

CS421 Fall 2005 Midterm

Thursday, October 13, 2005

Name:	
NetID:	

- You have **75 minutes** to complete this exam.
- This is a **closed-book** exam.. You are allowed one 3inch by 5 inch card of notes prepared by yourself. This card is **not to be shared**. 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.
- If you believe there is an error, or an ambiguous question, you may seek clarification from myself or one of the TAs. You must use a whisper, or write your question out. Speaking out aloud is not allowed.
- Including this cover sheet and rules at the end, there are 13 pages to the exam. Please verify that you have all 13 pages.
- Please write your name and NetID in the spaces above, and also at the top of every page.

Problems	Possible Points	Points Earned
1	6	
2	9	
3	16	
4	15	
5	12	
6	13	
7	10	
8	10	
9	9	
10	12	

1. (6 pts total) Suppose that the following code is input into OCaml:

```
let x = "Hello, ";;
let polite y = x^y;;
let x = "Hi, ";;
let z = polite "Mr. Jones";;
let x = 3.2
```

For each of the following circle the correct choice.

- a. (2 pts) **z** will have a value of

1) "Hi, Mr. Jones"

2) "Hello, Mr. Jones"

- b. (2 pts) The declaration **let x = 3.2;** will cause a type error.

1) true

2) false

- c. (2 pts) **x** will have an end value of

1) "Hi"

2) 3.2

2. (9 pts total)

- a. (3 pts) Write an OCaml function **f:int -> int -> int** that adds the square of its first argument to 2 times its second argument. Pay attention to the type given.

```
let f x y = (x * x) + (2 * y);;
```

2. (cont.) What is the result of each of the following applications:

- b. (2 pts) **f 7;**

$f\ 7 : \text{int} \rightarrow \text{int}$ is a function that add 49 to the double of its input

- c. (2 pts) **f 5 3;**

$f\ 5\ 3 = 31;$

- d. (2 pts) **f (4,2);;**

$f(4,2)$ would yield a type error: f needs its first argument as an int, not a pair of ints.

CS 421 Midterm

Name: _____

3. (16 pts total) Given the following OCAML datatype:

type int_seq = Null | Snoc of (int_seq * int)

a. (8 pts) Write a recursive function **fold** : (int -> 'a -> 'a) -> 'a -> int_seq -> 'a that folds a function over an **int_seq**, with a given value for the base case.

```
let rec fold f b s = match s with Null -> b | (Snoc (xs, x)) -> f x (fold f b xs)
```

b. (8 pts) Using the function **fold**, and no other recursion, write a function **prod**: int_seq -> int that returns the product of all the elements in a list. The product of **Null** is 1. You should have

prod (Snoc (Snoc (Snoc (Null, 2), 3), 5)) = 30

```
let prod = fold (fun n -> fun cum_prod -> n * cum_prod) 1;;
```

CS 421 Midterm

Name: _____

4. (15 pts total) Below I have given you an outline for the type derivation of the following expression:

let rec f = fun x -> x in f 7

Please complete the derivation by giving the results that go in the lettered blanks. Put your answers next to the corresponding letters below the type derivation outline

(#5)___ |- (#6)__: (#7)___ (#8)___ |- (#9)__: (#10)___ (#11)___ |- (#12)__: (#13)___

(#1)___ |- (fun x -> x):(#2)___ (#3)___ |- (f 7):(#4)___

{ } |- (let rec f = fun x -> x in f 7) :int

(#1) __ {f: int -> int} _____

(#2) __ int -> int _____

(#3) __ {f:int -> int} _____

(#4) __ int _____

(#5) __ {x: int, f:int -> int} _____

(#6) __ x _____

(#7) __ int _____

(#8) __ {f:int->int} _____

(#9) __ f _____

(#10) __ int -> int _____

(#11) __ {f:int->int} _____

(#12) __ 7 _____

(#13) __ int _____

CS 421 Midterm

Name: _____

5. (12 pts total) Give a (most general) unifier for the following unification problem. Capital letters (A,B,C,D) denote variables of unification. The lower-case letters (f, l, n, p) are constants or term constructors. (f and p have arity 2, s has arity 1, and n has arity 0 (is a constant).) Show your work by listing the operation performed in each step of the unification and the result of that step.

$$\{(f(A,B) = f(A, p(A,A))); (f(s(A), s(p(A,A))) = f(C,D)); (s(C) = s(s(n)))\}$$

$$\psi = \text{Unif}(\{(f(A,B) = f(A, p(A,A))); (f(s(A), s(p(A,A))) = f(C,D)); (s(C) = s(s(n)))\})$$

$$\text{Decompose: } \psi = \text{Unif}(\{(A=A), (B=p(A,A)), (f(s(A), s(p(A,A))) = f(C,D)); (s(C) = s(s(n)))\})$$

$$\text{Delete: } \psi = \text{Unif}(\{(B=p(A,A)), (f(s(A), s(p(A,A))) = f(C,D)); (s(C) = s(s(n)))\}) =$$

$$\text{Decompose: } \psi = \text{Unif}(\{(B=p(A,A)), (s(A) = C), (s(p(A,A)) = D); (s(C) = s(s(n)))\}) =$$

$$\text{Orient: } \psi = \text{Unif}(\{(B = p(A,A)), (C = s(A)), (s(p(A,A)) = D); (s(C) = s(s(n)))\})$$

$$\text{Orient: } \psi = \text{Unif}(\{(B = p(A,A)), (C = s(A)), (D = s(p(A,A))); (s(C) = s(s(n)))\})$$

$$\text{Eliminate: } \psi = \{B \mapsto \psi_1(p(A,A))\} \circ \psi_1$$

$$\text{where } \psi_1 = \text{Unif}(\{(C = s(A)), (D = s(p(A,A))); (s(C) = s(s(n)))\})$$

$$\text{Eliminate: } \psi_1 = \{C \mapsto \psi_2(s(A))\} \circ \psi_2$$

$$\text{where } \psi_2 = \text{Unif}(\{(D = s(p(A,A))); (s(s(A)) = s(s(n)))\})$$

$$\text{Decompose: } \psi_2 = \text{Unif}(\{(D = s(p(A,A))); (s(A) = s(n))\})$$

$$\text{Decompose: } \psi_2 = \text{Unif}(\{(D = s(p(A,A))); (A = n)\})$$

$$\text{Eliminate: } \psi_2 = \{D \mapsto \psi_3(s(p(A,A)))\} \circ \psi_3 \text{ where } \psi_3 = \text{Unif}(\{(A = n)\})$$

$$\text{Eliminate: } \psi_3 = \{A \mapsto n\}$$

$$\text{Thus, } \psi_2 = \{D \mapsto s(p(n,n))\} \circ \{A \mapsto n\}$$

$$\psi_1 = \{C \mapsto (s(n))\} \circ \{D \mapsto s(p(n,n))\} \circ \{A \mapsto n\}$$

$$\psi = \{B \mapsto (p(n,n))\} \circ \{C \mapsto (s(n))\} \circ \{D \mapsto s(p(n,n))\} \circ \{A \mapsto n\}$$

6. (13 points) Given the following lambda expression:

$$(\lambda x. \lambda y. x y x) (\lambda u. \lambda v. \lambda w. v u w) ((\lambda s. s) (\lambda t. t))$$

a. (4 pts) Evaluate this term as fully as possible using only Eager Evaluation:

$$\begin{aligned} & (\lambda x. \lambda y. x y x) (\lambda u. \lambda v. \lambda w. v u w) ((\lambda s. s) (\lambda t. t)) \text{-}\beta\text{->} \\ & (\lambda y. (\lambda u. \lambda v. \lambda w. v u w) y (\lambda u. \lambda v. \lambda w. v u w)) ((\lambda s. s) (\lambda t. t)) \text{-}\beta\text{->} \\ & (\lambda y. (\lambda u. \lambda v. \lambda w. v u w) y (\lambda u. \lambda v. \lambda w. v u w)) (\lambda t. t) \text{-}\beta\text{->} \\ & (\lambda u. \lambda v. \lambda w. v u w) (\lambda t. t) (\lambda u. \lambda v. \lambda w. v u w) \text{-}\beta\text{->} \\ & (\lambda v. \lambda w. v (\lambda t. t) w) (\lambda u. \lambda v. \lambda w. v u w) \text{-}\beta\text{->} \\ & (\lambda w. (\lambda u. \lambda v. \lambda w. v u w) (\lambda t. t) w) \end{aligned}$$

b. (4 pts) Evaluate this term as fully as possible using only Lazy Evaluation:

$$\begin{aligned} & (\lambda x. \lambda y. x y x) (\lambda u. \lambda v. \lambda w. v u w) ((\lambda s. s) (\lambda t. t)) \text{-}\beta\text{->} \\ & (\lambda y. (\lambda u. \lambda v. \lambda w. v u w) y (\lambda u. \lambda v. \lambda w. v u w)) ((\lambda s. s) (\lambda t. t)) \text{-}\beta\text{->} \\ & (\lambda u. \lambda v. \lambda w. v u w) ((\lambda s. s) (\lambda t. t)) (\lambda u. \lambda v. \lambda w. v u w) \text{-}\beta\text{->} \\ & (\lambda v. \lambda w. v ((\lambda s. s) (\lambda t. t)) w) (\lambda u. \lambda v. \lambda w. v u w) \text{-}\beta\text{->} \\ & (\lambda w. (\lambda u. \lambda v. \lambda w. v u w) ((\lambda s. s) (\lambda t. t)) w) \end{aligned}$$

CS 421 Midterm

Name: _____

(6 cont)

- c. (4 pts) Reduce this term to $\alpha\beta$ -normal form: (You may omit explicit mention or use of congruence closure)

(Copying over the computation for Eager Evaluation)

$$\begin{aligned}
 & (\lambda x. \lambda y. x y x) (\lambda u. \lambda v. \lambda w. v u w) ((\lambda s. s) (\lambda t. t)) \text{-}\beta\text{->} \\
 & (\lambda y. (\lambda u. \lambda v. \lambda w. v u w) y (\lambda u. \lambda v. \lambda w. v u w)) ((\lambda s. s) (\lambda t. t)) \text{-}\beta\text{->} \\
 & (\lambda y. (\lambda u. \lambda v. \lambda w. v u w) y (\lambda u. \lambda v. \lambda w. v u w)) (\lambda t. t) \text{-}\beta\text{->} \\
 & (\lambda u. \lambda v. \lambda w. v u w) (\lambda t. t) (\lambda u. \lambda v. \lambda w. v u w) \text{-}\beta\text{->} \\
 & (\lambda v. \lambda w. v (\lambda t. t) w) (\lambda u. \lambda v. \lambda w. v u w) \text{-}\beta\text{->} \\
 & (\lambda w. (\lambda u. \lambda v. \lambda w. v u w) (\lambda t. t) w) \text{-}\beta\text{->} \\
 & (\lambda w. (\lambda v. \lambda w. v (\lambda t. t) w) w) \text{-}\alpha\text{->} \\
 & (\lambda w. (\lambda v. \lambda z. v (\lambda t. t) z) w) \text{-}\beta\text{->} \\
 & \lambda w. \lambda z. w (\lambda t. t) z
 \end{aligned}$$

CS 421 Midterm

Name: _____

7. (10 pts total)

- a. (4 pts) In the style of Church numerals, write the lambda term that represents the constructor for pairs (comma). Your answer, if applied to a and b should return the representation of the pair (a,b).

$$\lambda a. \lambda b. \lambda x. x a b$$

- b. (6 pts) Write the lambda term the represents the function which takes a pair (a,b) and returns the pair (b,a),

$$\lambda p. p (\lambda a. \lambda b. \lambda x. x b a)$$

8. (10 pts total)

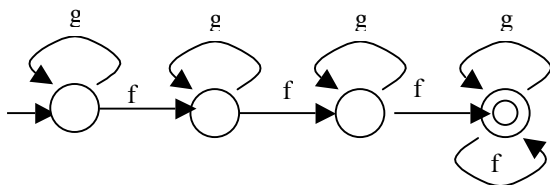
- a. (5 pts) Write a regular expression over the alphabet {a,b,c} generating exactly the set of all strings over {a,b,c} such that all the a's (if there are any) occur before any of the c's (if there are any).

$$(a \vee b)^* (b \vee c)^*$$

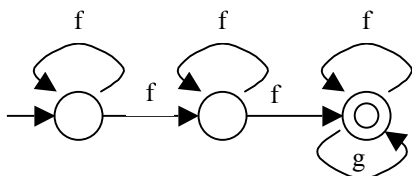
- b. (5 pts) Write a regular expression over the alphabet {0,1} generating exactly the set of all binary strings with an occurrence of 111 and a later occurrence of 000.

$$(0 \vee 1)^* 111 (0 \vee 1)^* 000 (0 \vee 1)^*$$

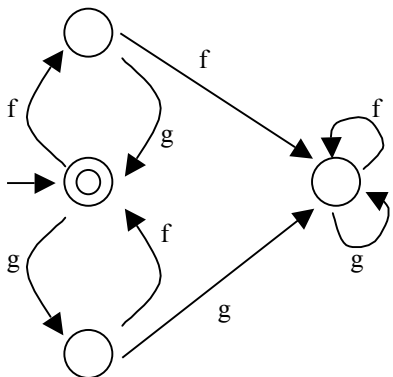
9. (9 pts) Below are three finite state automata over the alphabet {f,g}, each followed by a list of five strings. For each finite state automata, circle each string in the list that follows it that is accepted by the regular expression, and put an X through each one that is not.



- 1) gffgf 2) ~~ffggg~~ 3) fggfgggfg 4) fffggg 5) ~~ggff~~



- 1) gffgg 2) ~~ffgff~~ 3) fffffggg 4) fgfgfg 5) ~~fgf~~



- 1) gffggf 2) ~~fffggg~~ 3) fgfgfg 4) ~~ggffgg~~ 5) ~~gfgfgff~~

10. Extra Credit (12 pts total):

The following type may be used to represent terms of the lambda calculus as abstract syntax trees:

```
type lamb_exp =
  Var of string | App of (lamb_exp * lamb_exp) | Abs of (string*lamb_exp)
```

- a. (3pt) Write the function **free_vars lamb_exp -> string list** returning a list of all the names of variables having a free occurrence in a term.

```
let rec remove x l = match l with [ ] -> [ ]
  | y::ys -> if y = x then remove x ys else y::remove x ys
let rec free_vars tm =
  match tm with Var x -> [x]
  | App (tm1,tm2) -> free_vars tm1 @ free_vars tm2
  | Abs (x, body) -> remove x (free_vars body)
```

There are a number of other similar ways.

- b. (6pts) Write a function **subst_good : string -> string -> lamb_exp -> bool** that tells when it is permissible to replace free occurrences of the variable with the name of the second string by the variable with the name of the first string in the given expression. (Idea: we want know if $e[y/x]$ is valid.) You may find it useful to write and use an auxiliary function. You may always find it useful to use the previous function and **List.mem : 'a -> 'a list -> bool** for testing whether an element is in a list.

We need to know for every free occurrence of x , that if it is replaced by y it won't become bound.

```
let rec subst_good x y tm =
  match tm with Var _ -> true
  | App (tm1, tm2) -> subst_good x y tm1 && subst_good x y tm2
  | Abs (z, body) -> if z = x then true else
    if z = y then not(List.mem y (free_vars body))
    else subst_good x y body
```

CS 421 Midterm

Name: _____

(10 cont)

c. (3pt) Write a function

var_subst : string -> string -> lamb_exp -> lamb_exp option

that returns the result of substituting the variable of the first name for the variable of the second name in the given expression, provided the substitution is valid. It should return None if the substitution is invalid.

```
let rec naïve_subst x y tm =
  match tm with Var z -> if z = x then Var y else tm
  | App(tm1, tm2) -> App(naïve_subst x y tm1, naïve_subst x y tm2)
  | Abs (z, body) -> if z = x then tm else Abs(z, naïve_subst x y body)
```

```
let subst x y tm =
  if subst_good x y tm then None else Some (naïve_subst x y tm)
```

CS 421 Midterm

Name: _____

Rules for type derivations:

Constants:

 $\overline{\Gamma \vdash n : \text{int}}$ (assuming n is an integer constant) $\overline{\Gamma \vdash \text{true} : \text{bool}}$ $\overline{\Gamma \vdash \text{false} : \text{bool}}$

Variables:

 $\overline{\Gamma \vdash x : \sigma}$ if $\Gamma(x) = \sigma$ Primitive operators ($\oplus \in \{+, -, *, \dots\}$): $\overline{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}$
 $\Gamma \vdash e_1 \oplus e_2 : \text{int}$ Relations ($\sim \in \{<, >, =, \leq, \geq\}$): $\overline{\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int}}$
 $\Gamma \vdash e_1 \sim e_2 : \text{bool}$

Connectives :

 $\overline{\Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash e_2 : \text{bool}}$ $\overline{\Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash e_2 : \text{bool}}$
 $\Gamma \vdash e_1 \ \&\& \ e_2 : \text{bool}$ $\Gamma \vdash e_1 \ || \ e_2 : \text{bool}$

If_then_else rule:

 $\overline{\Gamma \vdash e_1 : \text{bool} \quad \Gamma \vdash e_2 : \tau \quad \Gamma \vdash e_3 : \tau}$
 $\Gamma \vdash (\text{if } e_1 \text{ then } e_2 \text{ else } e_3) : \tau$

Application rule:

 $\overline{\Gamma \vdash e_1 : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash e_2 : \tau_1}$
 $\Gamma \vdash (e_1 \ e_2) : \tau_2$

fun rule:

 $\overline{[x : \tau_1] \cup \Gamma \vdash e : \tau_2}$
 $\Gamma \vdash \text{fun } x \rightarrow e : \tau_1 \rightarrow \tau_2$

let rule:

 $\overline{\Gamma \vdash e_1 : \tau_1 \quad [x : \tau_1] \cup \Gamma \vdash e_2 : \tau_2}$
 $(\text{let } x = e_1 \text{ in } e_2) : \tau_2$

let rec rule:

 $\overline{[x : \tau_1] \cup \Gamma \vdash e_1 : \tau_1 \quad [x : \tau_1] \cup \Gamma \vdash e_2 : \tau_2}$
 $(\text{let rec } x = e_1 \text{ in } e_2) : \tau_2$