Solutions for Sample Questions for Midterm 2 (CS 421 Spring 2024)

Some of these questions may be reused for the exam.

1. Put the following function in full continuation passing style:
   let rec sum_odd n = if n <= 0 then 0 else ((2 * n) – 1) + sum_odd (n – 1);;
   Use **addk**, **subk**, **mulk, leqk**, for the CPS forms of the primitive operations (+, - , *, <=). All other procedure calls and constructs must be put in CPS.

Solution:

```
let add_k a b k = k(a + b)
let minus_k a b k = k(a - b)
let times_k a b k = k(a * b)
let leq_k a b k = k(a <= b)


let rec sum_odd_k n k =
    leq_k n 0 (fun b -> if b then k 0
                 else minus_k n 1
                 (fun d -> sum_odd_k d
                  (fun r -> times_k 2 n
                   (fun t -> minus_k t 1
                    (fun m -> add_k m r k )))))
```

2. Given the following OCAML datatype:
   type int_seq = Null | Snoc of (int_seq * int)
   write a tail-recursive function in OCAML **all_pos : int_seq -> bool** that returns **true** if every integer in the input **int_seq** to which **all_pos** is applied is strictly greater than 0 and **false** otherwise. Thus **all_pos (Snoc(Snoc(Snoc(Null, 3), 5), 7))** should returns **true,** but **all_pos (Snoc(Null, -1))** and **all_pos (Snoc(Snoc(Null, 3),0))** should both return **false.**

Solution:

```
let rec all_pos s =
  (match s with Null -> true
    | Snoc(seq, x) -> if x <= 0 then false else all_pos seq);;
```

3. Write the definition of an OCAML variant type **reg_exp** to express abstract syntax trees for regular expressions over a base character set of booleans. Thus, a boolean is a **reg_exp,** epsilon is a **reg_exp**, a parenthesized **reg_exp** is a **reg_exp**, the concatenation of two **reg_exp**'s is a **reg_exp,** the "choice" of two **reg_exp**'s is a **reg_exp**, and the Kleene star of a **reg_exp** is a **reg_exp**.

Solution:

```
type reg_exp =
  Char of bool
 | Epsilon
```

| Paren of reg_exp
| Concat of (reg_exp * reg_exp)
| Choice of (reg_exp * reg_exp)
| Kleene_star of reg_exp

4. Given the following rules for CPS transformation:
   [[x]] K => K x
   [[c]] K => K c
   [[let x = e1 in e2]] K=> [[e1]] (FN x -> [[e2]] K)
   [[e1 ⊕ e2]] K=> [[e2]] (FN a -> [[e1]] (FN b -> K (b ⊕ a)))
   where e1and e2 are OCaml expressions, K is any continuation, x is a variable and c is a constant, give the step-by-step transformation of

   [[let x = 2 + 3 in x – 4]] REPORTk

   Solution:
   [[let x = 2 + 3 in x – 4]] REPORTk =>
   [[2 + 3]] (FN x -> [[x – 4]] REPORTk) =>
   [[2 + 3]] (FN x -> [[4]] (FN n -> [[x]] (FN m -> REPORTk (m – n)))) =>
   [[2 + 3]] (FN x -> [[4]] (FN n -> (FN m -> REPORTk (m – n)) x)) =>
   [[2 + 3]] (FN x -> (FN n -> (FN m -> REPORTk (m – n)) x) 4) =>
   [[3]] (FN u -> [[2]] (FN v -> (FN x -> (FN n -> (FN m -> REPORTk (m – n)) x) 4) (v + u))) =>
   [[3]] (FN u -> (FN v -> (FN x -> (FN n -> (FN m -> REPORTk (m – n)) x) 4) (v + u)) 2) =>
   (FN u -> (FN v -> (FN x -> (FN n -> (FN m -> REPORTk (m – n)) x) 4) (v + u)) 2) 3

5. Review and be able to write any give clause of **cps_exp** from MP5. On the exam, you would be given all the information you were given in MP5.

Solution:

```
(* Problem 5 *)
let rec cps_exp e k =
  match e with
(*[[x]]k = k x*)
    VarExp x -> VarCPS (k, x)
(*[[c]]k = k x*)
  | ConstExp n -> ConstCPS (k, n)
(*[[~ e]]k = [[e]]_(fun r -> k (~ r)) *)
  | MonOpAppExp (m, e) ->
     let r = freshFor (freeVarsInContCPS k)
     in cps_exp e (FnContCPS (r, MonOpAppCPS (k, m, r)))
(*[[(e1 + e2)]]k = [[e2]]_ fun s -> [[e1]] _ fun r -> k (r + s)*)
  | BinOpAppExp (b, e1, e2) ->
     let v2 = freshFor (freeVarsInContCPS k @ freeVarsInExp e1)  in
     let v1 = freshFor (v2 :: (freeVarsInContCPS k)) in
     let e2CPS =
        cps_exp e1 (FnContCPS (v1, BinOpAppCPS(k, b, v1, v2))) in
```

```
      cps_exp e2 (FnContCPS (v2, e2CPS))
(*[[if e1 then e2 else e3]]k = [[e1]]_(fun r -> if r then [[e2]]k else [[e3]]k)*)
   | IfExp (e1,e2,e3) ->
      let r = freshFor (freeVarsInContCPS k @
                  freeVarsInExp e2 @ freeVarsInExp e3) in
      let e2cps = cps_exp e2 k in
      let e3cps = cps_exp e3 k in
      cps_exp e1 (FnContCPS(r, IfCPS(r, e2cps, e3cps)))
(*[[e1 e2]]k = [[e2]]_fun v2 -> [[e1]]_fun v1 -> k (v1 v2)*)
   | AppExp (e1,e2) ->
      let v2 = freshFor (freeVarsInContCPS k @ freeVarsInExp e1) in
      let v1 = freshFor (v2 :: freeVarsInContCPS k) in
      let e1cps = cps_exp e1 (FnContCPS (v1, AppCPS(k, v1, v2))) in
      cps_exp e2 (FnContCPS (v2, e1cps))
(*[[fun x -> e]]k = k(fnk x kx -> [[e]]kx) *)
   | FunExp (x,e) ->
      let ecps = cps_exp e (ContVarCPS Kvar) in
      FunCPS (k, x, Kvar, ecps)
(*[[let x = e1 in e2]]k = [[e1]]_fun x -> [[e2]]k) *)
   | LetInExp (x,e1,e2) ->
     let e2cps = cps_exp e2 k in
     let fx = FnContCPS (x, e2cps) in
     cps_exp e1 fx
(*[[let rec f x = e1 in e2]]k = (FN f -> [[e2]]_k))(FIX f. FUN x -> fn kx => [[e1]]kx) *)
   | LetRecInExp(f,x,e1,e2) ->
     let e1cps = cps_exp e1 (ContVarCPS Kvar) in
     let e2cps = cps_exp e2 k in
     FixCPS(FnContCPS (f,e2cps),f,x,Kvar,e1cps)
```

6. Given a polymorphic type derivation for {} |- let id = fun x -> x in id id true : bool

Solution:

Let Γ = {id : ∀ ‘a. ‘a -> ‘a}

```
                              Instance: ‘a → bool -> bool
                          Var ----------------------------------        Instance: ‘a → bool
                                Γ |-  id : (bool -> bool) ->      Var---------------------------
                                         bool -> bool                 Γ |-  id : bool -> bool        Const
 Var ----------------------     App------------------------------------------------------------     -------------------
     { x : ‘a }  |-  x : ‘a                  Γ |-  id id : bool -> bool                              Γ |-  true : bool
Fun -----------------------      App--------------------------------------------------------------------------------
       { } |- fun x -> x                      {id : ∀ ‘a. ‘a -> ‘a} |-  id id true
  Let ----------------------------------------------------------------------------------------
                          { } |- let id = fun x -> x in id id true : bool
```