CS421 Fall 2011 Midterm 2

Thursday, November 10, 2011

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 20 pages to the exam. Please verify that you have all 20 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	8	
2	16	
3	21	
4	12	
5	22	
6	12	
7	9	
PreTotal	100	
Extra Credit	10	
PostTotal	110	

CS 421 Midterm 2 **Name:**________________________________

1. (8 points) Recall that we use the following OCaml types to represent the types of PicoML, the language we have been implementing since MP4:

type typeVar = int **type constTy = {name : string; arity : int}**
type monoTy = TyVar of typeVar | TyConst of (constTy * monoTy list)

In MP6, you were asked to implement unification of systems of type constraints, where a system of type constraints is represented as a list of pairs of types. Each pair in the constraints represents an equation to be solved, and unification, if it succeeds returns a simultaneous substitution such that applying this substitution to every pair in the constraint system results in the first component becoming identical to the second. In addition to the base case, and the error case, there are four main steps to unification, as described in class. Give an implementation in OCaml of the **Decompose** step of

unify : (monoTy * monoTy) list -> (typeVar * monoTy) list option

described by:

If C is a nonempty constraint set such that $(s,t) \in C$ and $C' = C \setminus \{(s,t)\}$, then (**Decompose**) if s = **TyConst(**$name$, $[s_1,\ldots, s_n]$) and t = **TyConst(**$name'$, $[t_1,\ldots, t_m]$) then if $name = name'$ and $n = m$, then **unify** C = **unify** $(\{(s_1, t_1), \ldots, (s_1, t_1)\} \cup C'$, while if $name \neq name'$ or $n \neq m$, then no unifying substitution exists.

You may use **List.fold_left**, **List.fold_right**, **List.map** and **@**. Any other auxiliary functions should be defined.

Solution:

```
let rec addNewEqs lst1 lst2 acc =
   match lst1,lst2 with
     [],[] -> Some acc
   | t::tl, t'::tl' -> addNewEqs tl tl' ((t,t')::acc)
   | _ -> None

let rec unify c =
    match c with (s,t) :: c' ->
      (match (s,t) with (TyConst(str, tl), TyConst(str', tl'))::eqs when str=str' ->
        (match (addNewEqs tl tl' eqs) with
           None -> None
         | Some l -> unify l)
        | …)
       | … )
```

CS 421 Midterm 2 **Name:**___________________________________

2. (16 pts total) In parts a) and c) you are to give a regular expression generating each of the following languages over the alphabet Σ = {**a**, **b, c**}. You should use the notation fro basic regular expressions given in class: Regular expressions over an alphabet Σ are strings over Σ together with the five extra characters (,), *, ∨, and ε. **No other symbols should occur in your regular expression, and they will not be accepted**. In parts b) and d) you are asked to give a regular grammar for the languages in a) and c).

   a. (4 pts) Give a regular expression for the set of all strings, where **a** occurs exactly in positions that are multiples of 3. We will number positions starting from the left with 1.

      Solution:

      ((b ∨ **c**) (b ∨ **c**) **a**)* (b ∨ **c** ∨ ε) (b ∨ **c** ∨ ε)

   b. (4 pts) Give a right regular grammar for the set of all strings, where **a** occurs exactly in positions that are multiples of 3. We will number positions starting from the left with 1.

Solution:

S ::= b T | c T | ε
T ::= b R | c R | ε
R ::= a S

Start symbol S

CS 421 Midterm 2 **Name:___________________________________**

2. (cont) In parts a) and c) you are to give a regular expression generating each of the following languages over the alphabet Σ = {**a**, **b, c**}. You should use the notation fro basic regular expressions given in class: Regular expressions over an alphabet Σ are strings over Σ together with the five extra characters (,), *, ∨ , and **ε**. **No other symbols should occur in your regular expression, and they will not be accepted**. In parts b) and d) you are asked to give a regular grammar for the languages in a) and c).

   c. (4 pts) Give a regular expression for the set of all strings such that after each **a** eventually there is a **b** (not necessarily adjacent to the **a**).

Solution:

(b ∨ **c**)* **(a** (a ∨ b ∨ **c**)*)**c**)* (b ∨ **c**)*

   d. (4 pts) Give a regular grammar for the set of all strings such that after each **a** eventually there is a **b** (not necessarily adjacent to the **a**).

Solution:

S::= b S | c S | a C | ε
C::= a C | b C | c S

Start symbol S

3. (21 points total) Given the following BNF grammar, for each of the following strings, give a parse tree for it, if it parses starting with **E,** or write "None exists" if it does not parse starting with **E.** The terminals for this grammar are **{x, y, z, p, n, c, q, l}**.

E ::= V | E E p | E n | E E B c
B ::= E E q | E l
V::= x | y | z

a. (5 pts) **x n y n p**

Solution:

```
              E
           /  |  \
        E     E     p
     /  |     |  \
   E    n     E   n
   |          |
   V          V
   |          |
   x          y
```

3. (cont) (21 points total) Given the following BNF grammar, for each of the following strings, give a parse tree for it, if it parses starting with **E,** or write "None exists" if it does not parse starting with **E.** The terminals for this grammar are **{x, y, z, p, n, c, q, l}**.

E ::= V | E E p | E n | E E B c
B ::= E E q | E l
V::= x | y | z

b. (8 pts) **x y p x l c n**

Solution:

None exists

3. (cont) (21 points total) Given the following BNF grammar, for each of the following strings, give a parse tree for it, if it parses starting with **E,** or write "None exists" if it does not parse starting with **E.** The terminals for this grammar are **{x, y, z, p, n, c, q, l}**.

E ::= V | E E p | E n | E E B c
B ::= E E q | E l
V ::= x | y | z

c. (8 pts) **z x y x z p p n l c**

Solution:

```
            E
        /   /  \   \
     E     E   B    c
     |     |     |  \
     V     V    E    l
     |     |     |  \
     z     x    E    n
               /   |  \
             E    E    p
           /     / |\
         V    E   E   p
         |    |   |
         y    V   V
              |   |
              x   z
```

CS 421 Midterm 2 **Name:____________________________________**
Workspace

4. (12 points total) Consider the following grammar over the alphabet {**+, #, (,), x, y, z**}:

<exp> ::= <var> | <exp> + | <exp> # <exp> | (<exp>)
<var> ::= x | y | z

a. (4 pts) Show that the above grammar is ambiguous (using the definition of an ambiguous grammar).

Solution: x # y + has two parses

```
        <exp>                          <exp>
      /   |  \                       /       \
 <exp>  #  <exp>                 <exp>         +
   |       /      \             /   |  \
   x     <exp>     +        <exp>  #  <exp>
           |                  |          |
           y                  x          y
```

4. (cont) Consider the following grammar over the alphabet {**+, #, (,), x, y, z**}:

<exp> ::= <var> | <exp> + | <exp> # <exp> | (<exp>)
<var> ::= x | y | z

b. (8 pts) Write a new grammar accepting the same language accepted by **<exp>** above, and such that **#** (that is **<exp> # <exp>**) associates to the right and has higher precedence than + (that is **<exp> +**).

Solution:

<exp> ::= <no_plus> | <exp> + | <exp> + # <no_plus>
<no_plus> ::= <atom> | <atom> # <no_plus>
<atom> ::= x | y | z | (<exp>)

Start symbol <exp>

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

<expr> ::= <prop> | <prop> ∧ <expr> | ¬ <prop>
<prop> ::= true | x | (<expr>)

a. (3 pts) Write an Ocaml data type **token** for the tokens that lexer would generate as input to a parser for this grammar.

Solution:

```
type token = AndTk | NotTk | TrueTk | XTk | LeftParenTk | RightParenTk
```

b. (4 pts) Write Ocaml data types **expr** and **prop** to represent parse trees for each of the syntactic categories in the given grammar.

Solution:

```
type  expr = Prop2Expr of prop |And of prop*expr | Not of prop
and prop = True | X | Parens of expr
```

CS 421 Midterm 2 **Name:__________________________________**

5. (cont) Consider the following grammar:

<expr> ::= <prop> | <prop> ∧ <expr> | ¬ <prop>
<prop> ::= true | x | (<expr>)

c. (15 pts) 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 **token**s, returns an **expr** representing an **<expr>** parse tree. You should use

raise (Failure "no parse")

for cases where no parse exists.

Solution:

```
let rec expr tokens =
   match tokens with (NotTk::tokens_after_not) ->
     (match prop tokens_after_not  with (p, more_tokens) -> (Not p, more_tokens) )
   | _ ->
     (match prop tokens with (p, more_tokens) ->
       (match more_tokens with (AndTk::tokens_after_and) ->
         (match expr tokens_after_and with (e, rem_tokens) -> (And(p,e), rem_tokens))
        | _ -> (Prop2Expr p, more_tokens) ))

and prop tokens =
     match tokens with (TrueTk :: rem_tokens) -> (True,  rem_tokens)
       | (XTk :: rem_tokens) -> (X, rem_tokens)
       | (LeftParenTk :: tokens_after_lparen) ->
         (match expr tokens_after_lparen with (e, more_tokens) ->
           (match more_tokens with RightParenTk::rem_tokens -> (Parens e, rem_tokens)
             | _ -> raise (Failure "no parse")))
       | _ -> raise (Failure "no parse")

let parse tokens =
    match expr tokens with (e, [ ]) -> e
     | _ -> raise (Failure "no parse")
```

6. (12 points) Given the following grammar over nonterminal **<b>** and **<e>**, and terminals **z, l, r, p** and **eof**, with start symbol **<b>**:

P0: **<s> ::= <e> eof**
P1: **<e> ::= <e> <e> <b> c**
P2: **<e>::= x**
P3: **<b>::= <e> l**

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

State	**Action**					**Goto**		
	x	**c**	**l**	**[eof]**		**<b>**	**<e>**	**<s>**
st1	**s 3**	**err**	**err**	**err**			**st4**	**st2**
st2	**err**	**err**	**err**	**a**				
st3	**r 2**	**r 2**	**r 2**	**r 2**				
st4	**s 3**	**s 4**	**err**	**r 0**			**st5**	
st5	**s 3**	**err**	**err**	**err**		**st7**	**st6**	
st6	**s 3**	**err**	**s 8**	**err**		**st7**	**st6**	
st7	**err**	**s 9**	**err**	**err**				
st8	**r 3**	**r 3**	**r 3**	**r 3**				
st9	**r 1**	**r 1**	**r 1**	**r 1**				

where **st*i*** is state *i*, **s *i*** means **shift *i***, **r *i*** means **reduce *i*, a** means **accept** and [**eof]** means we have reached the end of input, describe how the string **xxxlc[eof]** 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.

Stack	**Current String**	**Action**
Empty	**xxxlc[eof]**	Initialize stack, go to state 1
st1	**xxxlc[eof]**	Shift, go to st3
st1::x::st3	xxlc[eof]	Reduce by P2, go to st4
st1::<e>::st4	xxlc[eof]	Shift, go to st3
st1::<e>::st4::x::st3	xlc[eof]	Reduce by P2, go to st5
st1::<e>::st4::<e>::st5	xlc[eof]	Shift, go to st3
st1::<e>::st4::<e>::st5::x::st3	lc[eof]	Reduce by P2, go to st6
st1::<e>::st4::<e>::st5::<e>::st6	lc[eof]	Shift, go to st8
st1::<e>::st4::<e>::st5::<e>::st6::l::st8	c[eof]	Reduce by P3, go to st7
st1::<e>::st4::<e>::st5::<b>::st7	c[eof]	Shift, go to st9
st1::<e>::st4::<e>::st5::<b>::st7::c::st9	[eof]	Reduce by P1, go to st4
st1::<e>::st4	[eof]	Table says: Reduce by P0, go to st2 This is an error in the table; should be accept

7. (9 points) Give a natural semantics (a.k.a. structured operational semantics) derivation of the evaluation of:

   ((if x + 2 > 5 then x:= 3 else x := x + 2), {x → 1})

   You may omit labeling your rule uses.

Solution:

Let EvalXPlus2 =

$$\frac{(x, \{x \to 1\}) \Downarrow 1 \quad (2, \{x \to 1\}) \Downarrow 2 \quad 1+2 = 3}{((x + 2), \{x \to 1\}) \Downarrow 3}$$

Then the derivation is:

$$\frac{\dfrac{\text{EvalXPlus2} \quad (5, \{x \to 1\}) \Downarrow 5 \quad (3 > 5) = \text{false}}{((x + 2 > 5), \{x \to 1\}) \Downarrow \text{false}} \qquad \dfrac{\text{EvalXPlus2}}{((x := x + 2), \{x \to 1\}) \Downarrow \{x \to 3\}}}{((\text{if } x + 2 > 5 \text{ then } x := 3 \text{ else } x := x + 2), \{x \to 1\}) \Downarrow \{x \to 3\}}$$

CS 421 Midterm 2 **Name:**___________________________________

8. (Extra Credit) (10 points total) Give the natural semantics rule(s) for the command

for $I = E_1$ to E_2 do C **od** in a memory m

where E_1 and E_2 are evaluated to fixed values before the loop is entered, the loop is entered only if the value of E_1 is less than or equal to the value of E_2 , I is incremented at the end of each pass through the loop, after C has been evaluated, and if the loop is entered, it is terminated when, after this incrementing, the value of I is strictly greater than the value of E_2 computed initially. You may assume that evaluating expressions does not alter the memory.

Solution:

$$\frac{(E_1, m) \Downarrow V_1 \quad (E_2, m) \Downarrow V_2 \quad ((\text{while } I <= V_2 \text{ do } C \text{ od}), m[I \leftarrow V_1]) \Downarrow m'}{((\text{for } I = E_1 \text{ to } E_2 \text{ do } C \text{ od}), m) \Downarrow m'}$$

CS 421 Midterm 2 **Name:___________________________________**

Workspace

CS 421 Midterm 2 **Name:__________________________________**

Simple Imperative Programming Language

$I \in Identifiers$ $\quad$ $N \in Numerals$

B ::= true | false | B & B | B or B | not B | $E < E$ | $E = E$

E::= N | I | $E + E$ | $E * E$ | $E - E$ | $- E$

C::= skip | C;C | I ::= E | if B then C else C fi | while B do C od

Natural Semantics Rules

Identifiers: $\dfrac{}{(I,m) \Downarrow m(I)}$ **Numerals are values:** $\dfrac{}{(N,m) \Downarrow N}$

Booleans: $\dfrac{}{(\text{true},m) \Downarrow \text{true}}$ $\quad$ $\dfrac{}{(\text{false},m) \Downarrow \text{false}}$

$$\frac{(B, m) \Downarrow \text{false}}{(B \,\&\, B', m) \Downarrow \text{false}} \qquad \frac{(B, m) \Downarrow \text{true} \quad (B', m) \Downarrow b}{(B \,\&\, B', m) \Downarrow b}$$

$$\frac{(B, m) \Downarrow \text{true}}{(B \text{ or } B', m) \Downarrow \text{true}} \qquad \frac{(B, m) \Downarrow \text{false} \quad (B', m) \Downarrow b}{(B \text{ or } B', m) \Downarrow b}$$

$$\frac{(B, m) \Downarrow \text{true}}{(\text{not } B, m) \Downarrow \text{false}} \qquad \frac{(B, m) \Downarrow \text{false}}{(\text{not } B, m) \Downarrow \text{true}} \qquad \frac{(E, m) \Downarrow U \quad (E', m) \Downarrow V \quad U \sim V = b}{(E \sim E', m) \Downarrow b}$$

(~ a relation)

Arithmetic Expressions:
(op an arith binary operation)

$$\frac{(E, m) \Downarrow U \quad (E', m) \Downarrow V \quad U \; op \; V = N}{(E \; op \; E', m) \Downarrow N}$$

Commands:

Skip: $\dfrac{}{(\text{skip}, m) \Downarrow m}$ **Assignment:** $\dfrac{(E,m) \Downarrow V}{(I::=E,m) \Downarrow m[I \leftarrow V]}$

Sequencing: $\dfrac{(C,m) \Downarrow m' \quad (C',m') \Downarrow m''}{(C;C', m) \Downarrow m''}$

If Then Else Command:

$$\frac{(B,m) \Downarrow \text{true} \quad (C,m) \Downarrow m'}{(\text{if } B \text{ then } C \text{ else } C' \text{ fi}, m) \Downarrow m'} \qquad \frac{(B,m) \Downarrow \text{false} \quad (C',m) \Downarrow m'}{(\text{if } B \text{ then } C \text{ else } C' \text{ fi}, m) \Downarrow m'}$$

While Command:

$$\frac{(B,m) \Downarrow \text{false}}{(\text{while } B \text{ do } C \text{ od}, m) \Downarrow m} \qquad \frac{(B,m) \Downarrow \text{true} \quad (C,m) \Downarrow m' \quad (\text{while } B \text{ do } C \text{ od}, m') \Downarrow m''}{(\text{while } B \text{ do C od}, m) \Downarrow m''}$$

CS 421 Midterm 2 **Name:___________________________________**

Transition Semantics:

Identifiers: $(I,m) \to (m(I), m)$ **Numerals are values.**

Booleans:

$$\frac{}{(\text{false} \,\&\, B, m) \to (\text{false}, m)} \qquad \frac{}{(\text{true} \,\&\, B, m) \to (B,m)} \qquad \frac{(B, m) \to (B'', m)}{(B \,\&\, B', m) \to (B'' \,\&\, B', m)}$$

$$\frac{}{(\text{true or } B, m) \to (\text{true}, m)} \qquad \frac{}{(\text{false or } B, m) \to (B,m)} \qquad \frac{(B, m) \to (B'', m)}{(B \text{ or } B', m) \to (B'' \text{ or } B',m)}$$

$$\frac{}{(\text{not true}, \text{m}) \to (\text{false}, m)} \qquad \frac{}{(\text{not false}, \text{m}) \to \text{true}, m)} \qquad \frac{(B, m) \to (B', m)}{(\text{not } B, m) \to (\text{not } B', m)}$$

$$\frac{(E, m) \to (E'',m)}{(E \sim E', m) \to (E''{\sim}E',m)} \qquad \frac{(E, m) \to (E',m)}{(V \sim E, m) \to (V{\sim}E',m)} \quad \sim \text{ a relation}$$

$$\frac{}{(U \sim V, m) \to (\text{true}, m) \text{ or } (\text{false}, m), \text{ depending on whether } U \sim V \text{ holds or not}}$$

Arithmetic Expressions:

$$\frac{(E, m) \to (E'',m)}{(E \; op \; E', m) \to (E'' \; op \; E',m)} \qquad \frac{(E, m) \to (E',m)}{(V \; op \; E, m) \to (V \; op \; E',m)}$$

$$\frac{}{(U \; op \; V, m) \to (N,m) \quad \text{where } N = U \; op \; V}$$

Commands:

$$\frac{}{(\text{skip}, m) \to m} \qquad \frac{(E,m) \to (E',m)}{(I::=E,m) \dashrightarrow (I::=E',m)} \qquad \frac{}{(I::=V,m) \to m[I \leftarrow V\,]}$$

$$\frac{(C,m) \to (C'',m'\,)}{(C;C', m) \to (C'';C',m'\,)} \qquad \frac{(C,m) \to m'}{(C;C', m) \to (C',m'\,)}$$

If Then Else Command:

$$\frac{}{(\text{if true then } C \text{ else } C' \text{ fi}, m) \to (C, m)} \qquad \frac{}{(\text{if false then } C \text{ else } C' \text{ fi}, m) \to (C', m)}$$

$$\frac{(B,m) \to (B',m)}{(\text{if } B \text{ then } C \text{ else } C' \text{ fi}, m) \to (\text{if } B' \text{ then } C \text{ else } C' \text{ fi}, m)}$$

While Command:

$$\frac{}{(\text{while } B \text{ do } C \text{ od}, m) \to (\text{if } B \text{ then } C; \text{ while } B \text{ do } C \text{ od else skip fi, m})}$$