) = let
val k1 = (x1-x0)
val k2 = (y1-y0)
val k3 = k1*y0 - k2*x0
val env = (k1,k2,k3)
in pack in pack ??1?? with
(env,fn ((env,(x2,y2)) => let
val (k1,k2,k3) = env
in (k1*y2) >= ((k2*x2) + k3) end)
as exists 'env in ('env * (('env * point) -> bool)) end

fun intersect(f1:shape,f2:shape) = let
val env = (f1,f2)
in pack ??2?? with
(env,(fn (env,p) => let
val (f1:shape,f2:shape) = env
in open f1 as 'env1 with clos1:??3?? in
open f2 as 'env2 with clos2:??4?? in
let val (env1,f1') = clos1
val (env2,f2') = clos2
in (f1'(env1,p)) andalso (f2'(env2,p)) end)
as exists 'env in ('env * (('env * point) -> bool)) end

For all four ??n?? write down the appropriate types so that the program will type check.

**Problem 5** [30 points] Complete the implementation of static-sem.sml. **Important Note:** The provided code does not include important checks related to the $\Delta \vdash \tau \ ok$ relation. In addition to extending the code to handle the new cases, you must modify the existing code that guarantees types are well formed. In particular the current implementation does not enforce the rules need to guarantee the lemma stated in problem 3. Fix the code so it does, and extend it to handle the new cases.

**Problem 6** [10 points] Complete the implementation of m-machine.sml.