

CS421 Fall 2008 Midterm 2

Thursday, November 6, 2008

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 18 pages to the exam. Please verify that you have all 18 pages.
- Please write your name and NetID in the spaces above, and also at the top of every page.

Problem	Possible Points	Points Earned
1	15	
2	18	
3	15	
4	12	
5	10	
6	20	
7	10	
PreTotal	100	
Extra Credit	8	
PostTotal	108	

CS 421 Midterm 2

Name: _____

1. (15 pts total) Using the rules described in class, and found at the back of this exam, give a full type derivation for the following type judgment:

$\{\} \vdash \text{let } y = \text{if true then false else let rec } f = \text{fun } y \rightarrow f \text{ y in } f \ 5 : \text{bool}$

Solution: Let C= Constant Rule, V= Variable Rule, I=If_then_else Rule, A=Application Rule, F=Function Rule, L = Let Rule, LR = Let Rec Rule

$$\begin{array}{c}
 \frac{}{\text{f:int} \rightarrow \text{bool}, \text{y:int} \vdash \text{f:int} \rightarrow \text{bool}} \text{V} \quad \frac{}{\text{f:int} \rightarrow \text{bool}, \text{y:int} \vdash \text{y:int}} \text{V} \\
 \frac{}{\text{f:int} \rightarrow \text{bool}} \text{V} \quad \frac{}{\text{f:int} \rightarrow \text{bool}} \text{C} \\
 \frac{}{\text{f:int} \rightarrow \text{bool}} \text{F} \quad \frac{}{\text{f:int} \rightarrow \text{bool}, \text{5:int}} \text{A} \\
 \frac{}{\text{f:int} \rightarrow \text{bool} \vdash \text{fun } y \rightarrow f \text{ y:int} \rightarrow \text{bool}} \text{C} \quad \frac{}{\text{f:int} \rightarrow \text{bool} \vdash \text{f } 5: \text{bool}} \text{LR} \\
 \frac{}{\text{f:int} \rightarrow \text{bool} \vdash \text{if true then false else let rec } f = \text{fun } y \rightarrow f \text{ y in } f \ 5: \text{bool}} \text{I} \quad \frac{}{\text{y:bool} \vdash \text{y:bool}} \text{V} \\
 \frac{}{\{\} \vdash \text{let } y = \text{if true then false else let rec } f = \text{fun } y \rightarrow f \text{ y in } f \ 5 \text{ in } y : \text{bool}} \text{L}
 \end{array}$$

2. (18 points) Give a most general unifier for the following unification problem. Capital letters **A,B,C,D,E** denote variables of unification and lower case letters **f, p, s, t** denote constructors. Show all your work by listing the operation being performed at each step, and all the results of that operation.

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

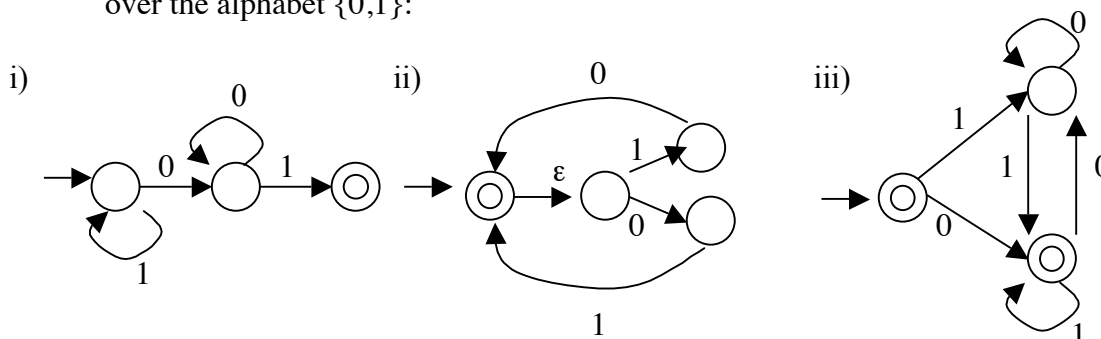
Solution:

$$\begin{aligned} & \text{Unify} \{ A = f(B, f(C, D)); B = f(E, D); p(s(E), t) = p(C, D) \} \\ & = \text{Unify} \{ B = f(E, D); p(s(E), t) = p(C, D) \} \circ \{ A \rightarrow f(B, f(C, D)) \} \text{ by Eliminate } A \\ & = \text{Unify} \{ p(s(E), t) = p(C, D) \} \circ \{ B \rightarrow f(E, D); A \rightarrow f(f(E, D), f(C, D)) \} \text{ by Eliminate } B \\ & = \text{Unify} \{ s(E) = C; t = D \} \circ \{ B \rightarrow f(E, D); A \rightarrow f(f(E, D), f(C, D)) \} \text{ by Decompose } p \\ & = \text{Unify} \{ C = s(E); t = D \} \circ \{ B \rightarrow f(E, D); A \rightarrow f(f(E, D), f(C, D)) \} \text{ by Orient } C \\ & = \text{Unify} \{ t = D \} \circ \{ C \rightarrow s(E); B \rightarrow f(E, D); A \rightarrow f(f(E, D), f(s(E), D)) \} \text{ by Eliminate } C \\ & = \text{Unify} \{ D = t \} \circ \{ C \rightarrow s(E); B \rightarrow f(E, D); A \rightarrow f(f(E, D), f(s(E), D)) \} \text{ by Orient } C \\ & = \text{Unify} \{ \} \circ \{ D \rightarrow t; C \rightarrow s(E); B \rightarrow f(E, t); A \rightarrow f(f(E, t), f(s(E), t)) \} \text{ by Eliminate } D \\ & = \{ D \rightarrow t; C \rightarrow s(E); B \rightarrow f(E, t); A \rightarrow f(f(E, t), f(s(E), t)) \} \end{aligned}$$

CS 421 Midterm 2

Name: _____

3. (15 pts total) Given the following nondeterministic finite state automata (NDFAs) over the alphabet $\{0,1\}$:



- a. (6pts) For each of the following strings, list every given NFA that accepts it:

i. 10101: _____ iii _____

ii. 0110: _____ ii _____

iii. 11001: _____ i _____ iii _____

- b. (9pts) For each of i, ii, and iii, write a regular expression that generates the same language as the corresponding NFA accepts. You should use the notation given in class: Regular expressions over an alphabet Σ are strings over Σ together with the five extra characters $(,)$, $*$, \vee , and ϵ . **No other symbols should occur in your regular expression, and they will not be accepted.**

i _____ 1^*00^*1 _____

ii _____ $(10 \vee 01)^*$ _____

iii _____ $\epsilon \vee (0 \vee 10^*1)(1^*\vee(00^*1)^*$ _____

4. (12 points) Given the following BNF grammar, for each of the following strings, give a parse tree for it if it parses starting with $\langle \text{Term} \rangle$, or write “None exists” if it does not parse starting with $\langle \text{Term} \rangle$.

$\langle \text{Term} \rangle ::= \langle \text{Atom} \rangle \mid \langle \text{Neg} \rangle \mid \langle \text{Par} \rangle$

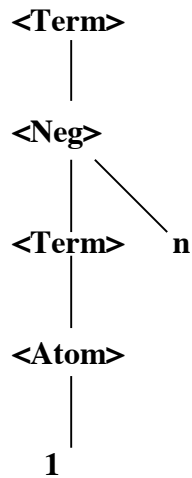
$\langle \text{Neg} \rangle ::= \langle \text{Term} \rangle \text{ n}$

$\langle \text{Par} \rangle ::= \langle \text{Term} \rangle \langle \text{Term} \rangle \text{ p}$

$\langle \text{Atom} \rangle ::= 0 \mid 1$

- a. (2 pts) **1 n**

Solution:



CS 421 Midterm 2

Name: _____

4. (cont) (12 pts total) Given the following BNF grammar, for each of the following strings, give a parse tree for it if it parses starting with **<Term>**, or write “None exists” if it does not parse starting with **<Term>**.

$$\langle \text{Term} \rangle ::= \langle \text{Atom} \rangle \mid \langle \text{Neg} \rangle \mid \langle \text{Par} \rangle$$
$$\langle \text{Neg} \rangle ::= \langle \text{Term} \rangle \text{ n}$$
$$\langle \text{Par} \rangle ::= \langle \text{Term} \rangle \langle \text{Term} \rangle \text{ p}$$
$$\langle \text{Atom} \rangle ::= 0 \mid 1$$

b. (5 pts) **0 1 p n 0 1 p**

Solution: None exists

4. (cont) (12 pts total) Given the following BNF grammar, for each of the following strings, give a parse tree for it if it parses starting with $\langle \text{Term} \rangle$, or write “None exists” if it does not parse starting with $\langle \text{Term} \rangle$.

$\langle \text{Term} \rangle ::= \langle \text{Atom} \rangle \mid \langle \text{Neg} \rangle \mid \langle \text{Par} \rangle$

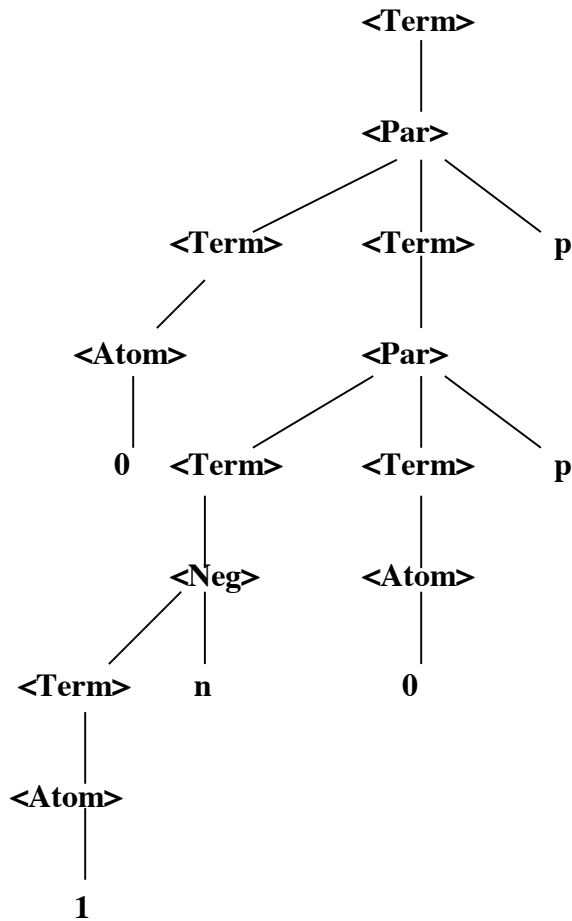
$\langle \text{Neg} \rangle ::= \langle \text{Term} \rangle n$

$\langle \text{Par} \rangle ::= \langle \text{Term} \rangle \langle \text{Term} \rangle p$

$\langle \text{Atom} \rangle ::= 0 \mid 1$

c. (5 pts) **0 1 n 0 p p**

Solution:



CS 421 Midterm 2

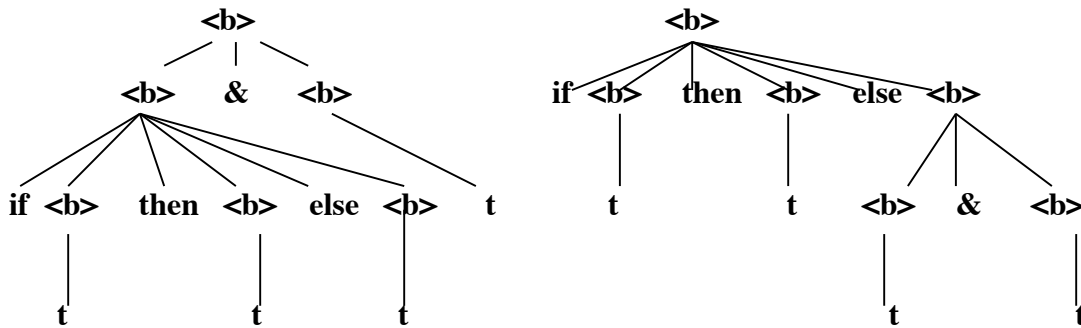
Name: _____

5.(10 pts) Consider the following extended BNF grammar over the alphabet of tokens **t, f, if, then, else, &**:

$$\langle b \rangle ::= t \mid f \mid \text{if } \langle b \rangle \text{ then } \langle b \rangle \text{ else } \langle b \rangle \mid \langle b \rangle \ \& \ \langle b \rangle$$

a. (3 pts) Show that the given grammar is ambiguous.

Solution: if t then t else t & t has two parse trees (also t & f & t)



b. (7 pts) Write a new grammar generating the same language as the one given above, but that is unambiguous, and such that **&** has higher precedence than **if_then_else_**, and such that **&** associates to the right.

Solution:

$$\langle b \rangle ::= \text{if } \langle b \rangle \text{ then } \langle b \rangle \text{ else } \langle b \rangle \mid \langle \text{noif} \rangle$$

$$\langle \text{noif} \rangle ::= \langle \text{noand} \rangle \ \& \ \langle b \rangle$$

$$\langle \text{noand} \rangle ::= t \mid f$$

CS 421 Midterm 2

Name: _____

6. (20 points) Consider the following grammar:

 $\langle \text{expr} \rangle ::= \langle \text{atom} \rangle \mid n \langle \text{expr} \rangle \mid p \langle \text{expr} \rangle \langle \text{expr} \rangle$ $\langle \text{atom} \rangle ::= 0 \mid 1 \mid (\langle \text{expr} \rangle)$

- a. (4 points) Write an Ocaml data type **token** for the tokens that lexer would generate as input to a parser for this grammar.

Solution:

```
type token = Ntk | Ptk | Ztk | Otk | LeftParen | RightParen
```

- b. (5 points) Write an Ocaml data type **expr** to represent parse trees for each of the syntactic categories in the given grammar.

Solution:

```
type expr = Atom of atom | N of expr | P of (expr * expr)
```

```
and atom = Zero | One | Parens of expr
```

CS 421 Midterm 2

Name: _____

6. (cont) Consider the following grammar:

```

<expr> ::= <atom> | n <expr> | p <expr> <expr>
<atom> ::= 0 | 1 | (<expr>)

```

c. (11 points) Using the types you gave in parts a. and b., write an Ocaml recursive descent parser **parse: token list -> expr** that, given a list of **tokens**, returns an **expr** representing an **<expr>** parse tree. You may omit handling the cases where parsing will fail.

Solution:

```

let rec expr tks =
  match tks with (Ntk::tks1) ->
    (match expr tks1 with (e, more_toks) -> (N e, more_toks))
  | (Ptk:: tks1) ->
    (match expr tks1 with (e1, tks2) ->
      (match expr tks2 with (e2, more_toks) -> (P (e1,e2),more_toks)))
  | _ -> (match atom tks with (a, more_toks) -> (Atom a, more_toks))
and atom tks =
  match tks with (Ztk::more_toks) -> (Zero, more_toks)
  | (Otk::more_toks) -> (One, more_toks)
  | (LeftParen::tks1) ->
    (match expr tks1 with (e, RightParen::more_toks) -> (Parens e, more_toks)
     | _ -> raise (Failure "no parse")) (* Not required *)
  | _ -> raise (Failure "no parse") (* Not required *)

let parse tks =
  match expr tks with (e, []) -> e
  | _ -> raise (Failure "no parse") (* Not required *)

```

CS 421 Midterm 2

Name: _____

7. (10 points) Given the following grammar over nonterminal **C** and **M**, and terminals **z**, **o**, **n**, **p** and **eol**, with start symbol **M**:

P1: **C ::= z**P2: **C ::= C n**P3: **C ::= C C p**P4: **M ::= C eol**

and Action and Goto tables generated by YACC for the above grammar:

State	Action					Goto	
	z	n	p	eol	<end>	C	M
st1	shift 3	error	error	error	error	st5	st2
st2	error	error	error	error	accept		
st3	reduce 1	reduce 1	reduce 1	reduce 1	error		
st5	shift 3	shift 6	error	shift 7	error	st8	
st6	reduce 2	reduce 2	reduce 2	reduce 2	error		
st7	reduce 4	reduce 4	reduce 4	reduce 4	error		
st8	shift 3	shift 6	shift 9	error	error	st8	
st9	reduce 3	reduce 3	reduce 3	reduce 3	error		

where st_i is state i , and **<end>** means we have reached the end of input, describe how the string **znp<eol>** would be parsed with an LR parser using these productions and tables by filling in the table on the next page. I have given you the first 5 cells to get started. **Caution:** There are strictly more rows than you will need, so do not expect to fill them all.

CS 421 Midterm 2

Name: _____

Stack	Current String	Action
<i>Empty</i>	znp<eol>	Initialize stack, go to state 1
st1	znp<eol>	Shift z; goto st1
st1:z:st3	znp<eol>	Reduce by P1, goto st5
st1:C:st5	znp<eol>	Shift z, goto st3
st1:C:st5:z:st3	np<eol>	Reduce by P1, goto st8
st1:C:st5:C:st8	np<eol>	Shift n, goto st6
st1:C:st5:C:st8:n:st6	p<eol>	Reduce by P2, goto st8
st1:C:st5:C:st8	p<eol>	Shift p. goto st9
st1:C:st5:C:st8:p:st9	<eol>	Reduce by P3, goto st5
st1:C:st5	<eol>	Shift <eol>, goto st7
st1:C:st5:<eol>:st7		Error

Comment: The information for st7 was inverted from my original intention. Therefore, the string does not successfully complete being parsed.

CS 421 Midterm 2

Name: _____

8. (Extra Credit) (8 pts total) In this problem you are asked to write a fragment of the code for **MP5, A Unification-Based Type Inferencer**. Below is a summary of the code given that you need to write your code.

Expressions and type are given by:

```

type exp = ... | VarExp of string | ConstExp of const | FunExp of string * exp | ...
type expType = TyVar of int | TyConst of (constTy * expType list)
type constTy = {name : string; arity : int}

```

You may use the side-effecting function **fresh:unit -> expType** that returns a fresh type variable. You may also use the following function for creating function types:

```

let mk_fun_ty ty1 ty2 = TyConst({name="->"; arity = 2},[ty1;ty2])

```

Environments, judgments, and proofs are given by

```

type env = (string * expType) list
type judgment = { gamma:env; exp:exp; expType:expType }
type proof = {antecedents : proof list; conclusion : judgment}

```

You may use the following environment functions:

```

val make_env : string -> expType -> env = <fun>
val lookup_env : env -> string -> expType option = <fun>
val sum_env : env -> env -> env = <fun>
val ins_env : env -> string -> expType -> env = <fun>

```

Your task is give a piece of the implementation of the function:

gather_ty_constraints : judgment -> (proof * (expType * expType) list) option, which takes in a judgment and returns **None** (on failure), or **Some** of a pair of a generic proof tree containing type variables, and a set of constraints to be unified. You were give the following code to start:

```

let rec gather_ty_constraints judgment =
  let {gamma = gamma; exp = exp; expType = expType} = judgment in
  match exp
  with ConstExp c ->
    let c_ty = const_signature c in
    Some ({antecedents = []; conclusion = judgment}, [(expType, c_ty)])

```

You are asked to implement the code for the case of functions:

$$\frac{\{x : \tau_1\} + \Gamma \vdash e : \tau_2 \mid C}{\Gamma \vdash \text{fun } x \rightarrow e : \tau \mid \{\tau = \tau_1 \rightarrow \tau_2\} \cup C}$$

CS 421 Midterm 2

Name: _____

Solution:

```
| FunExp (x,e) ->
  let (tau1, tau2) = (fresh(), fresh()) in
  (match gather_ty_constraints {gamma = (ins_env gamma x tau1);
                                exp = e;
                                expType = tau2}
   with Some (pf, const)
     -> Some ({antecedents = [pf]; conclusion = judgment},
              (expType, mk_fun_ty tau1 tau2) :: const)
   | None -> None)
```


CS 421 Midterm 2

Name: _____

Rules for type derivations:

Constants:

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

Variables:

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

Connectives :

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

If_then_else rule:

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

Application rule:

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

fun rule:

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

Let rule:

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

Let rec rule:

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

