CS421 Summer 2010 Final

Name:	
NetID:	

- You have **180 minutes** to complete this exam.
- This is a **closed-book** exam. All other materials, besides pens, pencils and erasers, are to be away.
- Do not share anything with other students. Do not talk to other students. Do not look at another student's exam. Do not expose your exam to easy viewing by other students. Violation of any of these rules will count as cheating.
- Please write your name and NetID in the spaces above, and also at the top of every page.

CS 421 Final **Name:______________________________________**

Problems	**Possible Points**	**Points Earned**
1	22	
2	15	
3	19	
4	16	
5	22	
6	20	
7	15	
8	15	
9	8	
10	18	
11	15	
Total		

CS 421 Final **Name:**___________________________________

1. (22 pts total) Write an Ocaml function **sum_bigger : int -> int list -> int** that, when applied to an integer **m** and a list of integers **l**, returns the sum of all elements of **l** that are strictly greater than **m** or 0 if there aren't any, in each of the following ways:
   (a) (6 pts) Using forward recursion over lists as the only form of recursion

   (b) (8 pts) Using tail recursion as the only form of recursion

   (c) (8 pts) Using no explicit recursion, but using the function
   List.fold_left : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a

2. (15 pts) Consider the following Ocaml code:

```
let all_nonneg list =
let rec all_nn b l =
   match l with [] -> b
   | (x::xs) -> all_nn (b && x >= 0) xs
in all_nn true list;;
```

a. (3 pts) What is the type of **all_nonneg**?

b. (4 pts) What is the type of **all_nonneg_k**, the result of transforming **all_nonneg** into continuation passing style?

c. (8 pts) Write the Ocaml function **all_nonneg_k** that is the full continuation passing style version of the Ocaml function given in part a. You must translate each procedure call into CPS, but you may treat the operations >= and **&&** as primitive, and not translate them into a form taking a continuation. All other procedures must take a continuation as an argument.

CS 421 Final **Name:**___________________________________

3.(19 pts total) The following is an outline of a type derivation.

$$\text{let rec rule}\quad \frac{A \qquad\qquad\qquad\qquad B}{\{\ \textbf{f: int -> int -> int; z : int }\}\ \textbf{|- let rec f = fun x -> f x z in f 5 : }\underline{\textbf{?}}}$$

a) (3 pts) The value for ? is:

b) (6 pts) Give the type derivation (not type inference) that **B** represents (hint: it is smaller than the one for **A**). Here and in the next part, you may give names to your environments to save space and writing. You may also want to do **A** in part c on the next page first. Label all inferences with the rule used.

3. (cont.) The following is an outline of a type derivation.

$$\text{let rec rule}\ \frac{\quad A \qquad\qquad\qquad\qquad B \quad}{\{\ \textbf{f: int -> int -> int; z : int }\}\ \textbf{|- let rec f = fun x -> f x z in f 5 : }\underline{\textbf{?}}}$$

c) (10 pts) Give the type derivation that A represents. Label all inferences with the rule used.

4. (16 pts total) For each of the following languages (ie, sets of strings), write a regular expression generating the set, and draw a **deterministic** finite state automaton accepting the set:
   (a) (8 pts) The set of all strings of a's, b's, and c's such that every third character is c.

   (b) (8 pts) The set of all strings of 0's, and 1's, such that between any two consecutive 1's, there are at least two 0's.

5. (22 pts total) Consider the following grammar:

<E> ::= ! <E> | <E> && <E> | (<E>) | 0 | 2

(a) (4pts) Demonstrate that the above grammar is ambiguous.

(b) (10 pts) Disambiguate the above grammar. Your grammar should have **!** bind more tightly than **&&**, and **&&** associate to the right.

5. (cont.)

(c) (8 pts) Using the grammar **you gave** above in b, give a parse tree for:

0 && (! 2 && 2) && 0

6. (20 pts total) Consider the following grammar:

<S> ::= <A> ! | <A> ; <S>
<A> ::= 0 | 1 | % <A>

(a) (3pts)Write an Ocaml type representing the tokens you would need to parse this language.

(b) (7pts) Write a collection of Ocaml types representing parse trees for the given grammar.

CS 421 Final **Name:____________________________________**

6.(cont.) Consider the following grammar:

<S> ::= <A> ! | <A> ; <S>
<A> ::= 0 | 1 | % <A>

(c) (10 pts) Write a function **parse** that returns an <S> parse tree (as represented by your types given in part (b)) when applied to a list of tokens (as given in part (a)).

CS 421 Final **Name:____________________________________**

7. (15pts) In the last MP , you were asked to write a function **eval_exp : exp * memory -> value.** You were given the following types (abbreviated here) and functions:

type exp = VarExp of string | ConstExp of const | AppExp of exp * exp
| FunExp of string * exp | . . .

type memory = (string * value) list
and value = Intval of int | Boolval of bool | Closure of string * exp * memory | . . .

val make_mem : string -> value -> memory = <fun>
val lookup_mem : memory -> string -> value = <fun>
val ins_mem : memory -> string -> value -> memory = <fun>

Write the clause(s) for **eval_exp** to handle function application as given by the following rule:

$$\frac{(e_1, m) \downarrow \langle x \to e', m' \rangle \quad (e_2, m) \downarrow v' \quad (e', m' + \{x \to v'\}) \downarrow v}{(e_1\ e_2, m) \downarrow v}$$

CS 421 Final **Name:**___________________________________

8. (15 pts) Which of the following rules are natural semantics rules and which are transition semantics rules for **if** *b* **then** *c* **fi**: (We are using ~ here for both forms of evaluation relations)

i) $\dfrac{}{\text{(if true then } C \text{ fi, m)} \sim \text{(C, m)}}$

ii) $\dfrac{\text{(B,m)} \sim \text{false}}{\text{(if B then C fi, m)} \sim \text{m}}$

iii) $\dfrac{\text{(B,m)} \sim \text{(B',m)}}{\text{(if B then C fi, m)} \sim \text{(if B' then C fi, m)}}$

iv) $\dfrac{\text{(B,m)} \sim \text{true} \quad \text{(C,m)} \sim \text{m'}}{\text{(if B then C fi, m)} \sim \text{m'}}$

v) $\dfrac{}{\text{(if false then C fi, m)} \sim \text{m}}$

Transition Semantics: ______________________________

Natural Semantics: ______________________________

9. (8 pts) For each of the following terms, write YES if the term is αβ-equivalent to **(λx. λy. x y) y** and write NO otherwise:

a. **(λy. λx. x y) y** ____________________

b. **(λx. λy. x y) z** ____________________

c. **(λy. λx. y x) y** ____________________

d. **(λx. λz. x z) z** ____________________

e. **(λy. x y)** ____________________

f. **(λw. y w)** ____________________

g. **(λy. y y)** ____________________

h. **(λy. z y)** ____________________

CS 421 Final **Name:___________________________________**

10. (18 pts total) Showing all your work, including labeling reductions, evaluate the following:

$$(\lambda x.\ x\ (\lambda y.\ x))\ ((\lambda u.\ u)\ (\lambda w.\ w))$$

(a) (9 pts) using eager evaluation

(b) (9 pts) using lazy evaluation

CS 421 Final **Name:___________________________________**

11. (15 pts total) Given a datatype for disjoint sums as follows:

type ‘a option = Some ‘a | None

(a) (5pts) In the style of Church numerals and Church booleans, write the lambda term that represents the constructors **Some** and **None**

(b) (10pts) Write a lambda term that corresponds to the Ocaml function

let option_map f x = match x with Some y -> Some (f y) | None -> None

CS 421 Final **Name:___________________________________**

Rules for type derivations:

Constants:

$$\frac{}{\Gamma \mid\!\!- \; n : \text{int}} \quad \text{(assuming } n \text{ is an integer constant)}$$

$$\frac{}{\Gamma \mid\!\!- \text{true} : \text{bool}} \qquad \frac{}{\Gamma \mid\!\!- \text{false} : \text{bool}}$$

Variables:

$$\frac{}{\Gamma \mid\!\!- x : \sigma} \quad \text{if } \Gamma(x) = \sigma$$

Primitive operators ($\oplus \in \{ +, -, *, \ldots \}$):

$$\frac{\Gamma \mid\!\!- e_1 : \text{int} \quad \Gamma \mid\!\!- e_2 : \text{int}}{\Gamma \mid\!\!- e_1 \oplus e_2 : \text{int}}$$

Relations ($\sim \in \{ <, >, =, <=, >= \}$):

$$\frac{\Gamma \mid\!\!- e_1 : \text{int} \quad \Gamma \mid\!\!- e_2 : \text{int}}{\Gamma \mid\!\!- e_1 \sim e_2 : \text{bool}}$$

Connectives :

$$\frac{\Gamma \mid\!\!- e_1 : \text{bool} \quad \Gamma \mid\!\!- e_2 : \text{bool}}{\Gamma \mid\!\!- e_1 \,\&\&\, e_2 : \text{bool}} \qquad \frac{\Gamma \mid\!\!- e_1 : \text{bool} \quad \Gamma \mid\!\!- e_2 : \text{bool}}{\Gamma \mid\!\!- e_1 \,||\, e_2 : \text{bool}}$$

If_then_else rule:

$$\frac{\Gamma \mid\!\!- e_1 : \text{bool} \quad \Gamma \mid\!\!- e_2 : \tau \quad \Gamma \mid\!\!- e_3 : \tau}{\Gamma \mid\!\!- (\text{if } e_1 \text{ then } e_2 \text{ else } e_3) : \tau}$$

Application rule:

$$\frac{\Gamma \mid\!\!- e_1 : \tau_1 \rightarrow \tau_2 \quad \Gamma \mid\!\!- e_2 : \tau_1}{\Gamma \mid\!\!- (e_1\ e_2) : \tau_2}$$

fun rule:

$$\frac{[x : \tau_1] \cup \Gamma \mid\!\!- e : \tau_2}{\Gamma \mid\!\!- \text{fun } x \text{ -> } e : \tau_1 \rightarrow \tau_2}$$

let rule:

$$\frac{\Gamma \mid\!\!- e_1 : \tau_1 \quad [x : \tau_1] \cup \Gamma \mid\!\!- e_2 : \tau_2}{\Gamma \mid\!\!- (\text{let x} = e_1 \text{ in } e_2) : \tau_2}$$

let rec rule:

$$\frac{[x : \tau_1] \cup \Gamma \mid\!\!- e_1 : \tau_1 \quad [x : \tau_1] \cup \Gamma \mid\!\!- e_2 : \tau_2}{\Gamma \mid\!\!- (\text{let rec x} = e_1 \text{ in } e_2) : \tau_2}$$